# ANALISIS
# Big Data

Dr. Joseph Teguh Santoso, S.Kom., M.Kom.

YAYASAN PRIMA AGUS TEKNIK

# ANALISIS Big Data

Dr. Joseph Teguh Santoso, S.Kom., M.Kom.

## BIODATA PENULIS

Dr. Joseph Teguh Santoso, S.Kom, M.Kom adalah Rektor dari Universitas Sains & Teknologi Komputer (Universitas STEKOM) Semarang yang memiliki banyak pengalaman praktis dalam bidang *e-commerce*sejak Tahun 2002. Beliau mempunyai 3 (tiga) toko *Official Online Store* di China untuk merek Sepeda Raleigh, dengan omzet tahunan pada Tahun 2019 mencapai lebih dari Rp. 35 Milyar rupiah dan terus meningkat. Dr. Joseph T.S memiliki lisensi tunggal sepeda merek "Raleigh" untuk penjualan *Online* di seluruh China. Di samping itu beliau juga memiliki pabrik sepeda dan sepeda listrik merek "Fengjiu", yaitu Pabrik Sepeda Listrik yang masih tergolong kecil di China. Pengalaman beliau malang melintang di dunia *online store* di China seperti Alibaba, Tmall, Taobao, JD, Aliexpress sangat membantu mahasiswa untuk memiliki pengalaman teknis dan praktis untuk membuka toko *online* bersama beliau.

YAYASAN PRIMA AGUS TEKNIK
Jl. Majapahit No. 605 Semarang
Telp. (024) 6723456. Fax. 024-6710144
Email : penerbit_ypat@stekom.ac.id

ISBN 978-623-95042-4-3

# ANALISIS Big Data

Dr. Joseph Teguh Santoso, S.Kom., M.Kom.

YAYASAN PRIMA AGUS TEKNIK

**Analisis Big Data**

**Penulis :**
Dr. Joseph Teguh Santoso, S.Kom., M.Kom.

**ISBN :**
9 786239 504243
**Editor :**
Dr. Joseph Teguh Santoso, S.Kom., M.Kom.

**Penyunting :**
Dr. Mars Caroline Wibowo. S.T., M.Mm.Tech

**Desain Sampul dan Tata Letak :**
Irdha Yunianto

**Penebit** :
Yayasan Prima Agus Teknik

**Redaksi :**
Jl. Majapahit no 605 Semarang
Telp. (024) 6723456
Fax. 024-6710144
Email : penerbit_ypat@stekom.ac.id

**Distributor Tunggal :**
STEKOM Semarang
Jl. Majapahit no 605 Semarang
Telp. (024) 6723456
Fax. 024-6710144
Email : info@stekom.ac.id

# Kata Pengantar

Puji syukur kami panjatkan kehadirat Tuhan Yang Maha Esa karena dengan rahmat,karunia, serta taufik dan hidayah-Nya kami dapat menyelesaikan penyusunan buku ***Analisis Big Data*** dengan harapan untuk dapat dipergunakan oleh kalangan para akademisi.

Tujuan utama penyusunan buku ini adalah untuk memudahkan mahasiswa dalam memahami dan menguasai dasar-dasar aplikasi yang digunakan dalam program penyimpanan dan penghitungan data yang super besar. Buku teks ini merupakan pedoman jenis-jenis komputasi yang digunakan dalam pengolahan data yang besar. Penjabaran dan penyampaian analisis Big Data dalam buku ini sangat jelas dan lengkap sehingga memudahkan mahasiswa untuk memahami dan menjalankan program pengolahan data komputasi super besar.

Semoga buku ini dapat dipahami bagi siapapun yang membacanya. Sekiranya buku yang telah disusun ini dapat berguna bagi kami sendiri maupun orang yang membacanya. Sebelumnya kami mohon maaf apabila terdapat kesalahan kata-kata yang kurang berkenan dan kami memohon kritik dan saran yang membangun demi perbaikan di masa depan.

Semarang, Nopember 2020

**Dr. Joseph Teguh Santoso, S.Kom, M.Kom**
Penulis

# DAFTAR ISI


# Konsep Analisis Big Data

# Pengantar Big Data

Bab 1

Bab ini mencakup :

- Pengertian Analisis
- Definisi Big Data
- Karakteristik Big Data
- Contoh Spesifik Domain dari Big Data
- Alur Analisis untuk Big Data
- Big Data Stack
- Pemetaan Alur Analisis ke Big Data Stack
- Pola Analisis

## 1.1 Pengertian Analisis

Pengertian Analisis adalah aktivitas yang memuat sejumlah kegiatan seperti mengurai, membedakan, memilah sesuatu untuk digolongkan dan dikelompokan kembali menurut kriteria tertentu kemudian dicari kaitannya dan ditafsir maknannya. Dalam definisi lain, Analisis adalah proses menyadari sesuatu dengan teliti dan hati-hati, atau menggunakan data dan metode statistik untuk memahami atau menjelaskan hal tersebut. Definisi ini merupakan rumusan umum tentang analisis.

Analisis adalah istilah luas yang mencakup proses, teknologi, kerangka kerja, dan algoritma untuk mengekstrak wawasan yang bermakna dari data. Data Mentah itu sendiri tidak memiliki arti sampai dikontekstualisasikan dan diolah menjadi informasi yang berguna. Analisis adalah proses mengekstraksi dan membuat informasi dari raw data dengan menyaring, memproses, mengkategorikan, memadatkan, dan mengontekstualisasikan data. Informasi yang diperoleh ini kemudian diatur dan disusun untuk pengetahuan tentang sistem dan / atau penggunanya, lingkungannya, dan operasi serta kemajuannya menuju tujuannya, sehingga membuat sistem lebih cerdas dan lebih efisien.

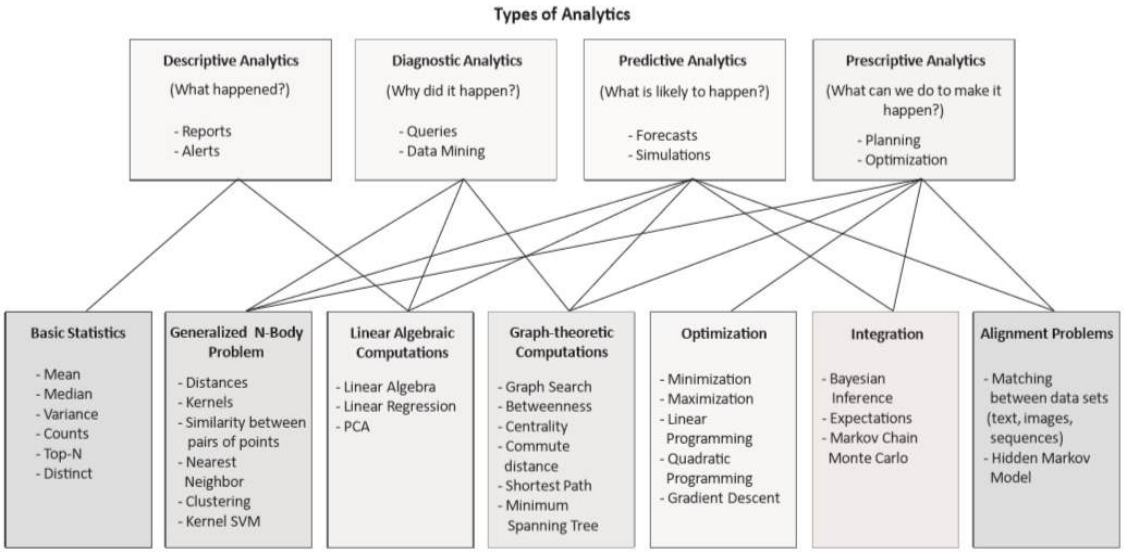
Pilihan teknologi, algoritma, dan kerangka kerja untuk analisis didorong oleh tujuan analisis dari aplikasi. Tujuan dari tugas analisis antara lain :

(1) untuk memprediksi sesuatu (misalnya, apakah transaksi itu tidak akan terjadi, apakah akan terjadi pada hari tertentu, atau apakah tumor itu jinak atau ganas),

(2) untuk menemukan pola dalam data (misalnya, menemukan pola urutan 10 hari terdingin dalam setahun, menemukan halaman mana yang paling banyak dikunjungi di situs web tertentu, atau menemukan selebriti yang paling banyak ditelusuri pada tahun tertentu),
(3) menemukan hubungan dalam data (misalnya, menemukan artikel berita yang serupa, menemukan pasien serupa dalam sistem catatan kesehatan elektronik, menemukan produk terkait di situs *web e-Commerce*, menemukan gambar serupa, atau menemukan korelasi antara item berita dan harga saham).

*National Research Council* [1] telah melakukan karakterisasi tugas komputasi untuk analisis Big Data-yang masif (disebut tujuh "raksasa"). Tugas komputasi ini meliputi: (1) Statistik Basis, (2) Masalah Umum N-Body, (3) Komputasi Aljabar Linear, (4) Komputasi Grafik-Teoretik, (5) Optimasi, (6) Integrasi dan (7) Masalah Alignment. Karakterisasi tugas komputasi ini bertujuan untuk memberikan taksonomi tugas yang telah terbukti berguna dalam analisis data dan mengelompokkan mereka secara kasar menurut struktur matematika dan strategi komputasi.

Kami juga akan membuat pemetaan antara jenis analisis dari tujuh komputasi super besar. Gambar 1.1 menunjukkan pemetaan antara jenis analisis dan tujuh komputasi "raksasa ".




Gambar 1.1 *Pemetaan antara Jenis Analisis dan Tujuh Raksasa Komputasi*

### 1.1.1 Analisis Deskriptif

Analisis deskriptif dapat digunakan untuk mengolah data kuantitatif. Cara ini dilakukan untuk melihat performa data di masa lalu agar dapat mengambil kesimpulan dari hal tersebut. Metode ini mengedepankan deskripsi yag memungkinkan kamu untuk belajar dari hal lalu. Biasanya, metode analisis jenis ini diaplikasikan pada data dengan volume yang sangat besar seperti data sensus misalnya. Analisis deskriptif bertujuan untuk menjawab "Apa yang terjadi?" Sebagian besar analisis yang dilakukan saat ini adalah analisis berbasis deskriptif melalui penggunaan fungsi statistik seperti hitungan, maksimum, minimum, mean, top-N, dan persentase. Statistik ini membantu dalam mendeskripsikan pola dalam data dan menyajikan data dalam bentuk ringkasan. Misalnya, menghitung jumlah total "suka" untuk pos tertentu, menghitung rata-rata curah hujan bulanan atau menemukan jumlah orang yang mengunjungi sebuah situs web perbulannya. Analisis deskriptif berguna untuk meringkas data. Di Bab-3, kami menjelaskan implementasi berbagai pola MapReduce untuk analisis deskriptif (seperti fungsi Count, Max/Min, Average, Distinct, dan Top-N).

Di antara tujuh tugas komputasi seperti yang ditunjukkan pada Gambar 1.1, tugas seperti Statistik Dasar dan Perhitungan Aljabar Linier dapat digunakan untuk analisis deskriptif.

### 1.1.2 Analisis Diagnostik

Analisis diagnostik terdiri dari analisis data lalu lintas dan data tentang berbagai alasan mengapa sebuah kejadian tertentu terjadi. Analisis diagnostik bertujuan untuk menjawab "Mengapa sesuatu itu terjadi?" Mari kita pertimbangkan contoh sistem yang mengumpulkan dan menganalisis data sensor dari mesin untuk memantau kesehatan dan memprediksi kegagalan. Meskipun analisis deskriptif dapat berguna untuk meringkas data dengan menghitung berbagai statistik (seperti mean, Count, Max/Min, Average, Distinct, dan Top-N), analisis diagnostik dapat memberikan lebih banyak wawasan tentang mengapa kesalahan tertentu telah terjadi berdasarkan pola dalam data sensor untuk kesalahan sebelumnya.

Di antara tujuh tugas komputasi, tugas komputasi seperti Perhitungan Aljabar Linier, Masalah Umum N-Body, dan Komputasi Grafik-teoretis dapat digunakan untuk analisis diagnostik.

### 1.1.3 Analisis Prediktif

Analisis prediktif terdiri dari memprediksi terjadinya suatu kejadian atau kemungkinan hasil dari suatu peristiwa atau memperkirakan nilai masa depan menggunakan model prediksi. Analisis prediktif bertujuan untuk menjawab "Apa yang mungkin terjadi?" Misalnya, analisis prediktif dapat digunakan untuk memprediksi kapan kesalahan akan terjadi pada mesin, memprediksi apakah tumor jinak atau ganas, memprediksi terjadinya keadaan darurat alami (misalnya, peristiwa kebakaran hutan atau banjir bandang) atau memperkirakan tingkat polusi. Analisis Prediktif dilakukan menggunakan model prediktif yang dukung dengan data yang tersedia. Model-model ini mempelajari pola dan tren dari data yang ada dan memprediksi terjadinya suatu kejadian atau kemungkinan hasil dari suatu kejadian (model klasifikasi) atau angka perkiraan (model regresi). Akurasi model prediksi bergantung pada kualitas dan volume data yang ada yang tersedia untuk memperkuat model, sehingga semua pola dan tren dalam data yang ada dapat dipelajari secara akurat. Sebelum model digunakan untuk prediksi, model harus divalidasi dengan data yang ada. Pendekatan tipikal yang diadopsi saat mengembangkan model prediksi adalah dengan membagi data yang ada menjadi kumpulan data pelatihan dan pengujian (misalnya 75% data digunakan untuk pelatihan dan 25% data digunakan untuk menguji model prediksi). Dalam Bab-11, kami menyediakan implementasi berbagai algoritme untuk analisis prediktif (termasuk algoritme pengelompokan, klasifikasi, dan regresi) menggunakan kerangka kerja seperti Spark MLlib dan H2O.

Di antara tujuh tugas komputasi, tugas-tugas seperti Perhitungan Aljabar Linear, Masalah Umum N-Body, Komputasi Grafik-teoretis, Masalah Integrasi dan Alignment dapat digunakan untuk analisis prediktif.

### 1.1.4 Analisis Preskriptif

Sementara analisis prediktif menggunakan model prediksi untuk memprediksi kemungkinan hasil dari suatu peristiwa, analisis preskriptif menggunakan beberapa model prediksi untuk memprediksi berbagai hasil dan tindakan terbaik untuk setiap hasil. Analisis preskriptif bertujuan untuk menjawab "Apa yang dapat kita lakukan untuk mewujudkannya?" Analisis Preskriptif dapat memprediksi hasil yang mungkin berdasarkan pilihan tindakan saat ini. Kami dapat mempertimbangkan analisis preskriptif sebagai jenis analisis yang menggunakan model prediksi berbeda untuk input berbeda. Analisis preskriptif menentukan tindakan atau opsi terbaik untuk diikuti dari opsi yang tersedia. Misalnya, analisis preskriptif dapat digunakan untuk menggambarkan pengobatan terbaik untuk pengobatan bagi

pasien berdasarkan hasil dari berbagai obat untuk pasien serupa. Contoh lain dari analisis preskriptif akan menyarankan paket data seluler terbaik untuk pelanggan berdasarkan pola penjelajahan pelanggan.

Di antara tujuh tugas komputasi, tugas-tugas seperti Masalah Umum N-Body, Perhitungan Graphtheoretic, Masalah Optimasi dan Alignment dapat digunakan untuk analisis preskriptif.

## 1.2 Definisi Big Data

Bigdata adalah kumpulan dari sekumpulan data yang volume, kecepatan, atau berbagai target yang sulit untuk disimpan, dikelola, diproses, dan dianalisis menggunakan database tradisional dan alat pemrosesan data. Dalam beberapa tahun terakhir, telah terjadi pertumbuhan eksponensial dalam data terstruktur dan tidak terstruktur yang dihasilkan oleh teknologi informasi, industri, perawatan kesehatan, Internet of Things, dan sistem lainnya.

Menurut perkiraan IBM, 2,5 quintillion byte data dibuat setiap hari [9]. Laporan terbaru dari DOMO memperkirakan jumlah data yang dihasilkan setiap menit pada platform online populer [10]. Berikut adalah beberapa bagian penting dari data dari laporan:

- Pengguna Facebook berbagi hampir 4,16 juta konten
- Pengguna Twitter mengirim hampir 300.000 tweet
- Pengguna Instagram menyukai hampir 1,73 juta foto
- Pengguna YouTube mengupload 300 jam konten video baru
- Pengguna Apple mengunduh hampir 51.000 aplikasi
- Pengguna Skype melakukan hampir 110.000 panggilan baru
- Amazon menerima 4300 pengunjung baru
- Penumpang Uber mengambil 694 perjalanan
- Pelanggan Net flix streaming hampir 77.000 jam video
-

Big Data memiliki potensi untuk mendukung aplikasi pintar generasi berikutnya yang akan memanfaatkan kekuatan data untuk membuat aplikasi menjadi cerdas. Aplikasi Big Data menjangkau berbagai domain seperti web, ritel dan pemasaran, perbankan dan keuangan, industri, perawatan kesehatan, lingkungan, Internet of Things dan sistem fisik siber.

Analisis Big Data berkaitan dengan pengumpulan, penyimpanan, pemrosesan, dan analisis data berskala besar ini. Alat dan kerangka kerja khusus diperlukan untuk analisis Big Data jika: (1) volume data yang terlibat sangat besar sehingga sulit untuk menyimpan, memproses, dan menganalisis data pada satu mesin, (2) kecepatan data sangat tinggi dan data tersebut perlu dianalisis secara real-time, (3) terdapat berbagai data yang terlibat, yang dapat terstruktur, tidak terstruktur atau semi-terstruktur, dan dikumpulkan dari berbagai sumber data, (5) berbagai jenis analisis perlu dilakukan untuk mengekstrak nilai dari data tersebut seperti deskriptif, diagnostik, analisis prediktif dan preskriptif. Kerangka kerja dan alat Big Data memiliki arsitektur pemrosesan terdistribusi dan paralel serta dapat memanfaatkan penyimpanan dan sumber daya komputasi dari sejumlah besar mesin.

Analisis Big Data melibatkan beberapa langkah mulai dari pembersihan data, data munging (atau pertengkaran), pemrosesan dan visualisasi data. Siklus hidup analisis Big Data dimulai dari pengumpulan data dari berbagai sumber data. Alat dan kerangka kerja khusus diperlukan untuk mencerna data dari berbagai sumber ke dalam back end analisis Big Data. Data disimpan dalam solusi penyimpanan khusus (seperti sistem file terdistribusi dan database non-relasional) yang dirancang sesuai skala. Berdasarkan persyaratan analisis (batch atau real-time), dan jenis analisis yang akan dilakukan (deskriptif, diagnostik, prediktif, atau prediktif) digunakan kerangka kerja khusus. Analisis Big Data diaktifkan oleh beberapa teknologi seperti komputasi awan, kerangka kerja pemrosesan terdistribusi dan paralel, database non-relasional, komputasi dalam memori, misalnya.
Beberapa contoh big data terdaftar sebagai berikut:

- Data yang dihasilkan oleh jejaring sosial termasuk data teks, gambar, audio dan video
- Data aliran klik yang dihasilkan oleh aplikasi web seperti e-Commerce untuk menganalisis perilaku pengguna
- Mesindapat dikumpulkan dari sensor yang tertanam dalam sistem industri dan energi untuk memantau kesehatannya dan mendeteksi kegagalan
- Data perawatan kesehatan dikumpulkan dalam sistem catatan kesehatan elektronik (EHR)
- Log yang dihasilkan oleh aplikasi web
- Data pasar saham
- Data transaksional yang dihasilkan oleh aplikasi perbankan dan keuangan

## 1.3 Karakteristik Big Data

Karakteristik yang mendasari big data meliputi:

### 1.3.1 Volume

Kita sudah tahu bahwa Big Data menunjukkan 'volume' data yang sangat besar yang dihasilkan setiap hari dari berbagai sumber seperti platform media sosial, proses bisnis, mesin, jaringan, interaksi manusia, dll. Sejumlah besar data disimpan di gudang data. Big data adalah suatu bentuk data yang volumenya sangat besar sehingga tidak akan muat pada satu mesin karena itu membutuhkan alat khusus dan kerangka kerja yang diperlukan untuk memproses dan menganalisis data seperti itu. Misalnya, aplikasi media sosial memproses miliaran pesan setiap hari, sistem industri dan energi dapat menghasilkan data sensor berukuran terabyte setiap hari, aplikasi agregasi kabin dapat memproses jutaan transaksi dalam sehari, dll. Volume data yang dihasilkan dibidang TI modern, industri, perawatan kesehatan, Internet of Things, dan sistem lainnya tumbuh secara eksponensial didorong oleh penurunan biaya penyimpanan data dan arsitektur pemrosesan dan kebutuhan untuk mengekstrak wawasan berharga dari data untuk meningkatkan proses bisnis, efisiensi, dan layanan kepada konsumen. Meskipun tidak ada batasan tetap untuk volume data yang akan dianggap sebagai Big Data, namun, biasanya, istilah Big Data digunakan untuk data skala besar yang sulit untuk disimpan, dikelola, dan diproses menggunakan database tradisional dan arsitektur pemrosesan data.

### 1.3.2 Kecepatan

*Velocity* atau kecepatan pada dasarnya mengacu pada kecepatan di mana data sedang dibuat secara *real-time*. Dalam prospek yang lebih luas, itu terdiri dari tingkat perubahan, menghubungkan set data yang masuk dengan kecepatan yang bervariasi, dan aktivitas meledak. Kecepatan data mengacu pada seberapa cepat data dihasilkan. Data yang dihasilkan oleh sumber tertentu dapat sampai dengan kecepatan yang sangat tinggi, misalnya data media sosial atau data sensor. Kecepatan adalah karakteristik penting lainnya dari big data dan alasan utama pertumbuhan data secara eksponensial. Kecepatan data yang tinggi mengakibatkan volume data yang terkumpul menjadi sangat besar, tidak lama. Beberapa aplikasi dapat menyelesaikan *tead lines for data analysis* (seperti perdagangan atau deteksi penipuan online) dan datanya perlu dianalisis secara real-time. Alat-alat khusus diperlukan untuk menangani semua itu dengan kecepatan tinggi untuk memelihara infrastruktur besar dan menganalisis data secara real-time.

### 1.3.3 Variasi

Model Big Data mengacu pada data terstruktur, tidak terstruktur, dan semi terstruktur yang dikumpulkan dari berbagai sumber. Sementara di masa lalu, data hanya dapat dikumpulkan dari spreadsheet dan database, hari ini data hadir dalam berbagai bentuk seperti email, PDF, foto, video, audio, audio, posting SM, dan banyak lagi lainnya. Sistem Big Data harus cukup fleksibel untuk menangani beragam data.

### 1.3.4 *Veracity* (Keakuratan)

*Verasiity* menyangkut tentang ke-valid-an sebuah data apakah bisa dipercaya atau tidak. Dengan banyaknya bentuk data, kebenaran tentang sebuah informasi menjadi kurang terkontrol, seperti contoh sistem akademik sebuah fakultas dimana untuk jenis kelamin dibedakan menjadi L (laki-laki) dan P (perempuan) sedangkan di sistem akademik fakultas lain menggunakan P (pria) dan W (wanita). Big data dengan teknologi analisis membantu kita untuk dapat bekerja dengan data tersebut melalui hasil analisis, karena semakin besar volume suatu data maka akan semakin tidak akurat data tersebut.

### 1.3.5 *Value* (Nilai)

Nilai data mengacu pada kegunaan data untuk tujuan yang dimaksudkan. Tujuan akhir dari sistem analisis Big Data adalah mengekstrak nilai dari data. Nilai data juga terkait dengan kebenaran atau akurasi data. Untuk beberapa aplikasi, nilai juga tergantung pada seberapa cepat kita dapat memproses datanya.

## 1.4 Contoh Spesifik Domain dari Big Data

Aplikasi Big Data menjangkau berbagai domain termasuk (tetapi tidak terbatas pada) rumah, kota, lingkungan, sistem energi, ritel, logistik, industri, pertanian, Internet of Things, dan perawatan kesehatan. Bagian ini memberikan gambaran umum tentang berbagai aplikasi big data untuk masing-masing domain ini. Pada bab-bab selanjutnya, mahasiswa dibimbing melalui implementasi referensi dan contoh yang akan membantu mahasiswa dalam mengembangkan aplikasi ini.

### 1.4.1 Web

- **Analisis Web**

  Analisis web berkaitan dengan pengumpulan dan analisis data pada kunjungan pengguna di situs web dan aplikasi cloud. Analisis data ini dapat memberikan wawasan tentang keterlibatan pengguna dan melacak kinerja kampanye iklan online. Untuk

mengumpulkan data tentang kunjungan pengguna, dua pendekatan digunakan. Dalam pendekatan pertama, kunjungan pengguna dicatat di server web yang mengumpulkan data seperti tanggal dan waktu kunjungan, permintaan sumber daya, alamat IP pengguna, kode status HTTP, forinstance. Pendekatan kedua, yang disebut penandaan halaman, menggunakan JavaScript yang disematkan di halaman web. Setiap kali pengguna mengunjungi halaman web, JavaScript mengumpulkan data pengguna dan mengirimkannya ke server pengumpulan data pihak ketiga. Cookie diberikan kepada pengguna yang mengidentifikasi pengguna selama kunjungan dan kunjungan berikutnya. Manfaat dari pendekatan penandaan halaman adalah memfasilitasi pengumpulan dan analisis data secara real-time. Pendekatan ini memungkinkan layanan pihak ketiga, yang tidak memiliki akses ke server web (melayani situs web) untuk mengumpulkan dan memproses data. Penyedia layanan analisis khusus ini (seperti Google Analisis) menawarkan analisis lanjutan dan laporan ringkasan. Metrik pelaporan utama mencakup sesi pengguna, kunjungan halaman, halaman masuk dan keluar teratas, rasio pentalan, halaman yang paling banyak dikunjungi, waktu yang dihabiskan di setiap halaman, jumlah pengunjung unik, jumlah pengunjung berulang, misalnya.

- **Pemantauan Kinerja**

  Aplikasi web dan cloud multitier seperti e-Commerce, Business-to-Business, Health care, Banking and Financial, Retail and Social Networking application, dapat mengalami perubahan cepat dalam beban kerjanya. Untuk memastikan kesiapan pasar dari aplikasi semacam itu, sumber daya yang memadai perlu disediakan sehingga aplikasi dapat memenuhi permintaan tingkat beban kerja yang ditentukan dan pada saat yang sama memastikan bahwa perjanjian tingkat layanan terpenuhi.

  Penyediaan dan perencanaan kapasitas adalah tugas yang menantang untuk aplikasi multi-tingkat yang kompleks karena setiap kelas aplikasi memiliki konfigurasi penerapan yang berbeda dengan server web, server aplikasi, dan server database. Penyediaan yang berlebihan sebelumnya untuk sistem semacam itu tidak layak secara ekonomi. Komputasi awan justru memberikan pendekatan yang menjanjikan yang secara dinamis akan meningkatkan atau menurunkan kapasitas berdasarkan beban kerja aplikasi. Untuk keputusan manajemen sumber daya dan perencanaan kapasitas, penting untuk memahami karakteristik beban kerja dari sistem tersebut, mengukur sensitivitas kinerja aplikasi terhadap atribut beban kerja, dan mendeteksi kemacetan dalam sistem. Pengujian kinerja aplikasi berbasis cloud sebelum penerapan dapat menunjukkan hambatan dalam sistem dan mendukung keputusan penyediaan dan perencanaan kapasitas.

Untuk pemantauan kinerja, berbagai jenis tes dapat dilakukan seperti tes beban (yang mengevaluasi kinerja sistem dengan beberapa pengguna dan tingkat beban kerja), tes stres (yang memuat aplikasi ke titik di mana ia rusak) dan tes rendam ( yang menjadikan aplikasi ke tingkat beban kerja tetap untuk jangka waktu yang lama). Sistem Big Data dapat digunakan untuk menganalisis data yang dihasilkan oleh pengujian tersebut, untuk memprediksi kinerja aplikasi di bawah beban kerja yang berat dan mengidentifikasi kemacetan dalam sistem sehingga kegagalan dapat dicegah. Hambatan, yang telah terdeteksi, dapat diatasi dengan menyediakan sumber daya komputasi tambahan, baik dengan sistem penskalaan (penskalaan vertikal menggunakan instans dengan kapasitas komputasi yang lebih) atau sistem penskalaan (penskalaan horizontal dengan menggunakan lebih banyak instans dari jenis yang sama).

- **Penargetan & Analisis Iklan**

  Penelusuran iklan dan tampilan adalah dua pendekatan yang paling banyak digunakan dalam iklan internet. Dalam pencarian iklan, pengguna menampilkan iklan ("iklan"), bersama dengan hasil pencarian, saat mereka mencari kata kunci tertentu di mesin pencari. Pengiklan dapat membuat iklan menggunakan jaringan periklanan yang disediakan oleh mesin pencari atau jaringan media sosial. Ini adalah pengaturan untuk kata kunci spesifik yang terkait dengan layanan produsen yang diiklankan. Pengguna yang mencari kata kunci ini akan melihat iklan bersama dengan hasil pencarian. Iklan tampilan, adalah bentuk lain dari iklan Internet, di mana iklan ditampilkan dalam situs web, video, dan aplikasi seluler yang berpartisipasi dalam jaringan periklanan. Iklan bergambar dapat berupa iklan berbasis teks atau gambar. Jaringan iklan mencocokkan iklan ini dengan konten situs web baru, aplikasi seluler, dan tempat iklan. Metode kompensasi yang paling umum digunakan untuk iklan Internet adalah *Pay-per-click* (PPC), di mana pengiklan membayar setiap kali pengguna mengklik iklan. Jaringan periklanan menggunakan sistem Big Data untuk mencocokkan dan menempatkan iklan dan menghasilkan laporan statistik periklanan. Pengiklan dapat menggunakan alat Big Data untuk melacak kinerja iklan, mengoptimalkan tawaran untuk iklan bayar per klik, melacak kata kunci mana yang paling banyak menautkan ke halaman arahan iklan dan mengoptimalkan alokasi anggaran ke berbagai kampanye iklan.

- **Rekomendasi Konten**

  Aplikasi pengiriman konten yang menyajikan konten (seperti aplikasi streaming musik dan video), mengumpulkan berbagai jenis data seperti pola pencarian pengguna dan riwayat penelusuran, riwayat konten yang dikonsumsi, dan peringkat pengguna. Aplikasi semacam itu dapat memanfaatkan sistem Big Data untuk merekomendasikan

konten baru kepada pengguna berdasarkan preferensi dan minat pengguna. Sistem rekomendasi menggunakan dua pendekatan kategori luas - rekomendasi berbasis pengguna dan rekomendasi berbasis item. Dalam rekomendasi berbasis pengguna, item baru disarankan untuk pengguna berdasarkan bagaimana pengguna yang serupa menilai item tersebut. Sedangkan dalam rekomendasi berbasis item, item baru direkomendasikan kepada pengguna berdasarkan bagaimana pengguna menilai item yang serupa. Di Bab-11, kami menjelaskan studi kasus tentang membangun sistem rekomendasi film.

### 1.4.2 Bidang Keuangan

- **Pemodelan Resiko Kredit**

  Perbankan dan lembaga keuangan menggunakan pemodelan risiko kredit untuk menilai aplikasi kredit dan memprediksi apakah peminjam akan default atau tidak di masa depan. Model risiko kredit dibuat dari data nasabah yang meliputi, skor kredit yang diperoleh dari biro kredit, riwayat kredibilitas, neraca saldo, data transaksi rekening, dan pola pengeluaran nasabah. Model kredit menghasilkan skor numerik yang merangkum kelayakan kredit pelanggan. Karena volume data pelanggan yang diperoleh dari berbagai sumber bisa sangat besar, sistem Big Data dapat digunakan untuk membangun model kredit. Sistem Big Data dapat membantu dalam menghitung skor risiko kredit dari sejumlah besar pelanggan secara teratur. Di Bab-11, kami menjelaskan framework Big Data untuk membuat model pembelajaran mesin. Kerangka kerja ini dapat digunakan untuk membangun model risiko kredit dengan menganalisis data pelanggan.

- ***Fraud Detection (Deteksi Penipuan/Penggelapan)***

  Perbankan dan lembaga keuangan dapat memanfaatkan sistem Big Data untuk mendeteksi penipuan seperti penipuan kartu kredit, pencucian uang dan penipuan klaim asuransi. Kerangka kerja analisis Big Data real-time dapat membantu dalam menganalisis data dari sumber yang berbeda dan melabeli transaksi secara real-time. Model pembelajaran mesin dapat dibangun untuk mendeteksi anomali dalam transaksi dan mendeteksi aktivitas penipuan. Kerangka kerja analisis batch dapat digunakan untuk menganalisis data historis pada transaksi pelanggan untuk mencari pola yang mengindikasikan penipuan.

### 1.4.3 Bidang Kesehatan

Lingkungan di bidang perawatan kesehatan terdiri dari banyak entitas termasuk penyedia perawatan kesehatan (dokter perawatan primer, spesialis, atau rumah sakit), pembayar (pemerintah, perusahaan asuransi kesehatan swasta, pemberi kerja), perusahaan farmasi, perangkat dan layanan medis, perusahaan penyedia konsultan dan layanan TI, dan pasien. Proses penyediaan perawatan kesehatan melibatkan data perawatan kesehatan besar-besaran yang ada dalam berbagai bentuk (terstruktur atau tidak terstruktur), disimpan dalam sumber data yang berbeda (seperti database relasional, atau server file) dan dalam berbagai format. Untuk lebuh memudahkan lebih banyak koordinasi perawatan di berbagai penyedia yang terlibat dengan pasien, informasi klinis mereka semakin dikumpulkan dari berbagai sumber ke dalam sistem Catatan Kesehatan Elektronik (*Electronic Health Record* /EHR). EHR menangkap dan menyimpan informasi tentang kesehatan pasien dan tindakan penyedia termasuk hasil laboratorium tingkat individu, diagnostik, pengobatan, dan data demografis. Meskipun penggunaan utama EHR sebagai catatan data medis untuk pasien individu dan untuk memberikan akses yang efisien ke data yang disimpan pada titik perawatan, EHR dapat menjadi sumber untuk informasi agregat yang berharga tentang populasi pasien secara keseluruhan [5, 6].

Dengan ledakan data klinis saat ini, masalah bagaimana mengumpulkan data dari sistem kesehatan terdistribusi dan heterogen dan bagaimana menganalisis data klinis secara signifikan telah menjadi penting. Sistem big data dapat digunakan untuk pengumpulan data dari pemangku kepentingan yang berbeda (pasien, dokter, pembayar, dokter, spesialis, dll) dan sumber data yang berbeda (database, format terstruktur dan tidak terstruktur, dll). Sistem analisis Big Data memungkinkan analisis data klinis berskala besar dan memfasilitasi pengembangan aplikasi perawatan kesehatan yang lebih efisien, meningkatkan akurasi prediksi, dan membantu pengambilan keputusan tepat waktu.
Mari kita lihat beberapa aplikasi perawatan kesehatan yang dapat memperoleh manfaat dari sistem Big Data:

- **Surveilans Epidemiologis**
  Surveilans epidemiologi adalah kegiatan analisis secara sistematis dan terus menerus terhadap penyakit atau masalah-masalah kesehatan dan kondisi yang mempengaruhi terjadinya peningkatan dan penularan penyakit atau masalah-masalah kesehatan tersebut, agar dapat melakukan tindakan penanggulangan secara efektif dan efisien melalui proses pengumpulan data, pengolahan dan penyebaran informasi epidemiologi kepada penyelenggara program kesehatan. Sistem EHR mencakup hasil laboratorium

tingkat individu, diagnostik, pengobatan, dan data demografis. Kerangka Big Data dapat digunakan untuk mengintegrasikan data dari beberapa sistem EHR dan analisis data yang tepat waktu untuk memprediksi wabah secara efektif dan akurat, upaya pengawasan kesehatan tingkat populasi, deteksi penyakit, dan pemetaan kesehatan masyarakat.

- **Analisi Data Pasien yang Sama berbasis Perangkat *Decesion Intelegence***
  Kerangka Big Data dapat digunakan untuk menganalisis data EHR untuk mengekstrak sekelompok catatan pasien yang paling mirip dengan pasien target tertentu. Pengelompokan catatan pasien juga dapat membantu dalam mengembangkan aplikasi prognosis medis yang memprediksi kemungkinan hasil dari suatu penyakit untuk pasien berdasarkan hasil untuk pasien serupa.
- **Prediksi Resiko Penggunan Obat-Obatan**
  Kerangka Big Data dapat digunakan untuk menganalisis data EHR dan memprediksi pasien mana yang paling berisiko mengalami tanggapan yang merugikan terhadap obat tertentu berdasarkan reaksi obat yang merugikan dari pasien lain.
- **Mendeteksi Keganjilan Klaim Kesehatan**
  Perusahaan asuransi kesehatan dapat memanfaatkan sistem Big Data untuk menganalisis klaim asuransi kesehatan guna mendeteksi penipuan, penyalahgunaan, pemborosan, dan kesalahan.
- **Evidence-based Medicine**
  Sistem Big Data dapat menggabungkan dan menganalisis data dari berbagai sumber, termasuk hasil laboratorium tingkat individu, data diagnostik, pengobatan, dan demografis, untuk mencocokkan pengobatan dengan hasil, memprediksi pasien yang berisiko terkena penyakit. Sistem untuk pengobatan berbasis bukti memungkinkan penyedia untuk membuat keputusan tidak hanya berdasarkan persepsi mereka sendiri tetapi juga dari bukti yang tersedia.
- **Pemantauan kesehatan secara real-time**
  Perangkat elektronik yang dapat dikenakan memungkinkan pemantauan parameter fisiologis non-invasif dan berkelanjutan. Perangkat yang dapat dikenakan ini mungkin dalam berbagai bentuk seperti ikat pinggang dan gelang tangan. Penyedia layanan kesehatan dapat menganalisis data perawatan kesehatan yang dikumpulkan untuk menentukan kondisi atau anomali kesehatan apa pun. Sistem Big Data untuk analisis data real-time dapat digunakan untuk analisis volume besar data yang bergerak cepat dari perangkat yang dapat dikenakan dan perangkat di rumah sakit atau di rumah lainnya, untuk pemantauan kesehatan pasien secara real-time dan prediksi kejadian buruk.

### 1.4.4 *Internet of Thing* (IoT)

*Internet of Things* (IoT) mengacu pada hal-hal yang memiliki identitas unik dan terhubung ke Internet. Kata "*Things*" dalam IoT adalah perangkat yang dapat melakukan penginderaan jarak jauh, penggerak dan pemantauan. Perangkat IoT dapat bertukar data dengan perangkat dan aplikasi lain yang terhubung (secara langsung atau tidak langsung), atau mengumpulkan data dari perangkat lain dan memproses data baik secara lokal atau mengirim data ke server terpusat atau back-end aplikasi berbasis cloud untuk memproses data, atau melakukan beberapa tugas secara lokal dan tugas lain dalam IoT infrastruktur, berdasarkan batasan temporal dan ruang (yaitu, memori, kapabilitas pemrosesan, latensi dan kecepatan komunikasi, serta tenggat waktu).

Sistem IoT dapat memanfaatkan teknologi Big Data untuk penyimpanan dan analisis data. Mari kita lihat beberapa aplikasi IoT yang dapat memperoleh manfaat dari sistem Big Data:

- **Deteksi Intrusi**
  Sistem deteksi intrusi menggunakan kamera dan sensor keamanan (seperti sensor PIR dan sensor pintu) untuk mendeteksi intrusi dan meningkatkan peringatan. Peringatan dapat berupa bentuk SMS atau email kepada pengguna. Sistem tingkat lanjut dapat mengirim peringatan rinci seperti pengambilan gambar atau klip video pendek yang dikirim sebagai lampiran email.
- **Smart Parking**
  Smart parking membuat pencarian tempat parkir lebih mudah dan nyaman bagi pengemudi. Smart parking didukung oleh sistem IoT yang mendeteksi jumlah slot parkir kosong dan mengirimkan informasi melalui Internet ke aplikasi smart parking. Aplikasi ini dapat diakses oleh pengemudi dari ponsel pintar, tablet, dan sistem navigasi dalam mobil. Dalam smart parking, sensor digunakan untuk setiap slot parkir, untuk mendeteksi apakah slot tersebut kosong atau terisi. Informasi ini dikumpulkan oleh pengontrol parkir cerdas di tempat dan kemudian dikirim melalui Internet ke backend analisis Big Data berbasis cloud.
- **Smart Road**
  Smart Road yang dilengkapi dengan sensor dapat memberikan informasi tentang kondisi mengemudi, perkiraan waktu tempuh dan peringatan jika terjadi kondisi mengemudi yang buruk, kemacetan lalu lintas dan kecelakaan. Informasi tersebut dapat membantu membuat jalan lebih aman dan membantu mengurangi kemacetan lalu lintas. Informasi yang dirasakan dari jalan dapat dikomunikasikan melalui Internet

ke aplikasi analisis Big Data berbasis *cloud*. Hasil analisis dapat disebarluaskan kepada para pengemudi yang berlangganan aplikasi tersebut atau melalui media sosial.

- **Pemantauan Kesehatan Struktural**
  Sistem Pemantauan Kesehatan Struktural menggunakan jaringan sensor untuk memantau tingkat getaran dalam struktur seperti jembatan dan gedung. Data yang dikumpulkan dari sensor ini dianalisis untuk menilai kesehatan struktur. Dengan menganalisis data, dimungkinkan untuk mendeteksi retakan dan kerusakan mekanis, menemukan kerusakan pada suatu struktur dan juga menghitung umur sisa struktur. Dengan menggunakan sistem seperti itu, peringatan dini dapat diberikan jika terjadi kegagalan struktur yang akan segera terjadi.
- ***Smart irrigation***
  Sistem irigasi pintar dapat meningkatkan hasil panen sekaligus menghemat air. Sistem irigasi pintar menggunakan perangkat IoT dengan sensor kelembapan tanah untuk menentukan jumlah kelembapan di tanah dan melepaskan aliran air melalui pipa irigasi hanya jika tingkat kelembapan berada di bawah ambang batas yang telah ditentukan. Sistem irigasi pintar juga mengumpulkan pengukuran tingkat kelembapan di awan tempat sistem Big Data dapat digunakan untuk menganalisis data guna merencanakan jadwal penyiraman.

### 1.4.5 Bidang Lingkungan

Sistem pemantauan lingkungan menghasilkan data berkecepatan tinggi dan volume tinggi. Analisis yang akurat dan tepat waktu dari data tersebut dapat membantu dalam memahami status lingkungan saat ini dan juga memprediksi tren lingkungan. Mari kita lihat beberapa aplikasi pemantauan lingkungan yang dapat memperoleh manfaat dari sistem Big Data:

- **Pemantauan Cuaca**
  Sistem pemantauan cuaca dapat mengumpulkan data dari sejumlah sensor yang terpasang (seperti suhu, kelembaban, atau tekanan) dan mengirim data ke aplikasi berbasis cloud dan backend analisis Big Data. Data ini kemudian dapat dianalisis dan divisualisasikan untuk memantau cuaca dan menghasilkan peringatan cuaca.
- **Pemantauan Polusi Udara**
  Sistem pemantauan polusi udara dapat memantau emisi gas berbahaya (CO2, CO, NO, atau NO2) oleh pabrik dan mobil menggunakan sensor gas dan meteorologi. Data yang dikumpulkan dapat dianalisis untuk membuat keputusan yang tepat tentang pendekatan pengendalian polusi.

- **Pemantauan Tingkat Kebisingan Suara**

  Karena perkembangan kota yang terus meningkat, tingkat kebisingan di kota-kota telah meningkat dan bahkan menjadi sangat tinggi di beberapa kota. Polusi suara dapat membahayakan kesehatan manusia akibat gangguan tidur dan stres. Pemantauan polusi suara dapat membantu menghasilkan map kebisingan untuk kota-kota. Map kebisingan kota dapat membantu para pembuat kebijakan dalam perencanaan kota dan membuat kebijakan untuk mengontrol tingkat kebisingan di sekitar pemukiman, sekolah dan taman. Sistem pemantauan polusi suara menggunakan sejumlah stasiun pemantauan kebisingan yang ditempatkan di berbagai tempat di kota. Data tingkat kebisingan dari stasiun dikirim ke aplikasi berbasis cloud dan backend analisis Big Data. Data yang terkumpul kemudian dikumpulkan untuk menghasilkan map kebisingan.
- **Deteksi Kebakaran Hutan**

  Kebakaran hutan dapat menyebabkan kerusakan pada sumber daya alam, harta benda dan kehidupan manusia. Ada berbagai penyebab kebakaran hutan termasuk petir, kelalaian manusia, letusan gunung berapi dan percikan api dari batu yang jatuh. Deteksi dini kebakaran hutan dapat membantu meminimalkan kerusakan. Sistem deteksi kebakaran hutan menggunakan sejumlah titik pemantauan yang ditempatkan di lokasi berbeda di dalam hutan. Setiap node pemantauan mengumpulkan pengukuran pada kondisi sekitar termasuk suhu, kelembaban, tingkat cahaya, misalnya.
- **Deteksi Banjir**

  Banjir sungai dapat menyebabkan kerusakan parah pada sumber daya alam dan manusia serta kehidupan manusia. Banjir sungai terjadi karena curah hujan yang terus menerus yang menyebabkan permukaan sungai naik dan laju aliran meningkat dengan cepat. Peringatan dini banjir dapat diberikan dengan memantau ketinggian air dan laju aliran. Sistem pemantauan banjir sungai menggunakan sejumlah node sensor yang memantau ketinggian air (menggunakan sensor ultrasonik) dan laju aliran (menggunakan sensor kecepatan aliran). Sistem Big Data dapat digunakan untuk mengumpulkan dan menganalisis data dari sejumlah node sensor tersebut dan meningkatkan peringatan ketika peningkatan cepat pada permukaan air dan laju aliran terdeteksi.
- **Pemantauan Kualitas Air**

  Pemantauan kualitas air dapat membantu untuk mengidentifikasi dan mengendalikan pencemaran dan pencemaran air akibat urbanisasi dan industrialisasi. Menjaga kualitas air yang baik penting untuk menjaga kesehatan tumbuhan dan hewan. Sistem pemantauan kualitas air menggunakan sensor untuk secara mandiri dan terus menerus memantau berbagai jenis kontaminasi di badan air (seperti bahan kimia, biologis,

dan radioaktif). Skala data yang dihasilkan oleh sistem semacam itu sangat besar. Sistem Big Data dapat membantu dalam analisis real-time dari data yang dihasilkan oleh sistem tersebut dan menghasilkan peringatan tentang setiap penurunan kualitas air, sehingga tindakan korektif dapat diambil.

### 1.4.6 Bidang Logistik dan Transportasi

- **Fleet Tracking Real-time**
  istem pelacakan kendaraan menggunakan teknologi GPS untuk melacak lokasi kendaraan secara real-time. Sistem pelacakan penerbangan berbasis cloud dapat ditingkatkan sesuai permintaan untuk menangani sejumlah besar kendaraan. Peringatan dapat dibuat jika terjadi penyimpangan pada rute yang direncanakan. Sistem Big Data dapat digunakan untuk mengumpulkan dan menganalisis lokasi kendaraan dan data rute untuk mendeteksi kemacetan dalam suplai chain seperti kemacetan lalu lintas di rute, penugasan dan pembuatan rute alternatif, dan optimalisasi suplai chain.
- **Pelacakan Kendaraan Real-time**
  Sistem pelacakan kendaraan menggunakan teknologi GPS untuk melacak lokasi kendaraan secara real-time. Sistem pelacakan penerbangan berbasis cloud dapat ditingkatkan sesuai permintaan untuk menangani sejumlah besar kendaraan. Peringatan dapat dibuat jika terjadi penyimpangan pada rute yang direncanakan. Sistem Big Data dapat digunakan untuk mengumpulkan dan menganalisis lokasi kendaraan dan data rute untuk mendeteksi kemacetan dalam suplai chain seperti kemacetan lalu lintas di rute, penugasan dan pembuatan rute alternatif, dan optimalisasi suplai chain.
- **Pengawasan Bongkar Muat Barang**
  Solusi manajemen pengiriman untuk sistem transportasi memungkinkan pemantauan kondisi di dalam kontainer. Misalnya, wadah yang membawa produk makanan segar dapat dipantau untuk mendeteksi pembusukan makanan. Sistem pemantauan pengiriman menggunakan sensor seperti suhu, tekanan, kelembaban, misalnya, untuk memantau kondisi di dalam wadah dan mengirim data ke cloud, yang dapat dianalisis untuk mendeteksi pembusukan makanan. Analisis dan interpretasi data tentang kondisi lingkungan dalam wadah dan posisi truk makanan dapat memungkinkan keputusan rute yang lebih efektif secara real time. Oleh karena itu, dimungkinkan untuk mengambil langkah-langkah perbaikan seperti - makanan yang memiliki anggaran waktu terbatas sebelum busuk dapat dialihkan kembali ke tujuan yang lebih dekat, peringatan dapat dinaikkan kepada pengemudi dan distributor tentang kondisi transit, seperti karena suhu wadah melebihi batas yang diizinkan, tingkat kelembapan keluar dari batas yang

diizinkan, misalnya, dan tindakan korektif dapat diambil sebelum makanan rusak. Untuk produk yang mudah pecah, tingkat getaran selama pengiriman dapat dilacak menggunakan sensor akselerometer dan giroskop. Sistem Big Data dapat digunakan untuk menganalisis pola getaran pengiriman guna mengungkapkan informasi terkait lingkungan operasi dan integritasnya selama pengangkutan, penanganan, dan penyimpanan.

- **Diagnostik Kendaraan Jarak Jauh**

  Sistem diagnostik kendaraan jarak jauh dapat mendeteksi kesalahan dalam kendaraan atau memperingatkan kesalahan yang akan datang. Sistem diagnostik ini menggunakan perangkat terpasang untuk mengumpulkan data tentang operasi kendaraan (seperti kecepatan, RPM engine, suhu cairan pendingin, atau nomor kode kerusakan) dan status berbagai sub-sistem kendaraan. Kendaraan komersial modern mendukung standar diagnostik on-board (OBD) seperti OBD-II. Sistem OBD menyediakan data-waktu yang nyata status dari sub-sistem kendaraan dan kode masalah diagnostik yang memungkinkan dengan cepat mengidentifikasi kesalahan dalam kendaraan. Sistem diagnostik kendaraan memindai akhir data analisis berbasis kendaraan yang tersimpan di data yang dapat dianalisis untuk menghasilkan peringatan dan menyarankan tindakan perbaikan.

- **Pembuatan & Schedulleran Rute**

  Sistem transportasi modern didorong oleh data yang dikumpulkan dari berbagai sumber yang diproses untuk memberikan layanan baru kepada para pemangku kepentingan. Dengan mengumpulkan data dalam jumlah besar dari berbagai sumber dan memproses data menjadi informasi yang berguna, sistem transportasi berbasi data dapat menyediakan layanan baru seperti panduan rute lanjutan, Schedulleran rute/jalur kendaraan yang dinamis, mengantisipasi permintaan pelanggan terhadap masalah pengambilan dan pengiriman, misalnya. Sistem pemutakhiran data dan Schedulleran rute dapat menghasilkan rute armada pulang pergi (PP) menggunakan kombinasi pola rute dan moda transportasi dan jadwal yang layak berdasarkan ketersediaan kendaraan. Ketika jaringan transportasi tumbuh dalam ukuran dan kompleksitas, jumlah kombinasi rute yang mungkin meningkat secara eksponensial. Sistem Big Data dapat memberikan respons cepat ke kueri pembuatan rute dan dapat ditingkatkan untuk melayani jaringan transportasi yang besar.

- **Pengiriman Hiper-Lokal**

  Platform pengiriman hiper-lokal semakin banyak digunakan oleh bisnis seperti restoran dan toko bahan makanan untuk memperluas jangkauan. Platform ini memungkinkan pelanggan untuk memesan produk (seperti bahan makanan dan

makanan) menggunakan aplikasi web dan seluler dan produk tersebut bersumber dari toko lokal (atau restoran). Karena platform ini berkembang untuk melayani sejumlah besar pelanggan (dengan ribuan transaksi setiap jam), mereka menghadapi berbagai tantangan dalam memproses pesanan secara real-time. Sistem Big Data untuk analisis real-time dapat digunakan oleh platform pengiriman hiper-lokal untuk menentukan toko terdekat dari mana sumber pesanan dan menemukan Agent pengiriman di dekat toko yang dapat mengambil pesanan dan mengirimkannya ke pelanggan.

- **Agregator Taksi / Taksi**

  Agregator teknologi transportasi sesuai permintaan (atau agregator taksi / taksi) memungkinkan pelanggan memesan taksi menggunakan aplikasi web atau seluler dan permintaan dialihkan ke taksi terdekat yang tersedia (terkadang bahkan pengemudi pribadi yang memilih mobil mereka sendiri untuk mempekerjakan). Platform agregasi kabin menggunakan sistem Big Data untuk pemrosesan permintaan real-time dan harga dinamis. Platform ini menyimpan catatan semua taksi dan menyesuaikan permintaan perjalanan dari pelanggan ke taksi terdekat dan paling sesuai. Platform ini mengadopsi model penetapan harga dinamis di mana harga naik atau turun berdasarkan permintaan dan kondisi lalu lintas.

### 1.4..7 Bidang Industri

- **Diagnosis & Prognosis Mesin**

  Prognosis mesin mengacu pada prediksi kinerja mesin dengan menganalisis data pada kondisi operasi saat ini dan penyimpangan dari kondisi operasi normal. Diagnosis mesin mengacu pada penentuan penyebab kerusakan mesin. Mesin industri memiliki banyak sekali komponen yang harus berfungsi dengan baik agar mesin dapat menjalankan operasinya. Sensor di mesin dapat memantau kondisi operasi seperti (suhu dan tingkat getaran). Pengukuran data sensor dilakukan pada skala waktu beberapa milidetik hingga beberapa detik, yang mengarah pada pembuatan data dalam jumlah besar. Sistem diagnostik mesin dapat diintegrasikan dengan penyimpanan berbasis cloud dan backend analisis Big Data untuk penyimpanan, pengumpulan, dan analisis sensordata mesin berskala besar. Sejumlah metode telah diusulkan untuk analisis keandalan dan prediksi kesalahan pada mesin. Penalaran berbasis kasus (CBR) adalah metode yang umum digunakan yang menemukan solusi untuk masalah baru berdasarkan pengalaman masa lalu. Pengalaman masa lalu ini diatur dan direpresentasikan sebagai kasus dalam basis kasus. CBR adalah teknik yang efektif untuk pemecahan masalah di bidang di mana sulit untuk menetapkan model matematika kuantitatif,

seperti diagnosis mesin dan prognosis. Karena untuk setiap mesin, data dari sejumlah besar sensor dikumpulkan, dengan menggunakan aldataforcreation berdimensi tinggi seperti itu dari perpustakaan kasus mengurangi efisiensi pengambilan kasus. Oleh karena itu, metode reduksi data dan ekstraksi fitur digunakan untuk menemukan kumpulan fitur yang representatif yang memiliki kemampuan klasifikasi yang sama dengan kumpulan fitur lengkap.

- **Analisis Risiko Operasi Industri**

  Di banyak industri, terdapat persyaratan yang ketat tentang kondisi lingkungan dan kondisi kerja peralatan. Pemantauan kondisi kerja para pekerja penting untuk memastikan kesehatan dan keselamatan mereka. Gas berbahaya dan beracun seperti karbon monoksida (CO), nitrogen monoksida (NO), Nitrogen Dioksida (NO2), misalnya, dapat menyebabkan gangguan kesehatan yang serius. Sistem pemantauan gas dapat membantu dalam memantau kualitas udara dalam ruangan menggunakan berbagai sensor gas. Sistem Big Data juga dapat digunakan untuk menganalisis risiko dalam operasi industri dan mengidentifikasi zona berbahaya, sehingga tindakan korektif dapat diambil dan peringatan tepat waktu dapat ditingkatkan jika terjadi kondisi abnormal.
- **Perencanaan dan Pengendalian Produksi**

  Sistem perencanaan dan kontrol produksi mengukur berbagai parameter proses produksi dan mengontrol seluruh proses produksi secara real-time. Sistem ini menggunakan berbagai sensor untuk mengumpulkan data pada proses produksi. Sistem Big Data dapat digunakan untuk menganalisis data ini untuk perencanaan produksi dan mengidentifikasi potensi masalah.

### 1.4.8 Bidang Retail

*Retailer* dapat menggunakan sistem Big Data untuk meningkatkan penjualan, meningkatkan keuntungan, dan meningkatkan kepuasan pelanggan. Mari kita lihat beberapa aplikasi analisis Big Data untuk ritel:

- **Manajemen Inventaris**

  Manajemen inventaris untuk ritel menjadi semakin penting dalam beberapa tahun terakhir dengan meningkatnya persaingan. Sementara stok produk yang berlebihan dapat mengakibatkan biaya dan risiko penyimpanan tambahan (dalam kasus yang mudah rusak), kekurangan stok dapat menyebabkan hilangnya pendapatan. Tag RFID yang ditempelkan pada produk memungkinkan mereka untuk dilacak secara real-time sehingga tingkat persediaan dapat ditentukan secara akurat dan produk yang persediaannya rendah dapat diisi ulang. Pelacakan dapat dilakukan dengan

menggunakan mahasiswa RFID yang dipasang di rak toko ritel atau di gudang. Sistem Big Data dapat digunakan untuk menganalisis data yang dikumpulkan dari mahasiswa RFID dan meningkatkan peringatan ketika tingkat persediaan untuk produk tertentu rendah. Pengisian ulang persediaan tepat waktu dapat membantu meminimalkan kerugian pendapatan karena persediaan habis. Analisis data persediaan dapat membantu dalam mengoptimalkan tingkat dan frekuensi persediaan ulang berdasarkan permintaan.

- **Rekomendasi Pelanggan**
  Sistem Big Data dapat digunakan untuk menganalisis data pelanggan (seperti data demografis, riwayat belanja, atau umpan balik pelanggan) dan memprediksi preferensi pelanggan. Produk baru dapat direkomendasikan kepada pelanggan berdasarkan preferensi pelanggan dan penawaran serta diskon yang dipersonalisasi dapat diberikan. Pelanggan dengan preferensi serupa dapat dikelompokkan dan kampanye bertarget dapat dibuat untuk pelanggan. Dalam Bab-11, kami menjelaskan studi kasus tentang membangun sistem rekomendasi berdasarkan filter kolaboratif. Pemfilteran kolaboratif memungkinkan untuk merekomendasikan item (atau memfilter item dari koleksi item) berdasarkan preferensi pengguna dan preferensi kolektif pengguna lain (yaitu memanfaatkan informasi kolaboratif yang tersedia pada peringkat item pengguna).
- **Pengoptimalan Tata Letak Toko**
  Sistem Big Data dapat membantu menganalisis data tentang pola belanja pelanggan dan umpan balik pelanggan untuk mengoptimalkan tata letak toko. Item yang kemungkinan besar dibeli bersama oleh pelanggan dapat ditempatkan di rak yang sama atau berdekatan.
- **Peramalan akan Permintaan**
  Karena banyaknya produk, variasi musiman dalam permintaan dan perubahan tren dan preferensi pelanggan, retailer merasa sulit untuk meramalkan permintaan dan volume penjualan. Sistem Big Data dapat digunakan untuk menganalisis pola pembelian pelanggan dan memprediksi permintaan dan volume penjualan.

## 1.5 Alur Analisis Big Data

Pada bagian ini kami mengusulkan metodologi desain sistem aplikasi sains dan analisis data baru yang dapat digunakan untuk analisis Big Data. Aliran generik untuk analisis Big Data, yang merinci langkah-langkah yang terlibat dalam penerapan aplikasi analisis khas dan opsi yang tersedia di setiap langkah, disajikan. Gambar 1.2 menunjukkan aliran

analisis dengan berbagai langkah. Untuk aplikasi, memilih opsi untuk setiap langkah dalam aliran analisis dapat membantu dalam menentukan alat dan kerangka kerja yang tepat untuk melakukan analisis.

### 1.5.1 Pengumpulan Data

Pengumpulan data adalah langkah pertama untuk aplikasi analisis apa pun. Sebelum data dapat dianalisis, data harus dikumpulkan dan diserap ke dalam Big Data Stack. Pilihan alat dan kerangka kerja untuk pengumpulan data bergantung pada sumber data dan jenis data yang diserap. Untuk pengumpulan data, berbagai jenis konektor dapat digunakan seperti kerangka perpesanan langganan-publikasi, antrean perpesanan, konektor sumber-sink, konektor database, dan konektor kustom. Bab-5 memberikan implementasi dari beberapa konektor ini.

### 1.5.2 Persiapan Data

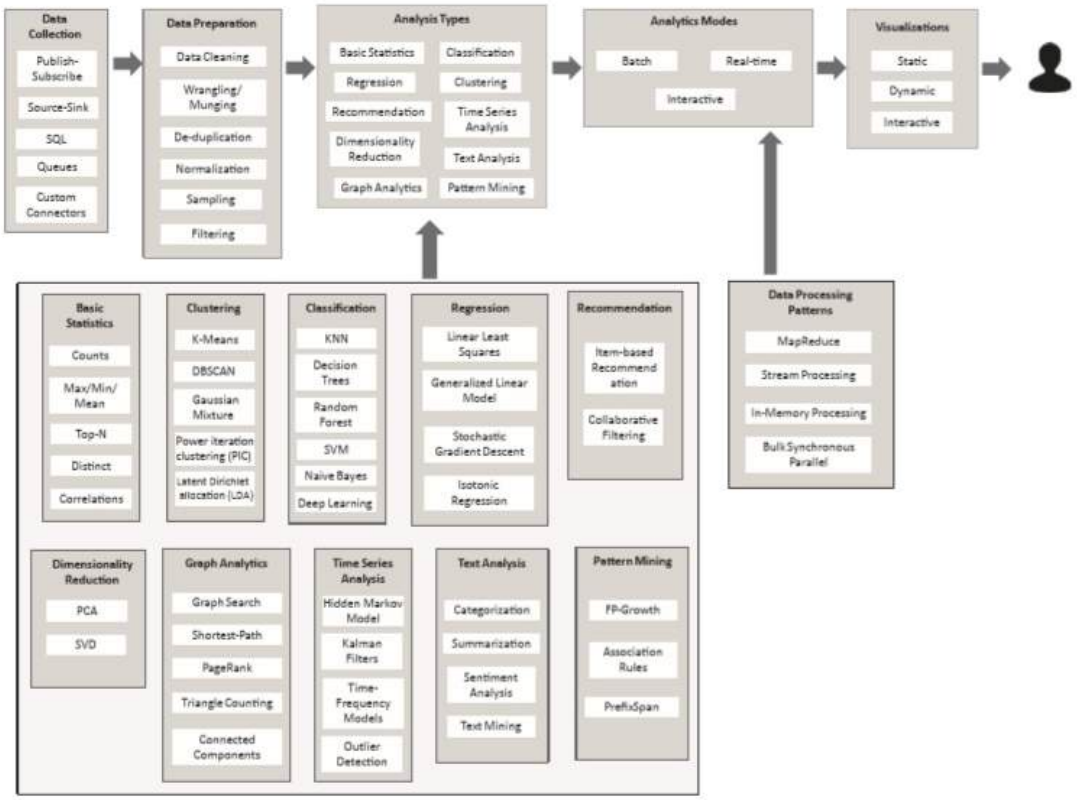
Data sering kali kotor dan dapat memiliki berbagai masalah yang harus diselesaikan sebelum data dapat diproses, seperti catatan rusak, nilai yang hilang, duplikat, singkatan yang tidak konsisten, unit yang tidak konsisten, kesalahan ketik, ejaan yang salah, dan format yang salah. Langkah persiapan data melibatkan berbagai tugas seperti pembersihan data, perselisihan atau munging data, de-duplikasi, normalisasi, pengambilan sampel dan pemfilteran. Pembersihan data mendeteksi dan menyelesaikan masalah seperti catatan rusak, catatan dengan nilai yang hilang, catatan dengan format yang buruk, misalnya. Perselisihan data atau munging berkaitan dengan mengubah data dari satu format mentah ke format lainnya. Sebagai contoh, ketika kita mengumpulkan *record* sebagai file teks mentah dari sumber yang berbeda, kita mungkin menemukan ketidakkonsistenan pada pemisah field yang digunakan pada file yang berbeda.

Beberapa file mungkin menggunakan koma sebagai pemisah bidang, yang lain mungkin menggunakan tab sebagai pemisah bidang. Perselisihan data menyelesaikan ketidakkonsistenan ini dengan mengurai raw data dari sumber yang berbeda dan mengubahnya menjadi satu format yang konsisten. Normalisasi diperlukan bila data dari sumber berbeda menggunakan satuan atau skala yang berbeda atau memiliki singkatan yang berbeda untuk hal yang sama. Misalnya, data cuaca yang dilaporkan oleh beberapa stasiun mungkin berisi suhu dalam skala Celcius sedangkan data dari stasiun lain mungkin menggunakan skala Fahrenheit. Pemfilteran dan pengambilan sampel mungkin berguna ketika kita ingin memproses hanya data yang memenuhi aturan tertentu. Pemfilteran

juga dapat berguna untuk menolak catatan buruk dengan nilai yang salah atau di luar kisaran.

### 1.5.3 Tipe Analisis

Langkah selanjutnya dalam aliran analisis adalah menentukan jenis analisis untuk aplikasi. Pada Gambar 1.2 kami telah membuat daftar berbagai opsi untuk jenis analisis dan algoritma populer untuk setiap jenis analisis. Di Bab-11, kami telah menjelaskan beberapa jenis analisis dan algoritme berikut implementasi algoritme menggunakan berbagai kerangka dan alat Big Data.




Gambar 1.2. *Arus analisis Big Data*

Dengan jenis analisis yang dipilih untuk suatu aplikasi, langkah selanjutnya adalah menentukan mode analisis, yang dapat berupa batch, real-time, atau interaktif. Pilihan mode tergantung pada persyaratan aplikasi. Jika aplikasi yang digunakan meminta hasil untuk diperbarui setelah interval waktu yang singkat (katakanlah setiap beberapa detik), maka mode analisis real-time dipilih. Namun jika aplikasi tersebut hanya membutuhkan hasil yang akan dibuat dan diperbarui pada skala waktu yang lebih besar (katakanlah harian

atau bulanan), maka mode batch dapat digunakan. Jika aplikasi menuntut fleksibilitas untuk meminta data sesuai permintaan, maka mode interaktif berguna. Setelah membuat pilihan jenis analisis dan mode analisis, selanjutnya dapat menentukan pola pemrosesan data yang dapat digunakan. Misalnya, untuk statistik dasar sebagai jenis analisis dan mode analisis batch, MapReduce bisa menjadi pilihan yang baik. Sedangkan untuk analisis regresi sebagai jenis analisis dan mode analisis real-time (memprediksi nilai secara real-time), pola Stream Processing merupakan pilihan yang baik. Pilihan jenis analisis, mode analisis, dan pola pemrosesan data dapat membantu dalam memilih alat dan kerangka kerja yang tepat untuk analisis data.

### 1.5.4 Visualisasi

Pilihan alat visualisasi, database penyajian, dan kerangka kerja web pengaruhi oleh persyaratan aplikasi. Visualisasi bisa statis, dinamis atau interaktif. Visualisasi statis digunakan saat memiliki hasil analisis yang disimpan dalam database penyajian dan hanya ingin menampilkan hasilnya. Namun, jika aplikasi meminta hasil diperbarui secara berkala, maka dibutuhkan visualisasi dinamis (dengan *widget,* plot, atau pengukur langsung). Jika ingin aplikasi yang digunakan menerima masukan dari pengguna dan menampilkan hasilnya, maka dibutuhkan adanya visualisasi interaktif.

## 1.6 Big Data Stack

Meskipun kerangka kerja Hadoop telah menjadi salah satu kerangka kerja paling populer untuk analisis Big Data, ada beberapa jenis tugas komputasi yang tidak berfungsi dengan baik oleh Hadoop. Dengan bantuan pemetaan antara jenis analisis dan komputasi"raksasa" seperti yang ditunjukkan pada Gambar 1.1, kami akan mengidentifikasi kasus di mana Hadoop bekerja dan di mana tidak, dan menjelaskan motivasi untuk memiliki Big Data Stack yang dapat digunakan untuk berbagai jenis analisis dan tugas komputasi.

Hadoop adalah kerangka kerja sumber terbuka untuk pemrosesan batch terdistribusi dari data skala besar menggunakan model pemrograman Map Reduce. Model pemrograman Map Reduce berguna untuk aplikasi di mana data yang terlibat sangat besar sehingga tidak akan muat pada satu mesin. Dalam aplikasi semacam itu, data biasanya disimpan pada sistem file terdistribusi (seperti Hadoop Distributed File System - HDFS). Program MapReduce memanfaatkan lokalitas data dan pemrosesan data berlangsung di node tempat data berada. Dengan kata lain, penghitungan dipindahkan ke tempat data berada, sebagai kebalikan dari cara tradisional memindahkan data dari tempatnya berada ke

tempat penghitungan selesai. MapReduce paling cocok untuk analisis deskriptif dan tugas komputasi statistik dasar karena operasi yang terlibat dapat dilakukan secara paralel (misalnya, menghitung hitungan, rata-rata, maks / min, distinct, top-N, filter, dan join). Banyak dari operasi ini diselesaikan dengan satu pekerjaan MapReduce. Untuk tugas yang lebih kompleks, beberapa pekerjaan MapReduce dapat digabungkan bersama. Namun, ketika komputasi bersifat iteratif, di mana tugas MapReduce harus dijalankan berulang kali, MapReduce mengalami penurunan kinerja karena overhead yang terlibat dalam pengambilan data dari HDFS di setiap iterasi.

Untuk jenis analisis dan tugas komputasi lainnya, ada kerangka kerja alternatif lain yang akan kita bahas sebagai bagian dari Big Data Stack. Dalam Bab ini, kami mengusulkan dan menjelaskan Big Data Stack yang terdiri dari kerangka kerja Big Data yang terbukti dan bersumber terbuka yang membentuk dasar buku ini. Gambar 1.3 menunjukkan Big Data Stack dengan nomor Bab yang disorot untuk berbagai blok dalam tumpukan. Bab-bab berikutnya dalam buku ini menjelaskan blok-blok ini secara rinci bersama dengan contoh-contoh langsung dan studi kasus. Kami telah menggunakan Python sebagai bahasa pemrograman utama untuk contoh dan studi kasus di seluruh buku ini. Mari kita lihat setiap blok satu per satu:

### 1.6.1 Sumber *Raw Data*

Dalam aplikasi atau platform analisis Big Data apa pun, sebelum data diproses dan dianalisis, data harus diambil dari sumber raw data ke dalam kerangka dan sistem Big Data. Beberapa contoh sumber Big Raw data meliputi:

- Log: Log yang dihasilkan oleh aplikasi web dan server yang dapat digunakan untuk pemantauan kinerja
- Data Transaksional: Data transaksional yang dihasilkan oleh aplikasi seperti e-Commerce, Perbankan dan Keuangan
- Media Sosial: Data yang dihasilkan oleh platform media sosial
- Database: Data terstruktur yang berada di database relasional
- Data Sensor: Data sensor yang dihasilkan oleh sistem Internet of Things (IoT)
- Data Clickstream: Data clickstream yang dihasilkan oleh aplikasi web yang dapat digunakan untuk menganalisis pola penjelajahan pengguna
- Data Pengawasan: Data sensor, gambar dan video yang dihasilkan oleh sistem pengawasan

- Data Perawatan Kesehatan: Data perawatan kesehatan yang dihasilkan oleh Electronic Health Record (EHR) dan aplikasi perawatan kesehatan lainnya
- Data Jaringan: Data jaringan yang dihasilkan oleh perangkat jaringan seperti router dan firewall

### 1.6.2 Konektor Akses Data

Konektor Akses Data mencakup alat dan kerangka kerja untuk mengumpulkan dan menyerap data dari berbagai sumber ke dalam kerangka penyimpanan Big Data dan kerangka kerja analisis. Pilihan konektor data ditentukan oleh jenis sumber data. Mari kita lihat beberapa konektor data dan kerangka kerja yang dapat digunakan untuk mengumpulkan dan menyerap data. Kerangka dan konektor data ini dijelaskan secara rinci pada Bab-5. Konektor ini dapat mencakup koneksi kabel dan nirkabel.

- ***Publish-Subscribe Messaging***

  *Publish-Subscribe* adalah model komunikasi yang melibatkan penerbit, broker dan konsumen. Penerbit adalah sumber data. Penerbit mengirimkan data ke topik yang dikelola oleh broker. Kerangka kerja perpesanan langganan-terbitkan seperti Apache Kafka dan Amazon Kinesis dijelaskan di Bab-5.
- **Konektor *Source-Sink***

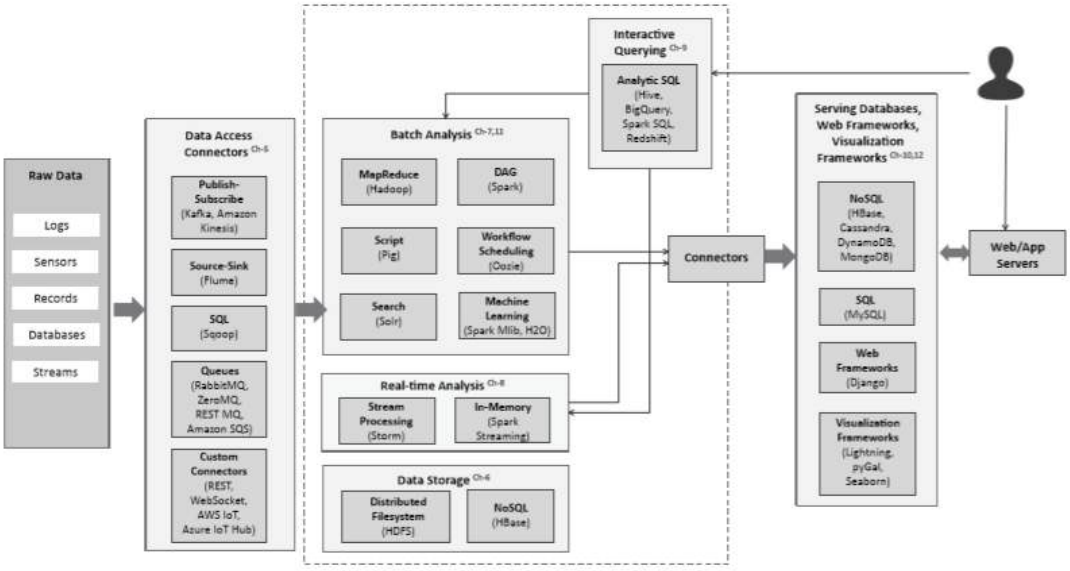
  Konektor Source-Sink memungkinkan pengumpulan, penggabungan, dan pemindahan data secara efisien dari berbagai sumber (seperti log server, database, media sosial, data sensor streaming dari perangkat Internet of Things dan sumber lainnya) ke dalam penyimpanan data terpusat (seperti sebagai sistem file terdistribusi). Di Bab-5 kami telah menjelaskan Apache Flume, yang merupakan kerangka kerja untuk mengumpulkan data dari berbagai sumber. Flume menggunakan model aliran data yang terdiri dari source, channel, dan sink.
- **Konektor Database**

  Konektor Database dapat digunakan untuk mengimpor data dari sistem manajemen Database relasional ke dalam penyimpanan Big Data dan kerangka kerja analisis untuk analisis. Di Bab-5 kami telah menjelaskan Apache Sqoop, yang merupakan alat yang memungkinkan mengimpor data dari database relasional.
- **Antrian Pesan**

  Antrian pesan berguna untuk pesan push-pull dimana produsen mendorong data ke antrian dan konsumen menarik data dari antrian. Produsen dan konsumen tidak perlu saling waspada. Di Bab-5 kami telah menjelaskan antrean perpesanan seperti RabbitMQ, ZeroMQ, RestMQ, dan Amazon SQS.

- **Konektor Khusus**

  Konektor khusus dapat dibuat berdasarkan sumber data dan persyaratan pengumpulan data. Beberapa contoh konektor kustom meliputi: konektor kustom untuk mengumpulkan data dari jejaring sosial, konektor kustom untuk database NoSQL, dan konektor untuk Internet of Things (IoT). Di Bab-5 kami telah menjelaskan konektor kustom berdasarkan REST, WebSocket, dan MQTT. Konektor IoT seperti AWS IoT dan Azure IoT Hub juga dijelaskan di Bab-5.




Gambar 1.3 *: Biga Data Stack*

### 1.6.3 Data Storage (Penyimpanan Data)

Blok penyimpanan data dalam Big Data Stack mencakup sistem file terdistribusi dan database non-relasional (NoSQL), yang menyimpan data yang dikumpulkan dari sumber raw data menggunakan konektor akses data. Di Bab-6, kami menjelaskan Hadoop Distributed File System (HDFS), sistem file terdistribusi yang berjalan pada cluster besar dan menyediakan akses throughput tinggi ke data.

Dengan data yang disimpan di HDFS, data dapat dianalisis dengan berbagai kerangka kerja analisis Big Data yang dibangun di atas HDFS. Untuk aplikasi analisis tertentu, sebaiknya menyimpan data dalam database NoSQL seperti HBase. HBase adalah database terukur, non-relasional, terdistribusi, berorientasi kolom yang menyediakan penyimpanan data terstruktur untuk tabel besar. Arsitektur HBase dan kasus penggunaannya dijelaskan pada Bab-4.

### 1.6.4 Analisis Batch

Blok analisis batch dalam Big Data Stack mencakup berbagai kerangka kerja yang memungkinkan analisis data dalam batch. Ini termasuk yang berikut:

- ***Hadoop-MapReduce***
  Hadoop adalah kerangka kerja untuk pemrosesan batch Big Data yang terdistribusi. Model pemrograman MapReduce digunakan untuk mengembangkan pekerjaan analisis batch yang dijalankan di cluster Hadoop. Contoh pekerjaan MapReduce dan studi kasus penggunaan Hadoop-MapReduce untuk analisis batch dijelaskan di Bab-7.
- **Pig**
  Pig adalah bahasa pemrosesan data tingkat tinggi yang memudahkan pengembang untuk menulis skrip analisis data yang diterjemahkan ke dalam program MapReduce oleh penyusun Pig. Contoh penggunaan Pig untuk analisis data batch dijelaskan di Bab-7.
- **Oozie**
  Oozie adalah sistem Scheduller aliran kerja yang memungkinkan pengelolaan pekerjaan Hadoop. Dengan Oozie, Anda dapat membuat alur kerja yang kumpulan area tindakannya (seperti pekerjaan MapReduce) disusun sebagai Direct Acyclic Graphs (DAG).
- **Spark**
  Apache Spark adalah kerangka kerja komputasi cluster open source untuk analisis data. Spark menyertakan berbagai alat tingkat tinggi untuk analisis data seperti Streaming Spark untuk pekerjaan streaming, Spark SQL untuk analisis data terstruktur, perpustakaan pembelajaran mesin MLlib untuk Spark, dan GraphX untuk pemrosesan grafik. Di Bab-7 kami menjelaskan arsitektur Spark, operasi Spark, dan cara menggunakan Spark untuk analisis data batch.
- **Solr**
  Apache Solr adalah kerangka kerja sumber terbuka dan skalabel untuk mencari data. Dalam Bab-7 kami menjelaskan arsitektur Solr dan contoh dokumen pengindeksan.
- **Pembelajaran Mesin**
  Di Bab-11 kami menjelaskan berbagai algoritme pembelajaran mesin dengan contoh-contoh yang menggunakan kerangka kerja Spark MLib dan H2O. Spark MLlib adalah pustaka pembelajaran mesin Spark yang menyediakan implementasi berbagai algoritma pembelajaran mesin. H2O adalah kerangka kerja analisis prediktif open source yang menyediakan implementasi berbagai algoritma pembelajaran mesin.

### 1.6.5 Analisis Real-time

Blok analisis real-time mencakup kerangka kerja Apache Storm dan Spark Streaming. Kerangka kerja ini dijelaskan secara rinci di Bab-8. Apache Storm adalah kerangka kerja untuk komputasi real-time terdistribusi dan toleran terhadap kesalahan. Storm dapat digunakan untuk pemrosesan aliran data secara real-time. Storm dapat menggunakan data dari berbagai sumber seperti kerangka perpesanan langganan-publikasi (seperti Kafka atau Kinesis), antrean perpesanan (seperti RabbitMQ atau ZeroMQ), dan konektor khusus lainnya.

Spark Streaming adalah komponen Spark yang memungkinkan analisis data streaming seperti data sensor, data aliran klik, log server web, misalnya. Data streaming diserap dan dianalisis dalam mikro-batch.

### 1.6.6 Kueri Interaktif

Sistem kueri interaktif memungkinkan pengguna untuk membuat kueri data dengan menulis pernyataan dalam bahasa mirip SQL. Kami menjelaskan sistem kueri interaktif berikut, dengan contoh, di Bab-9:

- **SparkSQL**
  Spark SQL adalah komponen Spark yang memungkinkan pembuatan kueri interaktif. Spark SQL berguna untuk membuat kueri data terstruktur dan semi-terstruktur menggunakan kueri seperti SQL.
- **Hive**
  Apache Hive adalah kerangka kerja data warehousing yang dibangun di atas Hadoop. Hive menyediakan bahasa kueri seperti SQL yang disebut Hive Query Language, untuk membuat kueri data yang berada di HDFS.
- **Amazon Redshift**
  Amazon Redshift adalah layanan gudang data terkelola skala besar dan cepat. Redshift mengkhususkan diri dalam menangani kueri pada kumpulan data dengan ukuran hingga satu mapbyte atau lebih yang memparalelkan kueri SQL di semua sumber daya di klaster Redshift.
- **Google BigQuery**
  Google BigQuery adalah layanan untuk menanyakan dataset yang sangat besar. BigQuery memungkinkan pembuatan kueri dataset menggunakan kueri seperti SQL.

### 1.6.7 Ukuran Database, Web & Kerangka Visualisasi

Sementara berbagai blok analisis memproses dan menganalisis data, hasilnya disimpan dalam database penyajian untuk tugas presentasi dan visualisasi selanjutnya. Penyajian Database ini memungkinkan data yang dianalisis untuk ditanyakan dan disajikan dalam aplikasi web. Di Bab-10, kami menjelaskan database SQL dan NoSQL berikut yang dapat digunakan sebagai penyajian database:

- **MySQL**
  MySQL adalah salah satu Relational Database Management System (RDBMS) yang paling banyak digunakan dan merupakan pilihan yang baik untuk digunakan sebagai pemyajian database untuk aplikasi analisis data terstruktur.
- **Amazon DynamoDB**
  Amazon DynamoDB adalah layanan database NoSQL yang dikelola sepenuhnya, dapat diskalakan, dan berperforma tinggi dari Amazon. DynamoDB adalah pilihan yang sangat baik untuk database penyajian untuk aplikasi analisis data karena memungkinkan penyimpanan dan pengambilan data dalam jumlah berapa pun dan kemampuan untuk menaikkan atau menurunkan throughput yang disediakan.
- **Cassandra**
  Apache Cassandra atau yang lebih dikenal dengan Cassandra adalah salah satu produk open source untuk menajemen database yang didistribusikan oleh Apache yang sangat scalable (dapat diukur) dan dirancang untuk mengelola data terstruktur yang berkapasitas sangat besar (Big Data) yang tersebar di banyak server. Cassandra merupakan salah satu implementasi dari NoSQL (Not Only SQL) seperti mongoDB. NoSQL merupakan konsep penyimpanan database dinamis yang tidak terikat pada relasi-relasi tabel yang kaku seperti RDBMS. Selain lebih scalable, NoSQL juga memiliki performa pengaksesan yang lebih cepat.
- **MongoDB**
  MongoDB adalah sistem database non-relasional yang berorientasi pada dokumen. MongoDB adalah database yang kuat, fleksibel, dan sangat dapat diskalakan yang dirancang untuk aplikasi web dan merupakan pilihan yang baik untuk database penyajian untuk aplikasi analisis data.

Dalam Bab-10, kami juga menjelaskan Django, yang merupakan kerangka aplikasi web sumber terbuka untuk mengembangkan aplikasi web dengan Python. Django didasarkan pada arsitektur Model-Tempelatee-View dan menyediakan pemisahan model data dari aturan bisnis dan interface pengguna. Meskipun aplikasi web dapat berguna untuk

menyajikan hasil, alat dan kerangka kerja visualisasi khusus dapat membantu dalam memahami data, dan hasil analisis dengan cepat dan mudah. Di Bab-12, kami menjelaskan alat dan kerangka visualisasi berikut:

- **Lightning**
  Lightning adalah kerangka kerja untuk membuat visualisasi interaktif berbasis web.
- **Pygal**
  Pustaka Python Pygal adalah pustaka diagram yang mudah digunakan yang mendukung diagram dari berbagai jenis.
- **Seaborn**
  Seaborn adalah pustaka visualisasi Python untuk merencanakan plot statistik yang menarik.

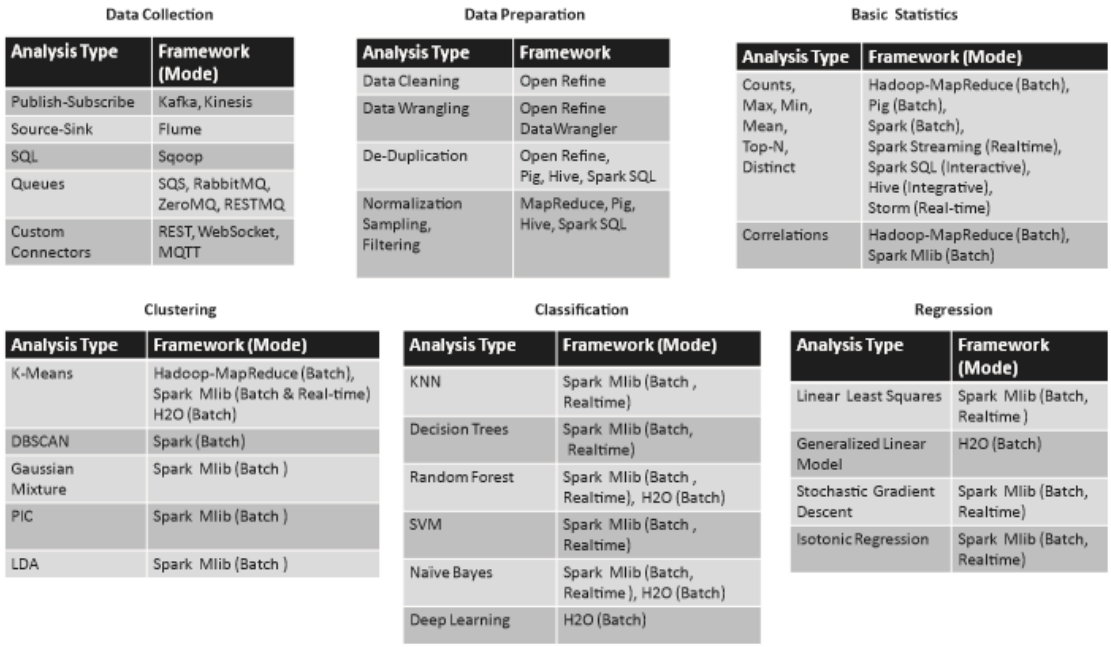
## 1.7 Memetakan Alur Analisis Tumpukan Data dalam Big Data (Big Data Stack)

Untuk aplikasi big data apa pun, setelah kami menemukan aliran analisis, langkah selanjutnya adalah memetakan aliran analisis ke alat dan kerangka kerja tertentu dalam Big Data Stack. Bagian ini memberikan beberapa pedoman dalam memetakan aliran analisis ke Big Data Stack. Untuk tugas pengumpulan data, pilihan alat atau kerangka kerja tertentu bergantung pada jenis sumber data (seperti file log, mesin yang menghasilkan data sensor, umpan media sosial, catatan dalam database relasional, misalnya) dan karakteristik data. Jika data akan diserap secara massal (seperti file log), maka sumber-sink seperti Apache Flume dapat digunakan. Namun, jika data berkecepatan tinggi akan diserap secara real-time, maka kerangka kerja perpesanan langganan-terbitkan seperti Apache Kafka atau Amazon Kinesis dapat digunakan. Untuk menyerap data dari database relasional, kerangka kerja seperti Apache Sqoop dapat digunakan. Konektor khusus dapat dibuat berdasarkan HTTP / REST, WebSocket atau MQTT, jika solusi lain tidak berfungsi dengan baik untuk aplikasi atau ada kendala tambahan. Misalnya, perangkat IoT yang menghasilkan data sensor mungkin memiliki sumber daya dan daya yang terbatas, dalam hal ini protokol komunikasi ringan seperti MQTT dapat dipilih dan konektor kustom berbasis MQTT dapat digunakan.

Untuk pembersihan dan transformasi data, alat seperti Open Refin [3] dan Stanford Data Wrangler [4] dapat digunakan. Alat ini mendukung berbagai format file seperti CSV, Excel, XML, JSON dan format berbasis baris. Dengan alat ini Anda dapat menghapus duplikat, menyaring catatan dengan nilai yang hilang, memangkas spasi di depan dan di

belakang, mengubah urutan baris ke kolom, mengubah nilai sel, mengelompokkan sel yang serupa, dan melakukan berbagai transformasi lainnya. Untuk memfilter, bergabung, dan transformasi lainnya, kerangka kerja skrip tingkat tinggi seperti Pig bisa sangat berguna. Manfaat menggunakan Pig adalah Anda dapat memproses data dalam jumlah besar dalam mode batch, yang mungkin sulit dilakukan dengan alat mandiri. Ketika Anda tidak yakin tentang transformasi apa yang harus diterapkan dan ingin menjelajahi data dan mencoba transformasi yang berbeda, kerangka kerja kueri interaktif seperti Hive, SparkSQL dapat berguna. Dengan alat ini, Anda dapat membuat kueri data dengan kueri yang ditulis dalam bahasa mirip SQL.

Untuk jenis analisis statistik dasar (dengan analisis seperti jumlah komputasi, maks, min, rata-rata, nilai-N, distinct, korelasi, misalnya), sebagian besar analisis dapat dilakukan menggunakan kerangka kerja Hadoop-MapReduce atau dengan skrip Pig. Baik MapReduce dan Pig memungkinkan analisis data dalam mode batch. Untuk statistik dasar dalam mode batch, kerangka kerja Spark juga merupakan pilihan yang bagus. Untuk statika dasar dalam mode real-time, kerangka kerja Spark Streaming dan Storm dapat digunakan. Untuk statistik dasar dalam mode interaktif, kerangka kerja seperti Hive dan SparkSQL dapat digunakan. Seperti, statistik dasar, kami juga dapat memetakan tipe analisis lain ke salah satu kerangka kerja di Big Data Stack. Gambar 1.4 dan 1.5 menunjukkan pemetaan antara berbagai jenis analisis dan kerangka Big Data.



Data Collection

| Analysis Type | Framework (Mode) |
|---|---|
| Publish-Subscribe | Kafka, Kinesis |
| Source-Sink | Flume |
| SQL | Sqoop |
| Queues | SQS, RabbitMQ, ZeroMQ, RESTMQ |
| Custom Connectors | REST, WebSocket, MQTT |

Data Preparation

| Analysis Type | Framework |
|---|---|
| Data Cleaning | Open Refine |
| Data Wrangling | Open Refine DataWrangler |
| De-Duplication | Open Refine, Pig, Hive, Spark SQL |
| Normalization Sampling, Filtering | MapReduce, Pig, Hive, Spark SQL |

Basic Statistics

| Analysis Type | Framework (Mode) |
|---|---|
| Counts, Max, Min, Mean, Top-N, Distinct | Hadoop-MapReduce (Batch), Pig (Batch), Spark (Batch), Spark Streaming (Realtime), Spark SQL (Interactive), Hive (Integrative), Storm (Real-time) |
| Correlations | Hadoop-MapReduce (Batch), Spark Mlib (Batch) |

Clustering

| Analysis Type | Framework (Mode) |
|---|---|
| K-Means | Hadoop-MapReduce (Batch), Spark Mlib (Batch & Real-time) H2O (Batch) |
| DBSCAN | Spark (Batch) |
| Gaussian Mixture | Spark Mlib (Batch ) |
| PIC | Spark Mlib (Batch ) |
| LDA | Spark Mlib (Batch ) |

Classification

| Analysis Type | Framework (Mode) |
|---|---|
| KNN | Spark Mlib (Batch , Realtime) |
| Decision Trees | Spark Mlib (Batch, Realtime) |
| Random Forest | Spark Mlib (Batch , Realtime), H2O (Batch) |
| SVM | Spark Mlib (Batch , Realtime) |
| Naïve Bayes | Spark Mlib (Batch, Realtime ), H2O (Batch) |
| Deep Learning | H2O (Batch) |

Regression

| Analysis Type | Framework (Mode) |
|---|---|
| Linear Least Squares | Spark Mlib (Batch, Realtime ) |
| Generalized Linear Model | H2O (Batch) |
| Stochastic Gradient Descent | Spark Mlib (Batch, Realtime) |
| Isotonic Regression | Spark Mlib (Batch, Realtime) |

Gambar 1.4: *Arus analisis Mapping untuk Big Data Stack – Part I*

Graph Analytics

| Analysis Type | Framework (Mode) |
|---|---|
| Graph Search | Spark GraphX (Batch) |
| Shortest-Path | Spark GraphX (Batch) |
| PageRank | Spark GraphX (Batch) |
| Triangle Counting | Spark GraphX (Batch) |
| Connected Components | Spark GraphX (Batch) |

Time Series Analysis

| Analysis Type | Framework (Mode) |
|---|---|
| Kalman Filter | Spark (Realtime) |
| Time Frequency Models | Spark (Realtime) |
| Outlier Detection | Storm (Realtime), Spark (Batch, Realtime) |

Dimensionality Reduction

| Analysis Type | Framework (Mode) |
|---|---|
| SVD | Spark Mlib (Batch ) |
| PCA | Spark Mlib (Batch ), H2O (Batch) |

Recommendation

| Analysis Type | Framework (Mode) |
|---|---|
| Item-bases Recommendation | Spark Mlib (Batch ) |
| Collaborative Filtering | Spark Mlib (Batch ) |

Text Analysis

| Analysis Type | Framework (Mode) |
|---|---|
| Categorization | Hadoop-MapReduce (Batch), Storm (Realtime), Spark (Batch, Realtime) |
| Summarization | Spark (Batch) |
| Sentiment Analysis | Storm (Realtime), Spark (Batch, Realtime) |
| Text Mining | Storm (Realtime), Spark (Batch, Realtime) |

Pattern Mining

| Analysis Type | Framework (Mode) |
|---|---|
| FP-Growth | Spark Mlib (Batch ) |
| Association Rules | Spark Mlib (Batch ) |
| PrefixSpan | Spark Mlib (Batch ) |

Visualization

| Analysis Type | Framework (Mode) |
|---|---|
| Web Frameworks | Django, Flask |
| SQL Databases | MySQL |
| NoSQL Databases | Hbase, DynamoDB, Cassandra, MongoDB |
| Visualization Frameworks | Lightning, pyGal, Seaborn |

Gmabar 1.5 *: Arus analitik Mapping untuk Big Data Stack – Part II*

## Studi Kasus

### Analisis Data Genomee

Mari kita lihat studi kasus penggunaan Big Data Stack untuk analisis data genome. Untuk studi kasus ini, kami akan menggunakan generator data sintetis yang disediakan dengan tolok ukur genomeik GenBase [2]. Generator data ini menghasilkan empat jenis dataset: (1) data microarray yang mencakup nilai ekspresi untuk sejumlah besar gen untuk pasien yang berbeda, (2) meta-data pasien yang berisi data demografis (usia pasien, jenis kelamin, kode pos) dan informasi klinis (penyakit dan respon obat) untuk setiap pasien yang data genomeiknya tersedia dalam dataset microarray, (3) meta-data gen yang berisi informasi seperti target gen (yaitu ID dari gen lain yang ditargetkan oleh protein dari gen saat ini), nomor kromosom, posisi (jumlah pasangan basa dari awal kromosom hingga awal gen), panjang (dalam pasangan basa) dan fungsi (dikodekan sebagai bilangan bulat), (4) Ontologi Gen ( GO) data yang menentukan kategori GO untuk gen yang berbeda. Gambar 1.6 menunjukkan sampel kecil untuk empat jenis dataset. Untuk mendapatkan pilihan alat dan kerangka kerja dari Big Data Stack yang dapat digunakan untuk analisis data genome, mari kita buat aliran analisis untuk aplikasi seperti yang ditunjukkan pada Gambar 1.7 (a).

***Koleksi Data***

Mari kita asumsikan bahwa kita memiliki kumpulan raw data yang tersedia dalam database SQL atau sebagai file teks mentah. Untuk mengimpor dataset dari database SQL ke dalam big data stack, kita dapat menggunakan konektor SQL. Sedangkan untuk mengimpor file dataset mentah, konektor source-sink dapat berguna.

***Persiapan Data***

Dalam langkah persiapan data, kami mungkin harus melakukan pembersihan data (untuk menghapus nilai yang hilang dan catatan yang rusak) dan perselisihan data (untuk mengubah catatan dalam format yang berbeda ke satu format yang konsisten).

***Tipe Analisis***

Katakanlah, untuk aplikasi ini kita ingin melakukan dua jenis analisis sebagai berikut: (1) memprediksi respon obat berdasarkan ekspresi gen, (2) menemukan korelasi antara nilai ekspresi dari semua pasang gen untuk menemukan gen yang memiliki ekspresi serupa pola dan gen yang memiliki pola ekspresi yang berlawanan. Analisis pertama berada di bawah kategori analisis regresi, di mana model regresi dapat dibangun untuk memprediksi respons obat. Variabel target untuk model regresi adalah respon obat pasien dan variabel independen adalah nilai ekspresi gen. Jenis analisis kedua berada di bawah kategori statistik dasar, di mana kami menghitung korelasi antara nilai ekspresi dari semua pasangan gen

***Mode Analisis***

Berdasarkan jenis analisis yang ditentukan pada langkah sebelumnya, diketahui bahwa mode analisis yang diperlukan untuk aplikasi bersifat batch dan interaktif.

***Visualisasi***

Aplikasi front end untuk memvisualisasikan hasil analisis akan bersifat dinamis dan interaktif.

***Memetakan Alur Analisis ke Big Data Stack***

Dengan aliran analisis untuk aplikasi yang dibuat, sekarang kita dapat memetakan pilihan di setiap langkah aliran ke Big Data Stack.

Gambar 1.7 (b) menunjukkan subset dari komponen Big Data Stack berdasarkan aliran analisis. Rincian implementasi aplikasi ini disediakan di Bab-11.

Gambar 1.8 menunjukkan langkah-langkah yang terlibat dalam membangun model regresi untuk memprediksi respons obat dan data di setiap langkah. Sebelum kita dapat membangun model regresi, kita harus melakukan beberapa transformasi dan penggabungan agar data sesuai untuk membangun model. Kami memilih gen dengan serangkaian fungsi tertentu dan menggabungkan meta-data gen dengan data meta pasien dan data microarray. Selanjutnya, kami melakukan pivot hasil untuk mendapatkan nilai ekspresi untuk setiap jenis gen untuk setiap pasien. Kemudian kami memilih ID pasien, penyakit dan respon obat dari data meta pasien. Selanjutnya,

kita menggabungkan tabel yang diperoleh dalam dua langkah sebelumnya untuk menghasilkan tabel baru yang memiliki semua data dalam format yang benar untuk membangun model regresi.

Gambar 1.9 menunjukkan langkah-langkah yang terlibat dalam menghitung korelasi antara tingkat ekspresi semua pasang gen dan data langkah mengajar. Kami memilih pasien dengan penyakit tertentu dan menggabungkan hasilnya dengan tabel microarray. Selanjutnya, kita pivot tabel pada langkah sebelumnya untuk mendapatkan nilai ekspresi semua gen untuk setiap pasien. Kami menggunakan tabel ini untuk membuat matriks korelasi yang memiliki korelasi antara nilai ekspresi dari semua pasangan gen.

Patient meta-data

| patientid | age | gender | zipcode | disease | drug response |
|---|---|---|---|---|---|
| 0 | 41 | 0 | 7494 | 15 | 84.77 |
| 1 | 45 | 1 | 38617 | 6 | 62.4 |
| 2 | 51 | 1 | 62817 | 17 | 49.43 |
| 3 | 62 | 0 | 53316 | 18 | 25.88 |
| 4 | 23 | 1 | 49685 | 7 | 41.03 |
| 5 | 60 | 0 | 48726 | 8 | 23.35 |
| 6 | 77 | 0 | 99103 | 18 | 87.86 |
| 7 | 83 | 1 | 5210 | 18 | 55.05 |
| 8 | 56 | 1 | 7359 | 5 | 97.09 |
| 9 | 63 | 1 | 59483 | 17 | 15.05 |

Gene meta-data

| geneid | target | position | length | function |
|---|---|---|---|---|
| 0 | -1 | 6.69E+08 | 175 | 633 |
| 1 | -1 | 2.74E+09 | 974 | 7 |
| 2 | -1 | 6.82E+08 | 260 | 909 |
| 3 | -1 | 2.4E+09 | 930 | 28 |
| 4 | 1 | 2.01E+09 | 836 | 462 |
| 5 | -1 | 1.64E+09 | 277 | 941 |
| 6 | -1 | 2.6E+09 | 428 | 487 |
| 7 | -1 | 1.02E+08 | 618 | 966 |
| 8 | 6 | 8.46E+08 | 635 | 328 |
| 9 | 3 | 2.77E+09 | 964 | 183 |

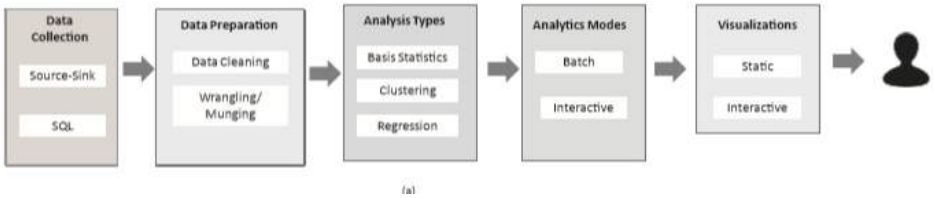
Micro-array data

| geneid | patientid | expression value |
|---|---|---|
| 0 | 0 | 7.51 |
| 0 | 1 | 5.92 |
| 0 | 2 | 8.12 |
| 0 | 3 | 3.47 |
| : | : | : |
| 1 | 1 | 7.43 |
| 1 | 2 | 5.54 |
| 1 | 3 | 2.86 |
| 2 | 0 | 7.69 |
| 2 | 1 | 7.66 |
| 2 | 2 | 9.76 |
| 2 | 3 | 1.41 |

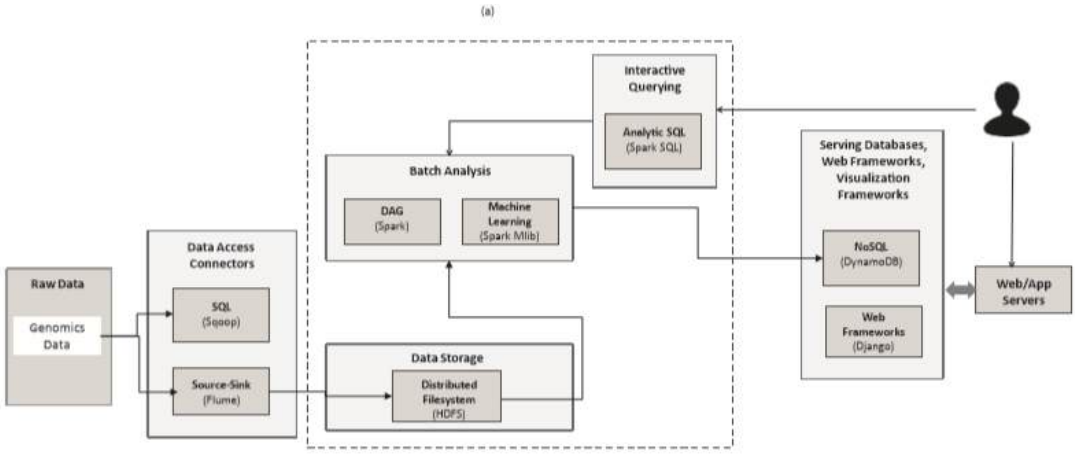
Gene Ontology data

| geneid | goid | whether gene belongs to go |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 2 | 1 |
| 0 | 3 | 1 |
| 0 | 6 | 0 |
| : | : | : |
| 9 | 59993 | 1 |
| 9 | 59994 | 1 |
| 9 | 59995 | 1 |
| 9 | 59996 | 1 |

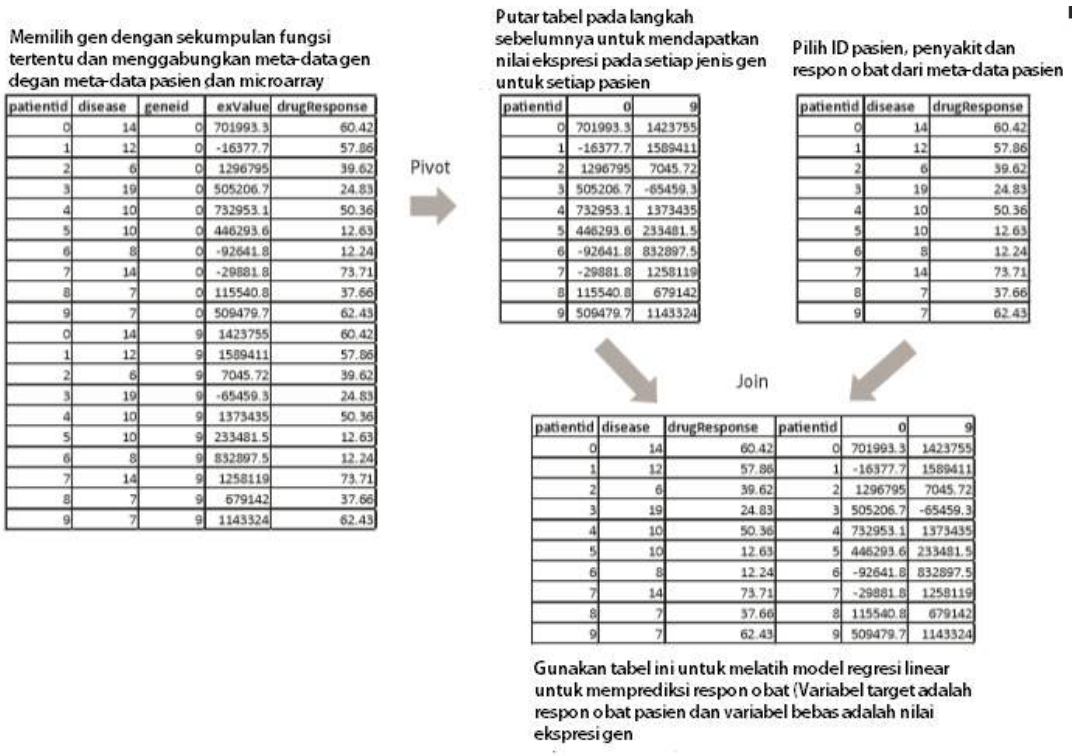
Gambar 1.6. *: Genome Dataset*




(a)




Gambar 1.7: *(a) Arus analisis utuk metadata analisis genome, (b) Penggunaan Big data stack pada metadata genome.*



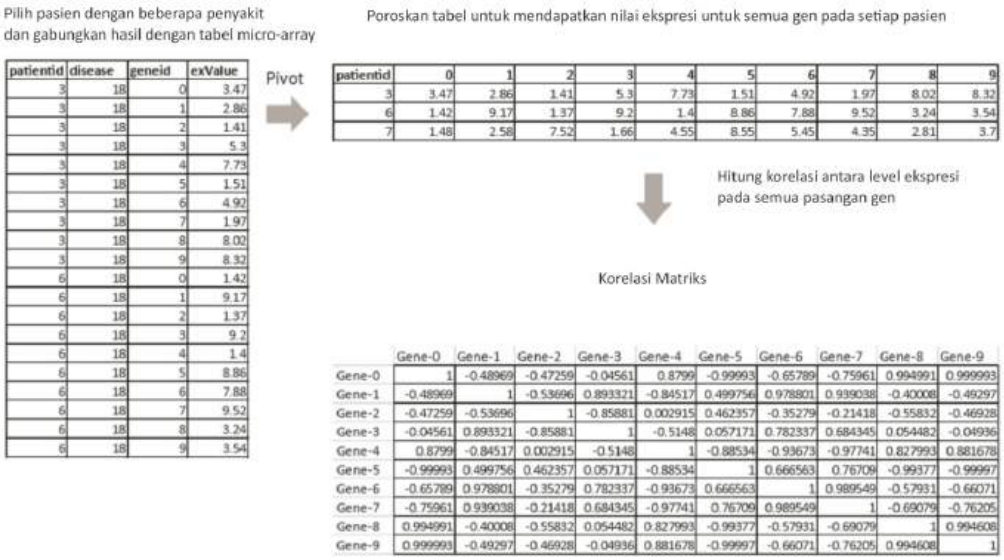

| patientid | disease | geneid | exValue | drugResponse |
|---|---|---|---|---|
| 0 | 14 | 0 | 701993.3 | 60.42 |
| 1 | 12 | 0 | -16377.7 | 57.86 |
| 2 | 6 | 0 | 1296795 | 39.62 |
| 3 | 19 | 0 | 505206.7 | 24.83 |
| 4 | 10 | 0 | 732953.1 | 50.36 |
| 5 | 10 | 0 | 446293.6 | 12.63 |
| 6 | 8 | 0 | -92641.8 | 12.24 |
| 7 | 14 | 0 | -29881.8 | 73.71 |
| 8 | 7 | 0 | 115540.8 | 37.66 |
| 9 | 7 | 0 | 509479.7 | 62.43 |
| 0 | 14 | 9 | 1423755 | 60.42 |
| 1 | 12 | 9 | 1589411 | 57.86 |
| 2 | 6 | 9 | 7045.72 | 39.62 |
| 3 | 19 | 9 | -65459.3 | 24.83 |
| 4 | 10 | 9 | 1373435 | 50.36 |
| 5 | 10 | 9 | 233481.5 | 12.63 |
| 6 | 8 | 9 | 832897.5 | 12.24 |
| 7 | 14 | 9 | 1258119 | 73.71 |
| 8 | 7 | 9 | 679142 | 37.66 |
| 9 | 7 | 9 | 1143324 | 62.43 |

| patientid | 0 | 9 |
|---|---|---|
| 0 | 701993.3 | 1423755 |
| 1 | -16377.7 | 1589411 |
| 2 | 1296795 | 7045.72 |
| 3 | 505206.7 | -65459.3 |
| 4 | 732953.1 | 1373435 |
| 5 | 446293.6 | 233481.5 |
| 6 | -92641.8 | 832897.5 |
| 7 | -29881.8 | 1258119 |
| 8 | 115540.8 | 679142 |
| 9 | 509479.7 | 1143324 |

| patientid | disease | drugResponse |
|---|---|---|
| 0 | 14 | 60.42 |
| 1 | 12 | 57.86 |
| 2 | 6 | 39.62 |
| 3 | 19 | 24.83 |
| 4 | 10 | 50.36 |
| 5 | 10 | 12.63 |
| 6 | 8 | 12.24 |
| 7 | 14 | 73.71 |
| 8 | 7 | 37.66 |
| 9 | 7 | 62.43 |

| patientid | disease | drugResponse | patientid | 0 | 9 |
|---|---|---|---|---|---|
| 0 | 14 | 60.42 | 0 | 701993.3 | 1423755 |
| 1 | 12 | 57.86 | 1 | -16377.7 | 1589411 |
| 2 | 6 | 39.62 | 2 | 1296795 | 7045.72 |
| 3 | 19 | 24.83 | 3 | 505206.7 | -65459.3 |
| 4 | 10 | 50.36 | 4 | 732953.1 | 1373435 |
| 5 | 10 | 12.63 | 5 | 446293.6 | 233481.5 |
| 6 | 8 | 12.24 | 6 | -92641.8 | 832897.5 |
| 7 | 14 | 73.71 | 7 | -29881.8 | 1258119 |
| 8 | 7 | 37.66 | 8 | 115540.8 | 679142 |
| 9 | 7 | 62.43 | 9 | 509479.7 | 1143324 |

Gambar 1.8 *: Langkah yang terlibat dalam membangun model regresi untuk memprediksi respon obat*




| patientid | disease | geneid | exValue |
|---|---|---|---|
| 3 | 18 | 0 | 3.47 |
| 3 | 18 | 1 | 2.86 |
| 3 | 18 | 2 | 1.41 |
| 3 | 18 | 3 | 5.3 |
| 3 | 18 | 4 | 7.73 |
| 3 | 18 | 5 | 1.51 |
| 3 | 18 | 6 | 4.92 |
| 3 | 18 | 7 | 1.97 |
| 3 | 18 | 8 | 8.02 |
| 3 | 18 | 9 | 8.32 |
| 6 | 18 | 0 | 1.42 |
| 6 | 18 | 1 | 9.17 |
| 6 | 18 | 2 | 1.37 |
| 6 | 18 | 3 | 9.2 |
| 6 | 18 | 4 | 1.4 |
| 6 | 18 | 5 | 8.86 |
| 6 | 18 | 6 | 7.88 |
| 6 | 18 | 7 | 9.52 |
| 6 | 18 | 8 | 3.24 |
| 6 | 18 | 9 | 3.54 |

| patientid | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 3 | 3.47 | 2.86 | 1.41 | 5.3 | 7.73 | 1.51 | 4.92 | 1.97 | 8.02 | 8.32 |
| 6 | 1.42 | 9.17 | 1.37 | 9.2 | 1.4 | 8.86 | 7.88 | 9.52 | 3.24 | 3.54 |
| 7 | 1.48 | 2.58 | 7.52 | 1.66 | 4.55 | 8.55 | 5.45 | 4.35 | 2.81 | 3.7 |

| | Gene-0 | Gene-1 | Gene-2 | Gene-3 | Gene-4 | Gene-5 | Gene-6 | Gene-7 | Gene-8 | Gene-9 |
|---|---|---|---|---|---|---|---|---|---|---|
| Gene-0 | 1 | -0.48969 | -0.47259 | -0.04561 | 0.8799 | -0.99993 | -0.65789 | -0.75961 | 0.994991 | 0.999993 |
| Gene-1 | -0.48969 | 1 | -0.53696 | 0.893321 | -0.84517 | 0.499756 | 0.978801 | 0.939038 | -0.40008 | -0.49297 |
| Gene-2 | -0.47259 | -0.53696 | 1 | -0.85881 | 0.002915 | 0.462357 | -0.35279 | -0.21418 | -0.55832 | -0.46928 |
| Gene-3 | -0.04561 | 0.893321 | -0.85881 | 1 | -0.5148 | 0.057171 | 0.782337 | 0.684345 | 0.054482 | -0.04936 |
| Gene-4 | 0.8799 | -0.84517 | 0.002915 | -0.5148 | 1 | -0.88534 | -0.93673 | -0.97741 | 0.827993 | 0.881678 |
| Gene-5 | -0.99993 | 0.499756 | 0.462357 | 0.057171 | -0.88534 | 1 | 0.666563 | 0.76709 | -0.99377 | -0.99997 |
| Gene-6 | -0.65789 | 0.978801 | -0.35279 | 0.782337 | -0.93673 | 0.666563 | 1 | 0.989549 | -0.57931 | -0.66071 |
| Gene-7 | -0.75961 | 0.939038 | -0.21418 | 0.684345 | -0.97741 | 0.76709 | 0.989549 | 1 | -0.69079 | -0.76205 |
| Gene-8 | 0.994991 | -0.40008 | -0.55832 | 0.054482 | 0.827993 | -0.99377 | -0.57931 | -0.69079 | 1 | 0.994608 |
| Gene-9 | 0.999993 | -0.49297 | -0.46928 | -0.04936 | 0.881678 | -0.99997 | -0.66071 | -0.76205 | 0.994608 | 1 |

Gambar 1.9 *: Langkah yang terlibat pada perhitungan korelasi antara tingkat level ekspresi pada semua pasangan gen*

## Studi Kasus

### Data Analisls Cuaca

Mari kita lihat studi kasus penggunaan Big Data Stack untuk analisis data cuaca. Untuk mendapatkan pilihan alat dan kerangka kerja dariBig Data Stack yang dapat digunakan untuk analisis data cuaca, pertama-tama mari kita buat aliran analisis untuk aplikasi seperti yang ditunjukkan pada Gambar 1.10.

***Koleksi Data***

Mari kita asumsikan, kita memiliki beberapa stasiun pemantau cuaca atau titik akhir yang dilengkapi dengan sensor suhu, kelembapan, angin, dan tekanan. Untuk mengumpulkan dan menyerap data sensor streaming yang dihasilkan oleh stasiun pemantauan cuaca, kita dapat menggunakan framework perpesanan langganan-publikasi untuk menyerap data untuk analisis real-time dalam Big Data Stack dan konektor Source-Sink untuk menyerap data ke dalam sistem file terdistribusi untuk analisis batch.

***Persiapan Data***

Karena data cuaca yang diterima dari stasiun pemantauan yang berbeda dapat memiliki nilai yang hilang, menggunakan unit yang berbeda dan memiliki format yang berbeda, kita mungkin perlu menyiapkan data untuk analisis dengan membersihkan, mengatur, menormalkan, dan memfilter data.

***Tipe Analisis***

Pilihan jenis analisis didorong oleh persyaratan aplikasi. Katakanlah, kita ingin aplikasi analisis cuaca kita mengumpulkan data pada berbagai skala waktu (menit, jam, harian atau bulanan) untuk menentukan mahasiswaan rata-rata, maksimum dan minimum untuk suhu, kelembaban, angin dan tekanan. Kami juga ingin aplikasi mendukung kueri interaktif untuk menjelajahi data, misalnya, kueri seperti: menemukan hari dengan suhu terendah setiap bulan dalam setahun, menemukan 10 hari paling basah teratas dalam setahun, misalnya. Jenis analisis ini termasuk dalam kategori statistik dasar. Selanjutnya kita juga ingin aplikasi membuat prediksi kejadian cuaca tertentu, misalnya memprediksi terjadinya kabut atau kabut asap. Untuk analisis seperti itu, kami membutuhkan model klasifikasi. Selain itu, jika kita ingin memprediksi nilai (seperti jumlah curah hujan), kita memerlukan model regresi.

***Mode Analisis***

Berdasarkan jenis analisis yang ditentukan pada langkah sebelumnya, diketahui bahwa mode analisis yang diperlukan untuk aplikasi adalah secara batch, real-time dan interaktif.

***Visualisasi***

Aplikasi front end untuk memvisualisasikan hasil analisis akan bersifat dinamis dan interaktif.

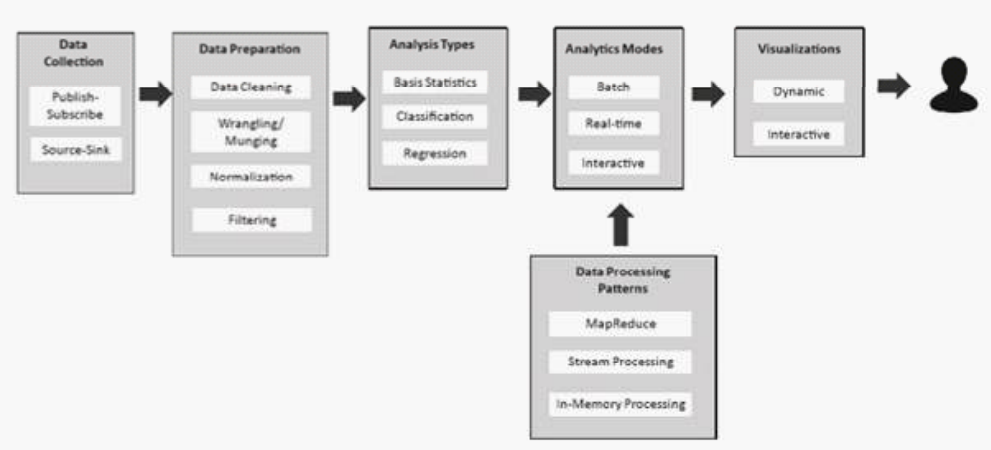
***Alur Analisis Pemetaan ke Big Data Stack***

Sekarang setelah kita memiliki aliran analisis untuk aplikasi, mari kita mapkan pilihan pada setiap langkah aliran ke Big Data Stack. Gambar 1.11 menunjukkan subset dari komponen Big Data Stack. berdasarkan aliran analisis. Untuk mengumpulkan dan mencerna data sensor streaming yang dihasilkan oleh stasiun pemantauan cuaca, kita dapat menggunakan kerangka perpesanan langganan-publikasi seperti Apache Kafka (untuk analisis real-time dalam Big Data Stack). Setiap stasiun cuaca mempublikasikan data sensor ke Kafka. Contoh Python untuk mempublikasikan data ke Kafka dijelaskan di Bab-5. Kerangka kerja analisis real-time seperti Storm dan Spark Streaming dapat menerima data dari Kafka untuk diproses. Contoh Python analisis real-time dengan Kafka sebagai sumber datanya dijelaskan di Bab-8.

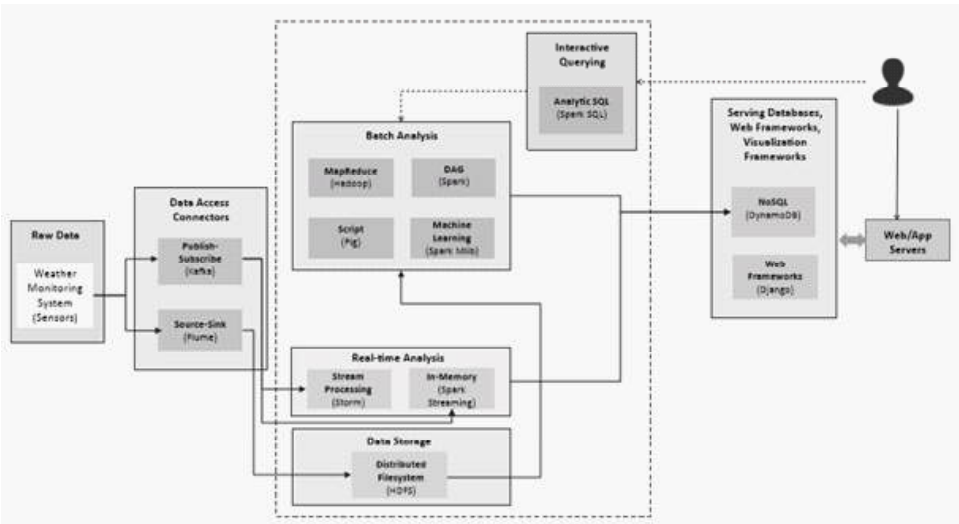
Untuk analisis batch, kita dapat menggunakan konektor sumber-sink seperti Flume untuk memindahkan data ke HDFS. Setelah data ada di HDFS, kita dapat menggunakan kerangka kerja pemrosesan batch seperti Hadoop-MapReduce. Kita juga dapat menggunakan Spark untuk transformasi Map dan Reduce tersebut. Contoh Python dari pemrosesan batch menggunakan Hadoop- MapReduce dan Spark dijelaskan di Bab-7.

Meskipun kerangka kerja pemrosesan batch dan real-time berguna saat tujuan dan persyaratan analisis diketahui sebelumnya, alat kueri interaktif dapat berguna untuk menjelajahi data. Kita bisa menggunakan kerangka kerja kueri interaktif seperti Spark SQL, yang bisa membuat kueri data dalam HDFS untuk kueri interaktif. Contoh Spark SQL untuk kueri interaktif data cuaca dijelaskan di Bab-9. Untuk menyajikan hasil analisis batch dan real-time, database NoSQL seperti DynamoDB dapat digunakan sebagai database penyajian. Contoh Python dalam menulis data ke tabel DynamoDB dan membaca data dari tabel dijelaskan di Bab-10.

Untuk mengembangkan aplikasi web dan menampilkan hasil analisis kita dapat menggunakan kerangka kerja web seperti Django. Contoh aplikasi Django dijelaskan di Bab-10.




Gambar 1.10: *Arus analisis untuk Aplikasi data analisis cuaca*




Gambar 1.11: *Penggunaan Big Data Stack untuk analisis data cuaca*

## 1.8 Pola Analisis

Pada bagian sebelumnya kami mengusulkan metodologi desain sistem aplikasi sains dan analisis data baru dan realisasinya melalui penggunaan kerangka Big Data sumber terbuka. Pada bagian ini kami mengusulkan, empat pola analisis: Alfa, Beta, Gamma dan Delta, yang terdiri dari alat dan kerangka kerja untuk mengumpulkan dan menyerap data dari berbagai sumber ke dalam infrastruktur analisis Big Data, sistem file terdistribusi, dan database non-relasional (NoSQL) untuk data penyimpanan, kerangka kerja pemrosesan untuk analisis batch dan real-time, kerangka kueri interaktif, database penyajian, serta kerangka web dan visualisasi. Pola-pola ini dan metodologi desain sistem aplikasi sains dan analisis data, membentuk fondasi pedagogis buku ini.

- **Pola Alpha**

  Gambar 1.12 (a) menunjukkan pola Alpha untuk analisis data batch. Pola ini dapat digunakan untuk menyerap volume data yang besar ke dalam sistem file terdistribusi (seperti HDFS) atau database NoSQL (seperti HBase) menggunakan konektor sumber-sink (seperti Flume) dan konektor SQL (seperti Sqoop). Setelah data dipindahkan ke tumpukan, data dapat dianalisis dalam mode batch dengan kerangka kerja analisis batch termasuk MapReduce (menggunakan Hadoop), kerangka kerja skrip (seperti Pig), kerangka grafik asiklik terdistribusi (seperti Spark), kerangka kerja pembelajaran mesin ( seperti Spark MLlib). Hasil analisis disimpan baik dalam database relasional maupun nonrelasional. Beberapa aplikasi khusus domain yang dijelaskan dalam bagian 1.4 yang dapat menggunakan pola Alpha meliputi: analisis web, pemantauan cuaca, pengawasan epidemiologi, dan diagnosis mesin.

- **Pola Beta**

  Gambar 1.12 (b) menunjukkan beta pattern untuk analisis real-time. Pola ini dapat digunakan untuk menyerap data streaming menggunakan framework perpesanan publish-subscribe, antrean, dan konektor khusus. Untuk analisis real-time, kita dapat menggunakan kerangka kerja pemrosesan aliran (seperti Storm) atau kerangka kerja pemrosesan dalam memori (seperti Spark). Beta pattern dapat digunakan oleh berbagai aplikasi Internet of Things dan aplikasi pemantauan real-time yang dijelaskan di bagian 1.4.

- **Pola Gamma**

  Gambar 1.13 (a) menunjukkan pola Gamma yang menggabungkan pola analisis batch dan real-time. Pola ini dimaksudkan untuk menyerap data streaming ke dalam Big Data Stack dan menganalisis data baik secara real-time maupun dalam mode batch. Untuk analisis batch, data dikumpulkan dan dianalisis selama interval tertentu. Misalnya, mari

kita lihat bagaimana pola ini dapat digunakan untuk sistem yang mendeteksi kebakaran hutan berdasarkan data sensor yang dikumpulkan dari sejumlah besar perangkat IoT yang diterapkan di hutan. Blok analisis real-time dalam pola ini dapat menyaring dan menganalisis data secara real-time dan membuat prediksi menggunakan model pembelajaran mesin terlatih. Sedangkan blok analisis batch dapat menganalisis data yang dikumpulkan selama interval tertentu (seperti per jam, harian, bulanan, atau tahunan).

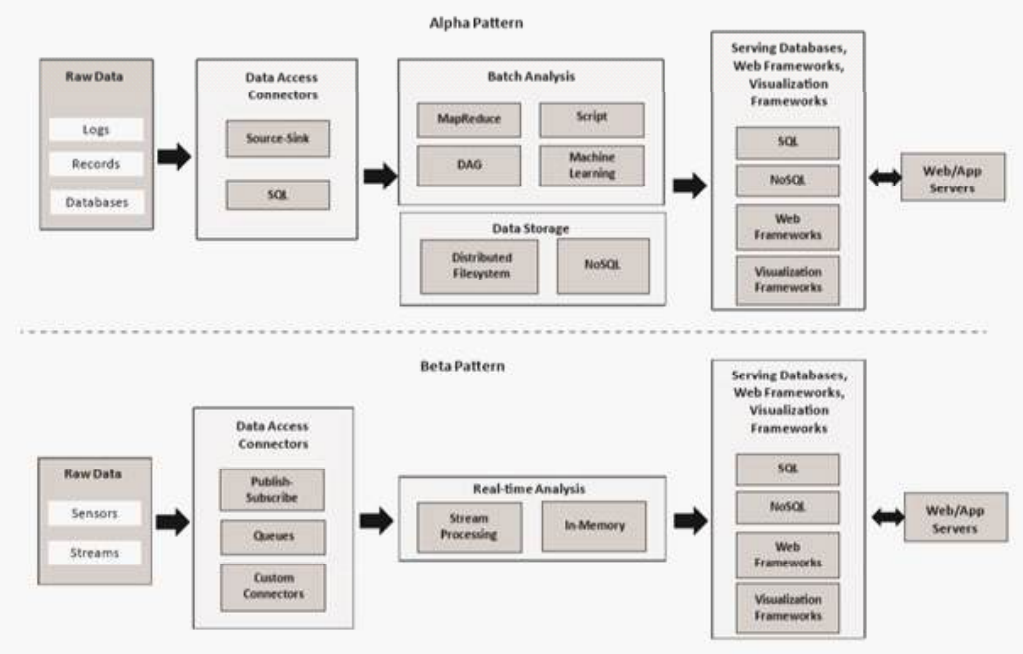
- **Pola Delta**

  Gambar 1.13 (b) menunjukkan pola Delta untuk kueri interaktif. Pola ini menggunakan, konektor sumber-sink (seperti Flume) atau konektor SQL (seperti Sqoop) untuk menyerap data massal ke dalam Big Data Stack. Setelah data dipindahkan ke sistem file terdistribusi, Anda dapat menggunakan kerangka kerja kueri interaktif (seperti Hive atau Spark SQL) untuk membuat kueri data dengan kueri seperti SQL dalam mode interaktif. Pola Delta dapat digunakan oleh aplikasi seperti analisis web, penargetan iklan, manajemen inventaris, perencanaan dan kontrol produksi, dan berbagai jenis aplikasi perusahaan.

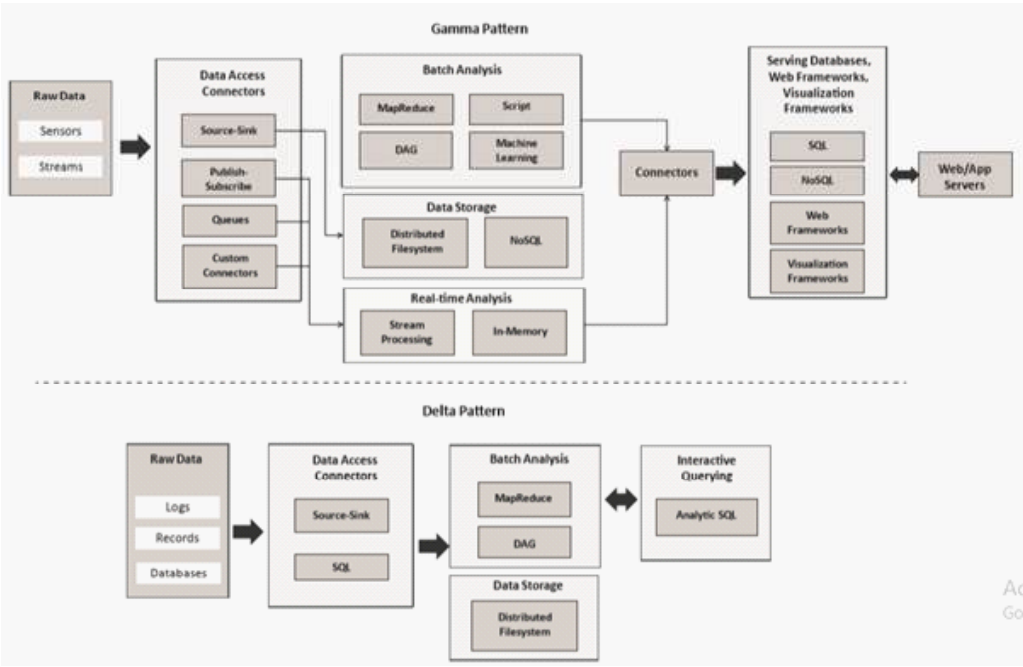
Pola yang diusulkan bersifat generik dan realisasi khusus dari pola ini dapat dibuat dengan memetakannya ke kerangka kerja tertentu atau layanan cloud dan analisis dari vendor cloud yang berbeda. Misalnya, pada Gambar 1.14 kami menyediakan pemetaan berbagai blok yang digunakan dalam pola analisis untuk layanan tertentu dari Amazon AWS dan Microsoft Azure. Pola tersebut memungkinkan penggunaan kembali kode, sehingga membuat proses desain lebih cepat. Tempelate kode untuk realisasi pola tertentu dapat digunakan untuk mengotomatiskan pembuatan atau konstruksi kode untuk berbagai komponen yang kemudian dapat disesuaikan untuk aplikasi tertentu.

Untuk masing-masing pola ini, terdapat beberapa tingkat kompleksitas dan opsi konfigurasi berdasarkan fitur seperti kinerja, skalabilitas, toleransi kesalahan, dan keamanan. Gambar1.15 menjelaskan berbagai level untuk pola ini. Misalnya, pola Alfa dapat diskalakan hingga ribuan node untuk menyimpan dan memproses beberapa mapbyte data. Demikian pula, beta pattern dapat diskalakan ke ratusan node untuk memproses sangat tinggi melalui data streaming. Sebagian besar kerangka kerja yang digunakan untuk pola ini telah didistribusikan dan arsitektur yang toleran terhadap kesalahan. Keamanan adalah aspek penting lainnya untuk aplikasi Big Data yang menyimpan dan memproses data sensitif. Untuk mengamankan kerangka Big Data, kerangka kerja keamanan khusus seperti Apache Ranger [16] dan Apache Knox [17] dapat digunakan. Apache Ranger,

misalnya, menghadirkan fitur keamanan seperti otorisasi, autentikasi, audit, enkripsi data, dan administrasi keamanan terpusat ke sebagian besar kerangka kerja yang dapat digunakan untuk mewujudkan pola ini. Apache Knox adalah Gateway API REST untuk cluster Hadoop, yang menyediakan fitur keamanan seperti otentikasi, federasi identitas otorisasi, dan audit.




Gambar 1.12:(a) *Analisis Batch Pola Alpha, (b) Pola Beta :Analisi Real-time*




Gambar 1.13 *: (a) Pola Gamma: Analisis Batch & Real-time, (b) Pola Delta: Kueri Interaktif*

| Data Access Connector | AWS Service | Azure Service |
|---|---|---|
| Publish-Subscribe | AWS Kinesis | Azure Event Hubs |
| Source-Sink | Flume on EMR | Flume on HDInsight |
| SQL | Sqoop on EMR | Sqoop on HDInsight |
| Queues | AWS SQS | Azure Queue Service |
| Custom Connectors | AWS IoT | Azure IoT Hub |

| Real-time Analysis | AWS Service | Azure Service |
|---|---|---|
| Stream Processing | Storm cluster on AWS EC2 | Storm on Azure HDInsight, Azure Stream Analytics |
| In-Memory Processing | Spark on AWS EMR | Spark on Azure HDInsight |

| Data Storage | AWS Service | Azure Service |
|---|---|---|
| Distributed Filesystem | HDFS on AWS EMR | HDFS on Azure HDInsight |
| NoSQL | AWS DynamoDB | Azure DocumentDB |

| Serving Databases, Web & Visualization Frameworks | AWS Service | Azure Service |
|---|---|---|
| SQL | AWS RDS | Azure SQL DB |
| NoSQL | AWS DynamoDB | Azure DocumentDB |
| Web Framework | Django on AWS EC2 instance | Django on Azure Virtual Machines instance |
| Visualization Framework | Lightning, Pygal, Seaborn, on AWS Virtual Machines instance | Lightning, Pygal, Seaborn, on Azure Virtual Machines instance, Azure Power BI |

| Batch Analysis | AWS Service | Azure Service |
|---|---|---|
| MapReduce | Hadoop on AWS EMR | Hadoop on Azure HDInsight |
| Script | Pig on AWS EMR | Pig on Azure HDInsight |
| DAG | Spark on AWS EMR | Spark on Azure HDInsight |
| Machine Learning | Spark MLlib on AWS EMR, AWS Machine Learning | Spark MLlib on Azure HDInsight, Azure Machine Learning |

Gambar 1.14: *Mapping blok pada pola analisis untuk AWS dan Azure Service*

| Level | | | | | |
|---|---|---|---|---|---|
| | Keamanan | • Apache Ranger : Otirosasi, autentikkasi, auditing, enkripsi data, Administrasi keamanan<br>• Apache Knox Gateway: REST API gateway, LDAP dan otentikasi direktori aktif, Federasi/SSO, Otorisasi, Auditing | | | |
| | Toleransi kesalahan | • Distribusi, Toleransi kesalahan dan ketersediaan yang tinggi<br>• Replikasi data, Failover otomatis | | | |
| | Skalabilitas | Dapat di skalakan ke ribuan node yang dapat menyimpan dan memproses beberapa pitabyte data | Skalabilitas pada ribuat node yang dapat memproses througput data streaming yang sangat tinggi | Skalabilitas pada ribuat node yang dapat memproses througput data streaming yang sangat tinggi | Skalabilitas pada ribuan node dapan menyimpan dan dapat memproses beberapa pitabyte pada data |
| | Performa | Memproses volume besar pada data dengan skala waktu pada beberapa menit | Proses streaming data skala waktu pada milisecon hingga menit | Kombinasi pemprosesan real-time dan batch untuk data streaming, dengan skala waktu milisecon, hingga menit untuk real-time, dan beberapa menit untuk Batch | Interaktivitas kueari volume besar oada data dalam skala waktu pada beberapa milisecond hingga menit |
| | Fungsionali-tas | Pemrosesan Batch | Pemrosesan Real time | Pemrosesan Real-time dan Batch | Kueri Interaktif |
| | | **Alpha** | **Beta** | **Gamma** | **Delta** |
| | **Pola Analalisis** | | | | |

Gambar 1.15: *Kompleksitas level untuk pola analisis*

## Ringkasan

Dalam bab ini, kami menjelaskan apa itu analisis dan berbagai jenis aplikasi analisis. Analisis adalah proses mengekstraksi dan membuat informasi dari raw data dengan menyaring, memproses, mengkategorikan, memadatkan, dan mengontekstualisasikan data. Analisis deskriptif berkaitan dengan menganalisis data masa lalu untuk disajikan dalam bentuk ringkasan yang dapat dengan mudah ditafsirkan. Analisis diagnostik berhubungan dengan analisis data masa lalu untuk mendiagnosis alasan terjadinya peristiwa tertentu. Analisis prediktif melibatkan prediksi terjadinya suatu peristiwa atau kemungkinan hasil dari suatu peristiwa atau perkiraan nilai masa depan menggunakan model prediksi. Analisis preskriptif menggunakan beberapa model prediksi untuk memprediksi berbagai hasil dan tindakan terbaik untuk setiap hasil. Analisis Big Data berhubungan dengan pengumpulan, penyimpanan, pemrosesan, dan analisis data skala besar.

Analisis Big Data melibatkan beberapa langkah mulai dari pembersihan data, data munging (atau pertengkaran), pemrosesan dan visualisasi data. Kami menjelaskan karakteristik big data termasuk volume, kecepatan, variasi, kebenaran, dan nilai. Contoh spesifik domain dan aplikasi big data dijelaskan. Selanjutnya, kami mengusulkan aliran umum untuk analisis Big Data, merinci langkah-langkah yang terlibat dalam tugas analisis khas dan opsi yang tersedia di setiap langkah. Langkah persiapan data melibatkan berbagai tugas seperti pembersihan data, perselisihan atau munging data, de-duplikasi, normalisasi, pengambilan sampel dan pemfilteran. Kami menggambarkan berbagai jenis analisis dan 'raksasa' komputasi. Kami mengusulkan dan mendeskripsikan Big Data Stack yang terdiri dari kerangka kerja Big Data yang terbukti dan bersumber terbuka yang membentuk dasar dari buku ini. Sebuah pendekatan untuk memetakan aliran analisis ke alat dan kerangka kerja tertentu dalam Big Data Stack telah dijelaskan. Akhirnya, kami mengusulkan empat pola analisis yang mencakup berbagai kategori kerangka Big Data seperti akuisisi data, penyimpanan data, analisis batch, analisis real-time, kueri interaktif, database penyajian, dan kerangka kerja web dan visualisasi. Pola yang diusulkan bersifat generik dan realisasi khusus dari pola ini dapat dibuat dengan memetakannya ke kerangka kerja tertentu atau layanan cloud dan analisis dari vendor cloud yang berbeda.

# Penyiapan Big Data Stack

Bab 2

Bab ini mencakup :

- Menyiapkan Big Data Stack
- Platform Data Hortonworks
- Cloudera CDH Stack
- Kluster Amazon EMR
- Kluster Azure HDInsight

Bab ini memberikan berbagai opsi untuk menyiapkan Big data stack dan kerangka Big data yang digunakan untuk contoh dalam buku ini. Merek dagang milik pemiliknya masing-masing.

## 2.1. Platform Data Hortonworks (HDP)

*Hortonworks Data Platform* (HDP) [7] adalah distribusi platform sumber terbuka yang terdiri dari berbagai kerangka Big data untuk akses dan integrasi data, pemrosesan batch, pemrosesan real-time, kueri interaktif, alat keamanan dan operasi. Kerangka kerja utama dalam tumpukan HDP yang digunakan untuk contoh dalam buku ini meliputi:

- Hadoop
- YARN
- HDFS
- HBase
- Hive
- Pig
- Sqoop
- Flume
- Oozie
- Storm
- Zookeeper
- Kafka
- Spark

Untuk menyiapkan kerangka kerja ini, kami merekomendasikan menggunakan Apache Ambari [15].

Ambari adalah alat untuk menyediakan, mengelola, dan memantau kluster yang menjalankan kerangka kerja ini. Bagian ini memberikan petunjuk untuk menyiapkan big data stack yang terdiri dari kerangka kerja yang tercantum di atas menggunakan Apache Ambari. Kluster Ambari dapat disiapkan di semua mesin yang menjalankan sistem operasi yang didukung berikut ini:

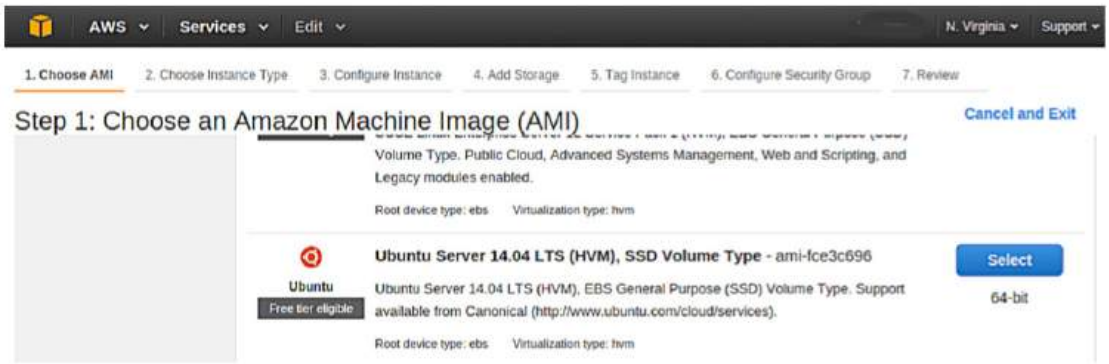
- Ubuntu Precise 12.04 atau 14.04
- Red Hat Enterprise Linux (RHEL) v6.x
- CentOS v6.x
- Oracle Linux v6.x
- SUSE Linux Enterprise Server (SLES) v11, SP1 and SP3

## 2.2 Launching Instan AWS EC2

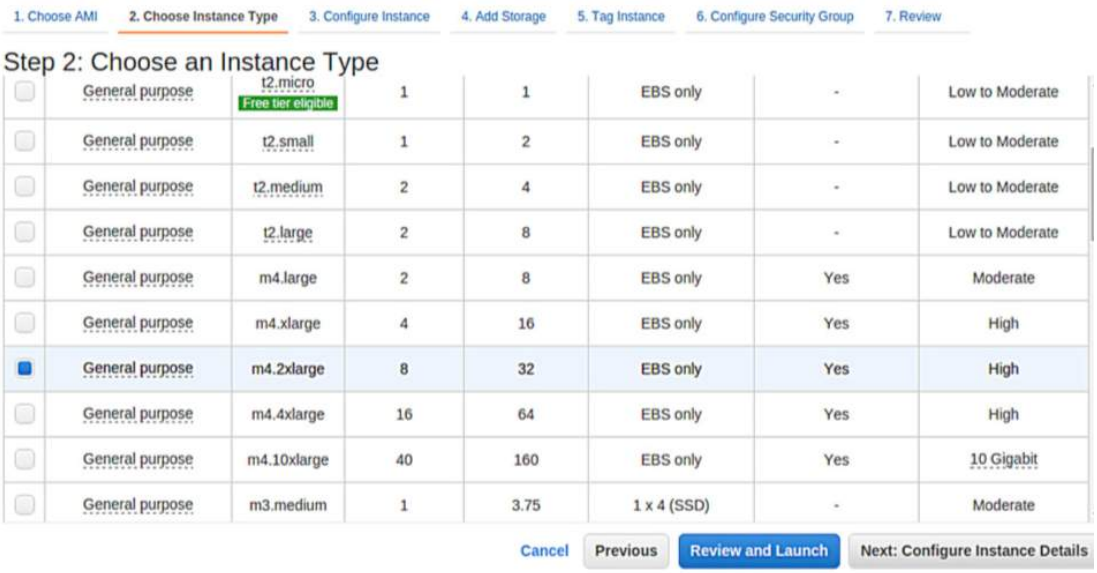
Di bagian ini, kami menjelaskan cara meluncurkan instans Amazon EC2 yang menjalankan Ubuntu 14.04, yang nantinya akan kami siapkan cluster Apache Ambari. Masuk ke akun Amazon AWS Anda dan buka konsol EC2. Untuk meluncurkan klik instan baru pada tombol instans peluncuran dari konsol Amazon EC2. Pilih Ubuntu 14.04 Amazon Machine Image (AMI) seperti yang ditunjukkan pada Gambar 2.1.

Selanjutnya, pilih tipe instance seperti yang ditunjukkan pada Gambar 2.2. Kami merekomendasikan jenis instans m4.2xlarge jika Anda ingin menyiapkan klaster Ambari pada satu instans.

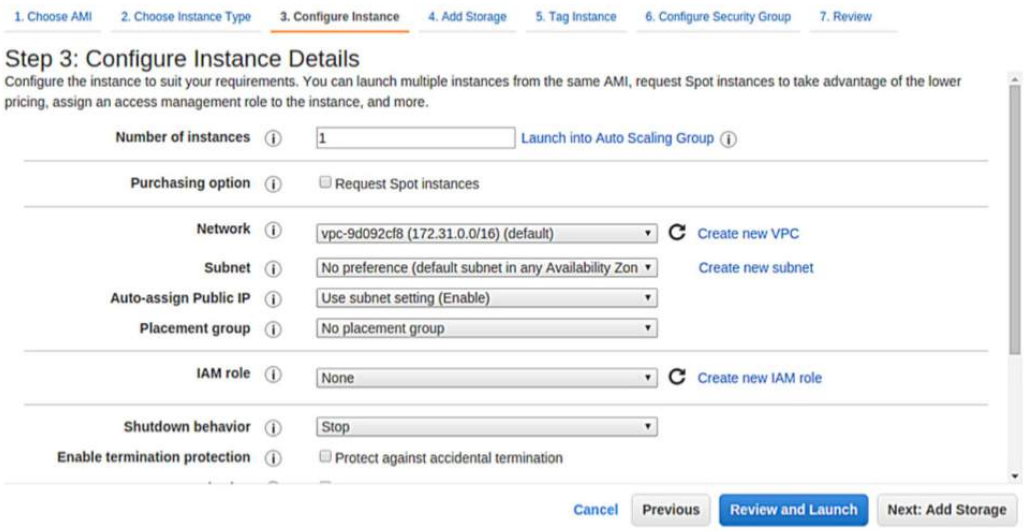
Pada langkah selanjutnya, konfigurasikan detail instance seperti yang ditunjukkan pada Gambar 2.3.




Gambar 2.1: *Pilih Ubuntu 14.04 AMI*

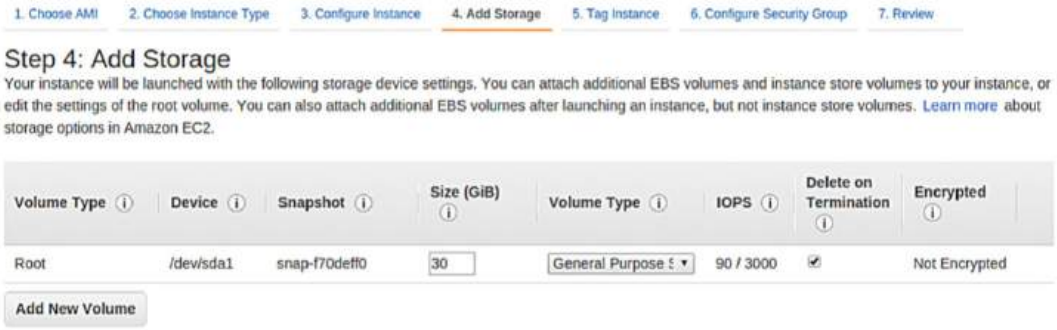



Gambar 2.2: *Pilih jenis instance*




Gambar 2.3: *Panduan peluncuran instans Amazon EC2 yang menampilkan detail instans*

Tambahkan penyimpanan pada langkah selanjutnya seperti yang ditunjukkan pada Gambar 2.4. Kami merekomendasikan penyimpanan minimal 20GB.




Gambar 2.4: *Panduan peluncuran instans Amazon EC2 yang menampilkan detail penyimpanan*

Tentukan tag instance yang dapat digunakan untuk mengidentifikasi instance seperti yang ditunjukkan pada Gambar 2.5.

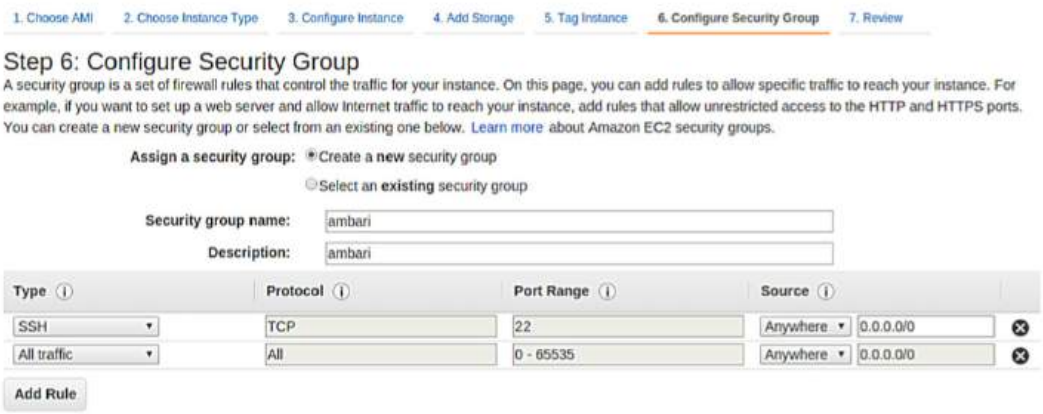

Gambar 2.5: *Panduan peluncuran instans Amazon EC2 yang menampilkan tag instans*

Gambar 2.6 menunjukkan halaman grup keamanan dari wizard peluncuran instance. Halaman ini memungkinkan Anda untuk memilih grup keamanan yang ada atau membuat grup keamanan baru. Grup keamanan digunakan untuk membuka atau memblokir port jaringan tertentu untuk instans yang diluncurkan. Buat grup keamanan baru dan buka semua lalu lintas TCP. Kerangka kerja yang akan kami siapkan dengan Apache Ambari menggunakan port yang berbeda untuk interface web mereka. Jadi lebih mudah (meskipun tidak direkomendasikan dalam lingkungan produksi) untuk membuka semua lalu lintas TCP tanpa harus mengidentifikasi port individu untuk berbagai kerangka kerja.

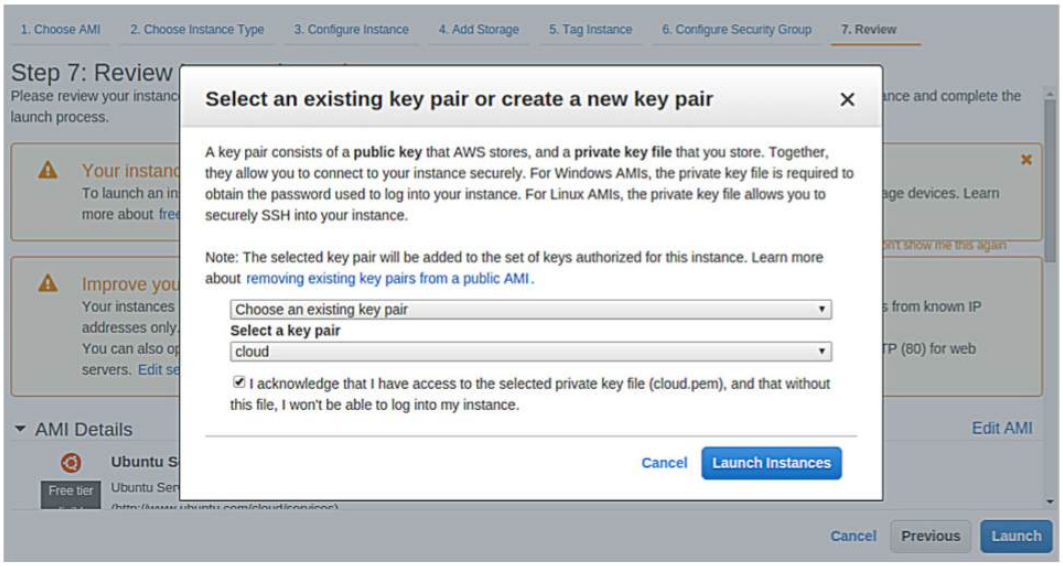
Pada langkah terakhir, tinjau detail instance dan buat pasangan kunci baru (atau pilih pasangan kunci yang ada) untuk instance dan luncurkan instance seperti yang ditunjukkan pada Gambar 2.7.

Status instans yang diluncurkan dapat dilihat di konsol EC2. Saat sebuah instance diluncurkan, statusnya menunggu keputusan. Perlu waktu beberapa menit agar instance menjadi berjalan. Saat instance masuk ke status berjalan, itu diberikan DNS publik, DNS

pribadi, IP publik dan IP pribadi. Kami akan menggunakan DNS publik untuk terhubung dengan aman ke instance menggunakan SSH.




Gambar 2.6: *Panduan peluncuran instans Amazon EC2 yang menampilkan grup keamanan*




Gambar 2.7: *Panduan peluncuran instans Amazon EC2 yang menampilkan window pemilihan pasangan kunci*

## 2.3 Pengaturan Apache Ambari

Hubungkan ke instans EC2 yang diluncurkan di bagian sebelumnya dari mesin lokal Anda menggunakan:

```
■ ssh -i myKeyPair.pem ubuntu@publicDNS
```

dengan public DNS adalah DNS Publik dari instance yang Anda buat.

Box 2.1 menunjukkan perintah untuk menginstal Apache Ambari, dependensi, dan beberapa paket lain yang digunakan sebagai contoh dalam buku ini.\

**Box 2.1: Commands for setting up Ambari**

```
sudo apt-get -q -y update
sudo ufw disable
sudo apt-get -q -y install ntp
sudo service ntp start
sudo wget http://public-repo-1.hortonworks.com/ambari/
ubuntu12/1.x/updates/1.7.0/ambari.list

sudo cp ambari.list /etc/apt/sources.list.d/
sudo apt-key adv -recv-keys -keyserver
keyserver.ubuntu.com B9733A7A07513CAD
sudo apt-get -q -y update

sudo apt-get -q -y install ambari-server

sudo apt-get -q -y install ant gcc g++ libkrb5-dev libmysqlclient-dev
libssl-dev libsasl2-dev libsasl2-modules-gssapi-mit libsqlite3-dev
libtidy-0.99-0 libxml2-dev libxslt-dev python-dev python-simplejson
python-setuptools maven libldap2-dev python2.7-dev make python-pip
```

Di mesin lokal Anda, jalankan perintah berikut untuk menyalin pasangan kunci ke instans EC2 (ubah DNS publik ke DNS publik instans Anda):

```
scp -i myKeyPair myKeyPair.pem ubuntu@publicDNS:/home/ubuntu/.ssh/
```

Pada instans EC2 Anda, ubah nama keypair yang baru saja Anda salin dari mesin lokal Anda menjadi id_rsa:

```
cd /home/ubuntu/.ssh/
mv myKeyPair.pem id_rsa
```

Jalankan perintah berikut untuk mengatur Apache Ambari:

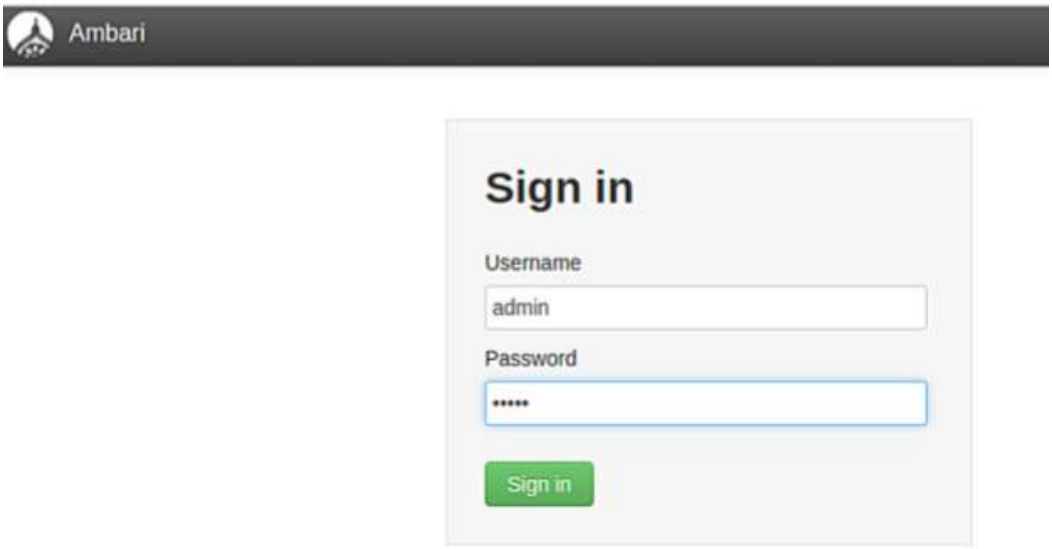
```
sudo ambari-server setup
```

Mulai Apache Ambari:

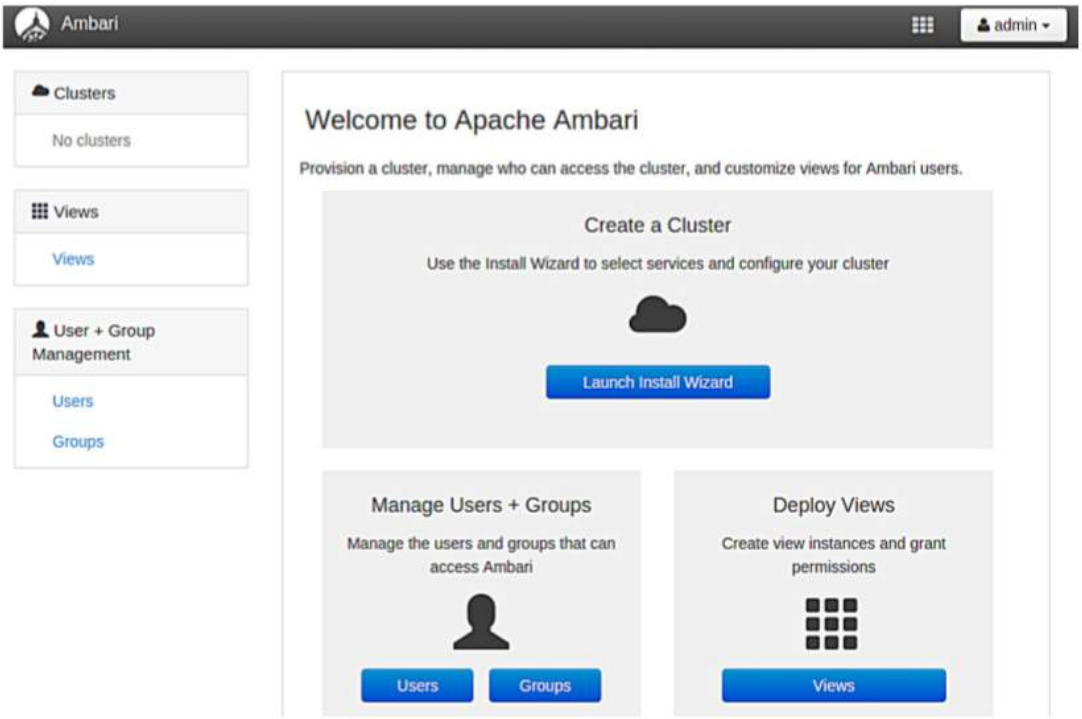
```
sudo ambari-server start
```

## 2.4 Menyiapkan HDP Stack dengan Ambari

Buka URL http: // <publicDNS>: 8080 di browser. Login ke Server Ambari dengan username admin dan password admin seperti yang ditunjukkan pada Gambar 2.8.

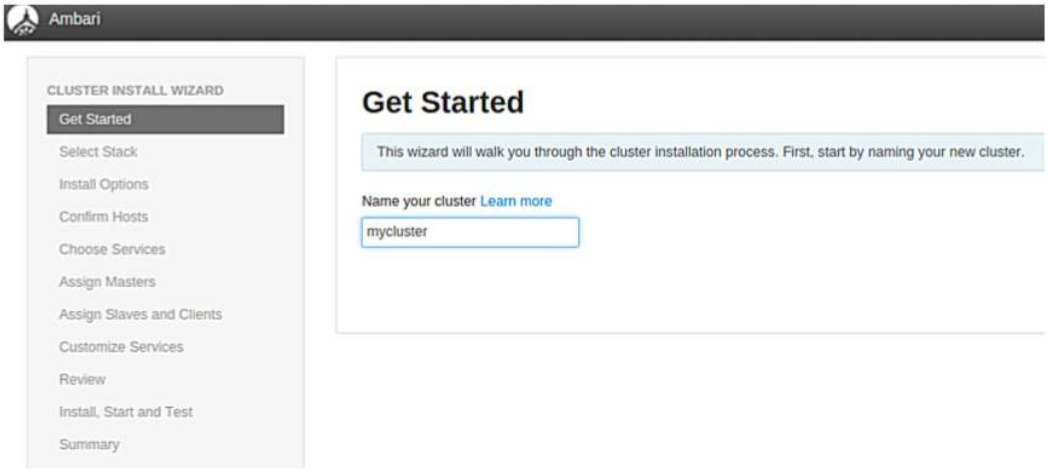



Gambar 2.8: *Halaman login Apache Ambari*

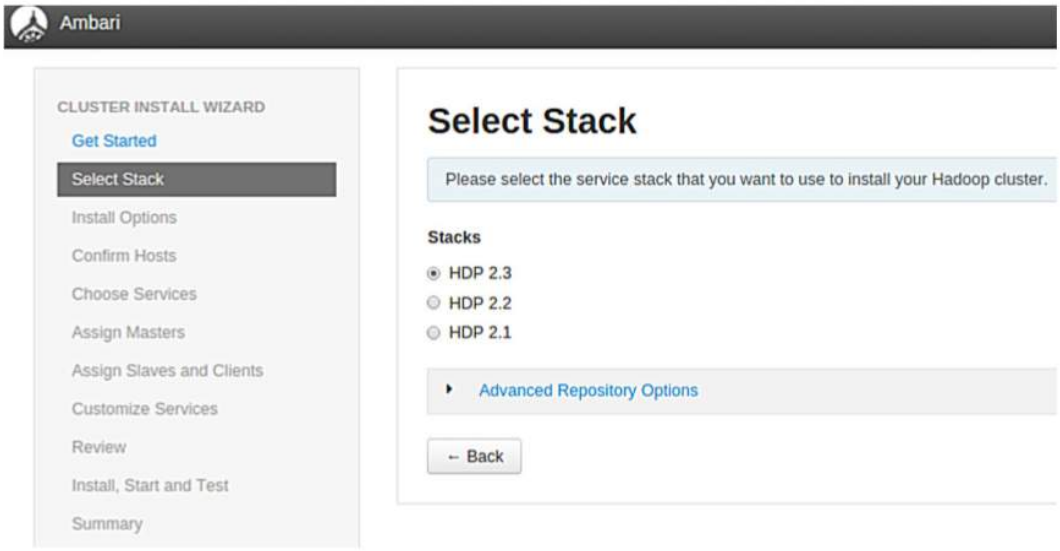



Gambar 2.9: *Wizard penyiapan Apache Ambari*

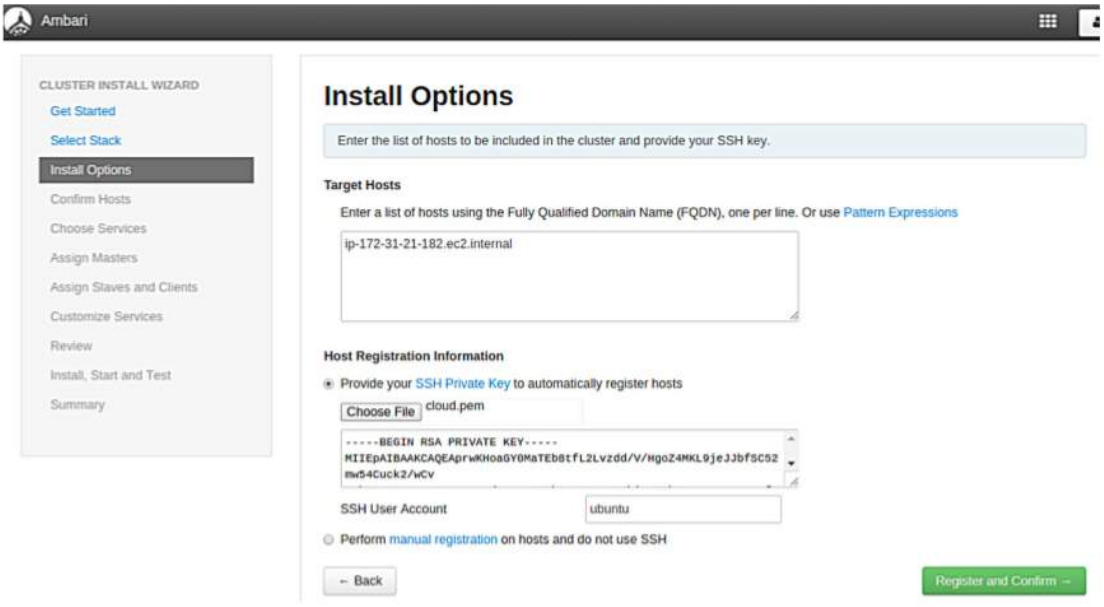
Selanjutnya, luncurkan wizard penginstalan seperti yang ditunjukkan pada Gambar 2.9. Berikan nama untuk cluster seperti yang ditunjukkan pada Gambar 2.10 dan kemudian pilih tumpukan HDP yang akan diinstal seperti yang ditunjukkan pada Gambar 2.11. Masukkan DNS pribadi dari instance tersebut di bagian Host Target. Pilih file pasangan kunci EC2 yang terkait dengan  dan ubah pengguna SSH ke ubuntu seperti yang ditunjukkan pada Gambar 2.11. Ikuti wizard untuk membuat cluster seperti yang ditunjukkan pada Gambar 2.12 - 2.18.



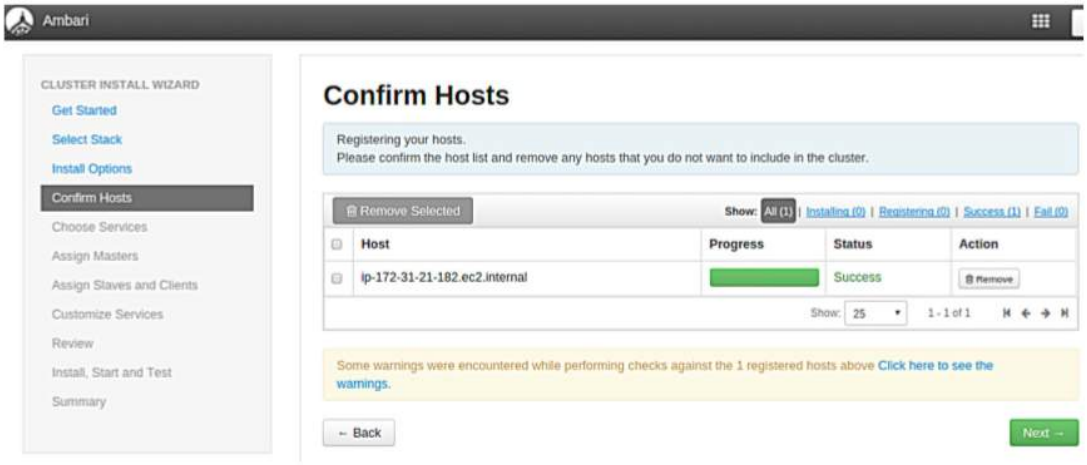

Gambar 2.10: *Wizard penyiapan Apache Ambari – Name your cluster*




Gambar 2.11: *Wizard penyiapan Apache Ambari - pilih tumpukan*

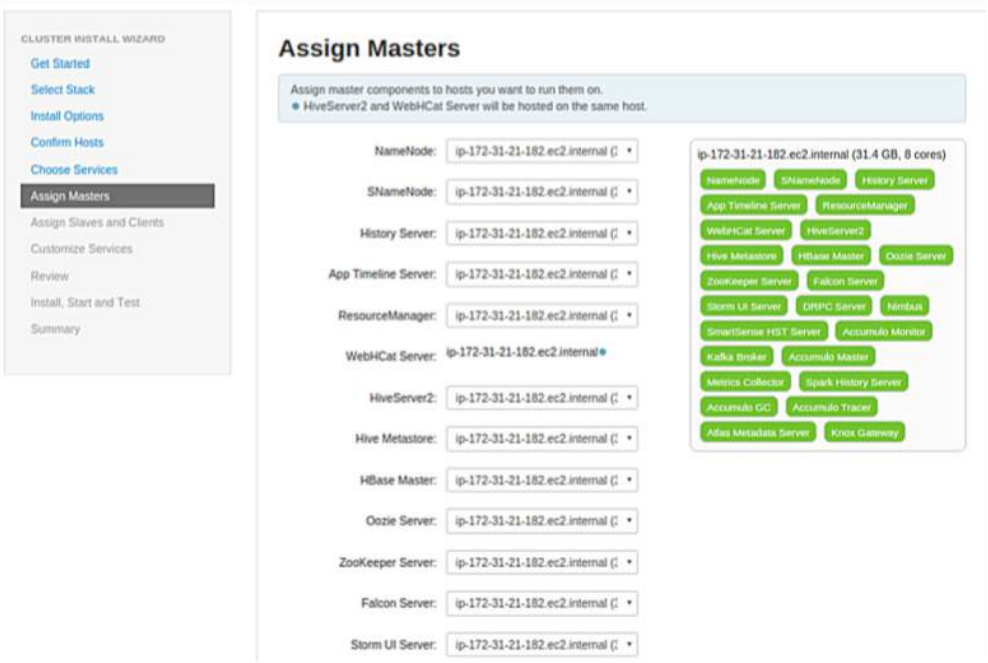



Gambar 2.12: *Wizard penyiapan Apache Ambari - opsi penginstalan*

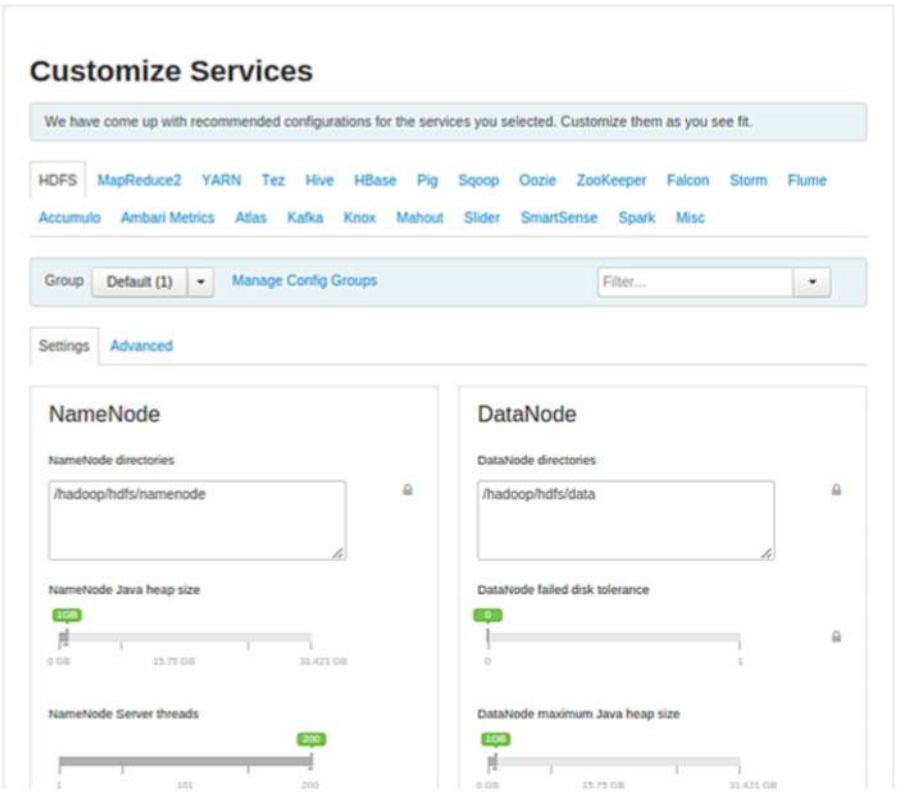



Gambar 2.13: *Wizard penyiapan Apache Ambari - host konfirmasi*

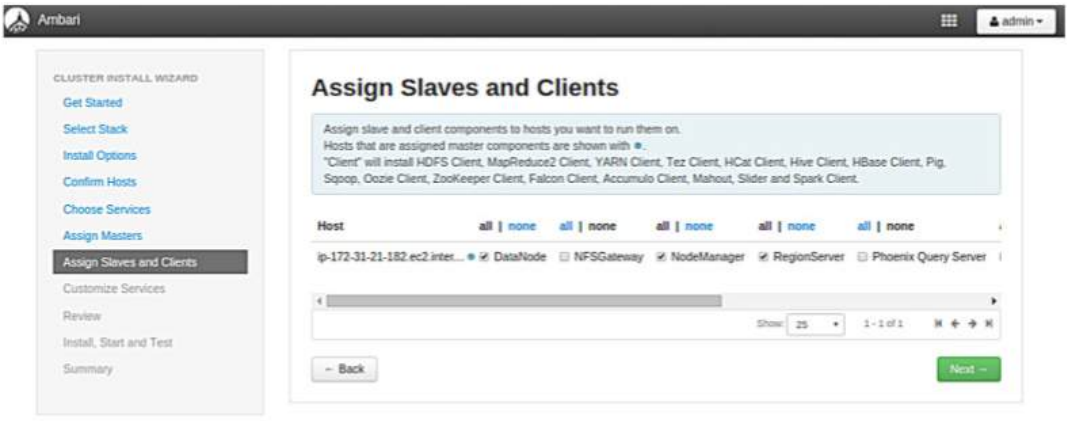

Gambar 2.14: *Wizard penyiapan Apache Ambari - pilih layanan*



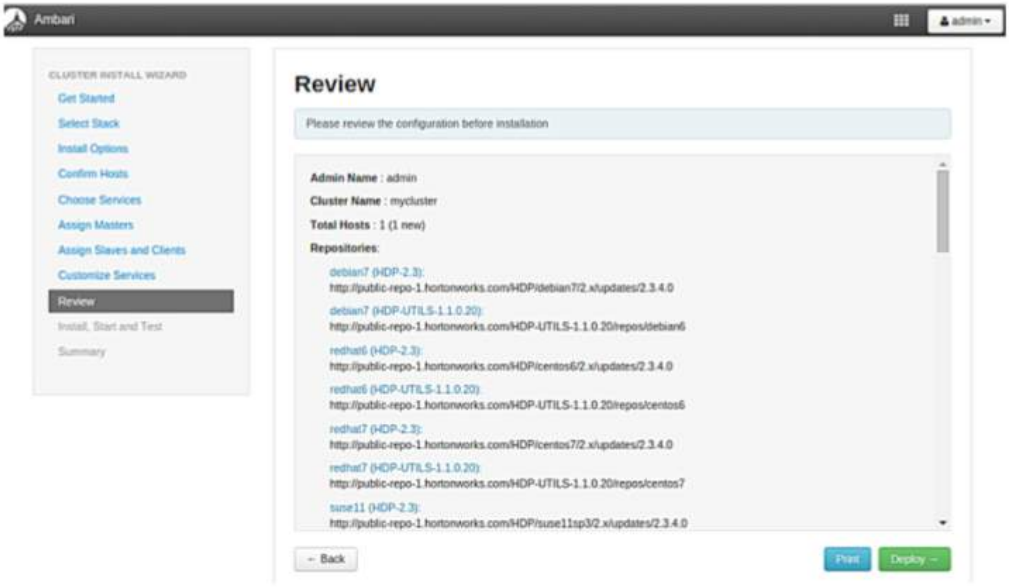

Gambar 2.15: *Wizard penyiapan Apache Ambari - tetapkan master*




Gambar 2.16: *Wizard penyiapan Apache Ambari - tetapkan budak dan klien*

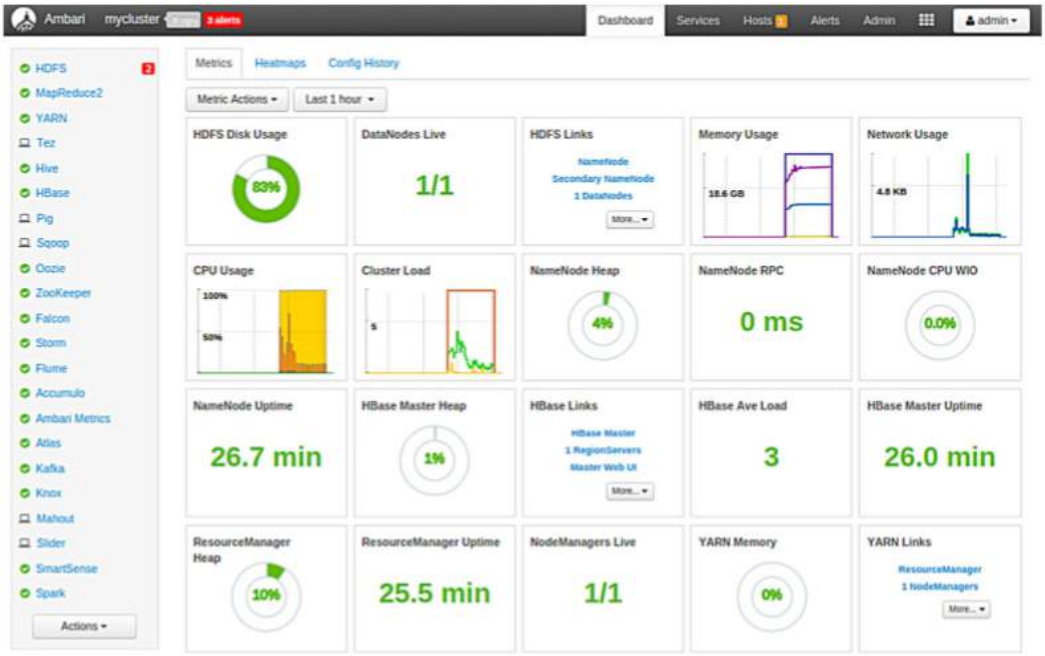



Gambar 2.17: *Wizard penyiapan Apache Ambari - layanan yang disesuaikan*




Gambar 2.18: *Wizard penyiapan Apache Ambari - tinjau dan terapkan*

Setelah wizard selesai, Anda akan memiliki cluster HDP yang berfungsi seperti yang ditunjukkan pada Gambar 2.19. Anda dapat memantau layanan dan kerangka kerja yang diinstal dari dasbor Ambari. Pastikan semua layanan berjalan. Untuk setiap layanan ini, Anda dapat mengedit konfigurasi, memantau, dan memulai ulang layanan dari dasbor Ambari.




Gambar 2.19: *Dasbor Apache Ambari menunjukkan kerangka kerja yang diinstal*

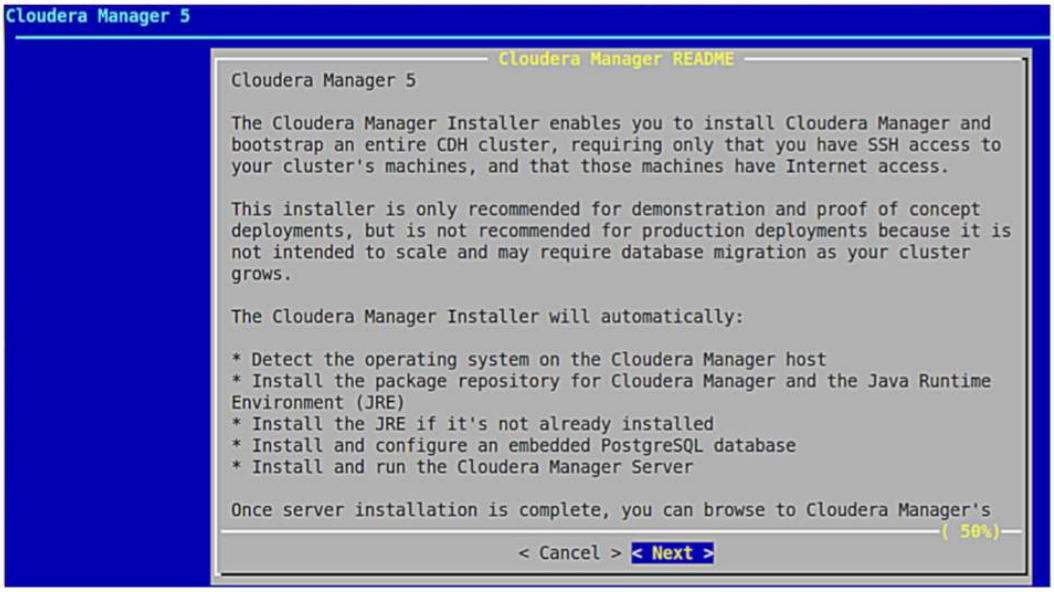
## 2.5 Cloudera CDH Stack

Cloudera CDH [8] adalah distribusi platform sumber terbuka yang mencakup berbagai kerangka kerja dan alat Big data. Kerangka kerja utama dalam tumpukan CDH yang digunakan untuk contoh dalam buku ini meliputi:

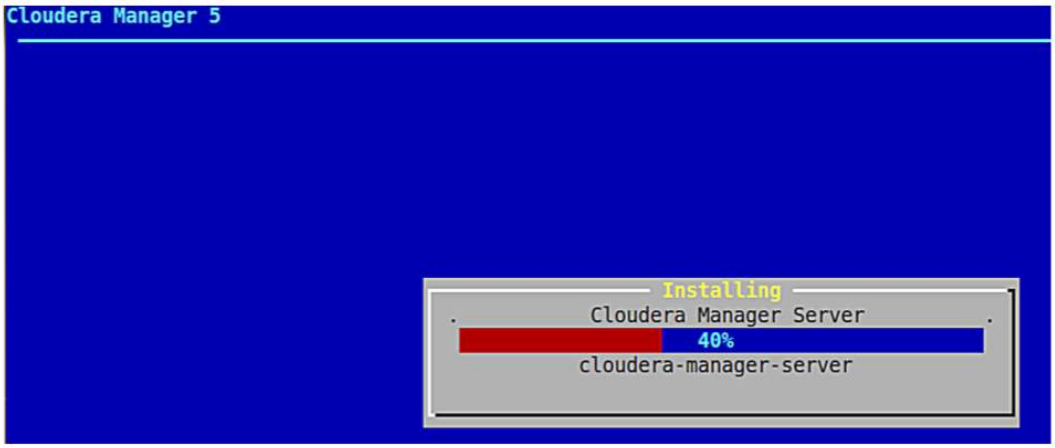
- Hadoop
- YARN
- HDFS
- HBase
- Hive
- Pig
- Sqoop
- Flume
- Zookeeper
- Kafka
- Spark

Ada berbagai metode untuk menyiapkan tumpukan CDH, yang termudah adalah metode otomatis menggunakan Cloudera Manager. Pemasang Cloudera Manager dapat diunduh dan dijalankan dari baris perintah seperti yang ditunjukkan pada Box di bawah ini:

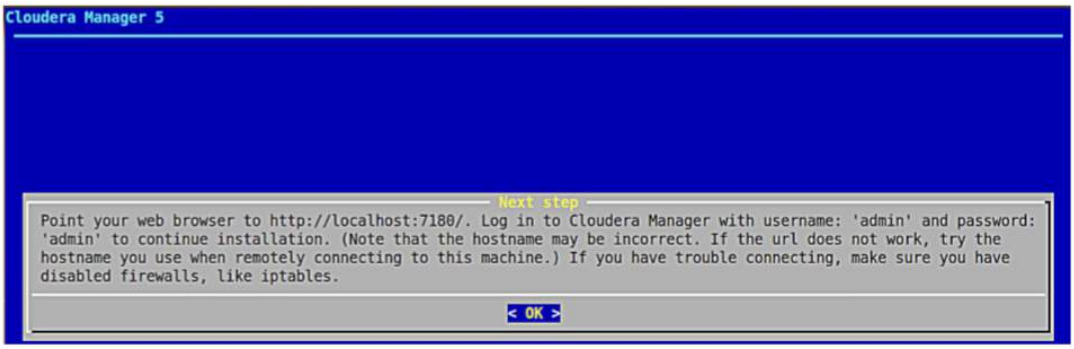
Ikuti langkah-langkah yang ditunjukkan pada Gambar 2.20-2.22 untuk menginstal Cloudera Manager Server.




Gambar 2.20: *Cloudera Manager install wizard*

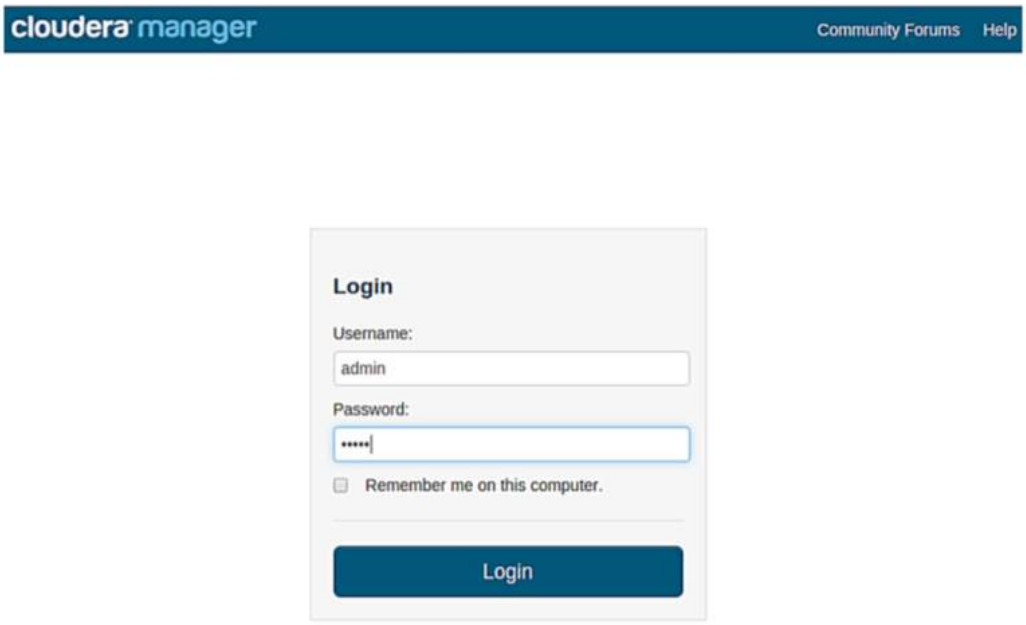



Gambar 2.21: *Instalasi Cloudera Manager dalam proses*

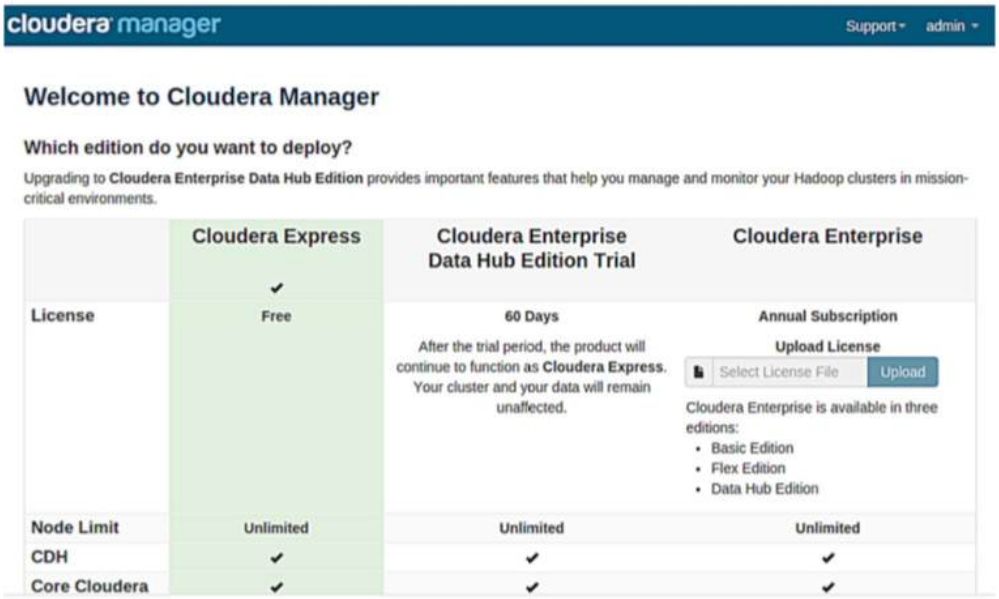



Gambar 2.22: *Konfirmasi Instalasi Cloudera Manager*

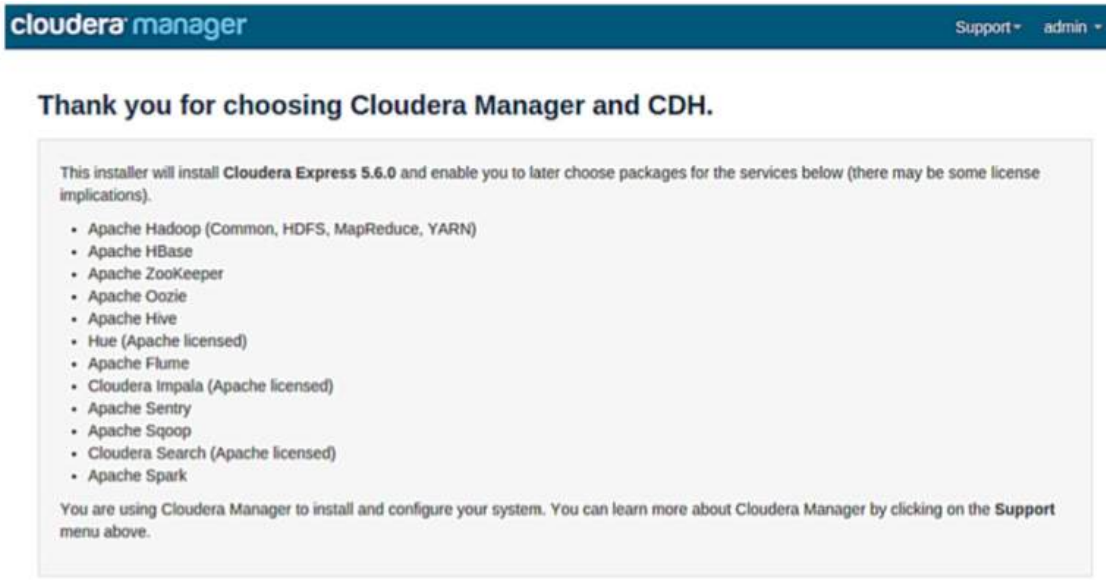
Setelah Server Cloudera Manager diinstal, Anda dapat membuka URL: http: // <hostname>: 7180 di browser untuk mengakses Cloudera Manager. Ikuti langkah-langkah seperti yang ditunjukkan pada Gambar 2.23-2.33 untuk menyelesaikan instalasi tumpukan CDH.



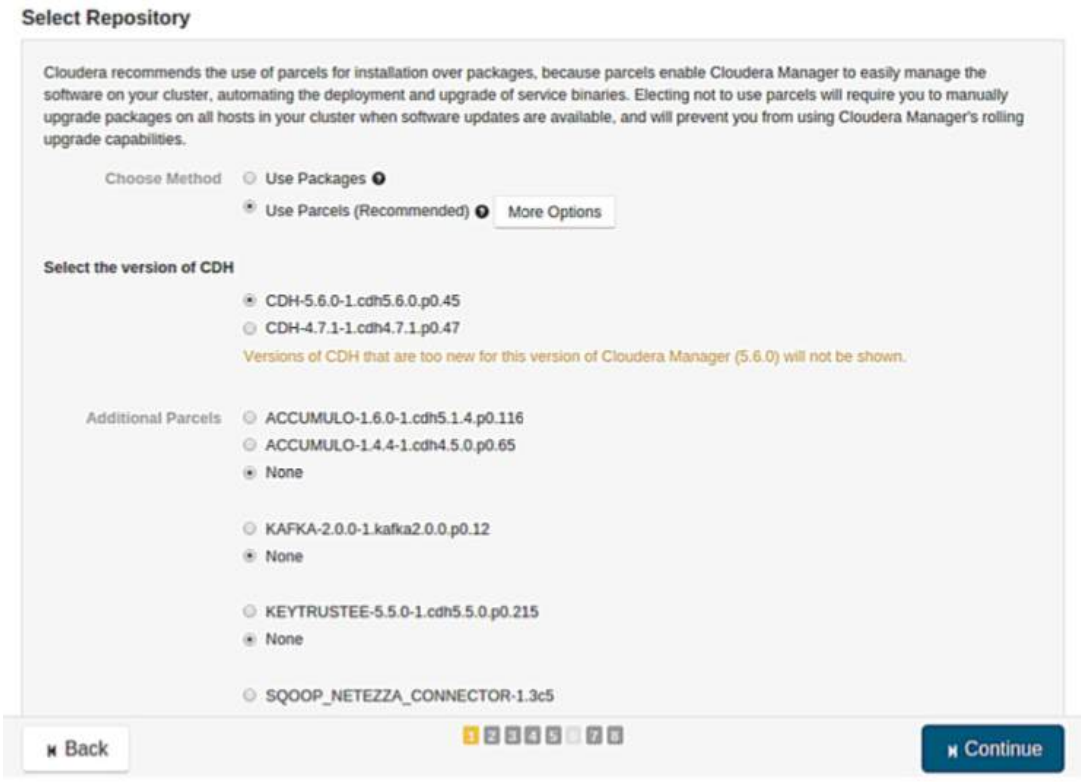

Gambar 2.23: *Login Page Cloudera Manager*



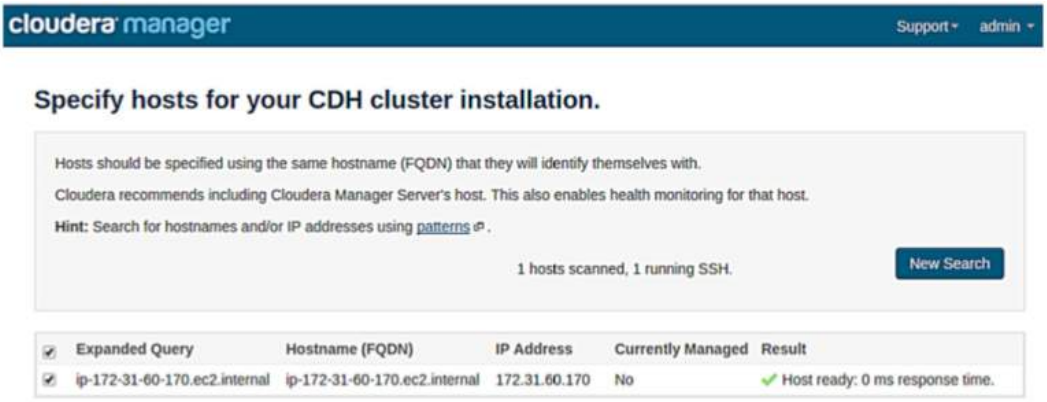

Gambar 2.24: *Select Cloudera edition untuk install*



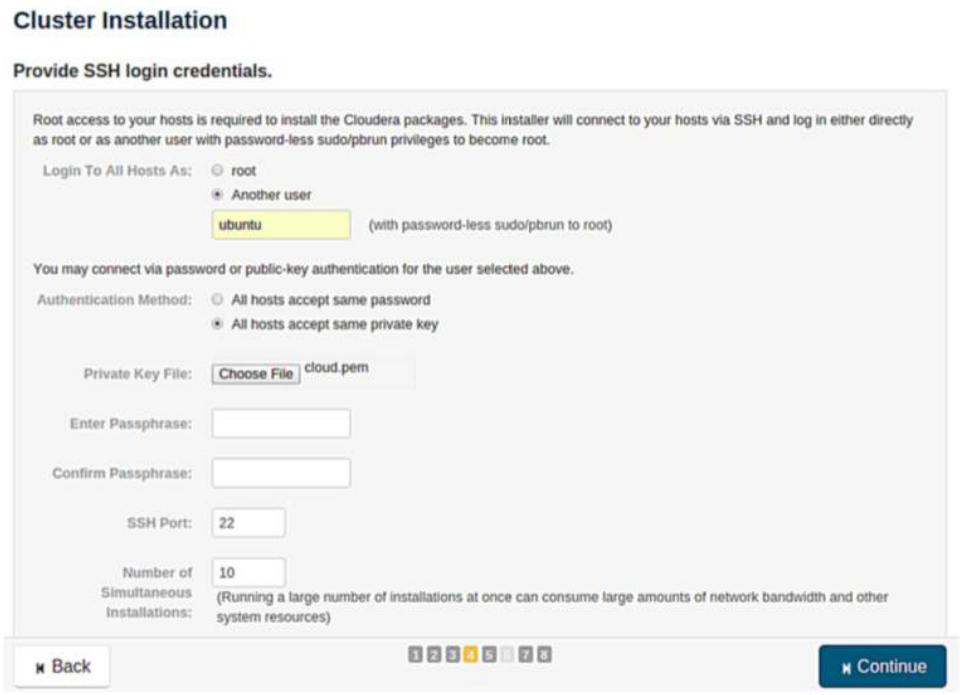

Gambar 2.25: *Layanan edisi Cloudera Express*



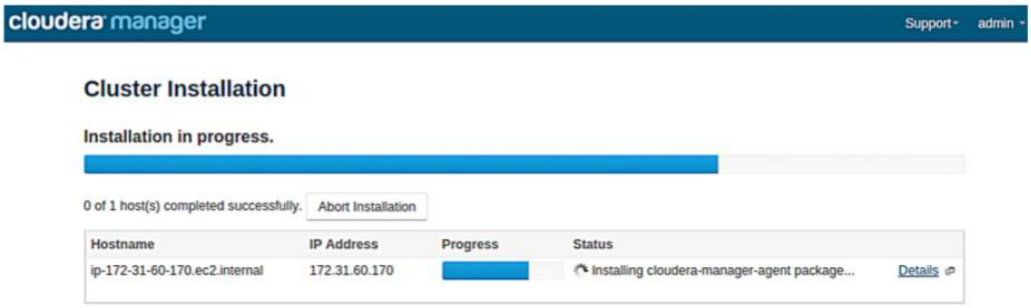

Gambar 2.26: *Pilih repositori untuk instalasi CDH*




Gambar 2.27: *Tentukan host untuk instalasi CDH*

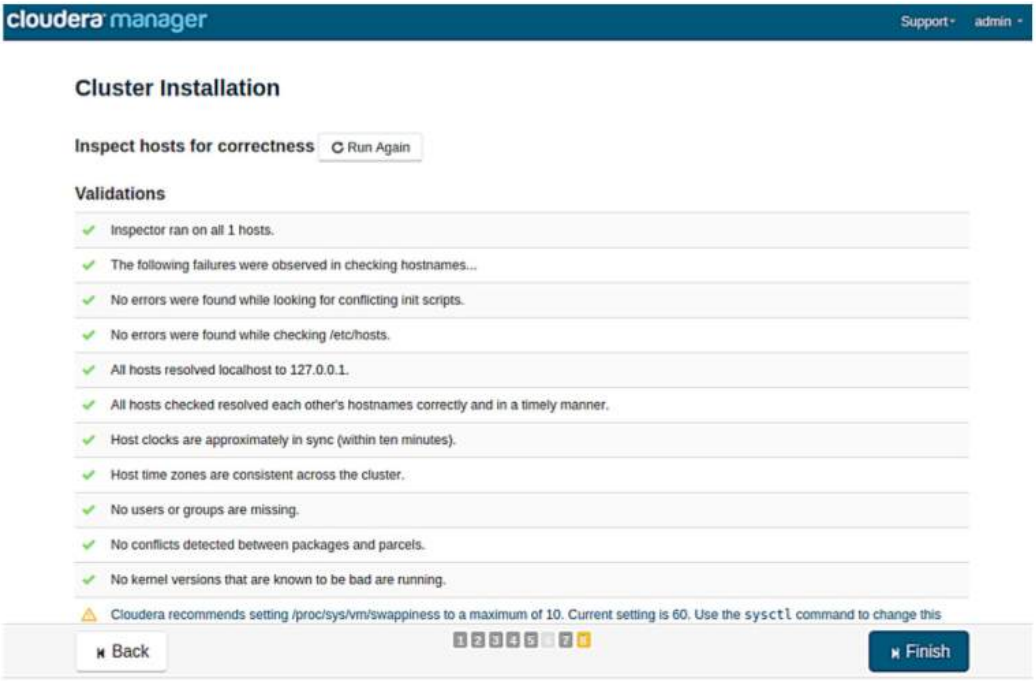



Gambar 2.28: *Memberikan kredensial login SSH untuk host*

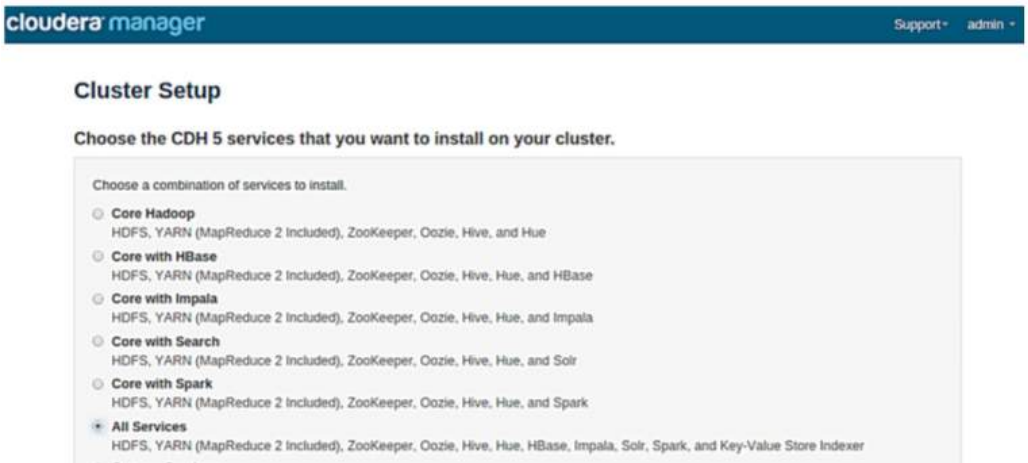


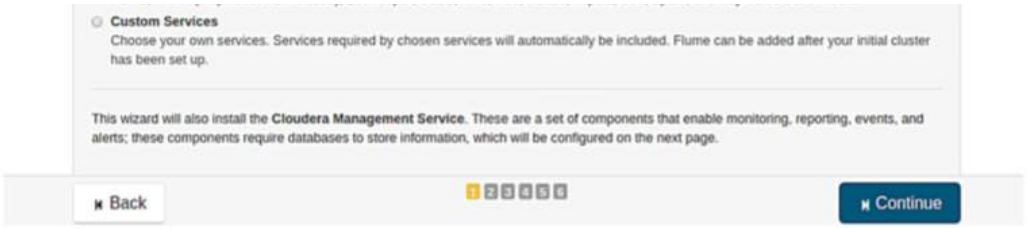

Gambar 2.29: *Cloudera Manager menunjukkan penginstalan cluster sedang berlangsung*


Gambar 2.30: *Cloudera Manager menunjukkan penginstalan paket yang dipilih sedang berlangsung*

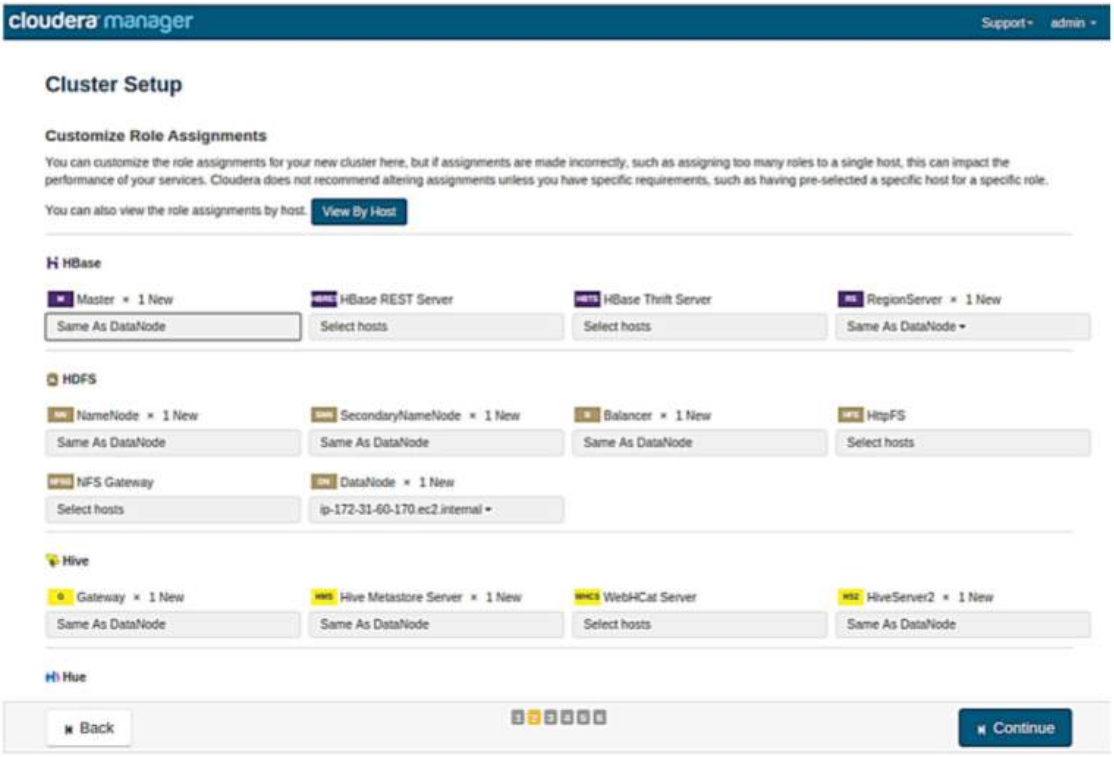



Gambar 2.31: *Cloudera Manager menampilkan ringkasan penginstalan cluster*



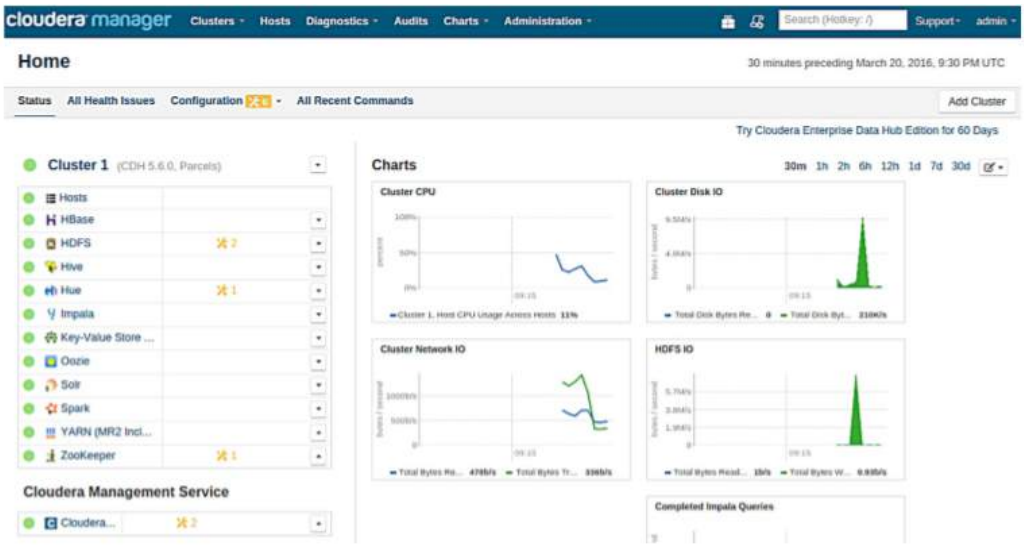



Gambar 2.32: *Select CDH services untuk install*




Gambar 2.33: *Sesuaikan penetapan peran*

Setelah wizard penginstalan selesai, Anda akan memiliki cluster CDH yang berfungsi seperti yang ditunjukkan pada Gambar 2.34. Anda dapat memantau layanan yang diinstal, mengedit konfigurasi layanan, dan memulai ulang layanan dari dasbor.




Gambar 2.34: *Dasbor Cloudera Manager menunjukkan layanan yang diinstal*

## 2.6 Amazon Elastic MapReduce (EMR)

Amazon Elastic MapReduce adalah platform klaster Big data terkelola yang mendukung kerangka kerja berikut:

- Hadoop
- Hive
- Hue
- Mahout
- Pig
- Spark

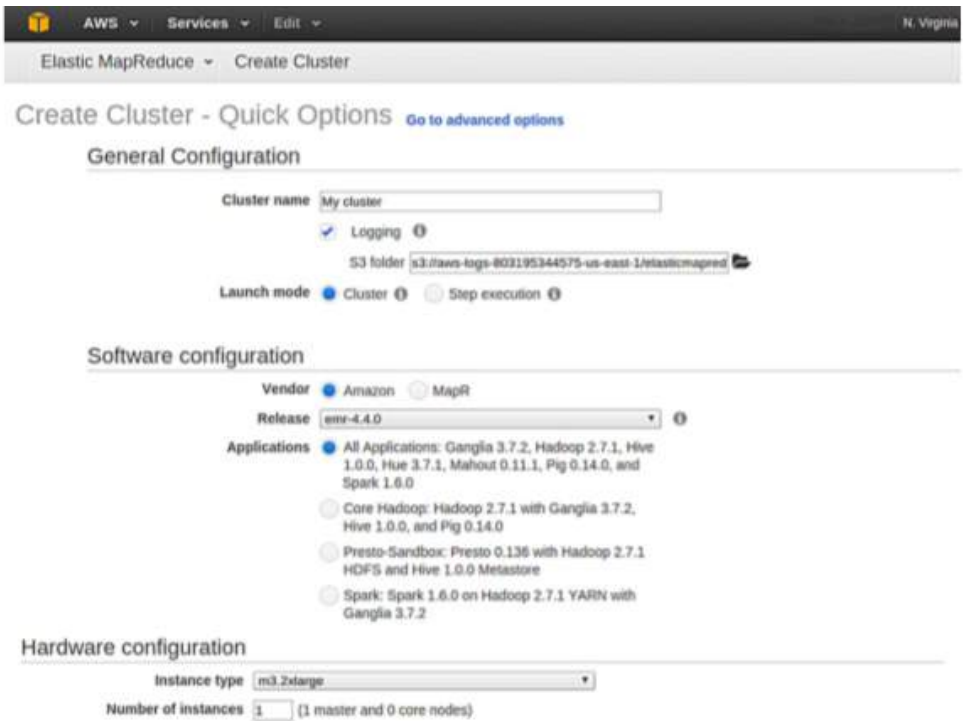
EMR mendukung dua mode peluncuran - Eksekusi Cluster dan Langkah. Dengan mode peluncuran cluster, Anda dapat membuat cluster yang menjalankan kerangka Big data. Anda dapat memilih jenis tumpukan yang akan digunakan untuk cluster (atau jenis vendor - EMR Stack atau Big data stack MapR).

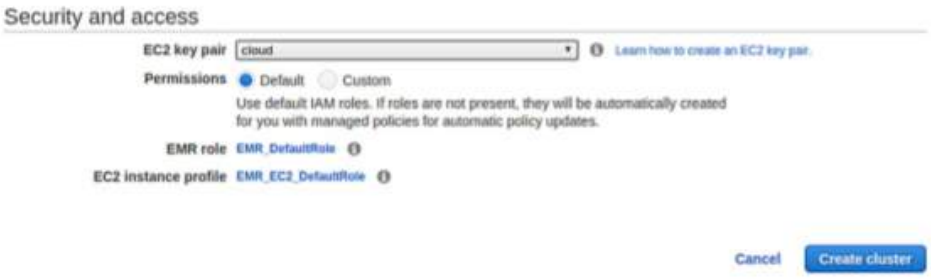
Dengan opsi Eksekusi langkah, Anda menambahkan langkah-langkah (yang merupakan unit atau pekerjaan) untuk dijalankan setelah cluster diluncurkan. Layanan EMR secara otomatis menentukan aplikasi / kerangka kerja yang diperlukan untuk menyelesaikan langkah-langkah tersebut. Ketika Anda membuat cluster, langkah-langkahnya selesai, dan cluster dihentikan secara otomatis.

EMR mendukung jenis langkah berikut:

- JAR Kustom: Alur tugas JAR Kustom menjalankan program Java yang telah Anda unggah ke Amazon S3.
- Program Hive : Anda dapat membuat aliran pekerjaan Hive dengan EMR yang bisa berupa pekerjaan Hive interaktif atau skrip Hive.
- Pekerjaan streaming: Aliran pekerjaan streaming menjalankan satu pekerjaan Hadoop yang terdiri dari map dan mengurangi fungsi yang diimplementasikan dalam skrip atau biner yang telah Anda unggah ke Amazon S3. Anda dapat menulis map dan mengurangi skrip di Ruby, Perl, Python, PHP, R, Bash, atau C ++.
- Program Pig: Anda dapat membuat aliran pekerjaan Pig dengan EMR yang dapat berupa pekerjaan Pig interaktif atau skrip Pig.
- Aplikasi Spark: Anda dapat menjalankan aplikasi Spark dengan menyediakan JAR aplikasi Spark dan opsi pengiriman Spark.

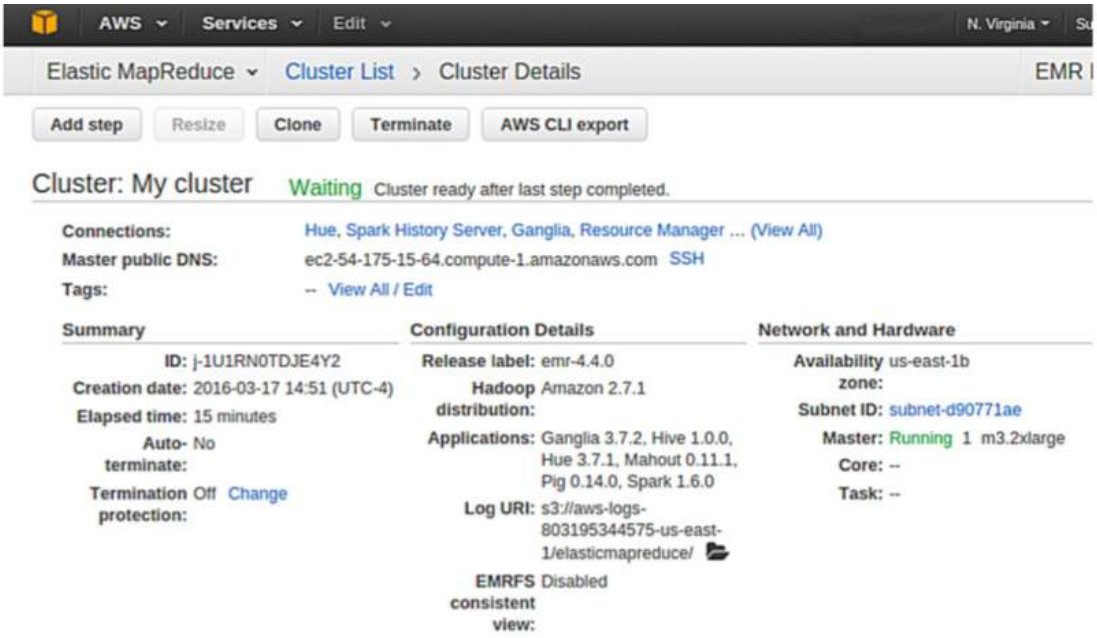
Cluster EMR dapat dibuat dari dashboard EMR seperti yang ditunjukkan pada Gambar 2.35. Berikan nama cluster, pilih rilis EMR dan aplikasi yang akan diinstal. Selanjutnya, pilih jenis instance yang akan digunakan untuk cluster dan jumlah instance di cluster. Selanjutnya, pilih pasangan kunci EC2 yang akan Anda gunakan untuk terhubung dengan aman ke cluster.





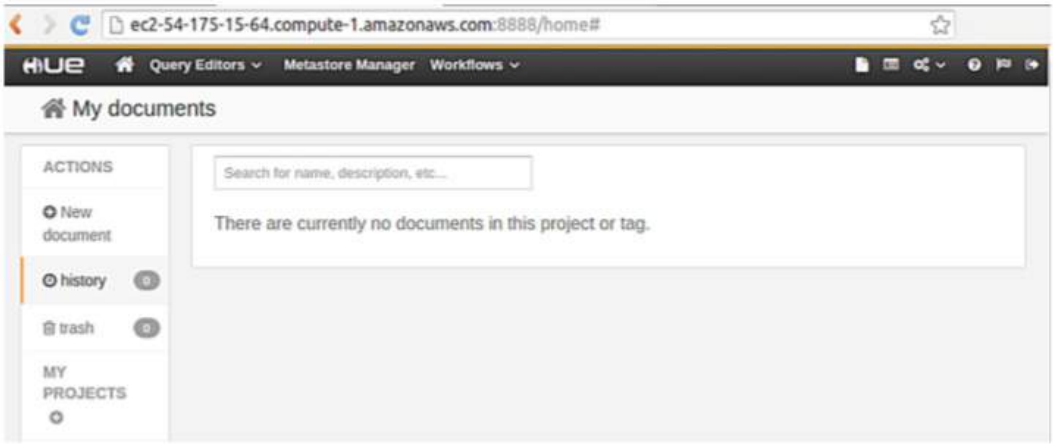

Gambar 2.35: *Membuat klaster Amazon EMR*

Setelah cluster disiapkan, Anda dapat memperoleh detail koneksi cluster dan link ke interface web dari masing-masing aplikasi dari halaman detail clusterseperti yang ditunjukkan pada Gambar 2.36.



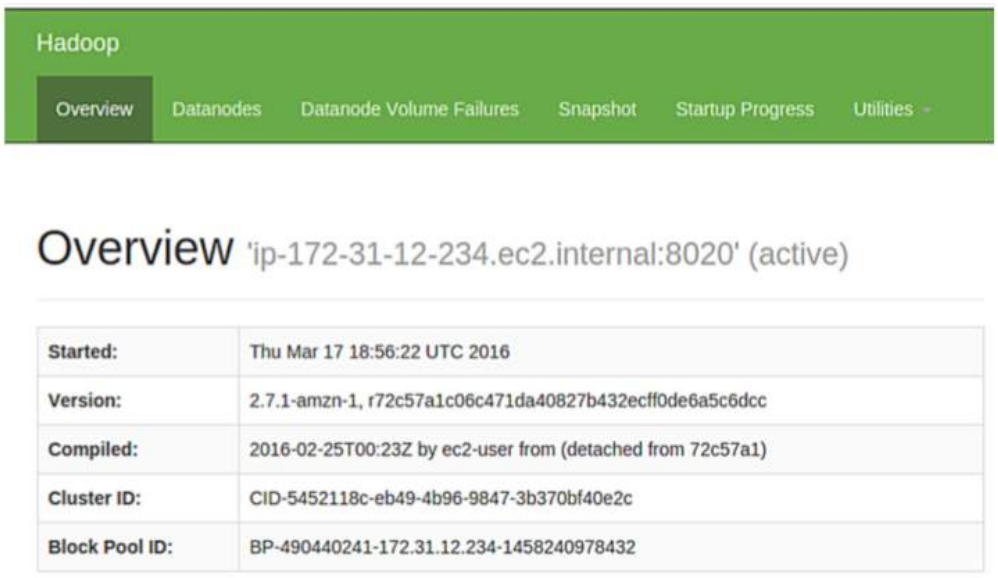

Gambar 2.36: *Detail klaster Amazon EMR*

Gambar 2.37-2.39 menunjukkan interface web dari beberapa aplikasi yang diinstal pada cluster EMR.




Gambar 2.37: *Interface Apache Hue*

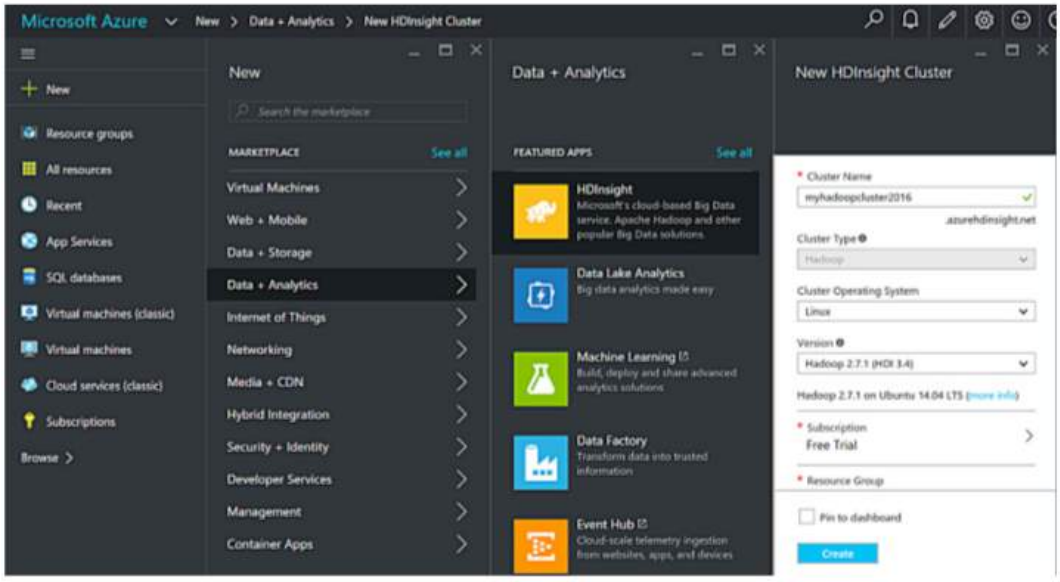

Gambar 2.38: *Interface Manajer Sumber Daya Hadoop*




Gambar 2.39: *Interface Node Nama Hadoop HDFS*

## 2.7 Azure HDInsight

Layanan Azure HDInsight memungkinkan Anda menyiapkan kluster terkelola yang menjalankan ekosistem komponen Hadoop. Kluster HDInsight dapat dibuat dari portal Azure seperti yang ditunjukkan pada Gambar 2.40. Jenis cluster yang didukung termasuk Hadoop, HBase, Spark, dan Storm.




Gambar 2.40: *Membuat kluster Azure HDInsight*

## Ringkasan

Dalam bab ini kami menyediakan empat opsi untuk menyiapkan Big data stack dan kerangka kerja untuk pemrosesan batch, real-time, dan interaktif yang digunakan sebagai contoh dalam buku ini. Mahasiswa dapat memilih stack apa pun yang sesuai dengan kebutuhan mereka. Meskipun layanan Amazon EMR dan Azure HDInsight mengharuskan Anda untuk mengatur akun di masing-masing platform layanan cloud, tumpukan HDP dan CDH juga dapat diatur di mesin lokal.

# Pola Big Data

Bab 3

Bab ini mencakup

- Komponen & Gaya Arsitektur Analytics
  - Load Leveling dengan Antrian
  - Load Balancing dengan Banyak Konsumen
  - Pemilihan Pemimpin
  - Sharding
  - Konsistensi, Ketersediaan & Toleransi Partisi (CAP)
  - Filter Bloom
  - Tampilan Terwujud
  - Arsitektur Lambda
  - Scheduler-Agentt-Supervisor
  - Pipa & Filter
  - Layanan web
  - Konsensus dalam Sistem Terdistribusi
- Pola MapReduce
  - Peringkasan Numerik
  - Top-N
  - Saring
  - Berbeda
  - Pengelompokan
  - Indeks Terbali
  - Penyortiran
  - Bergabung

Dalam bab ini, kami akan menjelaskan berbagai komponen arsitektur dan gaya desain untuk sistem dan aplikasi big data.

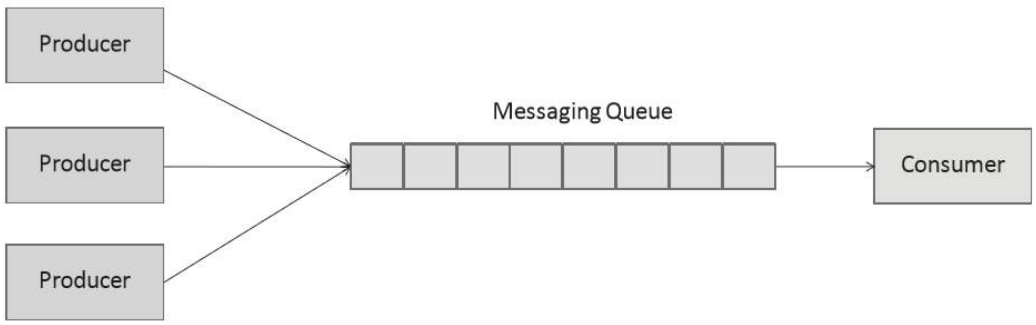
## 3.1. Analisis Komponen Arsitektur & Gaya Desain

### 3.1.1 Load Leveling dengan Queues

Antrian pesan dapat digunakan antara produsen data dan konsumen data dalam sistem Big data, untuk meratakan beban. Queuess/Antrian berguna untuk pesan push-pull di mana produsen mendorong data ke antrian dan konsumen menarik data dari antrian. Karena aplikasi Big data dapat mengalami kecepatan tinggi data yang diserap untuk diproses, antrean dapat digunakan sebagai penyangga antara produsen dan konsumen untuk mencegah konsumen kelebihan beban selama beban puncak. Antrian memungkinkan pemisahan produsen dan konsumen. Produsen dan konsumen tidak perlu saling waspada dan bisa berjalan serempak.

Saat antrian digunakan, konsumen dapat dinaikkan atau diturunkan berdasarkan tingkat muatan. Antrian membantu dalam meningkatkan keandalan dan ketersediaan sistem. Jika konsumen tidak tersedia untuk sementara, produsen masih dapat terus memasukkan pesan ke antrian.

Di Bab-5, kami menjelaskan antrean perpesanan seperti RabbitMQ, ZeroMQ, RESTMQ, dan Amazon SQS.

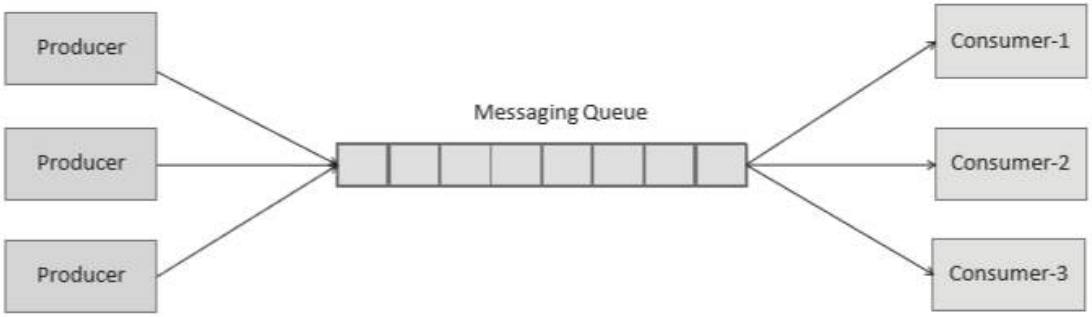



Gambar 3.1: *Load Leveling dengan Queues*

### 3.1.2 Load Balancing dengan Banyak Konsumen

Meskipun antrean perpesanan dapat digunakan antara produsen data dan konsumen untuk meratakan beban, memiliki banyak konsumen dapat membantu dalam penyeimbangan beban dan membuat sistem lebih skalabel, andal, dan tersedia. Produsen data mendorong pesan ke antrian dan konsumen mengambil dan memproses pesan. Banyak konsumen dapat membuat sistem lebih kuat karena tidak ada satupun titik kegagalan. Lebih banyak

konsumen dapat ditambahkan sesuai permintaan jika beban kerja tinggi. Penyeimbangan beban antar konsumen meningkatkan kinerja sistem karena banyak konsumen dapat memproses pesan secara paralel. Pola ini berguna jika pesan dapat diproses secara independen tanpa ketergantungan apa pun. Ketika beberapa konsumen digunakan, antrian pesan dikonfigurasi untuk mengirimkan setiap pesan setidaknya sekali. Ketika konsumen mengambil pesan, itu tidak dihapus dari antrian, tetapi disembunyikan dari konsumen lain (dengan batas waktu visibilitas) untuk mencegah pemrosesan duplikat. Setelah pesan diproses, konsumen menghapusnya dari antrian. Jika konsumen gagal saat memproses pesan, pesan tersebut akan tersedia untuk konsumen lain setelah periode batas waktu visibilitas. Ini memastikan bahwa pesan tidak hilang dan diproses oleh salah satu konsumen sebelum dihapus.




Gambar 3.2: *Load Balancing dengan Banyak Konsumen*

### 3.2.3 Pemilihan Pemimpin

Sistem Big data mencakup banyak node atau instance yang menyimpan, memproses, dan menganalisis data. Dalam sistem seperti itu, ada kebutuhan untuk mengoordinasikan tindakan yang dilakukan oleh instance, karena instance tersebut mungkin mengakses sumber daya bersama. Koordinasi menjadi penting ketika instans berjalan secara paralel dan setiap instans menghitung sebagian kecil dari komputasi yang kompleks. Dalam sistem terdistribusi seperti itu, seorang pemimpin diberi peran untuk mengelola contoh dan mengoordinasikan keadaan dan tindakan mereka.

Pemilihan pemimpin adalah mekanisme di mana instansi dalam sistem terdistribusi dapat memilih salah satu instansi sebagai pemimpin mereka. Dalam mekanisme pemilihan pemimpin yang paling sederhana, instance dengan ID tertinggi dipilih sebagai pemimpin. Mekanisme ini berfungsi ketika instance memiliki ID unik yang ditetapkan padanya. Saat proses pemilihan pemimpin dijalankan, instans berkomunikasi di antara mereka

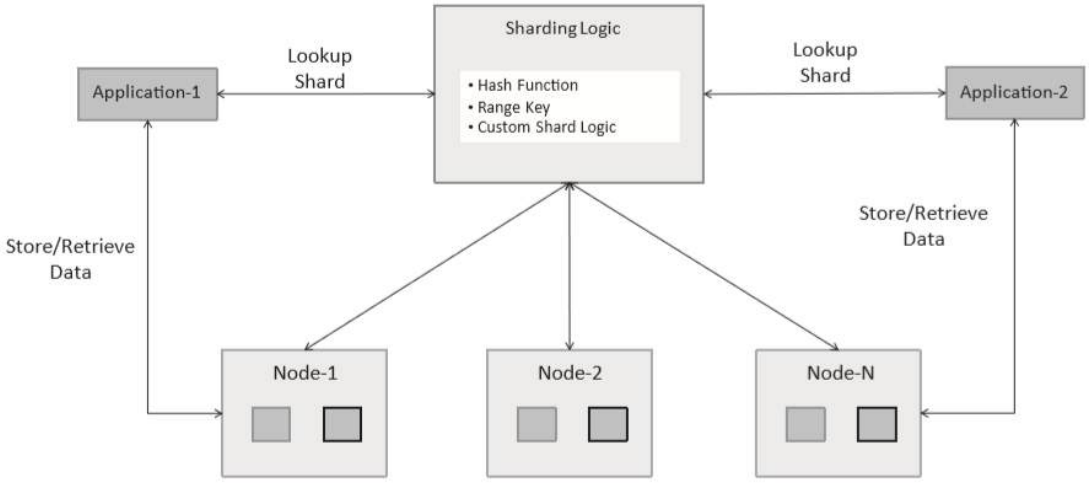
sendiri dan memilih instans dengan ID tertinggi sebagai pemimpinnya. Setiap instansi mengetahui pemimpinnya ketika proses pemilihan selesai. Persyaratan penting untuk setiap mekanisme pemilihan pemimpin adalah bahwa mekanisme tersebut harus dapat memilih tepat satu instansi sebagai pemimpin dan semua instansi harus mengetahui siapa pemimpinnya ketika proses pemilihan selesai.

Algoritma pemilihan pemimpin lainnya adalah algoritma Bully. Algoritme ini mengasumsikan bahwa setiap instance dapat berkomunikasi satu sama lain dan setiap instance mengetahui ID dari setiap instance lainnya. Algoritme Bully bekerja sebagai berikut. Contoh A memulai pemilihan pemimpin jika diketahui bahwa pemimpin telah gagal. Instance A mengirimkan pesan pemilihan ke instance lain dengan ID yang lebih tinggi dan menunggu tanggapan mereka. Jika tidak ada tanggapan yang diterima, A menyatakan dirinya sebagai pemimpin dan memberitahukan semua contoh lainnya. Jika instance A menerima tanggapan dari instance lain, instance tersebut keluar dari pemilihan dan menunggu instance lain menyatakan dirinya sebagai pemimpin. Jika instans tidak menerima respons apa pun dari instans lain yang menyatakan dirinya sebagai pemimpin, ia akan mengirim ulang pesan pemilihan ke instans dengan ID yang lebih tinggi. Jika instance A menerima pesan dari instance dengan ID yang lebih rendah yang menyatakan dirinya sebagai pemimpin, instance A memulai pemilihan baru.

### 3.1.4 Sharding

Sharding melibatkan partisi data secara horizontal di beberapa node penyimpanan dalam sistem penyimpanan data. Dalam aplikasi Big data, volume data yang terlibat sangat besar sehingga tidak memungkinkan untuk menyimpan data pada satu mesin. Meskipun penskalaan vertikal (dengan penambahan lebih banyak komputasi, penyimpanan, dan sumber daya memori) dapat menjadi solusi sementara untuk menangani volume data yang lebih besar, namun, data pada akhirnya akan menjadi terlalu besar untuk disimpan pada satu node penyimpanan. Selain itu, satu mesin hanya dapat melayani kueri pengguna serentak dalam jumlah terbatas. Jika aplikasi memiliki banyak pengguna bersamaan, sharding dapat membantu meningkatkan kinerja sistem dan waktu respons untuk pengguna karena kueri dilayani oleh beberapa node. Banyak sistem penyimpanan data menyediakan sharding otomatis, di mana data secara otomatis dipartisi dan didistribusikan di antara node penyimpanan. Sharding memungkinkan sistem penyimpanan diskalakan dengan menambahkan node baru. Pecahan data (partisi) dapat direplikasi di seluruh node penyimpanan untuk membuat sistem lebih andal. Saat shard direplikasi, load balancing kueri dapat dilakukan untuk meningkatkan waktu respons.

sharding dapat dilakukan dengan berbagai strategi sharding, yang dapat dikelola oleh aplikasi atau sistem penyimpanan data. Strategi sharding menentukan data mana yang harus dikirim ke shard. Kebanyakan strategi sharding menggunakan satu atau beberapa field dalam data sebagai kunci shard (atau kunci partisi). Kunci shard yang digunakan harus unik (dapat berupa kunci utama atau kunci komposit). Pendekatan yang paling umum adalah menggunakan fungsi hash yang membagi data secara merata di seluruh node penyimpanan. Fungsi hash diterapkan ke kunci beling untuk menentukan pecahan tempat data harus disimpan. Dengan menyebarkan data secara merata ke seluruh node penyimpanan, pendekatan hashing mencegah satu node penyimpanan menjadi hotspot. Pendekatan lain adalah menyimpan data dalam berbagai kunci pecahan dalam satu pecahan. Sistem memelihara map kunci jangkauan dan pecahan yang sesuai. Strategi ini berguna saat Anda ingin menyimpan data yang diambil dan diperbarui bersama dalam satu pecahan untuk menghindari pencarian berulang. Misalnya, pesanan terbaru (dengan ID pesanan dalam rentang tertentu) di aplikasi eCommerce dapat disimpan dalam satu pecahan.




Gambar 3.3: *Sharding*

### 3.1.5 Teorama *Consistency, Availability & Partition Tolerance (CAP)*

Teorema CAP adalah salah satu dalil atau pernyataan dalam ilmu komputer yang menjelaskan mengenai database yang terdistribusi. Untuk sistem data terdistribusi, ada trade-off antara konsistensi dan ketersediaan. Pertukaran ini dijelaskan dengan Teorema CAP, yang menyatakan bahwa di bawah partisi, sistem data terdistribusi dapat konsisten atau tersedia tetapi tidak keduanya pada waktu yang sama.

Sistem yang konsisten adalah sistem di mana semua mahasiswaan dijamin untuk menyertakan penulisan sebelumnya. Dalam sistem yang konsisten, setelah operasi pembaruan dilakukan oleh penulis, itu dilihat oleh semua mahasiswa. Ketersediaan mengacu pada kemampuan sistem untuk menanggapi semua pertanyaan tanpa menjadi tidak tersedia. Sistem data terdistribusi disebut tersedia ketika sistem dapat terus menjalankan operasinya bahkan jika terjadi kegagalan pada beberapa node. Toleransi partisi mengacu pada kemampuan sistem untuk terus menjalankan operasinya pada saat partisi jaringan. Parisi jaringan dapat terjadi ketika dua (atau lebih) kumpulan node tidak dapat terhubung satu sama lain.

Teorema CAP menyatakan bahwa sistem dapat mendukung konsistensi dan toleransi partisi daripada ketersediaan, atau mendukung ketersediaan dan toleransi partisi daripada konsistensi. Mari kita ambil contoh database NoSQL seperti Amazon DynamoDB dan Cassandra. Database ini lebih menyukai konsistensi dan toleransi partisi daripada ketersediaan. Sistem seperti itu dikatakan "pada akhirnya konsisten" karena semua penulisan pada akhirnya (tidak segera) dilihat oleh semua node. Dalam sistem yang pada akhirnya konsisten, klien dapat mengalami status sistem yang tidak konsisten saat pembaruan disebarkan ke semua node dan sistem belum mencapai kondisi stabil. Dalam hal partisi jaringan, semua node mungkin tidak memiliki pembaruan terbaru dan dapat mengembalikan informasi yang tidak konsisten atau sudah ketinggalan zaman. Ketika partisi jaringan diselesaikan, semua node akhirnya melihat pembaruan.

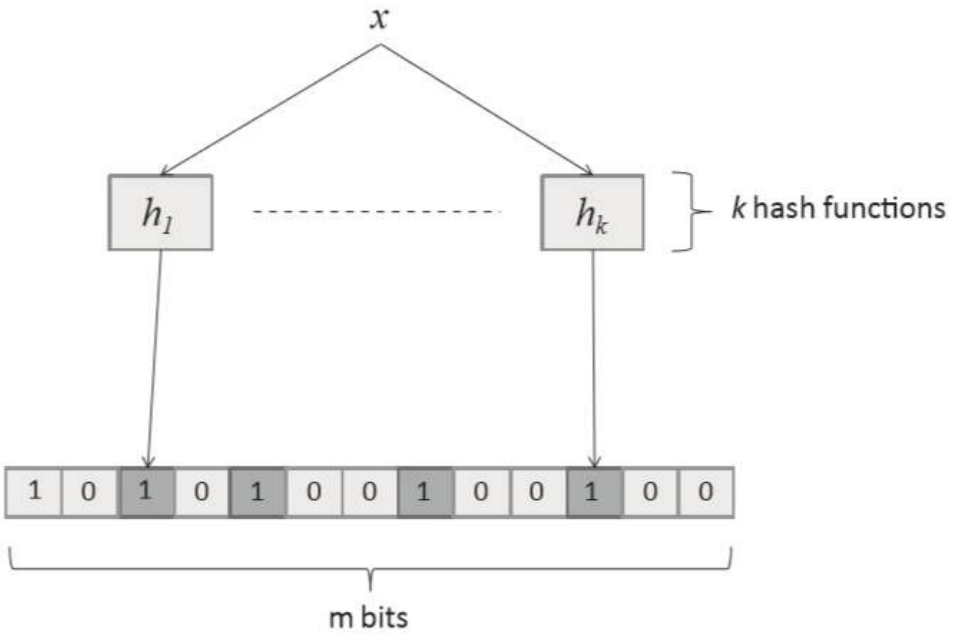
HBase, berbeda dengan DynamoDB dan Cassandra, lebih menyukai konsistensi dan toleransi partisi daripada ketersediaan. Dengan mengadopsi konsistensi yang kuat, HBase memastikan bahwa pembaruan segera tersedia untuk semua klien. Dalam hal partisi jaringan, sistem dapat menjadi tidak tersedia untuk memastikan konsistensi.

### 3.1.6 Bloom Filter

Dalam banyak sistem dan aplikasi big data, mungkin ada persyaratan untuk menguji apakah suatu elemen adalah anggota suatu himpunan. Misalnya, menguji apakah record adalah anggota dari sekumpulan nilai. Dalam sistem data terdistribusi, di mana tabel dipartisi di sekumpulan node, mungkin ada persyaratan untuk memeriksa apakah record (atau kunci baris tertentu) ada di partisi tertentu. Untuk sistem seperti itu, filter Bloom dapat berguna untuk menguji keanggotaan set. Bloomfilter adalah struktur data propiglistik yang direpresentasikan sebagai larik m bit yang awalnya disetel ke 0. Untuk

menambahkan elemen ke himpunan, Bloom filter menggunakan fungsi hash k, yang memetakan elemen ke salah satu posisi dalam larik bit dan bit tersebut disetel ke 1. Untuk menguji keanggotaan set suatu elemen, fungsi hash yang sama digunakan untuk memetakan ke posisi dalam larik bit. Jika bit pada posisi itu adalah 0, elemen tersebut pasti tidak dalam himpunan. Namun, bitnya adalah 1, elemen tersebut mungkin ada atau tidak ada dalam himpunan, karena beberapa elemen lain yang dimapkan ke posisi yang sama dapat menyetel bit ke 1. Sifat Bloom filter adalah sedemikian rupa sehingga dapat memberikan jawaban yang pasti apakah suatu elemen tidak ada dalam kumpulan, tetapi mungkin melaporkan positif palsu, mengklaim bahwa suatu elemen ada dalam kumpulan padahal tidak. Jumlah positif palsu dapat disesuaikan hingga kurang dari 1%. Positif palsu bukan masalah untuk penggunaan sistem.

Bloom filter karena sistemnya dirancang untuk memeriksa lebih lanjut keanggotaan yang ditetapkan setelah memeriksa dengan Bloom filter. Manfaat menggunakan Bloom filter adalah strukturnya yang hemat ruang dan hanya menyimpan sejumlah kecil bit daripada menyimpan elemen itu sendiri. Waktu yang diperlukan untuk menambahkan elemen baru ke himpunan atau pengujian keanggotaan himpunan adalah konstanta tetap (O (k)) dan tidak bergantung pada jumlah elemen yang sudah ada di himpunan.




Gambar 3.4: *Bloom Filter*

### 3.1.7 Materialized Views

Untuk aplikasi Big data yang melibatkan kueri tertentu yang sering dilakukan, sangat bermanfaat untuk menghitung sebelumnya kueri tersebut untuk meningkatkan waktu respons. Tampilan yang telah dihitung sebelumnya tersebut disebut Materialized Views. Materialized Views cocok untuk kueri yang terlalu kompleks untuk dihitung secara real-time (seperti gabungan kompleks) atau melibatkan volume data yang besar untuk digabungkan. Kueri semacam itu dapat dihitung sebelumnya dan disimpan di disk atau cache. Tampilan terwujud juga bermanfaat untuk aplikasi yang menggunakan sistem penyimpanan data terdistribusi seperti database NoSQL terdistribusi. Dalam sistem penyimpanan data seperti itu, data yang disimpan mungkin tidak dalam format yang memungkinkan penghitungan cepat kueri kompleks (karena data mungkin dalam bentuk normalisasi atau tidak terstruktur). Misalnya, mari kita pertimbangkan aplikasi eCommerce yang memungkinkan berbagai penjual untuk membuat daftar dan menjual produk mereka secara online. Dalam aplikasi seperti itu, setiap penjual dapat melihat total pesanan dalam 30 hari terakhir atau total volume penjualan dalam 30 hari terakhir. Untuk kueri semacam itu, tampilan yang terwujud dapat digunakan untuk meningkatkan kinerja aplikasi karena menghitung ini pada waktu proses mungkin memerlukan pengambilan pesanan dalam jumlah besar. Tampilan terwujud dapat diperbarui secara teratur (setiap jam atau setiap hari) atau setiap kali ada pembaruan pada data yang terlibat (misalnya, setiap kali ada pesanan baru). Tampilan terwujud cocok untuk kueri di mana hasil yang mendekati atau sedikit tidak konsisten akan mencukupi.

### 3.1.8 Arsitektur Lambda

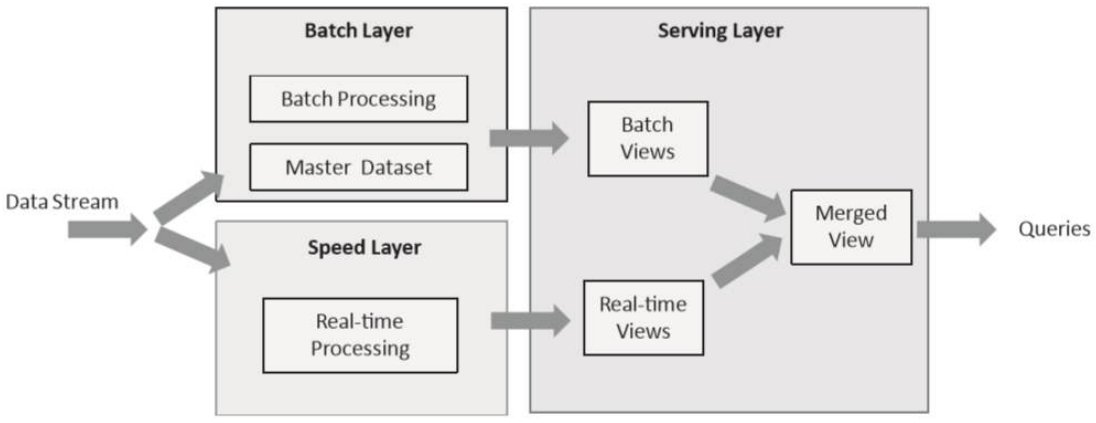
Arsitektur Lambda adalah arsitektur pemrosesan data yang dapat diskalakan dan toleran terhadap kesalahan yang cocok untuk aplikasi yang melibatkan aliran cepat data masuk (yang tidak dapat diubah dan diberi stempel waktu). Arsitektur Lambda memungkinkan kueri data real-time dan historis, dengan pra-komputasi kueri. Arsitektur Lambda terdiri dari layer berikut:

- **Batch Layer**
  Batch Layer bertanggung jawab untuk menyimpan dataset master (yang merupakan kumpulan data mentah yang tidak dapat diubah dan hanya menambahkan), dan pra-komputasi tampilan batch.
- **Serving Layer / Data Penyajian**
  Layer Penyajian mengindeks tampilan kumpulan untuk mengaktifkan kueri data dengan cepat.

- **Speed Layer**
  Layer Speed mengisi celah yang dibuat karena latensi tinggi pembaruan layer batch, dengan memproses data terbaru secara real time. Layer Speed menggunakan algoritme tambahan untuk memproses data terbaru dan menghasilkan tampilan real-time.

Untuk tampilan batch dan tampilan real-time digabungkan untuk merespons kueri yang masuk. Di Bab-7 kami menjelaskan kerangka kerja Hadoop yang dapat digunakan untuk layer batch. Di Bab-8, kami menjelaskan kerangka kerja Storm dan Spark yang dapat digunakan untuk layer kecepatan. Di Bab-10, kami menjelaskan berbagai database NoSQL yang dapat digunakan untuk melayani layer.

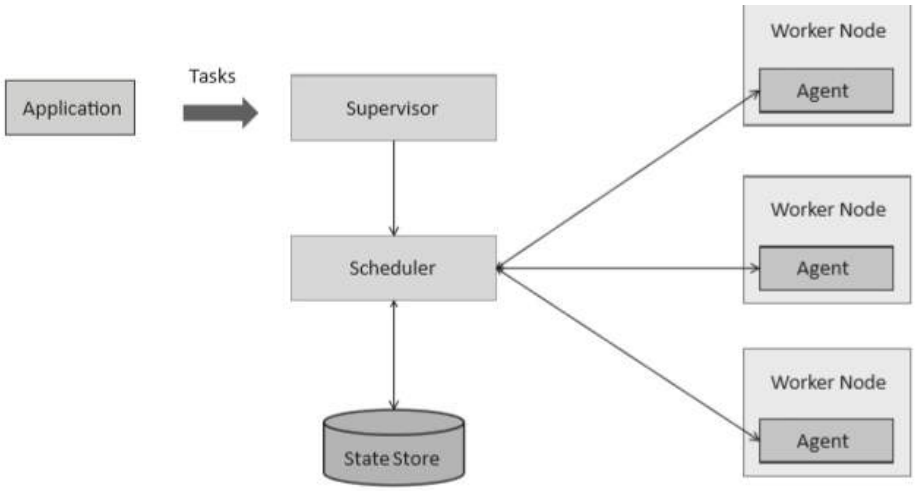
Arsitektur Lambda dapat digunakan untuk menanggapi kueri secara ad-hoc dengan melakukan pra-komputasi tampilan. Sementara layer batch memproses semua data, layer speed hanya memproses data terbaru. Speed Layer memproses data terbaru yang tidak direpresentasikan dalam tampilan batch, yaitu data yang masuk setelah tampilan batch diperbarui (dan saat layer batch memproses batch berikutnya). Layer speed tidak memproses semua data terbaru pada satu waktu. Sebaliknya, layer speed menggunakan algoritme tambahan yang memungkinkan layer speed memperbarui tampilan real-time saat data baru masuk. Jika kueri baru akan disajikan yang tidak dihitung sebelumnya, layer batch dan speed dapat diperbarui dengan logika untuk pra-komputasi kueri baru dan tugas analisis batch dapat dijalankan di seluruh kumpulan data ke tampilan pra-komputasi. Layer serving mendukung pembaruan batch, sehingga setiap kali tampilan baru dibuat, tampilan tersebut dapat diperbarui dan diindeks. Dengan mengindeks tampilan, layer serving memungkinkan mahasiswaan acak cepat untuk menyajikan kueri.




Gambar 3.5: *Lambda Architecture*

### 3.1.9 Scheduler-Agentt-Supervisor

Sistem Big data yang terdiri dari kumpulan node pekerja terdistribusi yang menjalankan tugas yang dikirimkan oleh aplikasi, memerlukan beberapa mekanisme untuk memastikan bahwa eksekusi tugas dapat dikoordinasikan dan tugas yang gagal dapat dicoba lagi. Pola Scheduler-Agentt-Supervisor dapat digunakan untuk membuat sistem lebih tangguh dan toleran terhadap kesalahan.




Gambar 3.6: *Scheduler-Agentt-Supervisor*

**Pola ini mendefinisikan tiga jenis komponen:**

- Scheduller

  Komponen Scheduller bertanggung jawab untuk menetapkan tugas ke node pekerja (*worker nodes*) dan melacak kemajuan setiap tugas. Aplikasi mengirimkan tugas baru ke Scheduller , yang menetapkannya ke node pekerja yang tersedia. Scheduller memelihara penyimpanan status (yang dapat dipertahankan dalam database NoSQL atau penyimpanan key-value dalam memori), di mana status setiap tugas disimpan dan diperbarui saat tugas berlangsung. Informasi status untuk setiap tugas dapat mencakup status tugas (seperti dijadwalkan, berjalan, selesai atau gagal) dan waktu yang diinginkan untuk menyelesaikan tugas (selesai pada waktu).

- Agent

  Scheduller menetapkan tugas ke node pekerja dengan memberi tahu Agent. Setiap node pekerja memiliki Agent (biasanya berjalan di node itu sendiri). Agent bertanggung jawab untuk berkomunikasi dengan Scheduller untuk tugas baru dan memberi tahu node pekerja untuk menyediakan sumber daya yang diperlukan untuk pelaksanaan tugas. Agent mengirimkan informasi tentang kemajuan tugas ke Scheduller. Dalam beberapa implementasi, Agent mungkin mengirim pesan heart-beast ke Scheduller

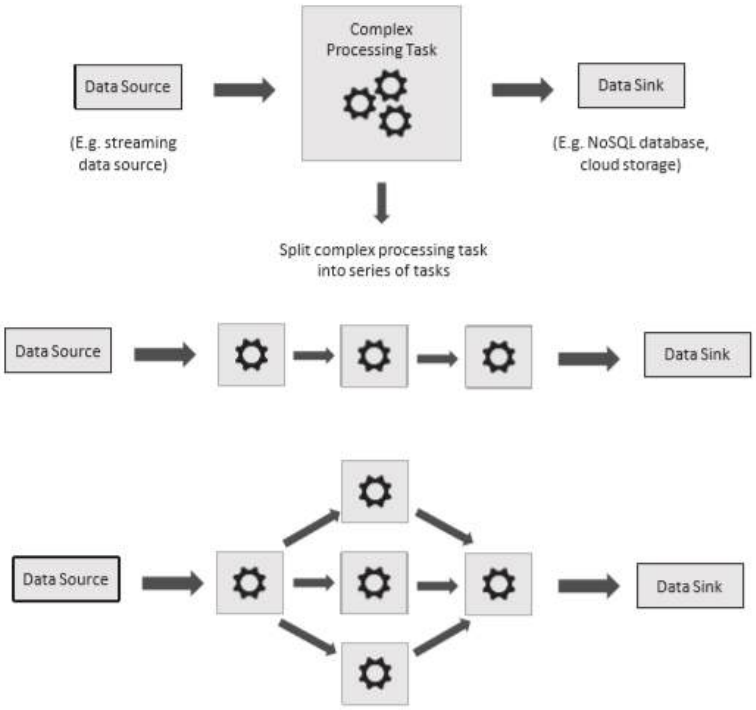
untuk memberi tahu tentang kemajuan tugas. Alternatifnya, Scheduller saat menetapkan tugas dapat memberi tahu Agent waktu selesai. Saat tugas selesai, Agent memberi tahu Scheduller . Jika Scheduller tidak menerima pemberitahuan penyelesaian tugas, Scheduller dapat menganggap bahwa tugas gagal.

- Supervisor

  Supervisor bertanggung jawab untuk memantau status setiap tugas (yang status tugasnya dilacak oleh Scheduller). Supervisor, memeriksa tugas yang gagal atau tugas yang waktunya habis, dan memberi tahu Scheduller untuk mencoba lagi tugas tersebut. Dalam beberapa implementasi, Supervisor dapat melacak jumlah percobaan ulang untuk setiap tugas untuk mencegah tugas yang terus-menerus gagal dicoba kembali. Supervisor juga dapat meningkatkan interval coba lagi setelah setiap upaya yang gagal untuk menyelesaikan tugas.

### 3.1.10 Pipes & Filter

Dalam banyak aplikasi Big data, tugas pemrosesan data yang kompleks dapat dibagi menjadi serangkaian tugas berbeda untuk meningkatkan kinerja, skalabilitas, dan keandalan sistem. Pola pembagian tugas yang kompleks menjadi serangkaian tugas berbeda disebut pola Pipes and Filters, di mana pipa adalah sambungan antara filter (komponen pemrosesan).



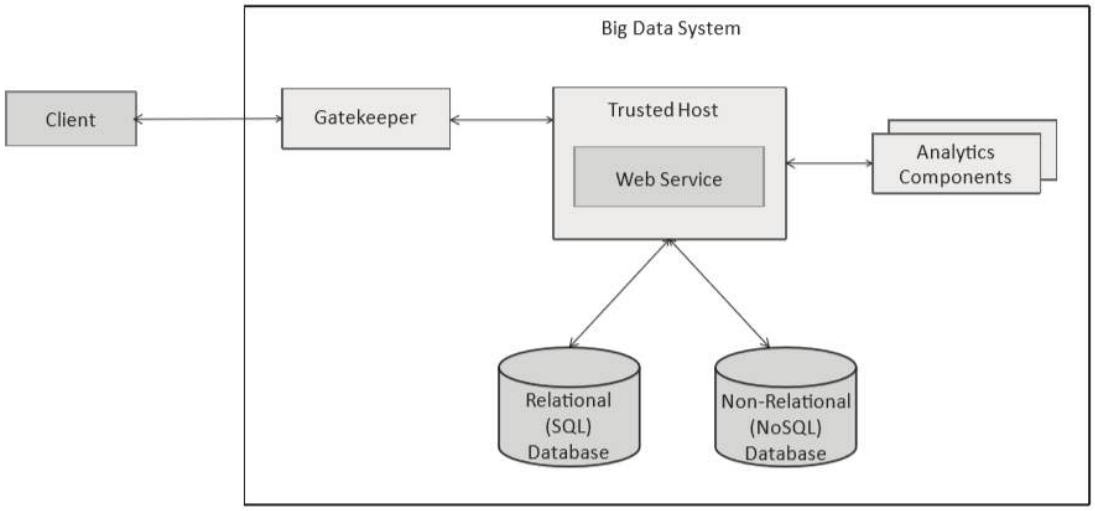

Gambar 3.7: *Pipes and Filters*

Manfaat dari pola ini adalah bahwa setiap tugas menjalankan sebagian pekerjaan dan dapat diskalakan secara independen. Dengan menjalankan banyak pekerja untuk setiap tugas, pemrosesan dapat dilakukan secara paralel. Menjalankan banyak pekerja juga membuat sistem lebih dapat diandalkan, karena meskipun beberapa pekerja gagal, seluruh pipa tidak akan gagal karena pekerja lain dapat melanjutkan pemrosesan. Pola ini sangat berguna untuk aplikasi analisis real-time yang melibatkan pemrosesan aliran data. Dalam Bab-8, kami menjelaskan bagaimana kerangka kerja Storm menggunakan pola serupa untuk pemrosesan aliran data yang andal.

### 3.1.11 Web Service

Sistem Big data dapat menggunakan sistem penyimpanan data dan database yang berbeda. Aplikasi klien yang mengakses sistem Big data untuk melakukan kueri dan mengambil data dapat dipisahkan dari sistem Big data dengan menggunakan layanan web. Layanan web dapat menyediakan interface yang memperlihatkan fungsionalitas sistem kepada klien dalam bentuk berbagai titik akhir. Ketika layanan web digunakan, klien tidak perlu mengetahui bagaimana data disimpan oleh sistem. Klien dapat mengirim permintaan ke layanan web dalam format standar (seperti REST) dan menerima respons dalam format standar (seperti JSON atau XML). Layanan web memungkinkan klien dan sistem untuk dikembangkan secara mandiri. Sistem dapat beralih ke database yang berbeda (dengan tetap menjaga interface layanan web tetap sama) dan klien tidak terpengaruh oleh perubahan tersebut.

Untuk membuat sistem lebih aman, komponen Gatekeepe dapat digunakan, yang biasanya digunakan pada node terpisah. Gatekeepe melakukan otentikasi, otorisasi, dan validasi permintaan. Otentikasi melibatkan verifikasi identitas klien sedangkan otentikasi melibatkan verifikasi jika klien memiliki izin yang memadai untuk operasi yang diminta. Penggunaan Gatekeeper membuat sistem lebih aman dan meminimalkan risiko klien mendapatkan akses ke sistem (atau server) yang memproses permintaan dan memiliki akses ke data.




Gambar 3.8: *Layanan web untuk memisahkan klien dari sistem Big data*

### 3.1.12 Konsensus dalam Sistem Terdistribusi

Dalam sistem terdistribusi, konsensus adalah masalah mendapatkan semua node untuk menyetujui sesuatu. Konsensus mungkin diperlukan untuk berbagai tujuan seperti menyetujui nilai data yang akan dilakukan, memutuskan apakah transaksi harus dilakukan, menyetujui node untuk bertindak sebagai pemimpin atau sinkronisasi jam. Masalah konsensus menjadi lebih rumit untuk sistem terdistribusi di mana node bisa tidak dapat diandalkan dan bisa gagal. Protokol konsensus untuk sistem seperti itu harus toleran terhadap kesalahan. Protokol konsensus untuk sistem terdistribusi benar jika memenuhi syarat-syarat sebagai berikut:

- Persetujuan: Node dalam sistem terdistribusi harus menyetujui beberapa nilai.
- Validitas: Hanya nilai yang telah diusulkan oleh beberapa node yang harus dipilih.
- Penghentian: Semua node pada akhirnya harus menyetujui beberapa nilai.

Protokol paling terkenal untuk konsensus dalam sistem terdistribusi adalah protokol Paxos yang diusulkan oleh Leslie Lamport [19]. Di bagian ini, kami akan menjelaskan secara singkat protokol Paxos. (Untuk penjelasan lebih rinci tentang Paxos, lihat makalah 'Paxos Made Simple' dari Leslie Lamport [20]). Paxos mendefinisikan aktor-aktor berikut:

- Proposer

  Proposer adalah node yang memulai protokol dan bertindak sebagai koordinator.
- Akseptor

  Akseptor adalah titik-titik yang mencoba menyepakati beberapa nilai yang diajukan oleh pengusul.

- Peserta didik

  Peserta didik adalah simpul yang mempelajari nilai yang disepakati oleh akseptor. Protokol melanjutkan dengan langkah-langkah berikut:
- Persiapan

  Pengusul mengirimkan proposal yang diidentifikasi dengan nomor urut N ke mayoritas Penerima (disebut Kuorum) dan menunggu tanggapan mereka.
- Janji

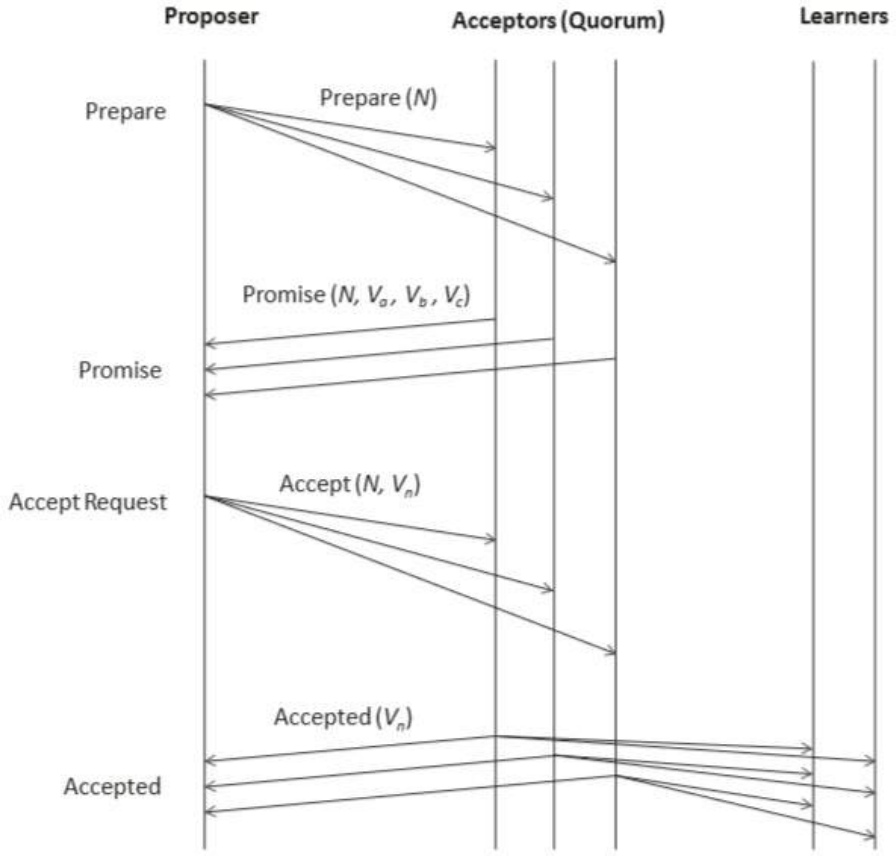
  Setiap Menerima atau menerima proposal membandingkannya dengan proposal bernomor tertinggi yang telah diterima. Jika proposal baru memiliki nomor urut lebih tinggi, akseptor setuju untuk menerima proposal dengan Janji bahwa semua proposal mendatang dengan nomor urut lebih rendah dari N akan ditolak. Ketika Acceptors menyetujui suatu proposal, mereka juga mengirimkan kembali kepada pengusul, nilai proposal pun sudah diterima. Jika proposal baru memiliki nomor urut lebih rendah dari beberapa proposal sebelumnya, Penerima menolak proposal baru.
- Terima Permintaan

  Ketika Pengusul menerima tanggapan dari mayoritas Penerima (yaitu Kuorum), yang menerima proposal, ia mengetahui nilai yang diterima oleh Akseptor dan nilai apa pun yang diterima sebelumnya. Pengusul kemudian mengambil nilai dengan nomor urut tertinggi dan mengirimkan Permintaan Terima ke Penerima. Jika Penerima tidak menyetujui nilai sebelumnya, Pengusul memilih nilainya dan mengirim Permintaan Terima. Jika mayoritas Akseptor menolak proposal, proposal ditinggalkan dan proses dimulai lagi.
- Diterima

  Saat Penerima menerima Permintaan Terima, mereka menerima nilainya, jika nilainya sama dengan nilai yang diterima sebelumnya dan nomor urutnya adalah nomor tertinggi yang telah mereka setujui. Jika tidak, permintaan ditolak. Ketika Permintaan Terima diterima, nilai yang diterima dikomunikasikan oleh penerima dan juga dikomunikasikan kepada semua Peserta.

Karena proses menyepakati nilai yang diajukan oleh pengusul juga mengidentifikasi pengusul yang nilainya diterima, proses yang sama juga dapat digunakan untuk pemilihan pemimpin. Protokol yang dibahas di atas merupakan bentuk dasar dari protokol Paxos. Namun, dalam praktiknya, versi yang lebih dioptimalkan dari protokol yang disebut Multi-Paxos digunakan yang menghilangkan langkah-langkah Mempersiapkan dan Menjanjikan dengan asumsi bahwa dalam kondisi mapan, salah satu pengusul telah dipilih sebagai pengusul terkemuka (atau pemimpin ) dan perubahan kepemimpinan akan jarang terjadi.

Penggunaan nomor urut proposal memungkinkan akseptor untuk memesan proposal dan menerima proposal dengan nomor urut tertinggi. Karena dalam sistem terdistribusi di mana pesan dikirim secara asinkron, pesan dapat memerlukan waktu lama untuk dikirim, penerima dapat memutuskan nilai mana yang akan disetujui dengan memesan proposal dengan nomor urut.



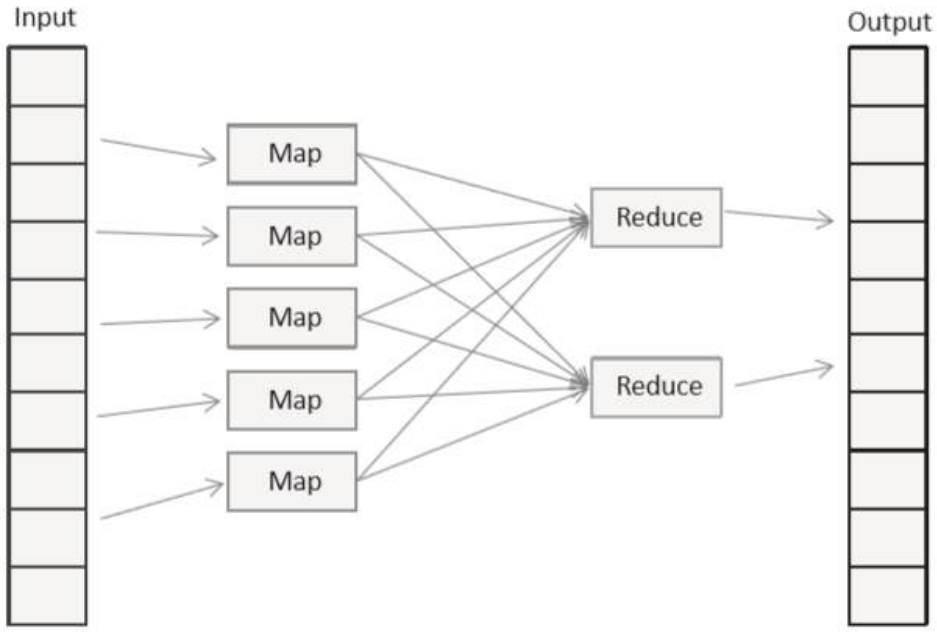

Gambar 3.9: *Protokol konsensus Paxos*

## 3.2 Pola MapReduce

MapReduce adalah model pemrograman populer untuk aplikasi intensif data. MapReduce telah digunakan untuk analisis batch data dalam berbagai aplikasi (seperti jejaring sosial, e-Commerce, keuangan, hiburan, pemerintahan, perawatan kesehatan, telekomunikasi, dll.). MapReduce memungkinkan pengembang untuk fokus pada pengembangan aplikasi intensif data tanpa harus khawatir tentang masalah seperti pemisahan input, Schedulleran, penyeimbangan beban, dan kegagalan.

Model pemrograman MapReduce untuk mengolah data pada cluster besar pada awalnya diusulkan oleh Dean dan Ghemawat [22]. Hadoop, yang merupakan kerangka kerja pemrosesan batch terdistribusi berskala besar sumber terbuka, menyediakan implementasi model MapReduce. Hadoop dan MapReduce telah mempermudah pengembangan aplikasi yang dapat diskalakan dan intensif data untuk lingkungan komputasi awan.

Model MapReduce memiliki dua fase: Map dan Reduce. Program MapReduce ditulis dalam gaya pemrograman fungsional untuk membuat fungsi Map dan Reduce. Data yang dimasukkan ke map dan fase reduksi berupa key-value pair. Sistem run-time untuk MapReduce biasanya merupakan kelompok besar yang dibangun dari perangkat keras komoditas. Sistem run-time MapReduce menangani tugas-tugas seperti mempartisi data, Schedulleran pekerjaan dan komunikasi antara node dalam cluster. Ini mempermudah pemrogram untuk menganalisis data berskala besar tanpa mengkhawatirkan tugas-tugas seperti partisi dan Schedulleran data.




Gambar 3.10: *Data flow pada MapReduce*

Gambar 3.10 menunjukkan aliran data untuk pekerjaan MapReduce. Program MapReduce mengambil satu set key-value pair masukan dan menghasilkan satu set key-value pair output. Dalam fase Map, data dibaca dari sistem file terdistribusi, dipartisi di antara sekumpulan node komputasi dalam kluster, dan dikirim ke node sebagai sekumpulan key-value pair . Tugas map memproses catatan input secara independen satu sama lain

dan menghasilkan hasil antara sebagai key-value pair . Hasil antara disimpan di disk lokal dari node yang menjalankan tugas Map. Pilihan pasangan kunci dan nilai pada langkah ini bergantung pada tugas analisis data yang akan diselesaikan.
Ketika semua tugas Map selesai, fase Reduce dimulai dengan langkah shuffle dan sort, di mana data antara diurutkan berdasarkan key dan key-value pair dikelompokkan dan diubah menjadi tugas pengurangan. Tugas pengurangan kemudian mengambil key-value pair yang dikelompokkan berdasarkan kunci dan menjalankan fungsi pengurangan untuk setiap grup key-value pair . Logika pemrosesan data dalam fungsi pengurangan tergantung pada tugas analisis yang akan diselesaikan.

Tugas Gabungkan opsional dapat digunakan untuk melakukan agregasi data pada data perantara dari kunci yang sama untuk output mapper sebelum mentransfer output ke tugas Kurangi.

Di bagian ini, kami akan menjelaskan berbagai pola MapReduce untuk analisis data, bersama dengan implementasinya dengan Python. Implementasi Python menggunakan pustaka MRJob Python yang memungkinkan Anda menulis pekerjaan MapReduce dengan Python dan menjalankannya di beberapa platform termasuk mesin lokal, klaster Hadoop dan Amazon Elastic MapReduce (EMR). Kerangka kerja Hadoop dibahas secara rinci di Bab-7, di mana studi kasus yang lebih rinci untuk analisis batch data menggunakan MapReduce dijelaskan. Scheduller MapReduce juga dijelaskan di Bab-7.

### 3.2.1 Peringkasan Numerik

Pola peringkasan numerik digunakan untuk menghitung berbagai statistik seperti hitungan, maksimum, minimum, rata-rata, dll. Statistik ini membantu dalam menyajikan data dalam bentuk ringkasan. Misalnya, menghitung jumlah total suka untuk posting tertentu, menghitung curah hujan bulanan rata-rata atau menemukan jumlah rata-rata pengunjung per bulan di situs web.

Untuk contoh di bagian ini, kami akan menggunakan data sintetis yang mirip dengan data yang dikumpulkan oleh layanan analisis web yang menunjukkan berbagai statistik untuk kunjungan halaman situs web. Setiap halaman memiliki beberapa kode pelacakan yang mengirimkan alamat IP pengunjung bersama dengan stempel waktu ke layanan analisis web. Layanan analisis web menyimpan catatan dari semua kunjungan halaman dan alamat IP pengunjung dan menggunakan program MapReduce untuk menghitung

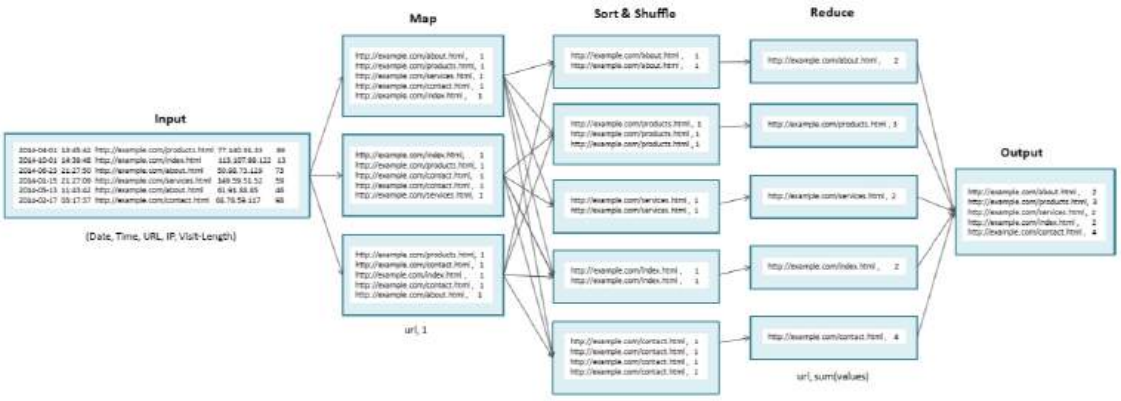
berbagai statistik. Setiap kunjungan ke halaman dicatat sebagai satu baris di log. File log berisi kolom berikut: Tanggal (YYYY-MM-DD), Waktu (HH: MM: SS), URL, IP, Kunjungan-Panjang.

***Count (Menghitung)***

Untuk menghitung hitungan, fungsi mapper memancarkan key-value pair di mana kuncinya adalah bidang yang akan dikelompokkan dan nilainya adalah '1' atau item terkait apa pun yang diperlukan untuk menghitung jumlah. Fungsi peredam menerima key-value pair yang dikelompokkan berdasarkan kunci yang sama dan menjumlahkan nilai untuk setiap grup untuk menghitung jumlah. Mari kita lihat contoh penghitungan jumlah total kunjungan setiap halaman pada tahun 2014, dari log layanan analisis web. Box 3.1 menunjukkan program Python untuk menghitung kunjungan halaman menggunakan MapReduce. Gambar 3.11 menunjukkan data dan key-value pair di setiap langkah pekerjaan MapReduce untuk menghitung hitungan. Fungsi mapper dalam contoh ini mengurai setiap baris input dan memancarkan key-value pair dengan kuncinya adalah URL dan nilainya adalah '1'. Peredam menerima daftar nilai yang dikelompokkan berdasarkan kunci dan menjumlahkan nilai untuk menghitung jumlah.

**■ Box 3.1: Python program for computing count with MapReduce**

```
#Total number of times each page is visited in the year 2014

from mrjob.job import MRJob

class MRmyjob(MRJob):
  def mapper(self, _, line):
    #Split the line with tab separated fields
    data=line.split('')

    #Parse line
    date = data[0].strip()
    time = data[1].strip()
    url = data[2].strip()
    ip = data[3].strip()

    #Extract year from date
    year=date[0:4]

    #Emit URL and 1 if year is 2014
    if year=='2014':
      yield url, 1


  def reducer(self, key, list_of_values):
    yield key,sum(list_of_values)


if __name__ == '__main__':
  MRmyjob.run()
```




Gambar 3.11 *: Menghitung count dnegan MapReduce*

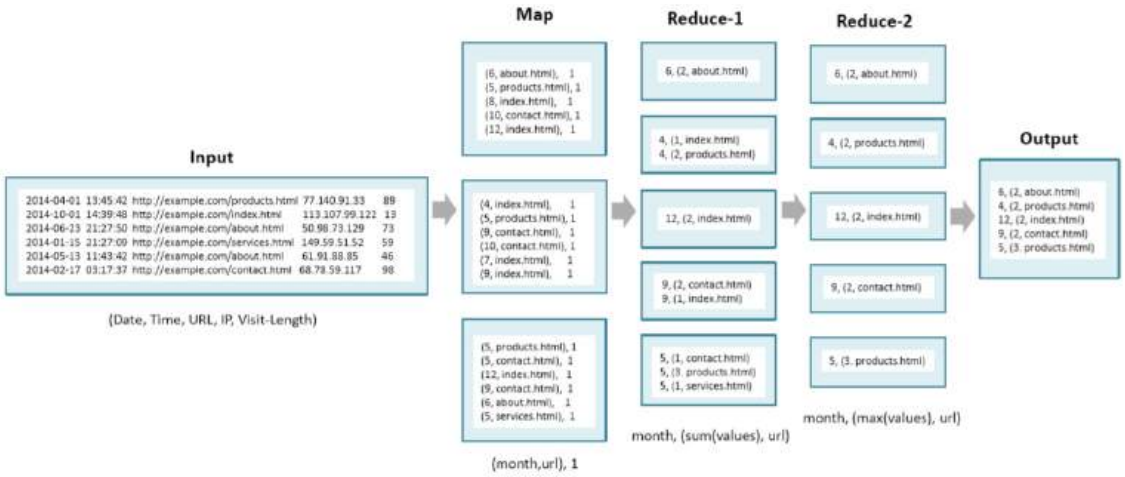
### *Max/Min*

Untuk menghitung maksimum atau minimum, fungsi mapper memancarkan key-value pair di mana kuncinya adalah bidang yang akan dikelompokkan dan nilai berisi item terkait yang diperlukan untuk menghitung maksimum atau minimum. Fungsi peredam menerima daftar nilai yang dikelompokkan berdasarkan kunci yang sama dan menemukan nilai minimum atau maksimum. Mari kita lihat contoh penghitungan halaman yang paling banyak dikunjungi setiap bulan di tahun 2014, dari log layanan analisis web. Box 3.2 menunjukkan program Python untuk menghitung halaman yang paling banyak dikunjungi menggunakan MapReduce. Gambar 3.12 menunjukkan data dan key-value pair di setiap langkah pekerjaan MapReduce untuk menemukan halaman yang paling sering dikunjungi. Contoh ini menggunakan fitur multi-langkah pustaka mrJob Python yang memungkinkan Anda memiliki beberapa fase pengurangan dalam sebuah pekerjaan. Kerangka kerja eksekusi MapReduce dapat menganggapnya sebagai beberapa pekerjaan MapReduce yang dirangkai bersama. Fungsi mapper dalam contoh ini mengurai setiap baris input dan memancarkan key-value pair dengan kuncinya adalah tupel yang terdiri dari bulan dan URL, dan nilainya adalah '1'. Peredam menerima daftar nilai yang dikelompokkan berdasarkan kunci dan merangkum nilai tersebut untuk menghitung kunjungan untuk setiap halaman di setiap bulan. Bulan remit remit sebagai kunci dan tupel yang terdiri dari jumlah kunjungan halaman dan URL halaman sebagai nilainya. Peredam kedua menerima daftar pasangan (jumlah kunjungan, URL) yang dikelompokkan berdasarkan bulan yang sama, dan menghitung jumlah kunjungan maksimum dari daftar nilai. Peredam kedua mengeluarkan bulan sebagai kunci dan tupel yang terdiri dari jumlah kunjungan halaman maksimum dan URL halaman serta nilainya. Dalam contoh ini, pekerjaan dua langkah diperlukan karena kita perlu menghitung jumlah kunjungan halaman terlebih dahulu sebelum menemukan jumlah maksimum.

**Box 3.2: Python program for computing maximum with MapReduce**

```
# Most visited page in each month of year 2014

from mrjob.job import MRJob

class MRmyjob(MRJob):
  def mapper1(self, _, line):
    #Split the line with tab separated fields
    data=line.split('')

    #Parse line
    date = data[0].strip()
    time = data[1].strip()
    url = data[2].strip()
    ip = data[3].strip()

    #Extract year from date
    year=date[0:4]
    month=date[5:7]

    #Emit (month,url) pair and 1 if year is 2014
    if year=='2014':
      yield (month,url), 1


  def reducer1(self, key, list_of_values):
    yield key[0], (sum(list_of_values), key[1])
```

```
  def reducer2(self, key, list_of_values):
    yield key, max(list_of_values)


  def steps(self):
    return [self.mr(mapper=self.mapper1,
    reducer=self.reducer1), self.mr(reducer=self.reducer2)]


if __name__ == '__main__':
  MRmyjob.run()
```




Gambar 3.12: *Perhitungan maximum dengan MapReduce*

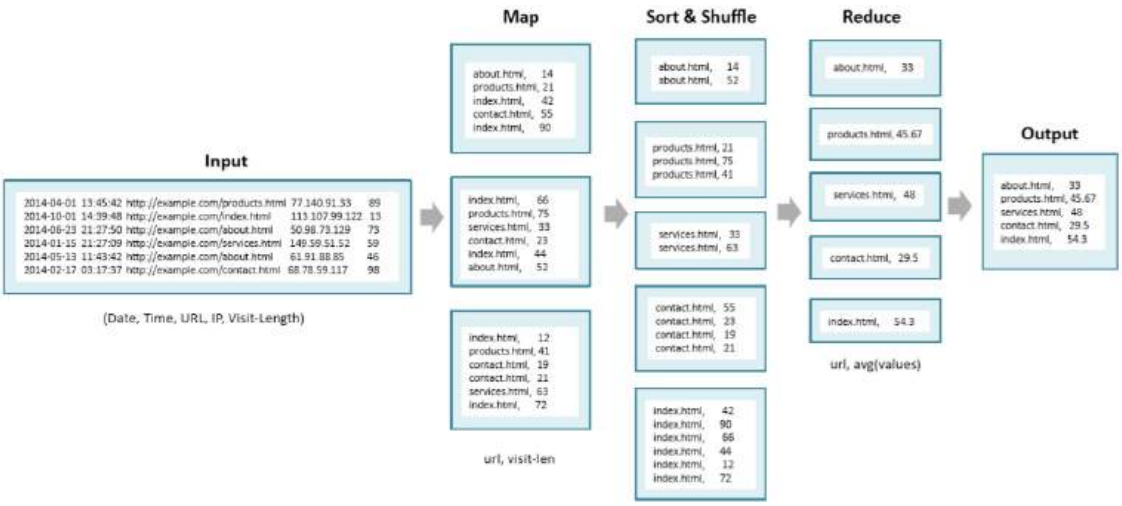
### *Rata-rata / Mean*

Untuk menghitung rata-rata, fungsi mapper memancarkan key-value pair di mana kuncinya adalah bidang yang akan dikelompokkan dan nilai berisi item terkait yang diperlukan untuk menghitung rata-rata. Fungsi peredam menerima daftar nilai yang dikelompokkan berdasarkan kunci yang sama dan menemukan nilai rata-rata. Mari kita lihat contoh penghitungan lama kunjungan rata-rata untuk setiap halaman. Box 3.3 menunjukkan program Python untuk menghitung lama kunjungan rata-rata menggunakan MapReduce. Gambar 3.13 menunjukkan data dan key-value pair di setiap langkah pekerjaan MapReduce. Fungsi mapper dalam contoh ini mengurai setiap baris masukan dan memancarkan key-value pair dengan kuncinya adalah URL dan nilai adalah panjang kunjungan. Peredam menerima daftar nilai yang dikelompokkan berdasarkan kunci (yang merupakan URL) dan menemukan rata-rata nilai ini.

**Box 3.3: Python program for computing average with MapReduce**

```
#Average visit length for each page

from mrjob.job import MRJob

class MRmyjob(MRJob):
  def mapper(self, _, line):
    #Split the line with tab separated fields
    data=line.split('')

    #Parse line
    date = data[0].strip()
    time = data[1].strip()
    url = data[2].strip()
    ip = data[3].strip()
```

```
    visit_len = int(data[4].strip())

    year=date[0:4]
    month=date[5:7]

    yield url, visit_len


  def reducer(self, key, list_of_values):
    count = 0
    total = 0.0
    for x in list_of_values:
      total = total+x
      count=count+1

    avgLen = ("%.2f" % (total/count))
    yield key, avgLen


if __name__ == '__main__':
  MRmyjob.run()
```




Gambar 3.13: *Perhitungan rata-rata menggunakan MapReduce*

### 3.2.2 Top-N

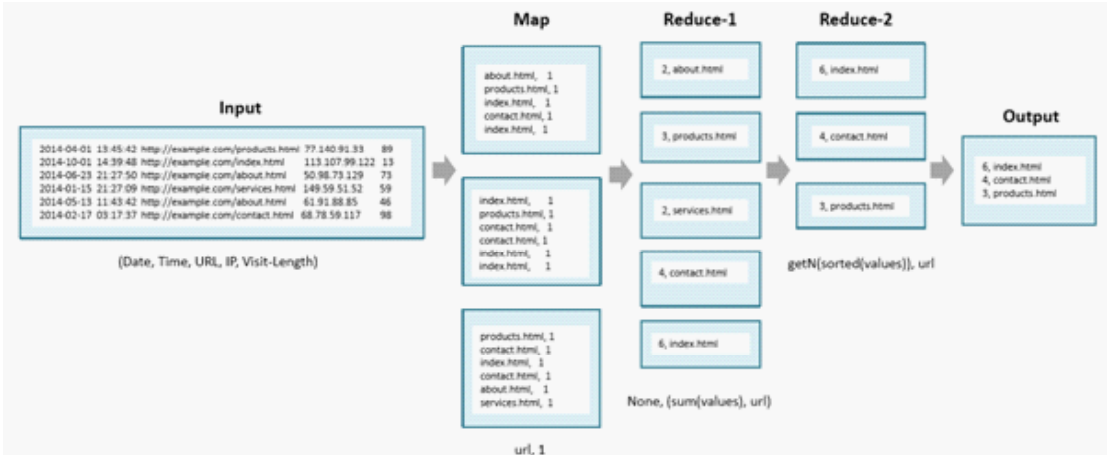
Untuk menemukan record N teratas, fungsi mapper memancarkan key-value pair di mana kuncinya adalah bidang yang akan dikelompokkan dan nilai berisi item terkait yang diperlukan untuk menghitung N teratas. Fungsi peredam menerima daftar nilai yang dikelompokkan berdasarkan kunci yang sama, mengurutkan nilai dan memancarkan nilai N teratas untuk setiap kunci. Dalam pendekatan alternatif, setiap pembuat map memancarkan rekaman N teratas lokalnya dan peredam kemudian menemukan N teratas

global. Mari kita lihat contoh penghitungan 3 halaman teratas yang dikunjungi setiap bulan di tahun 2014. Box 3.4 menunjukkan program Python untuk menghitung halaman yang paling banyak dikunjungi menggunakan MapReduce. Gambar 3.14 menunjukkan data dan key-value pair di setiap langkah pekerjaan MapReduce. Fungsi mapper dalam contoh ini mengurai setiap baris input dan memancarkan key-value pair dengan kuncinya adalah URL dan nilainya adalah '1'. Peredam menerima daftar nilai yang dikelompokkan berdasarkan kunci dan merangkum nilai tersebut untuk menghitung kunjungan setiap halaman. Peredam mengeluarkan None sebagai kunci dan tupel yang terdiri dari jumlah kunjungan halaman dan URL halaman serta nilai. Peredam kedua menerima daftar pasangan (jumlah kunjungan, URL) yang semuanya dikelompokkan bersama (karena kuncinya adalah Tidak Ada). Peredam mengurutkan jumlah kunjungan dan mengeluarkan 3 jumlah kunjungan teratas bersama dengan URL halaman. Dalam contoh ini, pekerjaan dua langkah diperlukan karena kita perlu menghitung jumlah kunjungan halaman terlebih dahulu sebelum menemukan 3 halaman teratas yang dikunjungi.

**Box 3.4: Python program for computing Top-N with MapReduce**

```
# Top 3 visited page in each month of year 2014

from mrjob.job import MRJob

class MRmyjob(MRJob):
  def mapper(self, _, line):
    #Split the line with tab separated fields
    data=line.split('')

    #Parse line
    date = data[0].strip()
    time = data[1].strip()
    url = data[2].strip()
    ip = data[3].strip()
    visit_len=int(data[4].strip())
    #Extract year from date
    year=date[0:4]
    month=date[5:7]

    #Emit url and 1 if year is 2014
    if year=='2014':
      yield url, 1


  def reducer(self, key, list_of_values):
    total_count = sum(list_of_values)
    yield None, (total_count, key)


  def reducer2(self, _, list_of_values):
```

```
    N = 3
    list_of_values = sorted(list(list_of_values), reverse=True)
    return list_of_values[:N]


  def steps(self):
    return [self.mr(mapper=self.mapper1,
      reducer=self.reducer1), self.mr(reducer=self.reducer2)]


if __name__ == '__main__':
  MRmyjob.run()
```



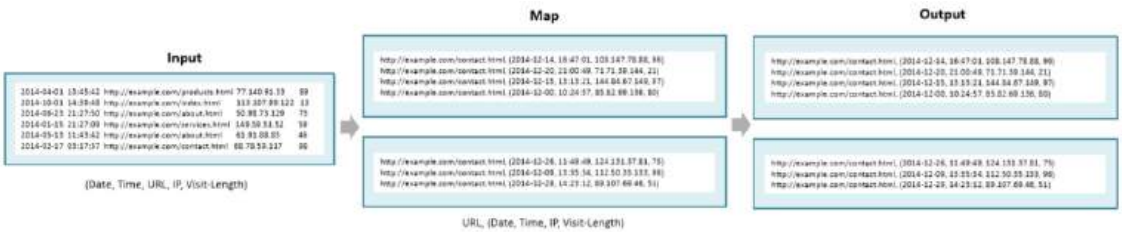

Gambar 3.14 *: Perhitungan Top-N dengan MapReduce*

### 3.2.3 Filter

Pola pemfilteran digunakan untuk menyaring subset rekaman berdasarkan kriteria pemfilteran. Catatan itu sendiri tidak diubah atau diproses. Pemfilteran berguna ketika Anda ingin mendapatkan subkumpulan data untuk diproses lebih lanjut. Pemfilteran hanya membutuhkan tugas Map. Setiap pembuat map menyaring catatan lokalnya berdasarkan kriteria penyaringan dalam fungsi map. Mari kita lihat contoh pemfilteran semua kunjungan halaman untuk halaman 'contact.html' pada bulan Desember 2014, di log layanan analisis web. Box 3.5 menunjukkan program Python untuk memfilter record menggunakan MapReduce. Gambar 3.15 menunjukkan data dan key-value pair di setiap langkah pekerjaan MapReduce. Fungsi mapper dalam contoh ini mengurai setiap baris input, mengekstrak bulan, tahun, dan URL halaman dan mengeluarkan key-value pair jika bulan dan tahun adalah Des 2014 dan URL halamannya adalah 'http://example. com/contact .html '. Kuncinya adalah URL, dan nilainya adalah tupel yang berisi bidang parsing lainnya.

**Box 3.4: Python program for computing Top-N with MapReduce**

```
# Top 3 visited page in each month of year 2014

from mrjob.job import MRJob

class MRmyjob(MRJob):
  def mapper(self, _, line):
    #Split the line with tab separated fields
    data=line.split('')

    #Parse line
    date = data[0].strip()
    time = data[1].strip()
    url = data[2].strip()
    ip = data[3].strip()
    visit_len=int(data[4].strip())
    #Extract year from date
    year=date[0:4]
    month=date[5:7]

    #Emit url and 1 if year is 2014
    if year=='2014':
      yield url, 1


  def reducer(self, key, list_of_values):
    total_count = sum(list_of_values)
    yield None, (total_count, key)


  def reducer2(self, _, list_of_values):
    N = 3
    list_of_values = sorted(list(list_of_values), reverse=True)
    return list_of_values[:N]


  def steps(self):
    return [self.mr(mapper=self.mapper1,
      reducer=self.reducer1), self.mr(reducer=self.reducer2)]


if __name__ == '__main__':
  MRmyjob.run()
```



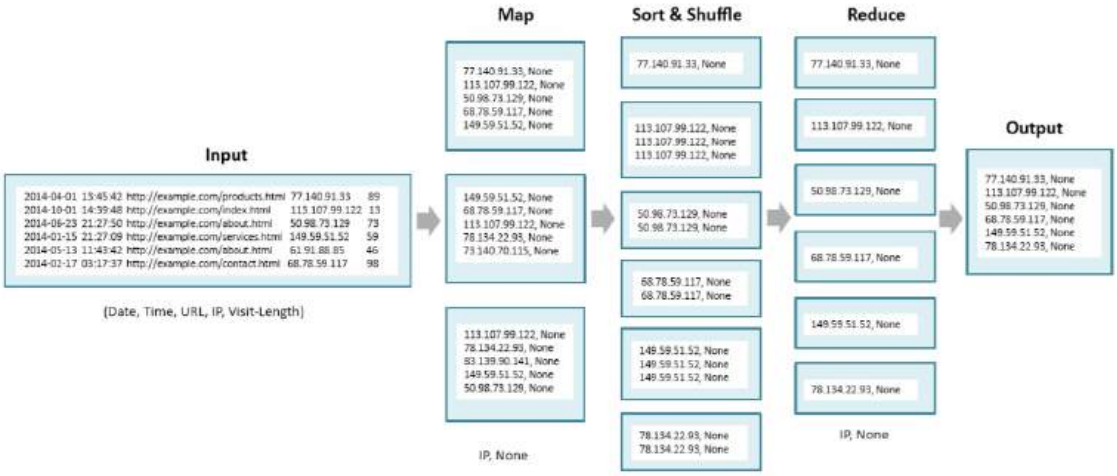

Gambar 3.15: *Filtering dengan MapReduce*

### 3.2.4 Distinct

*Distinct pattern*/Pola yang berbeda digunakan untuk menyaring catatan duplikat atau memancarkan nilai yang berbeda dari sub-bidang dalam dataset. Menemukan catatan yang berbeda mudah dilakukan dengan MapReduce karena catatan dengan kunci yang sama dikelompokkan bersama dalam fase pengurangan. Fungsi mapper memancarkan pasangan kunci-nilai di mana kunci adalah field yang ingin kita temukan berbeda (mungkin subfield dalam record atau seluruh record) dan nilainya adalah None. Fungsi peredam menerima pasangan kunci-nilai yang dikelompokkan berdasarkan kunci yang sama dan mengeluarkan kunci dan nilai sebagai Tidak Ada. Mari kita lihat contoh alamat IP berbeda di log layanan analisis web. Box 3.6 menunjukkan program Python untuk menemukan nilai yang berbeda menggunakan MapReduce. Gambar 3.16 menunjukkan data dan key-value pair di setiap langkah pekerjaan MapReduce. Fungsi map dalam contoh berikut ini masing-masing baris masukan dan memancarkan key-value pair dengan kuncinya adalah alamat IP dan nilainya adalah Tidak Ada. Peredam menerima daftar nilai (semua Tidak Ada) yang dikelompokkan berdasarkan kunci (alamat IP unik) dan mengeluarkan kunci dan nilai sebagai Tidak Ada.

**Box 3.6: Python program for computing distinct with MapReduce**

```
# Distint IP addresses

from mrjob.job import MRJob

class MRmyjob(MRJob):
  def mapper(self, _, line):
    #Split the line with tab separated fields
    data=line.split('')

    #Parse line
    date = data[0].strip()
    time = data[1].strip()
    url = data[2].strip()
    ip = data[3].strip()
    visit_printlen=int(data[4].strip())
    yield ip, None


  def reducer(self, key, list_of_values):
    yield key, None


if __name__ == '__main__':
  MRmyjob.run()
```




Gambar 3.16: *Menemukan perbedaan dengan MapReduce*

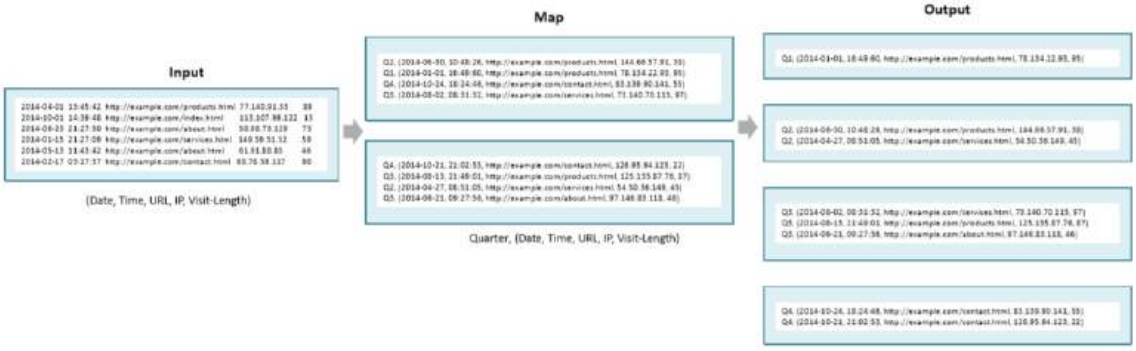
### 3.2.5 Binning

Pola Binning digunakan untuk mempartisi record ke dalam bin atau kategori. Binning berguna saat Anda ingin mempartisi dataset Anda ke dalam bin yang berbeda (berdasarkan kriteria partisi) dan memproses lebih lanjut catatan di setiap bin secara terpisah atau memproses catatan di nampan tertentu. Pengelompokan membutuhkan tugas Map saja. Dalam fungsi mapper, setiap record diperiksa menggunakan daftar kriteria dan ditetapkan ke bin tertentu. Fungsi mapper memancarkan key-value pair di mana kuncinya adalah bin dan nilai adalah catatannya. Tidak ada pemrosesan yang dilakukan untuk rekaman dalam pola ini. Mari kita lihat contoh pemartisian catatan per kuartal (Q1-Q4) di log layanan analisis web. Box 3.7 menunjukkan program Python untuk mempartisi record menggunakan MapReduce. Gambar 3.17 menunjukkan data dan key-value pair di setiap langkah pekerjaan MapReduce.

**Box 3.7: Python program for binning data with MapReduce**

```
# Parition records by Quarter

from mrjob.job import MRJob

class MRmyjob(MRJob):
  def mapper(self, _, line):
    #Split the line with tab separated fields
    data=line.split('')

    #Parse line
    date = data[0].strip()
    time = data[1].strip()
```

```
    url = data[2].strip()
    ip = data[3].strip()
    visit_len=int(data[4].strip())

    #Extract year from date
    year=date[0:4]
    month=int(date[5:7])

    #Emit url and 1 if year is 2014
    if year=='2014':
      if month<=3:
        yield "Q1", (date, time, url, ip, visit_len)
      elif month<=6:
        yield "Q2", (date, time, url, ip, visit_len)
      elif month<=9:
        yield "Q3", (date, time, url, ip, visit_len)
      else:
        yield "Q4", (date, time, url, ip, visit_len)


if __name__ == '__main__':
  MRmyjob.run()
```




Gambar 3.17 *: Binning dengan MapReduce*

### 3.2.6 Indeks Terbalik

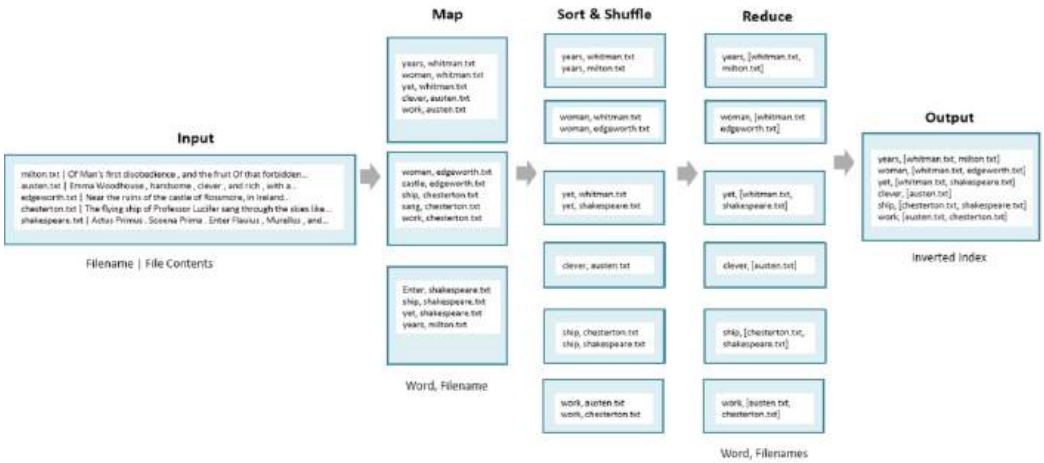
Indeks terbalik adalah struktur data indeks yang menyimpan pemetaan dari konten (seperti kata-kata dalam dokumen atau halaman web) ke lokasi konten (seperti nama file dokumen atau URL halaman). Mesin pencari menggunakan indeks terbalik untuk memungkinkan pencarian dokumen atau halaman yang berisi beberapa konten tertentu lebih cepat.

Untuk menghasilkan indeks terbalik, fungsi mapper memancarkan key-value pair di mana kunci berisi bidang untuk indeks (seperti setiap kata dalam dokumen), dan nilainya adalah pengidentifikasi unik dari dokumen tersebut. Fungsi peredam menerima daftar nilai (seperti ID dokumen) yang dikelompokkan dengan kunci yang sama (kata) dan mengeluarkan kunci dan daftar nilai. Mari kita lihat contoh indeks terbalik untuk beberapa buku. Mari kita

asumsikan bahwa kita memiliki semua konten dari semua buku yang digabungkan menjadi satu file besar dengan dua bidang yang dipisahkan oleh simbol pipa ('|'). Bidang pertama adalah nama file dan bidang kedua berisi semua konten di file. Box 3.8 menunjukkan program Python untuk menghasilkan indeks terbalik menggunakan MapReduce. Gambar 3.18 menunjukkan data dan key-value pair di setiap langkah pekerjaan MapReduce. Fungsi mapper dalam contoh ini mengurai setiap baris input dan memancarkan kkey-value pair di mana kuncinya adalah setiap kata dalam baris dan nilainya adalah nama file. Peredam menerima daftar nilai (nama file) yang dikelompokkan berdasarkan kunci (kata) dan mengeluarkan kata dan daftar nama file tempat kata tersebut muncul.

**Box 3.8: Python program for computing inverted index with MapReduce**

```
#Inverted index

from mrjob.job import MRJob

class MRmyjob(MRJob):
  def mapper(self, _, line):
    doc_id, content = line.split('|')
    words = content.split()
    for word in words:
      yield word, doc_id

  def reducer(self, key, list_of_values):
    docs=[]
    for x in list_of_values:
      docs.append(x)

    yield key, docs


if __name__ == '__main__':
  MRmyjob.run()
```




Gambar 3.18: *Perhitungan inversi index dengan MapReduce*

### 3.2.7 Sorting

Pola penyortiran digunakan untuk mengurutkan record berdasarkan field tertentu. Mari kita lihat contoh pengurutan catatan di log layanan analisis web berdasarkan lama kunjungan. Box 3.9 menunjukkan program Python untuk menyortir menggunakan MapReduce. Gambar 3.19 menunjukkan data dan key-value pair di setiap langkah pekerjaan MapReduce. Fungsi mapper dalam contoh ini mengurai setiap baris input dan memancarkan key-value pair di mana kuncinya adalah None dan nilai adalah tupel yang terdiri dari panjang kunjungan dan bidang lainnya dalam catatan (sebagai tupel berhive). Peredam menerima daftar nilai yang semuanya dikelompokkan bersama (karena kuncinya adalah None). Peredam menggunakan fungsi terurut dari Python, yang mengurutkan daftar tupel berdasarkan elemen pertama dalam tupel (yang merupakan panjang kunjungan).

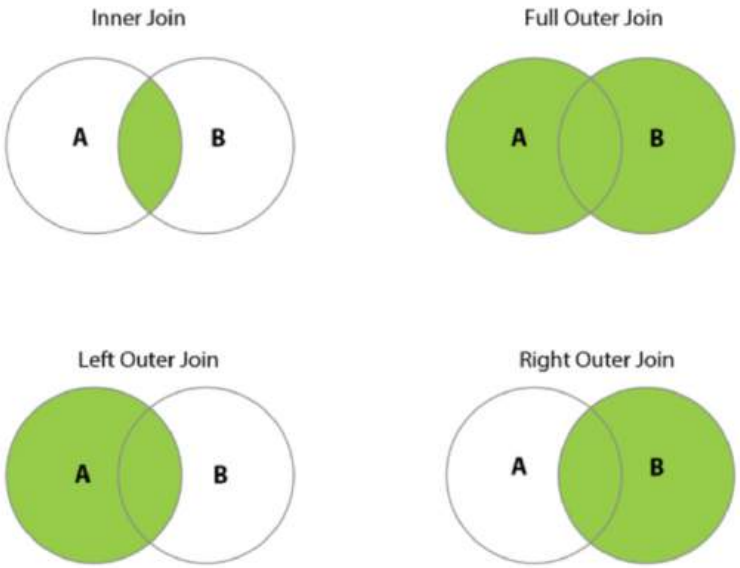
**Box 3.9: Python program for sorting with MapReduce**

```
# Sort by visit length

from mrjob.job import MRJob

class MRmyjob(MRJob):
  def mapper(self, _, line):
    #Split the line with tab separated fields
    data=line.split('')

    #Parse line
    date = data[0].strip()
    time = data[1].strip()
    url = data[2].strip()
    ip = data[3].strip()
    visit_len=int(data[4].strip())
    #Extract year from date
    year=date[0:4]
    month=date[5:7]

    #Emit url and 1 if year is 2014
    if year=='2014':
      yield None, (visit_len, (date, time, url, ip))


  def reducer(self, key, list_of_values):
    list_of_values = sorted(list(list_of_values), reverse=True)
    return list_of_values


if __name__ == '__main__':
  MRmyjob.run()
```


Gambar 3.19: *Sorting dengan MapReduce*

### 3.2.8 Join

Ketika dataset berisi banyak file, gabungan digunakan untuk menggabungkan catatan dalam file untuk diproses lebih lanjut. Gabungan menggabungkan dua atau lebih dataset atau record dalam beberapa file, berdasarkan sebuah field (disebut atribut join atau foreign key). Mari kita ambil contoh penggabungan dua tabel A dan B. Gambar 3.20 menunjukkan berbagai tipe gabungan. Inner Join mengembalikan baris dari kedua tabel yang memiliki nilai yang sama dari kolom yang cocok atau kunci asing. Outputnya berisi kolom dari kedua tabel dengan kunci asing yang cocok. Semua rekaman yang tidak cocok dari kedua tabel tidak disertakan dalam output. Gabungan Luar Penuh adalah jenis gabungan lain yang menyertakan semua rekaman yang cocok dan tidak cocok dari kedua tabel. Full Outer Join mengembalikan semua baris dari kedua tabel dan mengembalikan nilai NULL di kolom setiap tabel di mana tidak ada baris yang cocok. Di Left Outer Join, kolom yang tidak cocok dalam tabel di sisi kiri gabungan disertakan dan bersama dengan semua rekaman yang cocok dari kedua tabel. Left Outer Join mengembalikan semua baris dari tabel di sisi kiri gabungan dan mengembalikan NULL dalam kolom tabel di sebelah kanan di mana tidak ada baris yang cocok. Di Right Outer Join, kolom yang tidak cocok dalam tabel di sisi kanan gabungan disertakan dan bersama dengan semua rekaman yang cocok dari kedua tabel. Right Outer Join mengembalikan semua baris dari tabel di sisi kanan gabungan dan mengembalikan NULL dalam kolom tabel di sebelah kanan di mana tidak ada baris yang cocok.



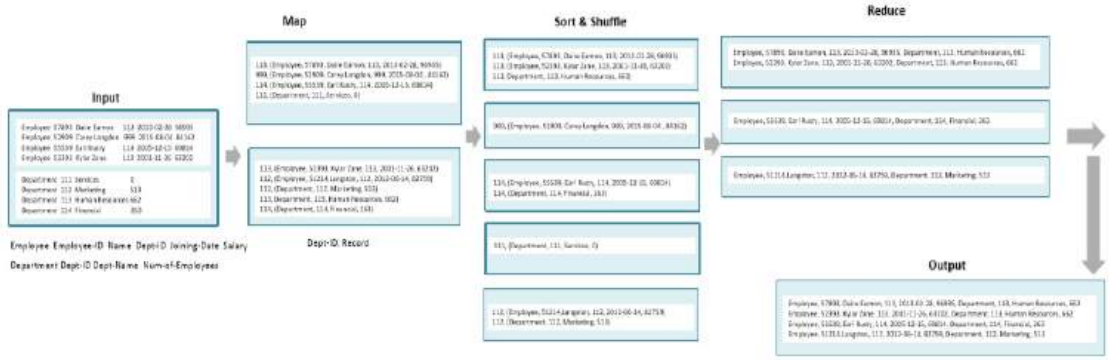

Gambar 3.20 *: Tipe Join*

Mari kita lihat contoh gabungan menggunakan MapReduce. Untuk contoh ini, kami akan menggunakan dua dataset yang berisi catatan karyawan dan departemen dalam suatu organisasi. Dataset karyawan berisi beberapa bidang seperti ID Karyawan, Nama Karyawan, ID Departemen, Data Bergabung, dan Gaji. Dataset departemen berisi bidang-bidang seperti ID Departemen, Nama Departemen dan Jumlah Karyawan. Bidang pertama dalam kumpulan data karyawan berisi kata 'Karyawan' diikuti oleh bidang lainnya yang berisi detail karyawan. Demikian pula, bidang pertama dalam kumpulan data departemen berisi kata 'Departemen', diikuti oleh bidang lainnya yang berisi detail departemen. Mari kita lihat contoh penggabungan kumpulan data karyawan dan departemen dengan bidang ID Departemen. Box 3.10 menunjukkan program Python untuk inner join menggunakan MapReduce. Gambar 3.21 menunjukkan data dan key-value pair di setiap langkah pekerjaan MapReduce. Fungsi mapper dalam contoh ini mengurai setiap baris input dan memancarkan key-value pair di mana kuncinya adalah ID Departemen dan nilainya adalah catatan lengkap. Peredam menerima daftar nilai yang semuanya dikelompokkan berdasarkan ID Departemen. Di reducer, kami memeriksa bidang pertama dari setiap nilai dan jika bidang tersebut adalah 'Karyawan', kami menambahkan nilai ke daftar karyawan dan jika bidang pertama adalah 'Departemen', kami menambahkan nilai ke daftar departemen. Selanjutnya, kami mengulang kedua daftar dan melakukan penggabungan.

Box 3.11, 3.12 dan 3.13 menunjukkan program Python untuk left outer join, right outer join, dan fulllouterjoin masing-masing. Satu-satunya perbedaan dalam program-program ini adalah pada peredam di mana kami mengulang daftar karyawan dan departemen untuk melakukan penggabungan.

**Box 3.10: Python program for computing inner join with MapReduce**

```
from mrjob.job import MRJob

class MyMRJob(MRJob):
  def mapper(self, _, line):
    data=line.split(',')
    if data[0]=='Employee':
      deptID = data[3]
    elif data[0]=='Department':
      deptID = data[1]
    yield deptID, data

  def reducer(self, key, list_of_values):
    values = list(list_of_values)
    employees = [ ]
    departments=[ ]
    for v in values:
      if v[0]=='Employee':
         employees.append(v)
      elif v[0]=='Department':
         departments.append(v)

    #Inner Join
    for e in employees:
      for d in departments:
        yield None, (e+d)


if __name__ == '__main__':
  MyMRJob.run()
```




Gambar 3.21:Perhitungan *Join dengan MapReduce*

**Box 3.11: Python program for computing left outer join with MapReduce**

```
from mrjob.job import MRJob

class MyMRJob(MRJob):
  def mapper(self, _, line):
    data=line.split(',')
    if data[0]=='Employee':
      deptID = data[3]
    elif data[0]=='Department':
      deptID = data[1]
    yield deptID, data

  def reducer(self, key, list_of_values):
    values = list(list_of_values)
    employees = []
    departments=[]
    for v in values:
      if v[0]=='Employee':
         employees.append(v)
      elif v[0]=='Department':
         departments.append(v)

    #Left Outer Join
    for e in employees:
      if len(departments)>0:
        for d in departments:
          yield None, (e+d)
      else:
        yield None, (e)

if __name__ == '__main__':
  MyMRJob.run()
```

**Box 3.12: Python program for computing right outer join with MapReduce**

```
from mrjob.job import MRJob

class MyMRJob(MRJob):
  def mapper(self, _, line):
    data=line.split(',')
    if data[0]=='Employee':
      deptID = data[3]
    elif data[0]=='Department':
      deptID = data[1]
    yield deptID, data

  def reducer(self, key, list_of_values):
    values = list(list_of_values)
    employees = []
    departments=[]
    for v in values:
```

```
      if v[0]=='Employee':
         employees.append(v)
      elif v[0]=='Department':
         departments.append(v)

    #Right Outer Join
    for d in departments:
      if len(employees)>0:
        for e in employees:
          yield None, (e+d)
      else:
        yield None, (d)

if __name__ == '__main__':
  MyMRJob.run()
```

**Box 3.13: Python program for computing full outer join with MapReduce**

```
from mrjob.job import MRJob

class MyMRJob(MRJob):
  def mapper(self, _, line):
    data=line.split(',')
    if data[0]=='Employee':
      deptID = data[3]
    elif data[0]=='Department':
      deptID = data[1]
    yield deptID, data

  def reducer(self, key, list_of_values):
    values = list(list_of_values)
    employees = []
    departments=[]
    for v in values:
      if v[0]=='Employee':
         employees.append(v)
      elif v[0]=='Department':
         departments.append(v)

    #Full Outer Join
    if len(employees)>0:
      for e in employees:
        if len(departments)>0:
          for d in departments:
            yield None, (e+d)
        else:
          yield None, (e)
    else:
      for d in departments:
        yield None, (d)

if __name__ == '__main__':
  MyMRJob.run()
```

## Ringkasan

Dalam bab ini kami menjelaskan berbagai pola komponen arsitektur dan gaya desain untuk sistem dan aplikasi big data. Antrian pesan dapat digunakan antara produsen data dan konsumen data dalam sistem Big data, untuk meratakan beban. Antrian memungkinkan pemisahan produsen data dari konsumen. Memiliki banyak konsumen dapat membantu dalam penyeimbangan beban dan membuat sistem lebih skalabel, andal, dan tersedia. Kami menjelaskan pendekatan untuk pemilihan pemimpin, di mana contoh dalam sistem terdistribusi dapat memilih salah satu contoh sebagai pemimpin mereka. Selanjutnya, kami menjelaskan sharding yang melibatkan partisi data secara horizontal di beberapa node penyimpanan dalam sistem penyimpanan data. Untuk sistem data terdistribusi, ada trade-off antara konsistensi dan ketersediaan. Pertukaran ini dijelaskan dengan Teorema CAP, yang menyatakan bahwa di bawah partisi, sistem data terdistribusi dapat konsisten atau tersedia tetapi tidak keduanya pada waktu yang sama. Selanjutnya, kami mendeskripsikan Bloomfilter, sebuah struktur data propiglistik untuk menguji keanggotaan set. Untuk aplikasi Big data yang melibatkan kueri tertentu yang sering dilakukan, ada baiknya untuk menghitung sebelumnya kueri tersebut. Kueri yang telah dihitung sebelumnya ini disebut tampilan terwujud. Kami menjelaskan arsitektur Lambda yang terdiri dari layer batch, kecepatan, dan penyajian. Arsitektur Lambda memungkinkan kueri data real-time dan historis, dengan pra-komputasi kueri. Selanjutnya, kami menjelaskan pola Scheduler-Agentt-Supervisor, yang dapat digunakan untuk membuat sistem big data lebih tangguh dan toleran terhadap kesalahan. Pola pipa dan filter melibatkan pembagian tugas kompleks menjadi serangkaian tugas berbeda. Kami menjelaskan manfaat menggunakan layanan web, yang menyediakan interface yang menampilkan fungsionalitas sistem kepada klien dalam bentuk berbagai titik akhir. Selanjutnya, kami menjelaskan protokol konsensus Paxos. Di bagian kedua dari bab ini, kami menjelaskan berbagai pola MapReduce untuk analisis data, bersama dengan penerapannya dengan Python.

# NoSQL

Bab 4

Bab ini mencakup

- Database Key-Value
- Database Dokumen
- Database Column Family
- Database grafik

## Pendahuluan

Database non-relasional ("database NoSQL") menjadi populer dengan meningkatnya penggunaan layanan komputasi awan. Database non-relasional memiliki kemampuan penskalaan horizontal yang lebih baik dan peningkatan kinerja untuk big data dengan mengorbankan model konsistensi yang kurang ketat.

Database NoSQL populer untuk aplikasi di mana skala data yang terlibat sangat besar dan datanya mungkin tidak terstruktur. Selain itu, kinerja real-time lebih penting daripada konsistensi. Sistem ini dioptimalkan untuk pengambilan cepat dan operasi penambahan pada catatan. Tidak seperti database relasional, database NoSQL tidak memiliki skema yang ketat. Catatan dapat berupa key-value pair atau dokumen. Kebanyakan database NoSQL diklasifikasikan dalam hal model penyimpanan data atau jenis catatan yang dapat disimpan.

Pada bab ini, kami menjelaskan beberapa database NoSQL yang sering digunakan, karakteristik setiap tipe database, contoh setiap jenis database dan implementasi dengan kode sumber yang lengkap untuk membaca dan menulis data.

## 4.1 Database Key-value

Database key-value adalah bentuk paling sederhana dari database NoSQL. Database ini menyimpan data dalam bentuk key-value pair. Kunci digunakan untuk mengidentifikasi secara unik nilai yang disimpan dalam database. Aplikasi yang ingin menyimpan data menghasilkan kunci unik dan mengirimkan key-value pair ke database. Database menggunakan kunci untuk menentukan di mana nilai harus disimpan. Sebagian besar

database key-value telah mendistribusikan arsitektur yang terdiri dari beberapa node penyimpanan. Data dipartisi di seluruh node penyimpanan oleh kunci. Untuk menentukan partisi tombol, fungsi hash digunakan. Nomor partisi untuk kunci diperoleh dengan menerapkan fungsi hash ke kunci tersebut. Fungsi hash dipilih sedemikian rupa sehingga tombol-tombolnya didistribusikan secara merata ke seluruh partisi.

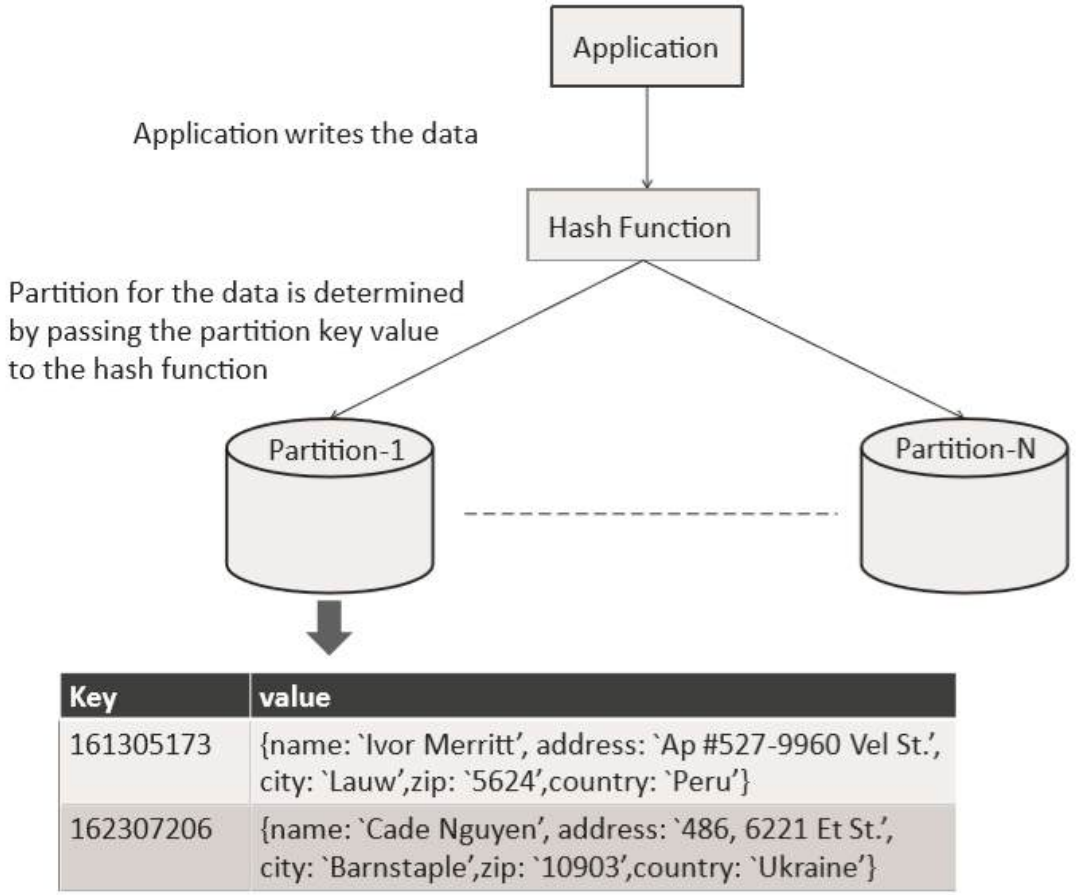
Database key-value memberikan banyak fleksibilitas dalam hal jenis nilai yang dapat disimpan. Nilainya bisa hampir semua jenis (seperti string, integer, float, binary large object (BLOB), dll.). Sebagian besar penyimpanan key-value memiliki dukungan untuk tipe data bahasa pemrograman asli. Ada batasan ukuran nilai yang dapat disimpan. Tidak seperti database relasional di mana tabel memiliki skema tetap dan ada batasan pada kolom, dalam database key-value, tidak ada batasan seperti itu. Database key-value tidak memiliki tabel seperti di database relasional. Namun, beberapa database key-value mendukung tabel, keranjang, atau koleksi untuk membuat ruang nama terpisah untuk kunci tersebut. Kunci dalam tabel, keranjang, atau koleksi bersifat unik.

Database key-value cocok untuk aplikasi yang memerlukan penyimpanan data tidak terstruktur tanpa skema tetap. Database ini dapat ditingkatkan secara horizontal dan dapat menyimpan key-value pair dalam jumlah yang sangat besar. Tidak seperti database relasional yang menyediakan bahasa kueri khusus (seperti SQL), database key-value hanya menyediakan kemampuan kueri dan pencarian dasar. Database key-value cocok untuk aplikasi di mana kemampuan untuk menyimpan dan mengambil data secara cepat dan efisien lebih penting daripada memaksakan struktur atau batasan pada data. Misalnya, database key-value dapat digunakan untuk menyimpan data konfigurasi, data pengguna, data sementara atau perantara (seperti data keranjang belanja), atribut item dan BLOB (seperti audio dan gambar).

### 4.1.1 Amazon DynamoDB

Amazon DynamoDB adalah layanan database NoSQL berperforma tinggi yang terkelola sepenuhnya, dapat diskalakan, dan berperforma tinggi dari Amazon. DynamoDB memberikan kinerja yang cepat dan dapat diprediksi serta skalabilitas yang mulus tanpa overhead operasional. DynamoDB adalah pilihan yang sangat baik untuk database penyajian untuk aplikasi analisis data karena memungkinkan penyimpanan dan pengambilan data dalam jumlah berapa pun dan kemampuan untuk menaikkan atau menurunkan yang

disediakan melalui put tergantung pada persyaratan kinerja aplikasi. DynamoDB adalah layanan yang sangat tersedia dan dapat diandalkan. Data yang disimpan di DynamoDB direplikasi di beberapa zona ketersediaan.



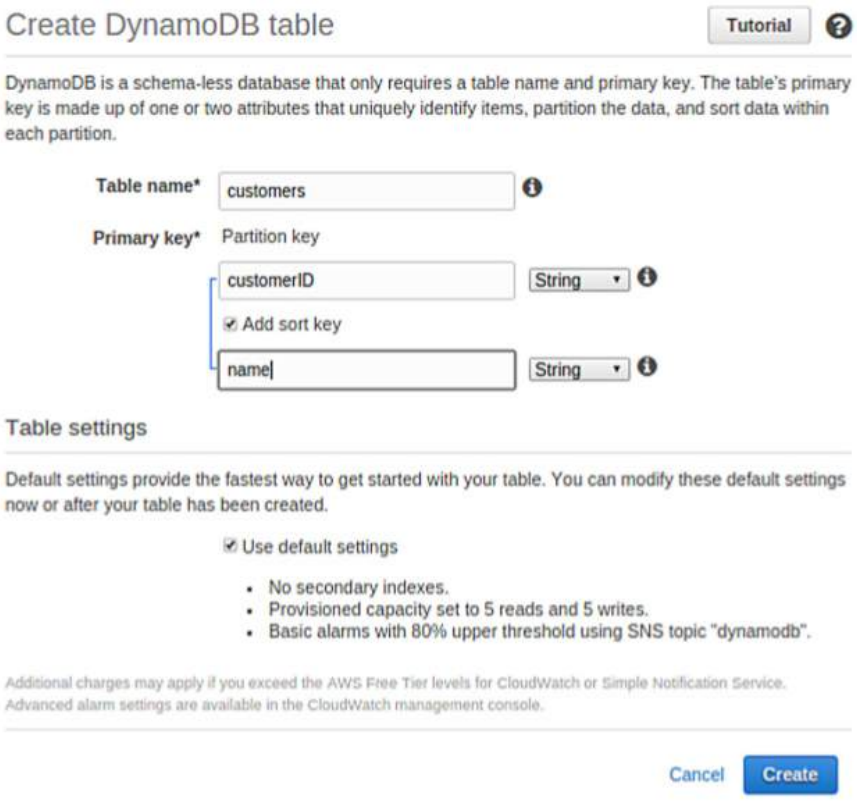

| Key | value |
|---|---|
| 161305173 | {name: `Ivor Merritt', address: `Ap #527-9960 Vel St.', city: `Lauw',zip: `5624',country: `Peru'} |
| 162307206 | {name: `Cade Nguyen', address: `486, 6221 Et St.', city: `Barnstaple',zip: `10903',country: `Ukraine'} |

Gambar 4.1: *Menggunakan database key-value untuk menyimpan catatan pelanggan*

Model data DynamoDB mencakup Tabel, Item, dan Atribut. Tabel adalah kumpulan item dan setiap item adalah kumpulan atribut. Tabel di DynamoDB tidak memiliki skema tetap. Saat membuat tabel, hanya kunci utama yang perlu ditentukan. Kunci utama secara unik mengidentifikasi item dalam tabel. Kunci utama adalah kombinasi dari kunci partisi dan kunci sortir opsional. Kunci partisi di-hash menggunakan fungsi hash untuk menentukan partisi tempat penyimpanan item. Key-value partisi harus unik di semua item jika tidak ada penyortiran yang ditentukan. Kunci sortir opsional dapat ditentukan yang digunakan untuk mengurutkan item di dalam partisi. Jika kunci utama yang digunakan adalah kombinasi dari kunci hash dan kunci urutkan maka ada kemungkinan dua item memiliki nilai yang sama dari kunci partisi, tetapi kunci urutkan harus memiliki nilai yang berbeda. Item terdiri

dari atribut. Atribut dapat ditambahkan saat runtime. Item yang berbeda dalam sebuah tabel dapat memiliki atribut yang berbeda. Setiap atribut adalah key-value pair.

Untuk membaca item, DynamoDB menyediakan tehnologi pengoperasian scan dan query. Operasi pemindaian digunakan untuk mengambil semua item dalam tabel. Anda dapat menentukan kriteria pemfilteran opsional. Kriteria penyaringan dapat mencari nilai tertentu dari atribut atau rentang nilai. Operasi kueri digunakan untuk meminta item dengan kunci utama (baik hanya kunci partisi atau kunci partisi dan kunci urutkan). Untuk membuat kueri tabel menggunakan atribut selain kunci utama, indeks sekunder dapat ditambahkan. Mari kita lihat contoh penggunaan DynamoDB untuk menyimpan informasi pelanggan untuk aplikasi eCommerce. Langkah pertama adalah membuat tabel DynamoDB. Anda dapat membuat tabel dari dasbor DynamoDB atau menggunakan API DynamoDB. Gambar 4.2 menunjukkan contoh pembuatan tabel DynamoDB. Dalam contoh ini, customerID ditentukan sebagai kunci partisi dan nama pelanggan sebagai kunci sortir. Kami menggunakan setelan default lainnya untuk indeks sekunder, kapasitas yang disediakan, dan alarm.




Gambar 4.2: *Membuat Tabel DynamoDB*

Box 4.1 menunjukkan contoh Python dalam menulis data ke tabel DynamoDB. Untuk contoh ini, kami membuat data pelanggan sintetis dari www.generatedata.com dan menyimpan data dalam file CSV. Dalam contoh Python, setiap baris file CSV dibaca satu per satu dalam satu putaran dan data pelanggan ditulis ke tabel DynamoDB.

**■ Box 4.1: Writing data to DynamoDB table**

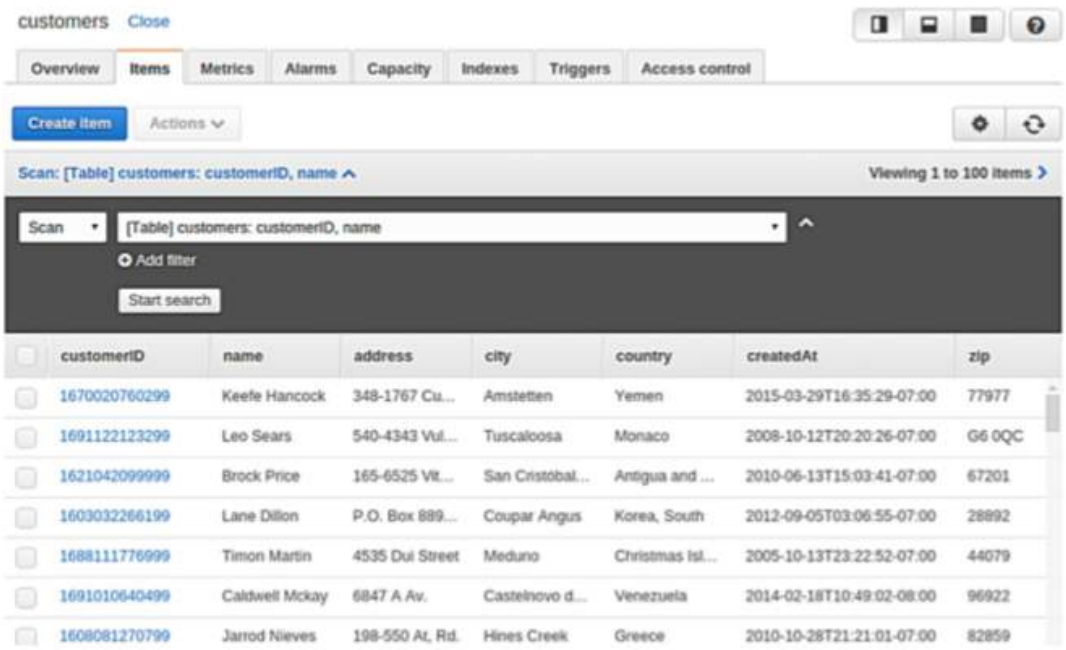
```
import boto.dynamodb2
from boto.dynamodb2.table import Table
from awscredentials import ACCESS_KEY,SECRET_KEY
import csv

REGION="us-east-1"

conn = boto.dynamodb2.connect_to_region(REGION,
aws_access_key_id=ACCESS_KEY,
aws_secret_access_key=SECRET_KEY)


table=Table('customers',connection=conn)
desc = table.describe()
print desc

reader = csv.reader(open('customers.csv','r'))
header=reader.next()
for row in reader:
 item = table.put_item(data={
  'customerID':row[0],
  'name':row[1],
  'address': row[2],
  'city': row[3],
  'zip': row[4],
  'country': row[5],
  'createdAt': row[6]
 },overwrite=True)
```

Gambar 4.3 menunjukkan contoh penggunaan operasi pemindaian dari dasbor *DynamoDB* untuk mengambil data. Box 4.2 menunjukkan contoh Python membaca data dari *DynamoDB* menggunakan operasi pemindaian dan kueri.

**■ Box 4.2: Reading data from DynamoDB table with query and scan operations**

```
import boto.dynamodb2
from boto.dynamodb2.table import Table
from awscredentials import ACCESS_KEY,SECRET_KEY,EC2_KEY_HANDLE

REGION="us-east-1"

conn = boto.dynamodb2.connect_to_region(REGION,
aws_access_key_id=ACCESS_KEY,
aws_secret_access_key=SECRET_KEY)
```

```
table=Table('customers',connection=conn)
desc = table.describe()

#Scan table
result=table.scan()

for item in result:
 print item.items()

#Scan table with filter
result = table.scan(country__eq='India')

for item in result:
 print item.items()

#Scan table with filters
result = table.scan(name__beginswith='A',
 createdAt__between=['2012-03-26T00:00:00-00:00',
 '2016-03-26T00:00:00-00:00'])

for item in result:
 print item.items()

#Query table with partition key
result = table.query_2(customerID__eq='1623072020799')

for item in result:
 print item.items()
```




Gambar 4.3: *Memindai tabel dari Dasbor DynamoDB*

## 4.2 Database Dokumen

Database penyimpanan dokumen menyimpan data semi-terstruktur dalam bentuk dokumen yang dikodekan dalam standar yang berbeda seperti JSON, XML, BSON atau YAML. Dengan data semi-terstruktur yang kami maksud adalah bahwa dokumen yang disimpan serupa satu sama lain (bidang, kunci, atau atribut serupa) tetapi tidak ada persyaratan ketat untuk skema. Dokumen diatur dengan cara yang berbeda dalam database dokumen yang berbeda seperti dalam bentuk koleksi, bucket, atau tag.

Setiap dokumen yang disimpan dalam database dokumen memiliki kumpulan bidang bernama dan nilainya. Setiap dokumen diidentifikasi dengan kunci atau ID unik. Tidak perlu mendefinisikan skema apa pun untuk dokumen sebelum menyimpannya dalam database. Meskipun dimungkinkan untuk menyimpan dokumen seperti JSON atau XML sebagai nilai dalam database key-value, manfaat menggunakan database dokumen di atas database key-value adalah bahwa database ini memungkinkan secara efisien melakukan kueri dokumen berdasarkan nilai atribut dalam dokumen. Database dokumen berguna untuk aplikasi yang ingin menyimpan data semi-terstruktur dengan jumlah bidang yang bervariasi.

Sedangkan dalam database relasional data disimpan dalam bentuk yang dinormalisasi untuk menghilangkan duplikat, dalam database dokumen data disimpan dalam bentuk yang dinormalisasi. Database dokumen tidak menyediakan fungsionalitas gabungan yang disediakan oleh database relasional. Oleh karena itu, semua data yang perlu diambil bersama disimpan dalam sebuah dokumen. Misalnya, dalam aplikasi eCommerce, semua data yang terkait dengan produk tertentu biasanya diambil bersama. Dalam hal ini, dokumen dapat dibuat untuk setiap produk. Setiap dokumen terdiri dari data fitur dan atribut produk.

### 4.2.1 MongoDB

MongoDB adalah sistem database non-relasional berorientasi dokumen. MongoDB adalah database yang kuat, fleksibel, dan sangat dapat diskalakan yang dirancang untuk aplikasi web dan merupakan pilihan yang baik untuk database penyajian untuk aplikasi analisis data. Unit dasar dari data yang disimpan oleh MongoDB adalah dokumen. Dokumen menyertakan sekumpulan key-value pair seperti JSON.

**Box 4.3: Commands for setting up and running MongoDB**

```
#Import the public key used by the package management system
sudo apt-key adv -keyserver hkp://keyserver.ubuntu.com:80 -recv 7F0CEB10

#Create a list file for MongoDB
echo "deb http://repo.mongodb.org/apt/ubuntu trusty/mongodb-org/3.0
multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-3.0.list

#Reload local package database
sudo apt-get update

#Install MongoDB
sudo apt-get install -y mongodb-org

#Start MongoDB service
sudo service mongod start
```

Dokumen dikelompokkan bersama untuk membentuk koleksi. Koleksi tidak memiliki skema tetap dan dokumen berbeda dalam sebuah koleksi dapat memiliki kumpulan key-value pair yang berbeda. Koleksi diatur ke dalam database, dan mungkin ada beberapa database yang berjalan pada satu instance MongoDB. Box 4.3 menunjukkan perintah untuk menyiapkan dan menjalankan MongoDB.

| ID | Document |
|---|---|
| 56fd4f59849f6367af489537 | { "title" : "Motorola Moto G (3rd Generation)", "features" : [ "Advanced water resistance", "13 MP camera", "5in HD display", "Quad core processing power", "5MP rear camera", "Great 24hr battery", "4G LTE Speed" ], "specifications" : { "Color" : "Black", "Size" : "16 GB", "Dimensions" : "0.2 x 2.9 x 5.6 inches", "Weight" : "5.4 ounces" }, "price" : 219.99 } |
| 56fd504d849f6367af489538 | { "title" : "Canon EOS Rebel T5", "features" : [ "18 megapixel CMOS (APS-C) sensor", "EF-S 18-55mm IS II standard zoom lens", "3-inch LCD TFT color, liquid-crystal monitor", "EOS 1080p full HD movie mode" |

```
        ],
        "specifications" : {
                "Color" : "Black",
                "MaximumAperture" : "f/3.5",
                "Dimensions" : "3.94 x 3.07 x 5.12 inches",
                "Weight" : "1.06 pounds"
        },
        "price" : 399
}
```

Gambar 4.4: *Menggunakan database dokumen untuk menyimpan catatan produk*

Box 4.4 menunjukkan contoh penggunaan perintah shell MongoDB untuk menulis data ke database MongoDB dan membuat kueri data. Data yang digunakan dalam contoh ini adalah data produk untuk aplikasi eCommerce. Data untuk setiap produk disimpan sebagai satu dokumen.

**■ Box 4.4: Using MongoDB shell commands**

```
#Launch MongoDB shell
mongo localhost:27017

#Switch to new database named storedb
> use storedb
switched to db storedb

post = {
 "title" :  "Motorola Moto G (3rd Generation)",
 "features" :  [
  "Advanced water resistance",
  "13 MP camera which includes a color-balancing dual LED Flash",
  "5in HD display",
  "Quad core processing power",
  "5MP rear camera",
  "Great 24hr battery performance with a 2470mAh battery",
  "4G LTE Speed"
 ],
 "specifications" :  {
  "Color" :  "Black",
  "Size" :  "16 GB",
  "Dimensions" :  "0.2 x 2.9 x 5.6 inches",
  "Weight" :  "5.4 ounces"
 },
 "price" :  219.99
}

> db.collection.insert(post)
WriteResult({ "Inserted" :  1 })

#Get all documents
> db.collection.find()
{ "_id" :  ObjectId("56fd4f59849f6367af489537"),
"title" :  "Motorola Moto G (3rd Generation)",
"features" :  [ "Advanced water resistance",
```

```
"13 MP camera which includes a color-balancing dual LED Flash",
"5in HD display", "Quad core processing power", "5MP rear camera",
"Great 24hr battery performance with a 2470mAh battery", "4G LTE Speed" ],
"specifications" :  { "Color" :  "Black", "Size" :  "16 GB",
"Dimensions" :  "0.2 x 2.9 x 5.6 inches", "Weight" :  "5.4 ounces" },
"price" :  219.99 }

{ "_id" :  ObjectId("56fd504d849f6367af489538"),
"title" :  "Canon EOS Rebel T5",
"features" :  [ "18 megapixel CMOS (APS-C) sensor",
"EF-S 18-55mm IS II standard zoom lens",
"3-inch LCD TFT color, liquid-crystal monitor",
"EOS 1080p full HD movie mode" ],
"specifications" :  { "Color" :  "Black",
"MaximumAperture" :  "f3.5", "Dimensions" :  "3.94 x 3.07 x 5.12 inches",
"Weight" :  "1.06 pounds" }, "price" :  399 }
```

Box 4.5 menunjukkan program Python untuk menulis data ke MongoDB dan membaca data. Untuk contoh ini, kami menggunakan library pada pyMongo python.

**Box 4.5: Python program for writing data to MongoDB and reading the data**

```
import time
from datetime import date
import datetime
import cPickle
from pymongo import MongoClient

client = MongoClient()
db = client['storedb']
collection = db['current']

item = {
 "title" :  "Motorola Moto G (3rd Generation)",
 "features" :  [
  "Advanced water resistance",
  "13 MP camera which includes a color-balancing dual LED Flash",
  "5in HD display",
  "Quad core processing power",
  "5MP rear camera",
  "Great 24hr battery performance with a 2470mAh battery",
  "4G LTE Speed"
 ],
 "specifications" :  {
  "Color" :  "Black",
  "Size" :  "16 GB",
  "Dimensions" :  "0.2 x 2.9 x 5.6 inches",
  "Weight" :  "5.4 ounces"
 },
 "price" :  219.99
}
```

```
#Insert an item
collection.insert_one(item)

#Retrieve all items
results=db.collection.find()
for item in results:
 print item
```

## 4.3 Family Databases

Dalam family database, unit dasar penyimpanan data adalah kolom, yang memiliki nama dan nilai. Kumpulan dari column membentuk satu baris yang diidentifikasi oleh sebuah row-key. Column yang ada dikelompokkan menjadi column family. Tidak seperti, database relasional, family database tidak perlu memiliki skema tetap dan jumlah kolom tetap di setiap baris. Jumlah kolom dalam aplikasi family database dapat bervariasi di berbagai baris. Kelompok kolom dapat dianggap sebagai map yang memiliki key-value pair dan map ini dapat bervariasi di berbagai baris. Family database menyimpan data dalam bentuk yang didenormalisasi sehingga semua informasi relevan yang terkait dengan entitas yang dibutuhkan oleh aplikasi dapat diambil dengan membaca satu baris. Family database mendukung penulisan dengan throughput tinggi dan telah mendistribusikan dan memiliki arsitektur yang sangat mumpuni.

### 4.3.1 Hbase

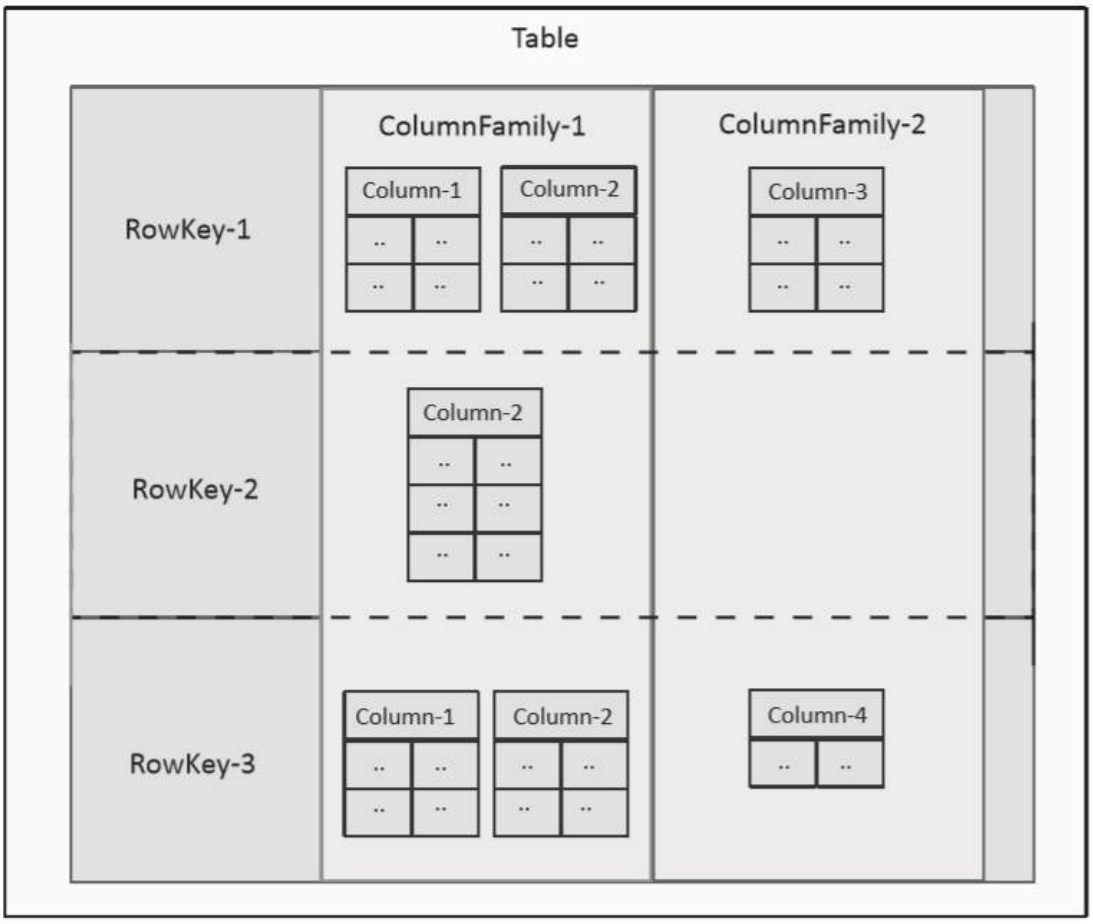
HBase memiliki struktur data yang cukup sederhana, yang hanya terdiri atas Key dan Value. Model ini dikenal dengan istilah Key Value Store (KVS). Key adalah kunci, sedangkan Value adalah data. Tiap kunci berpasangan dengan satu data. Key disusun berurutan, sedangkan data disimpan berdasarkan lokasi yang ditunjuk oleh Key-nya. Pasanangan Key dan Value ini dapat diibaratkan seperti halnya sebuah kamus. Key adalah kata yang ingin kita cari artinya, kata-kata dalam kamus disusun berurutan berdasarkan urutan alfabet. Sedangkan Value adalah arti dari kata itu sendiri, yang disimpan berdasarkan lokasi Key-nya. Pada HBase, Key terdiri atas Row Key, Column Family, Column, dan Timestamp. Sedangkan Value adalah data yang disimpan dalam bentuk 'byte array' yang bisa berupa data teks, angka, website pages, maupun data binary. Row Key juga berupa 'byte array' dan bertindak sebagai 'Primary Key'. HBase sedari awal memang didesain untuk dapat mengelola data berukuran super besar dalam suatu sistem terdistribusi dan memiliki fungsi sharding original bawaan yang dapat bekerja secara otomatis maupun manual. HBase memiliki karakteristik 'fault tolerance' atau mampu menjamin keutuhan

data meskipun terjadi kegagalan pada beberapa komputer yang diperkerjakannya. HBase juga mampu menangani input data yang terjadi secara terus-menerus dari ribuan user yang selama menjadi 'bottle neck' pada sistem database sebelumnya.

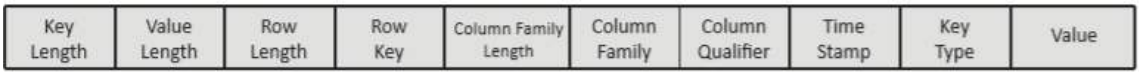
Tidak seperti tabel database relasional, tabel HBase tidak memiliki skema tetap. Kelompok kolom HBase dideklarasikan pada saat pembuatan tabel dan tidak dapat diubah nanti. Kolom dapat ditambahkan secara dinamis, dan HBase dapat memiliki jutaan kolom. HBase sering dideskripsikan sebagai map yang tersortir secara tersebar, persisten, dan multi-dimensi. Mari kita lihat fitur-fitur ini secara detail:

- Tersebar: Dalam database relasional tradisional, tabel memiliki skema tetap. Setiap baris dalam tabel memiliki jumlah kolom yang sama. Setiap baris memiliki semua kolom meskipun semuanya tidak diisi. HBase, sebaliknya, memiliki tabel renggang karena setiap baris tidak perlu memiliki semua kolom. Hanya kolom yang diisi dalam satu baris yang disimpan.
- Terdistribusi: HBase adalah database terdistribusi. Tabel HBase dipartisi berdasarkan Row Key menjadi beberapa wilayah. Setiap wilayah berisi serangkaian kunci baris. Penerapan HBase tipikal berisi beberapa Server Wilayah. Setiap Server Wilayah berisi beberapa wilayah dari tabel yang berbeda.
- Persisten: HBase bekerja di atas HDFS dan semua data yang disimpan dalam tabel HBase disimpan di HDFS.
- Multi-dimensi: HBase menyimpan data sebagai key-value pair dengan kuncinya multi-dimensi. Kunci meliputi: (Table, Row Key, Column Family, Column, TimeStamp) seperti yang ditunjukkan pada Gambar 4.6. Untuk setiap entri / sel, beberapa versi disimpan, yang diberi timestamp.
- Urut: Baris HBase diurutkan berdasarkan Row Key dalam urutan leksikografik. Kolom dalam column family diurutkan berdasarkan column key.

Sel HBase tidak bisa ditulis berlebihan. Karena sel diversi dengan stempel waktu, ketika nilai yang lebih baru ditambahkan, nilai yang lebih lama juga dipertahankan. Data disimpan dalam sel sebagai array byte. Aplikasi bertanggung jawab untuk menafsirkan tipe data dengan benar.




Gambar 4.5: *Struktur Tabel HBase*

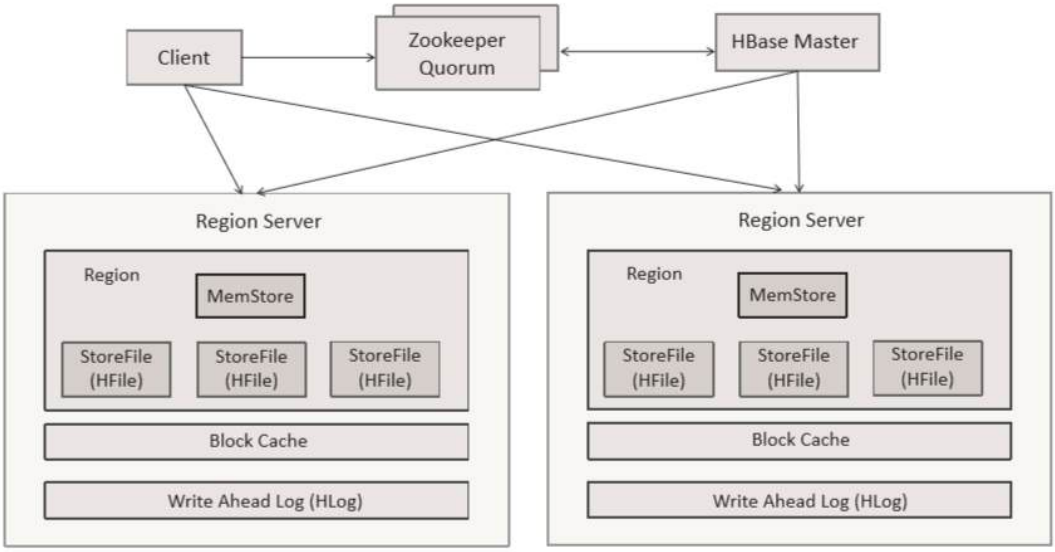



Gambar 4.6: *Format HBase key-value*

### *Arsitektur*

Sebuah cluster HBase terdiri HMaster, RegionServer, ZooKeeper, dan HDFS (Gambar 7). HMaster adalah server pada HBase yang bertugas men-start HBase, mendistribusikan Region ke RegionServer yang terdaftar, mendeteksi dan memulihkan RegionServer yang rusak. HMaster tidak menangani data yang disimpan pada HBase. RegionServer adalah server yang bertugas menyimpan dan mengelola Region-region yang diterimanya dari HMaster, menangani permintaan client, dan mempartisi Region yang sudah melewati ukuran maksimal, kemudian melaporkan Region yang telah dipartisi tersebut kepada HMaster. ZooKeeper bertugas mengelola informasi pokok tentang kondisi HBase itu sendiri. Ia senantiasa mengetahui kondisi terkini dari para RegionServer, kemudian memberikan informasi ini kepada HMaster. Berdasarkan informasi dari ZooKeeper ini lah HMaster

mengatur pembagian Region ke RegionServer dan memulihkan RegionServer yang mengalami kerusakan. ZooKeeper juga menyimpan informasi tentang lokasi RootKatalog dan alamat HMaster. Bila suatu client hendak mengakses HBase untuk pertama kalinya, maka ia harus memulai koneksinya melalui ZooKeeper. HDFS (Hadoop Distributed File System) berfungsi sebagai media penyimpanan data bagi HBase. Semua data yang diloading ke HBase dan data log HBase disimpan dalam HDFS..




Gambar 4.7: *Arsitektur HBase*

**Data Storage & Operasi**

Setiap Server Wilayah menyimpan dua jenis file - file penyimpanan (HFile) dan log depan tulis (HLog). HFile berisi sejumlah variabel blok data dan blok tetap untuk informasi file dan trailer. Setiap blok data berisi angka ajaib dan beberapa key-value pair . Ukuran default dari blok data adalah 64 KB. Blok indeks menyimpan offset untuk data dan meta-blok. Trailer menyimpan petunjuk ke blok lain. HFiles disimpan di HDFS.

Setiap Server Wilayah memiliki Memstore dan Block Cache. Memstore menyimpan hasil edit terbaru ke data dalam memori dan Block Cache menyimpan cache data blok.
Setiap Server Wilayah juga mempertahankan log depan tulis (WAL) yang dikenal sebagai Hlog yang mencatat penulisan (yang juga ditulis ke Memstore). Karena HLog disimpan di HDFS, HLog memastikan bahwa bahkan jika Memstore hilang (yang merupakan buffer dalam memori), penulisan tidak akan pernah hilang.

Setiap Server Wilayah memiliki cache Blok, yang merupakan penyimpanan dalam memori yang menyimpan cache blok yang paling baru digunakan untuk pencarian cepat.
HBase mendukung operasi berikut:

- Get: Operasi get digunakan untuk mengembalikan nilai untuk kunci baris tertentu.
- Scan: Operasi pemindaian mengembalikan nilai untuk berbagai tombol baris.
- Put: Operasi Put digunakan untuk menambah entri baru.
- Delete: Operasi penghapusan menambahkan penanda khusus yang disebut Tombstone ke entri. Entri yang ditandai dengan Batu Nisan dibuang selama proses pemadatan (dibahas nanti di bab ini).

Struktur penyimpanan yang digunakan oleh HBase adalah Log Structured Merge (LSM) Tree. LSM Tree menggunakan dua pohon, yang satu merupakan pohon dalam memori yang lebih kecil (dalam Memstore) dan yang lainnya adalah pohon yang lebih besar yang disimpan pada disk (sebagai berkas penyimpanan). Ketika ukuran pohon dalam memori melebihi batas tertentu, ia digabungkan dengan pohon yang lebih besar pada disk menggunakan algoritma pengurutan gabungan dan pohon dalam memori baru dibuat. LSM Tree memungkinkan HBase menangani mahasiswaan dan penulisan acak yang cepat untuk data dalam jumlah besar. LSM Tree mencapai ini dengan mengubah akses data acak menjadi akses data berurutan.

**Read Path**

Untuk operasi baca (dapatkan atau pindai) klien pertama menghubungi Zookeeper untuk mendapatkan lokasi tabel ROOT. Klien kemudian memeriksa tabel ROOT untuk tabel META yang benar yang berisi kunci baris dan mendapatkan nama Server Wilayah yang bertanggung jawab untuk melayani permintaan untuk kunci baris tersebut. Klien kemudian menghubungi Server Wilayah secara langsung untuk menyelesaikan operasi baca. Pencarian tabel ROOT dan META di-cache oleh klien sehingga dalam operasi baca berikutnya klien dapat langsung menghubungi Server Wilayah yang benar untuk kunci baris tertentu.

**Write Path**

Semua permintaan tulis pertama kali login ke WAL (HLog) secara berurutan. Setelah data dicatat, itu juga ditulis ke Memstore. Memstore menyimpan pembaruan terkini untuk mengaktifkan pencarian cepat. Seiring waktu, Memstore mulai terisi saat pembaruan baru disimpan. Ketika Memstore sudah terisi, itu akan dialihkan ke disk untuk membuat file penyimpanan baru (HFile).

### Pemadatan

Setiap kali Memstore dialihkan ke disk, file penyimpanan baru dibuat. Seiring waktu, banyak file penyimpanan kecil dibuat di HDFS. Karena HDFS dirancang untuk bekerja lebih baik dengan sejumlah kecil file besar (dibandingkan dengan sejumlah besar file kecil), proses yang disebut pemadatan dilakukan untuk menggabungkan file kecil menjadi satu file. Proses pemadatan meningkatkan efisiensi mahasiswaan karena sejumlah besar file kecil tidak perlu dicari. Pemadatan HBase terdiri dari dua jenis - minor dan mayor. Pemadatan kecil menggabungkan file-file kecil menjadi satu file bila jumlahnya melebihi ambang batas. Pemadatan kecil dilakukan secara teratur (biasanya beberapa kali dalam sehari). Pemadatan besar menggabungkan semua file penyimpanan menjadi satu file penyimpanan besar. Dalam proses pemadatan utama, nilai yang kadaluwarsa dan dihapus (sel yang memiliki versi kedaluwarsa atau sel yang ditandai dengan Batu Nisan) dihapus. Proses pemadatan meningkatkan kinerja HBase karena mencari file penyimpanan tunggal yang besar untuk operasi get atau scan lebih efisien daripada mencari beberapa file penyimpanan kecil.

### Bloom Filter

HBase menggunakan Filter Bloom untuk mengecualikan file penyimpanan yang perlu dicari saat melayani permintaan baca untuk kunci baris tertentu. Bloomfilter adalah struktur data propiglistik yang digunakan untuk menguji apakah suatu elemen adalah anggota suatu himpunan. Bloomfilter dapat memberikan jawaban definitif jika elemen tersebut ada di dalam himpunan, namun, filter juga dapat mengatakan bahwa sebuah elemen ada di dalam himpunan sementara itu tidak. Dengan kata lain, positif palsu dimungkinkan, tetapi negatif palsu tidak dimungkinkan. Jumlah positif palsu dapat disetel kurang dari 1%.

## 4.3.2 Contoh Penggunaan HBase

### Garis komando

HBase hadir dengan shell interaktif dari mana pengguna dapat melakukan berbagai operasi HBase. Shell HBase dapat diluncurkan sebagai berikut:

```
# Launch HBase Shell
$ hbase shell
hbase(main):001:0>
```

Untuk membuat tabel, perintah create digunakan seperti yang ditunjukkan di bawah ini. Nama tabel dan kelompok kolom ditentukan saat membuat tabel.

```
# Create HBase table
> create 'products', 'details', 'sale'

=> Hbase::Table - products
```

Dalam contoh ini, kami membuat tabel bernama produk yang memiliki dua kelompok kolom bernama detail dan penjualan. Perintah daftar dapat digunakan untuk membuat daftar semua tabel di HBase, seperti yang ditunjukkan di bawah ini:

```
#List HBase tables
> list
TABLE
products
1 row(s) in 1.7470 seconds
=> ["products"]
```

Untuk menulis data ke HBase, perintah put dapat digunakan. Box di bawah ini menunjukkan contoh penulisan ke tabel produk. Untuk baris dengan kunci baris baris ke-1 dan baris ke-2 ditulis dengan data keluarga kolom dan kolom (nama).

```
> put 'products', 'row-1', 'details:name', 'Cloud Book'
> put 'products', 'row-2', 'details:name', 'IoT Book'
```

Kolom dapat ditambahkan secara dinamis. Box di bawah ini menunjukkan contoh penambahan kolom baru ke baris yang telah dibuat sebelumnya.

```
> put 'products', 'row-1', 'details:ISBN', '9781494435141'
> put 'products', 'row-2', 'sale:Price', '50'
```

Untuk membaca data, HBase menyediakan operasi get dan scan. Box di bawah ini menunjukkan contoh membaca baris dengan baris-kunci baris-1.

```
hbase(main):027:0> get 'products', 'row-1'
COLUMN CELL
details:name timestamp=1434772884378, value=Cloud Book
details:ISBN timestamp=1434772890556, value=9781494435141
```

Hasil operasi get menunjukkan dua sel di baris-1. Nilainya diberi stempel waktu, dan beberapa versi dapat disimpan untuk sebuah sel. Box di bawah ini menunjukkan contoh operasi pemindaian yang mengembalikan semua baris dalam sebuah tabel.

```
■ > scan 'products'
ROW COLUMN+CELL
row-1 column=details:name, timestamp=1434772884378, value=Cloud Book
row-2 column=details:name, timestamp=1434772923678, value=IoT Book
2 row(s) in 0.0210 seconds
```

**Contoh HBase - Python**

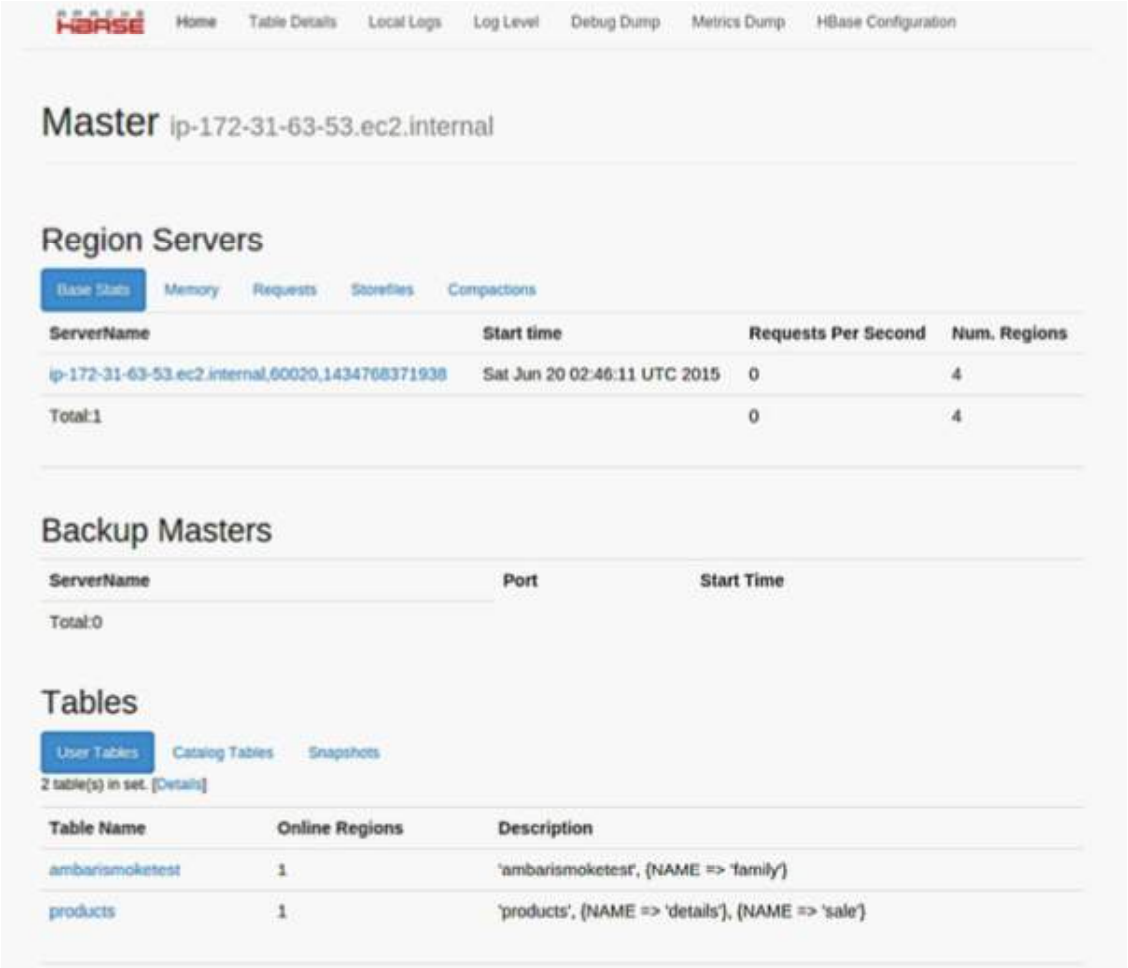
Mari kita lihat beberapa contoh operasi HBase menggunakan Python. Kami akan menggunakan library Python happybase untuk contoh-contoh ini.

**■ Box 4.6: HBase operations using Python**

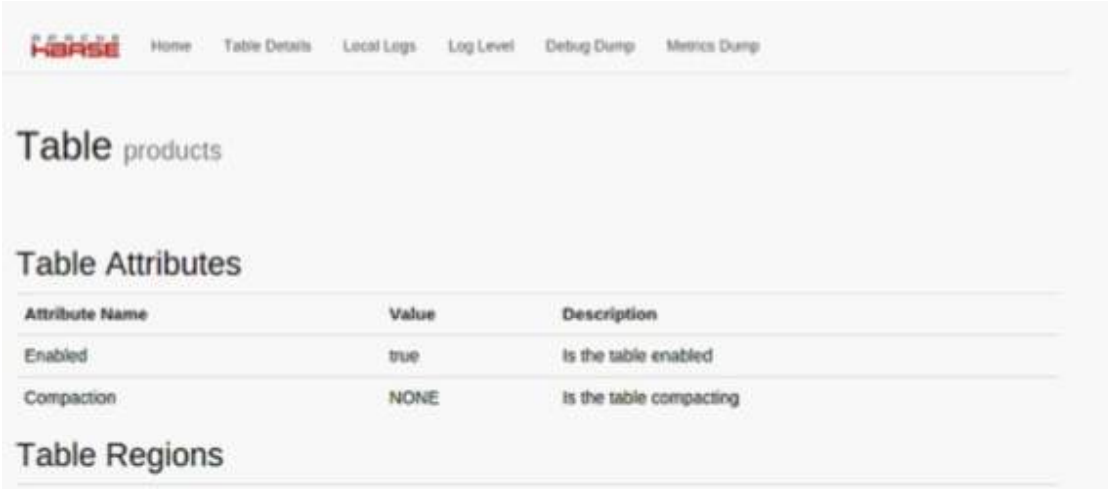
```
# Start thrift server first:
# hbase thrift start

import happybase

connection = happybase.Connection(host='localhost')
table = connection.table('products')

# Put
table.put('row-1', 'details:name': 'Cloud Book')

# Get
row = table.row('row-1')
print row['details:name']

# Scan
for key, data in table.scan():
print key, data

# Delete row
row = table.delete('row-1')

# Batch put
b = table.batch()
b.put('row-key-1', 'details:name': 'Cloud Book',
'details:ISBN': '9781494435141',
'sale:StartSale': '01-01-2014', 'sale:Price':'50')


b.put('row-key-2', 'details:name': 'IoT Book',
'details:ISBN': '9780996025515',
'sale:StartSale': '01-01-2015', 'sale:Price':'55')
b.send()
```

### Interface Web Hbase

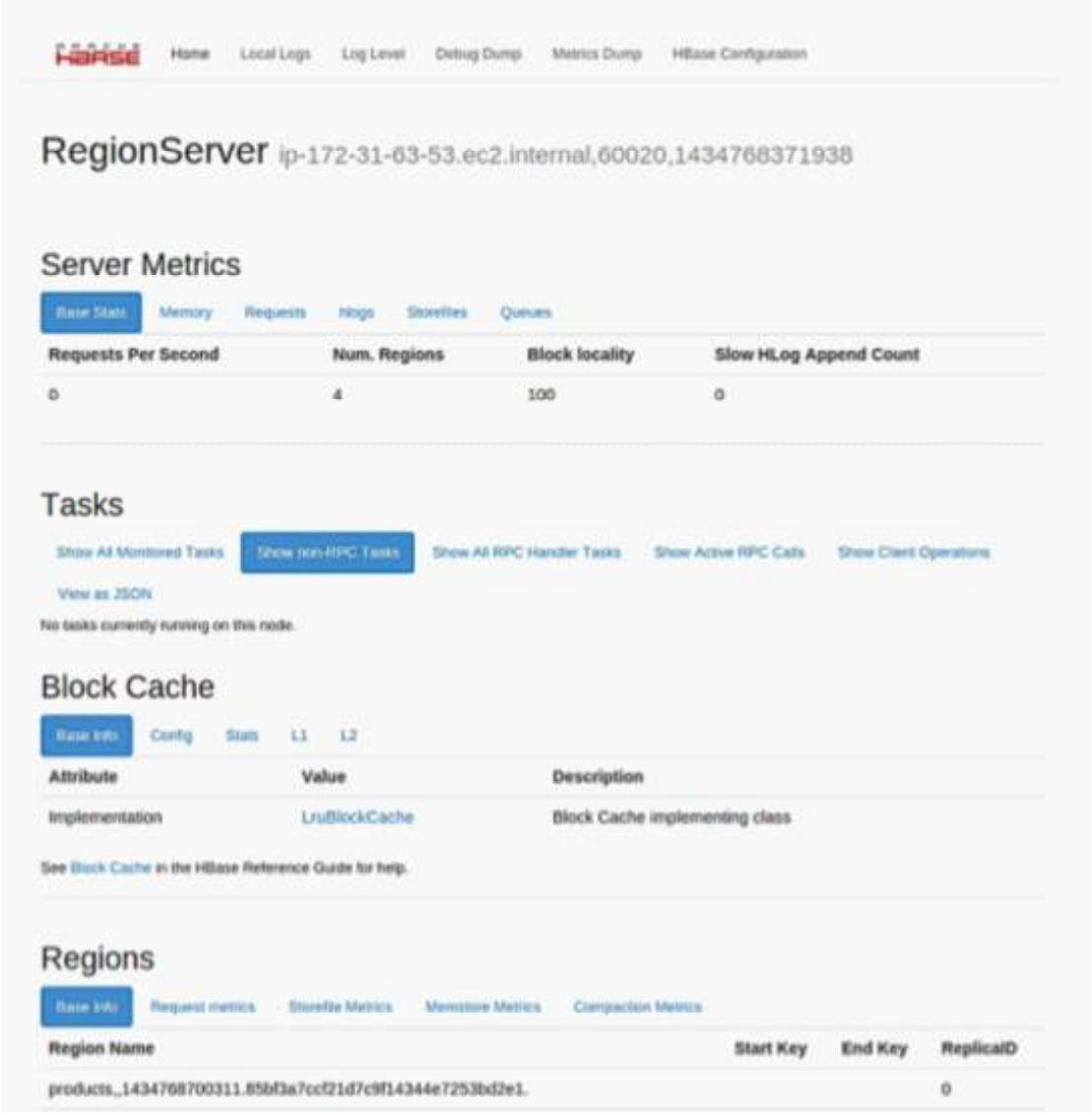
Gambar 4.8 - 4.10 menunjukkan Interface web HBase.




Gambar 4.8: *Interface web HBase menampilkan rincian HBase Master dan daftar tabel*




Gambar 4.9: *Interface web HBase menampilkan detail tabel*




Gambar 4.10: *Interface web HBase menampilkan detail server wilayah*

## 4.4 Database Grafik

Database grafik adalah database NoSQL yang dirancang untuk menyimpan data yang memiliki struktur grafik dengan node dan edge. Sedangkan data model database relasional berupa baris dan kolom, sedangkan data model database grafik berupa node dan relasi. Node merepresentasikan entitas dalam model. Nodeshaveasetofatribut. Node dapat mewakili berbagai jenis entitas, misalnya, orang, tempat (seperti kota, restoran, atau gedung) atau objek (seperti mobil). Hubungan antar entitas direpresentasikan dalam bentuk link antar node. Tautan juga memiliki sekumpulan atribut. Tautan bisa diarahkan atau tidak. Tautan terarah menunjukkan bahwa hubungan itu searah. Misalnya, untuk dua entitas penulis dan buku, ada hubungan searah yang disebut 'menulis' di antara mereka, seperti penulis menulis buku. Sedangkan untuk dua orang teman, katakanlah A dan B, hubungan pertemanan antara A dan B adalah dua arah. Dalam terminologi teori graf, simpul-simpul dalam sebuah graf adalah simpul-simpul yang merepresentasikan entitas-entitas dan tepi-tepi antar simpul-simpul tersebut adalah penghubung antar simpul-simpul yang merepresentasikan hubungan antar entitas. Sekumpulan node bersama dengan tautan di antara mereka membentuk jalur.

Database grafik berguna untuk berbagai aplikasi, di mana Anda mungkin perlu memodelkan entitas dan hubungan di antara mereka, seperti media sosial, keuangan, jaringan, atau berbagai jenis aplikasi perusahaan. Dalam database relasional, hubungan antar entitas dimodelkan dalam bentuk pada tabel yang berbeda dengan kunci primer dan kunci asing. Langkah-langkah yang terlibat dalam pemetaan diagram hubungan entitas ke dalam tabel relasional dijelaskan di Bab-10. Menghitung hubungan dan membuat kueri entitas terkait dalam database relasional memerlukan operasi gabungan yang kompleks antara tabel database. Database grafik, berbeda dengan database relasional, model hubungan dalam bentuk link antar node. Karena hubungan antara entitas secara eksplisit disimpan dalam bentuk tautan, kueri untuk entitas terkait dalam database grafik jauh lebih sederhana dan lebih cepat daripada database relasional karena operasi gabungan yang kompleks dihindari. Database grafik cocok untuk aplikasi yang fokus utamanya adalah menanyakan hubungan antara entitas dan menganalisis hubungan.

### 4.4.1 Neo4j

Di bagian ini, kami akan menjelaskan database grafik Neo4j dengan bantuan beberapa contoh. Neo4j adalah salah satu database grafik populer yang menyediakan dukungan untuk Atomicity, Consistency, Isolation, Durability (ACID). Neo4j mengadopsi model grafik

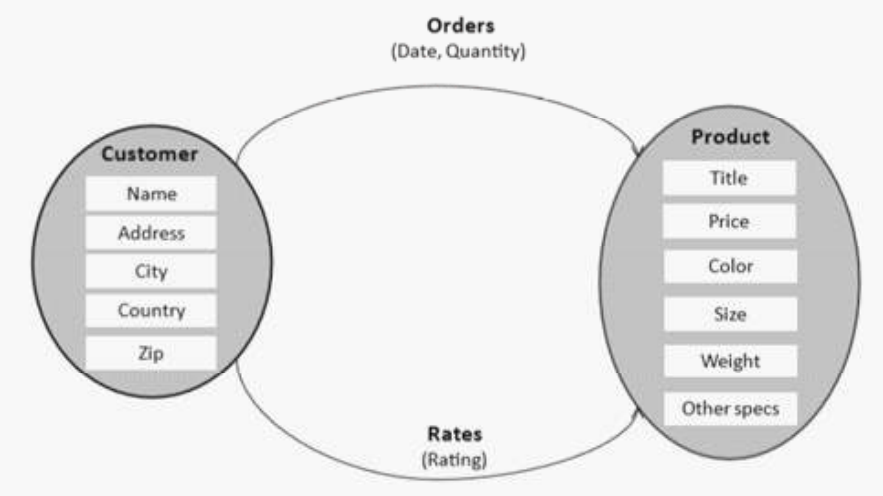
yang terdiri dari node dan relasi. Node dan relasi memiliki properti yang ditangkap dalam bentuk beberapa atribut (key-value pair ). Node ditandai dengan label yang digunakan untuk merepresentasikan peran berbeda dalam domain yang dimodelkan. Box 4.7 menunjukkan perintah untuk menyiapkan Neo4j.

**■ Box 4.7: Setting up Neo4j graph database**

```
#Download the stable release of Neo4j for Linux
#from http://neo4j.com/download/

#Extract the archive
tar -xf neo4j-*.tar.gz
```

```
#Run neo4j
NEO4J_HOME/bin/neo4j start

#Visit http://localhost:7474 in your web browser.
```

Mari kita lihat contoh penggunaan database Grafik untuk aplikasi eCommerce. Gambar 4.11 menunjukkan model grafik properti berlabel untuk aplikasi eCommerce. Dalam grafik ini, kami memiliki dua jenis node: Pelanggan dan Produk. Node Pelanggan memiliki atribut seperti nama pelanggan, alamat, kota, negara, dan kode pos. Node Produk memiliki atribut seperti judul produk, harga dan berbagai properti khusus produk lainnya (seperti warna, ukuran, berat, dll.). Ada dua jenis hubungan antara pelanggan dan node produk: Pesanan atau Tarif. Hubungan pesanan antara pelanggan dan produk memiliki properti seperti tanggal dan kuantitas pesanan. Hubungan Tarif antara pelanggan dan produk memiliki properti tunggal untuk menangkap peringkat pelanggan.




Gambar 4.11: *Contoh grafik properti berlabel*

Untuk operasi buat, baca, perbarui, dan hapus (CRUD), Neo4j menyediakan bahasa kueri yang disebut Cypher. Cypher memiliki beberapa kesamaan dengan bahasa kueri SQL yang digunakan untuk database relasional. Gambar 4.12 menjelaskan penggunaan beberapa konstruksi Cypher yang umum digunakan.


Gambar 4.12: *Penggunaan umum Cypher constructs*

Box 4.8 menunjukkan contoh penggunaan Cypher untuk membuat node pelanggan dan produk dan hubungan antara node.

**Box 4.8: Examples of Neo4j Cypher statements**

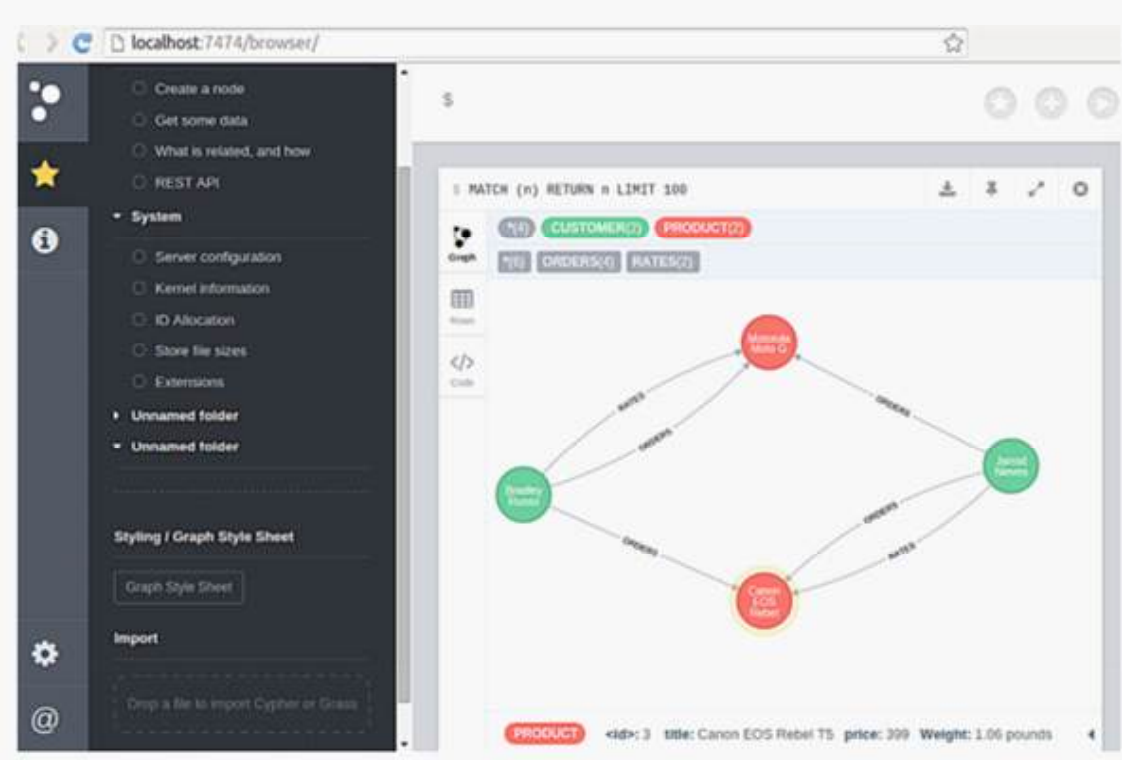
```
#Create customer
CREATE (c:CUSTOMER {name:  "Bradley Russo",
  address:"P.O. Box 486, 6221 Et St.,Barnstaple",
  country:"Ukraine", zipcode:"10903"});

#Create product
CREATE (p:PRODUCT {title :  "Motorola Moto G",
  Color :  "Black", Size :  "16 GB",
  Weight :  "5.4 ounces", price :  219.99 });

#Create relationship between customer and product
MATCH(c:CUSTOMER{name:"Bradley Russo"}),
 (p:PRODUCT{title:"Motorola Moto G"}) WITH c, p
 CREATE (c)-[:RATES]->(p);

#Return all data
MATCH (n) RETURN n;

#Query for a customer
MATCH (n:CUSTOMER {name:  "Bradley Russo"}) RETURN n;

#Query for a product
MATCH (n:PRODUCT) WHERE n.price>200 RETURN n;
```

Neo4j juga mengekspos berbagai REST API untuk melakukan operasi CRUD. REST API ini memungkinkan pengembangan pustaka klien khusus bahasa untuk Neo4j. Mari kita lihat beberapa contoh penggunaan pustaka klien Python Py2neo untuk Neo4j. Box 4.9 menunjukkan contoh penggunaan pustaka klien Py2neo. Dalam contoh ini, kami pertama kali mengotentikasi dengan server Neo4j dengan memberikan nama host server, nama pengguna, dan kata sandi. Selanjutnya, kita membuat instance kelas Graph yang menyediakan metode untuk membuat node dan hubungan, mencari node, menjalankan kueri Cypher, dan berbagai metode lainnya. Untuk mendefinisikan node, kita membuat instance dari kelas Node dengan memberikan label node, nama dan properti. Dalam contoh ini, kami mendefinisikan tiga node (c1, c2, c3) untuk mewakili tiga pelanggan dan dua node (p1, p2) untuk mewakili dua produk. Selanjutnya, kami mendefinisikan hubungan antara node dengan membuat instance dari kelas Relationship. Untuk setiap hubungan, kami menyediakan node terkait, label untuk properti hubungan dan hubungan. Akhirnya, kami menggunakan metode create dari kelas Graph untuk membuat node dan hubungan.

**Box 4.9: Using Python Py2neo client library for Neo4j**

```
import py2neo
from py2neo import Graph, Node, Relationship

# Authenticate the user using py2neo.authentication
py2neo.authenticate("localhost:7474", "neo4j", "password")

# Connect to Graph and get the instance of Graph
graph = Graph("http://localhost:7474/db/data/")

#Define nodes
c1 = Node("CUSTOMER", name="Bradley Russo",
 address="486, 6221 Et St.,Barnstaple",
 country="Ukraine", zipcode="10903")

c2 = Node("CUSTOMER", name="Jarrod Nieves",
 address="198-550 At, Rd.,Hines Creek",
 country="Greece", zipcode="10903")

c3 = Node("CUSTOMER", name="Ivor Merritt",
 address="527-9960 Vel Street,Lauw",
 country="Peru", zipcode="5624")

p1 = Node("PRODUCT",title = "Motorola Moto G (3rd Generation)",
 features = ["Advanced water resistance", "13 MP camera",
 "5in HD display", "Quad core processing power",
 "5MP rear camera", "4G LTE Speed"],
 Color ="Black", Size = "16GB",
 Dimensions = "0.2 x 2.9 x 5.6 inches",
  Weight = "5.4 ounces",price = 219.99)

p2=Node("PRODUCT",title = "Canon EOS Rebel T5",
 features = ["18 megapixel CMOS (APS-C) sensor",
 "EF-S 18-55mm IS II standard zoom lens", "3-inch LCD TFT color,
 liquid-crystal monitor", "EOS 1080p full HD movie mode"],
 Color = "Black", MaximumAperture = "f3.5",
 Dimensions = "3.94 x 3.07 x 5.12 inches",
 Weight = "1.06 pounds", price = 399)

#Define relationships
r1 = Relationship(c1,"ORDERS",p1,date="2015-11-03", quantity="2")
r2 = Relationship(c2,"ORDERS",p1,date="2015-11-03", quantity="1")
r3 = Relationship(c1,"ORDERS",p2,date="2015-11-03", quantity="1")
r4 = Relationship(c2,"ORDERS",p2,date="2015-11-03", quantity="1")

r5 = Relationship(c1,"RATES",p1,rating="4.8")
r6 = Relationship(c2,"RATES",p2,quantity="4.5")

#Create nodes
result = graph.create(c1, c2, p1, p2)

#Create relationships
result = graph.create(r1, r2, r3, r4, r5, r6)

#Print the results
print result
```

Neo4j menyediakan interface web dari mana Anda dapat menjalankan pernyataan Cypher dan melihat grafik di database. Gambar 4.13 menunjukkan grafik yang dibuat menggunakan program Python di Box 4.9.




Gambar 4.13: *Interface web Neo4j menunjukkan node dan hubungan yang dibuat*

| | Key-Value DB | Dokumen DB | Kolom Family DB | Grafik DB |
|---|---|---|---|---|
| Model data | Pasangan keky-value yang unik di identifikasi oleh kunci | Dokumen (yang memiliki key-value pair ), digabungkan pada kolom fmily | Kolom memiliki nama dan nilai, yag digabungkan pada kolom family | Graph comprising pada node dan relationship |
| Kueri | Item kueri menutut kunci, Database spesifik APIs | Dokumen kueri sesuai ID-dokumen, Database spesifik APIs | Query row by the key, Database spesifik APIs | Grafik bahasa kueri : Cypher, Database spesifik APIs |
| Penggunaan | Aplikasi yang melibatkan frekuensi mahasiswaan kecil dan menulis dengan model data simpel | Aplikasi yang melibatkan data pada form dokumen encode dalam format JSON atau XML, dokumen dapat memiliki jumlah yang bervariasi pada atribut | Aplikasi yang melibatkan volume yang besar pada data, high troughput read and write, Ketersediaan permintaan yang tinggi | Aplikasi yang melibatkan data pada entitas dan hubungan antara entitas, data spasial |
| Contoh | DynamoDB, Cassandra | MongoDB, Google BigTable | Hbase, Google BigTable | Neo4j, AlegroGraph |

## Ringkasan

Database non-relasional atau database NoSQL populer untuk aplikasi di mana skala datanya sangat besar dan datanya mungkin tidak terstruktur. Selain itu, kinerja real-time dianggap lebih penting daripada konsistensi. Dalam bab ini kami menjelaskan empat jenis database NoSQL. Gambar 4.14 memberikan perbandingan dari empat jenis database NoSQL ini. Database key-value menyimpan data dalam bentuk key-value pair di mana kunci tersebut digunakan untuk mengidentifikasi nilai yang disimpan secara unik. Fungsi hash diterapkan ke kunci untuk menentukan di mana nilai harus disimpan. Database penyimpanan dokumen menyimpan data semi-terstruktur dalam bentuk dokumen yang dikodekan dalam standar yang berbeda seperti JSON, XML, BSON atau YAML. Manfaat menggunakan database dokumen di atas database key-value adalah bahwa database ini memungkinkan kueri dokumen secara efisien berdasarkan nilai atribut dalam dokumen. Database keluarga kolom menyimpan data sebagai kolom di mana kolom memiliki nama dan nilai. Kolom dikelompokkan ke dalam keluarga kolom dan kumpulan kolom membentuk satu baris yang diidentifikasi dengan kunci baris. Database kelompok kolom mendukung mahasiswaan dan penulisan dengan throughput tinggi dan telah mendistribusikan dan memiliki arsitektur yang sangat tersedia. Grafik model database data berupa node dan relasi. Node merepresentasikan entitas dalam model data dan memiliki sekumpulan atribut. Hubungan antar entitas direpresentasikan dalam bentuk link antar node.


# Implemetasi Analisis Big Data

# Akuisasi Data

Bab 5

Bab ini mencakup

- Pertimbangan Akuisisi Data
- Publikasikan - Berlangganan Kerangka Perpesanan
- Sistem Pengumpulan Big data
- Antrian Pesan
- Konektor Kustom

## Pendahuluan

Di Bab-1, kami mengusulkan big data stack dan berbagai pola analisis. Komponen penting dari kumpulan analisis dan pola yang ada di konektor yang memungkinkan pengumpulan data dari berbagai sumber data ke dalam sistem file terdistribusi atau database NoSQL untuk analisis batch data, atau yang menghubungkan sumber data ke aliran atau kerangka kerja pemrosesan dalam memori untuk analisis data secara real-time.

## 5.1 Pertimbangan Akuisisi Data

Sebelum kita melihat alat dan kerangka kerja khusus untuk akuisisi data dan konektor data, pertama-tama mari kita lihat berbagai pertimbangan untuk akuisisi data, yang akan mengarahkan pilihan alat atau kerangka kerja.

### 5.1.1 Jenis Sumber

Jenis sumber data harus dipertimbangkan saat membuat pilihan untuk konektor data. Sumber data dapat memublikasikan data massal dalam kelompok atau data dalam kelompok kecil (mikro-batch) atau streaming data real-time.
Beberapa contoh sumber data batch adalah:

- File
- Log
- Database relasional

Beberapa contoh sumber data real-time adalah:

- Mesin menghasilkan data sensor

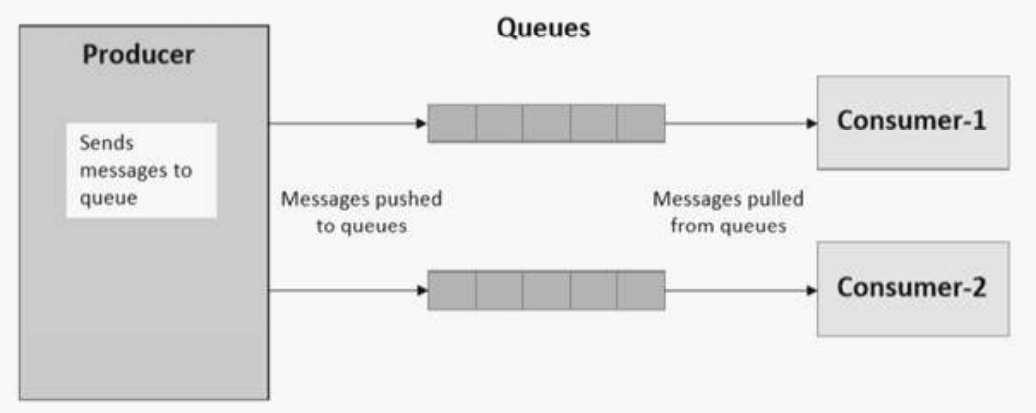
- Sistem Internet of Things (IoT) mengirimkan data real-time
- Umpan media sosial
- Umpan pasar saham

### 5.1.2 Velocity (Kecepatan)

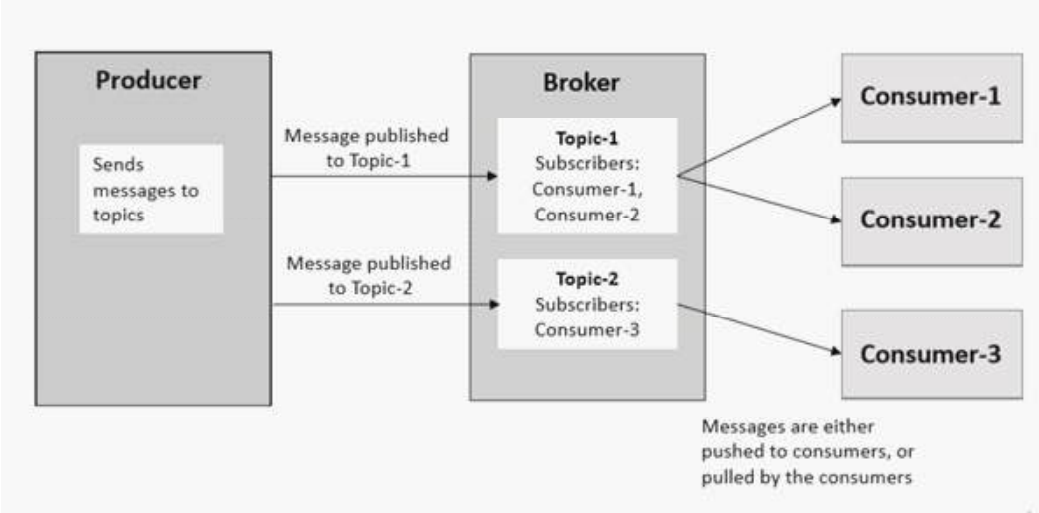
Kecepatan data mengacu pada seberapa cepat data dibuat dan seberapa sering bervariasi. Untuk data dengan kecepatan tinggi (real-time atau streaming data), diperlukan mekanisme komunikasi yang memiliki overhead rendah dan latensi rendah. Nanti di Bab ini, kami menjelaskan bagaimana konektor berbasis WebSocket dan MQTT dapat digunakan untuk menyerap data streaming dan real-time. Untuk aplikasi semacam itu, framework perpesanan publish-subscribe seperti Apache Kafka juga merupakan pilihan yang baik karena mendukung throughput tinggi dan komunikasi latensi rendah. Dengan kerangka kerja seperti itu, konsumen data hilir dapat berlangganan umpan data dan menerima data hampir secara real-time.

### 5.1.3 Mekanisme Ingestion

Mekanisme penyerapan data dapat berupa mekanisme dorong atau tarik. Pilihan alat atau kerangka kerja khusus untuk penyerapan data akan ditentukan oleh konsumen data. Jika konsumen memiliki kemampuan (atau persyaratan) untuk menarik data, publikasikan-langganan kerangka kerja perpesanan yang memungkinkantheconsumerstopullthedata (sepertiApacheKafka) atau antrean pesan dapat digunakan. Produsen data mendorong data ke kerangka pengiriman pesan atau antrean tempat konsumen dapat menarik data. Pendekatan desain alternatif yang diadopsi dalam sistem seperti Apache Flume adalah pendekatan push, di mana sumber data pertama kali mendorong data ke framework dan framework kemudian mendorong data ke data sink. Gambar 5.1 dan 5.2 menunjukkan aliran data dalam pesan push-pull dan publish-subscribe.




Gambar 5.1: *Push-Pull messaging*




Gambar 5.2: *Publish - Subscribe Messaging*

## 5.2 Kerangka Kerja Publish - *Subscribe Messaging*

*Publish-Subscribe* adalah model komunikasi yang terdiri dari penerbit, broker dan konsumen. Penerbit adalah sumber data. Penerbit mengirimkan data ke topik yang dikelola oleh broker. Penerbit tidak sadar akan konsumen. Konsumen berlangganan topik yang dikelola oleh broker. Ketika broker menerima data untuk suatu topik dari penerbit, ia mengirimkan data tersebut ke semua konsumen yang berlangganan. Alternatifnya, konsumen dapat menarik data untuk topik tertentu dari broker.

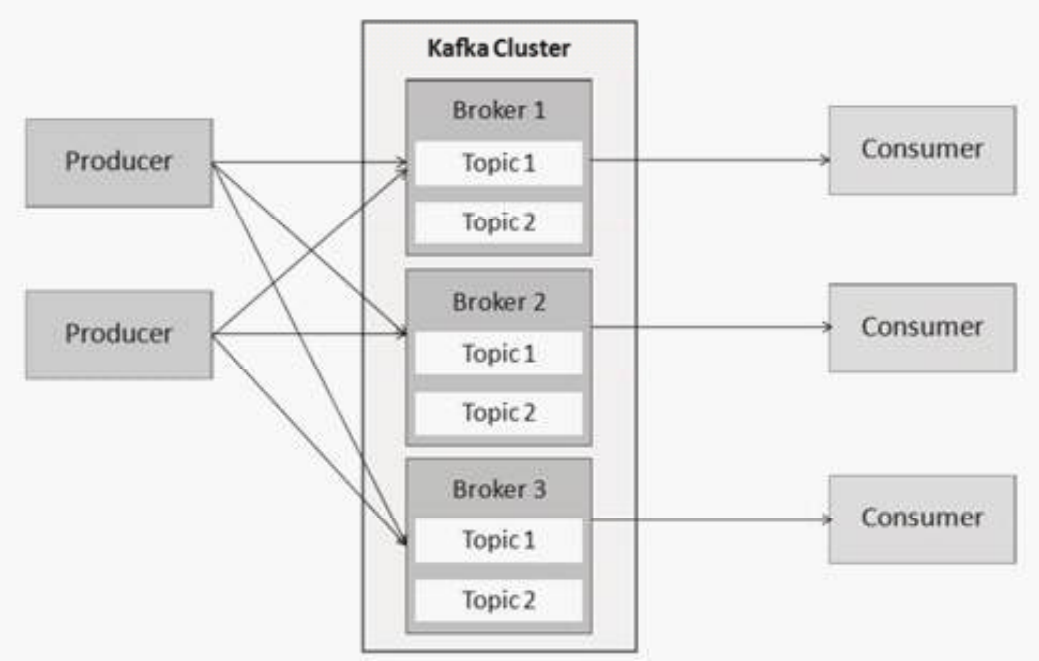
Di bagian ini, kami akan menjelaskan dua kerangka kerja perpesanan langganan-terbitkan - Apache Kafka dan Amazon Kinesis.

### 5.2.1 Apache Kafka

Apache Kafka adalah sistem pesan terdistribusi dengan throughput tinggi. Kafka juga dapat dianggap sebagai layanan log komit terdistribusi, dipartisi, dan direplikasi. Kafka dapat digunakan untuk aplikasi seperti pemrosesan streaming, perpesanan, pelacakan aktivitas situs web, pengumpulan dan pemantauan metrik, agregasi log, dan lain-lain.

#### Arsitektur

Gambar 5.3 menunjukkan arsitektur dan komponen Kafka.




Gambar 5.3 *: Arsitektur Kafka*

Sistem Kafka mencakup komponen-komponen berikut:

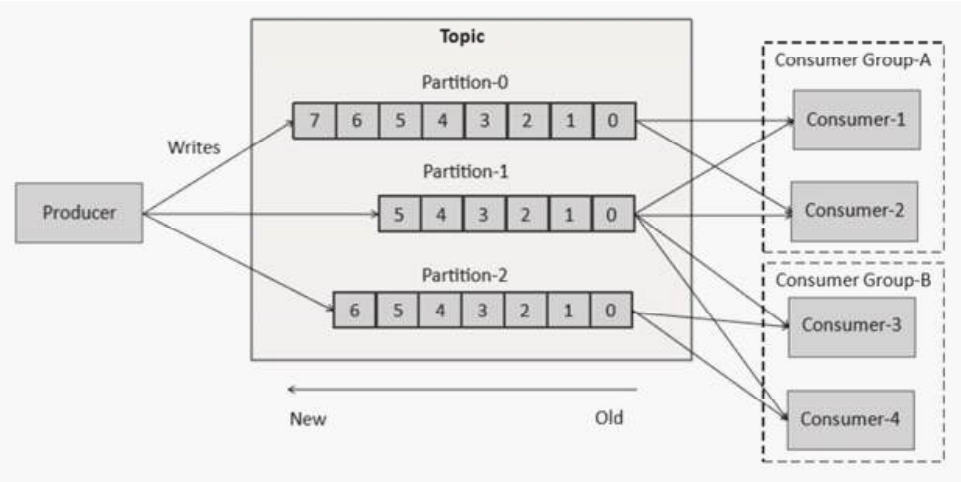
- Topik: Topik adalah kategori yang ditentukan pengguna tempat pesan dipublikasikan.
- Produser: Produser adalah komponen yang menerbitkan pesan ke satu atau beberapa topik.
- Konsumen: Konsumen adalah komponen yang berlangganan satu atau lebih topik dan memproses pesan.
- Broker: Broker adalah komponen yang mengelola topik dan menangani persistensi, partisi, dan replikasi data. Sebuah cluster Kafka dapat memiliki beberapa Broker Kafka (atau server), dengan setiap Broker mengelola banyak topik.

**Partisi**

Topik Kafka dibagi menjadi beberapa partisi seperti yang ditunjukkan pada Gambar 5.4. Setiap partisi adalah urutan pesan yang teratur dan tidak dapat diubah. Topik disimpan di disk dalam bentuk log yang dipartisi (log komit). Manfaat menggunakan partisi adalah bahwa log dapat berskala besar, yang tidak akan muat ke disk dari satu server. Partisi juga memungkinkan banyak konsumen menggunakan pesan secara paralel.

Partisi didistribusikan di antara broker, di mana setiap broker biasanya merupakan server fisik yang terpisah. Partisi juga direplikasi di beberapa broker untuk tujuan toleransi

kesalahan. Untuk setiap partisi, salah satu server bertindak sebagai 'pemimpin' untuk partisi dan menangani semua penulisan dan mahasiswaan partisi. Server lain yang menyimpan replika partisi disebut 'pengikut'. Partisi dan replika dapat dialihkan antar broker karena lebih banyak broker tersedia.




Gambar 5.4: *Topik Partisi Kafka*

### *Publishing Message*

Produsen mempublikasikan pesan ke topik. Produsen memutuskan pesan mana yang harus dipublikasikan ke partisi topik mana. Produsen dapat mempublikasikan pesan ke partisi yang berbeda dari sebuah topik secara round-robin (untuk tujuan load balancing) atau mempublikasikan pesan dengan kunci spesifik ke partisi tertentu.

### *Consuming Message*

Konsumen menggunakan (dan kemudian memproses) pesan dari topik. Setiap pesan di partisi diberi ID urutan yang disebut offset. Offset digunakan oleh konsumen untuk melacak pesan mana yang telah dikonsumsi. Kafka Broker tidak bertanggung jawab untuk melacak pesan yang dikonsumsi oleh konsumen. Pesan yang diterbitkan disimpan di disk untuk durasi waktu yang dapat dikonfigurasi. Karena topik disimpan pada disk, konsumen dapat memutar ulang pesan menggunakan offset. Konsumen menambah offset saat mereka menggunakan pesan secara berurutan.

Konsumen dapat dikelompokkan menjadi beberapa kelompok konsumen. Setiap pesan yang diterbitkan ke topik oleh produsen dikirimkan ke satu konsumen dalam kelompok konsumen. Konsumen dalam kelompok konsumen dapat berupa proses terpisah atau contoh terpisah. Memiliki banyak konsumen dalam satu kelompok konsumen membuat

sistem dapat diskalakan dan toleran terhadap kesalahan. Partisi dalam topik ditetapkan ke konsumen sehingga setiap partisi dikonsumsi oleh satu konsumen dalam sebuah grup konsumen. Proses konsumen individu dapat memproses berbagai partisi secara paralel. Karena hanya satu proses konsumen yang mengkonsumsi dari partisi tertentu, pesan dikirim secara berurutan. Kafka menyediakan pengurutan pesan dalam sebuah partisi, tetapi tidak di antara partisi.

### *Log Storage* & Pemadatan

Kafka disusun untuk menyimpan pesan dalam bentuk log khusus lampiran. Untuk setiap partisi dari setiap topik, file log dipertahankan yang berisi urutan pesan yang teratur dan tidak dapat diubah. Pesan tersedia untuk konsumen hanya setelah mereka berkomitmen pada log. Hal ini memastikan bahwa semua pesan yang dikonsumsi oleh konsumen tetap ada di dalam log sehingga dapat dikonsumsi oleh konsumen lain jika diperlukan. Tidak seperti, sistem antrian yang menghapus pesan setelah dikonsumsi, Kafka menyimpan pesan dalam file log, yang disimpan selama jangka waktu tertentu (yang dapat dikonfigurasi). Kafka menyediakan dua opsi untuk kebijakan pembersihan log - hapus atau ringkas. Log untuk partisi topik disimpan sebagai direktori file segmen. Ukuran maksimum yang dapat dikembangkan oleh file segmen sebelum segmen baru di-roll over di log dapat dikonfigurasi. Jika kebijakan pembersihan log disetel ke hapus, segmen log dihapus saat mencapai ukuran atau batas waktu yang ditetapkan. Jika kebijakan pembersihan log disetel ke padat, maka pemadatan log digunakan untuk membersihkan catatan usang. Sementara untuk data temporal kebijakan penyimpanan penghapusan berfungsi dengan baik, kebijakan kompak berguna ketika Anda memiliki data yang dikunci dan dapat diubah. Misalnya, jika setiap pesan yang diterbitkan ke topik secara unik diidentifikasi oleh kunci utama (seperti baris dalam tabel database), dan nilai pesan akan berubah seiring waktu, maka lebih baik menggunakan pemadatan log yang menghapus nilai lama dan usang untuk sebuah kunci. Pemadatan log memastikan bahwa setidaknya nilai terakhir yang diketahui untuk setiap kunci pesan untuk setiap partisi topik dipertahankan.

### Menggunakan Kafka

Pada bagian ini, kami akan menjelaskan beberapa contoh produsen dan konsumen Kafka. Kami akan menggunakan pustaka python Kafka yang disebut kafka-python untuk contohnya. Pustaka kafka-python dapat diinstal menggunakan pip sebagai berikut:

```
sudo pip install kafka-python
```

Kafka menggunakan Zookeeper untuk koordinasi negara. Topik Kafka dapat dibuat sebagai berikut:

```
■ bin/kafka-topics.sh --create --zookeeper localhost:2181
--replication-factor 1 --partitions 1 --topic test
```

Jika Zookeeper dijalankan pada mesin yang terpisah dari Kafka, ubah nama host Zookeeper dari localhost menjadi nama host yang benar.
Untuk memastikan bahwa topik telah dibuat, Anda dapat menggunakan perintah berikut untuk mencantumkan semua topik:

```
■ bin/kafka-topics.sh --list --zookeeper localhost:2181
```

Box 5.1 menunjukkan contoh produser Kafka yang mengirim pesan secara sinkron.

**■ Box 5.1: Kafka Producer for sending messages synchronously**

```
import time
from datetime import datetime
from kafka.client import KafkaClient
from kafka.producer import SimpleProducer

client = KafkaClient("localhost:6667")
producer = SimpleProducer(client)

while True:
   ts=time.time()
   timestamp = datetime.fromtimestamp(ts).strftime('%Y-%m-%d %H:%M:%S')
   data = "This is a test string generated at:  " + str(timestamp)

   producer.send_messages('test', data)

   time.sleep(1)
```

Box 5.2 menunjukkan contoh produser Kafka yang mengirim pesan secara asynchronous.

**■ Box 5.2: Kafka Producer for sending messages asynchronously**

```
import time
from datetime import datetime
from kafka.client import KafkaClient
from kafka.producer import SimpleProducer
```

```
client = KafkaClient("localhost:6667")
producer = SimpleProducer(client, async=True)

while True:
   ts=time.time()
   timestamp = datetime.fromtimestamp(ts).strftime('%Y-%m-%d %H:%M:%S')
   data = "Test message sent asynchronously at:  " + str(timestamp)

   producer.send_messages('test', data)

   time.sleep(1)
```

Box 5.3 menunjukkan contoh produsen Kafka yang mengirim pesan secara berkelompok. Produser mengumpulkan pesan dan mengirim pesan setelah 20 pesan dikumpulkan atau setelah setiap 60 detik.

**Box 5.3: Kafka Producer for sending messages in batch**

```
import time
from datetime import datetime
from kafka.client import KafkaClient
from kafka.producer import SimpleProducer

client = KafkaClient("localhost:6667")

#The following producer will collect messages in batch
#and send them to Kafka after 20 messages are
# collected or every 60 seconds

producer = SimpleProducer(client,batch_send=True,
        batch_send_every_n=20,
        batch_send_every_t=60)

while True:
   ts=time.time()
   timestamp = datetime.fromtimestamp(ts).strftime('%Y-%m-%d %H:%M:%S')
   data = "This is a test string generated at:  " + str(timestamp)

   producer.send_messages('test', data)

   time.sleep(1)
```

Box 5.4 menunjukkan contoh produser Kafka yang mengirimkan pesan-pesan kunci. Kelas pemartisi default untuk mempartisi pesan adalah Hashed Partitioner yang membuat partisi berdasarkan hash kunci. Pilihan lainnya adalah menggunakan Partisi Round Robin yang mengirimkan pesan ke partisi yang berbeda dengan cara round-robin.

**Box 5.4: Kafka Producer for sending keyed messages**

```
import time
import datetime
from kafka import KafkaClient, KeyedProducer,
from kafka import HashedPartitioner, RoundRobinPartitioner

kafka = KafkaClient("localhost:6667")

#Default partitioner is HashedPartitioner
producer = KeyedProducer(kafka)
producer.send("test", "key1", "Test message with key1")
producer.send("test", "key2", "Test message with key2")

#Using RoundRobinPartitioner
producer = KeyedProducer(kafka, partitioner=RoundRobinPartitioner)
producer.send("test", "key3", "Test message with key3")
producer.send("test", "key4", "Test message with key4")
```

Box 5.5 menunjukkan contoh konsumen Kafka yang mengkonsumsi pesan dari suatu topik.

**Box 5.5: Kafka Consumer**

```
from kafka.client import KafkaClient
from kafka.consumer import SimpleConsumer

client = KafkaClient("localhost:6667")
consumer = SimpleConsumer(client, "test-group", "test")

for message in consumer:
   #Print message object
   print message

   #Print only message value
   print message.message.value
```

Box 5.6 menunjukkan implementasi alternatif dari konsumen Kafka.

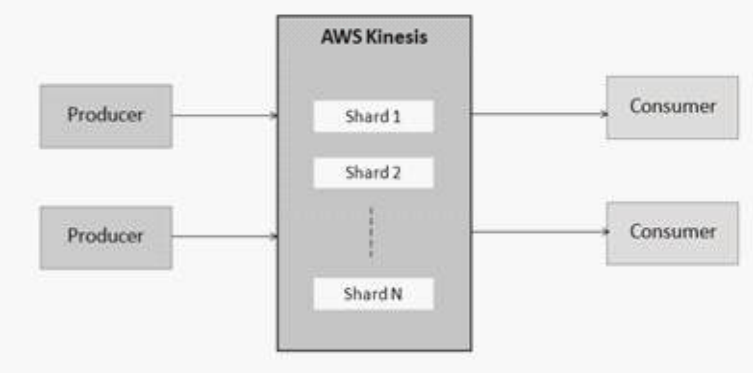
**Box 5.6: Kafka Consumer - alternative implementation**

```
from kafka.client import KafkaClient
from kafka.consumer import KafkaConsumer

client = KafkaClient("localhost:6667")
consumer = KafkaConsumer("test", metadata_broker_list=['localhost:6667'])

while True:
   data = consumer.next().value
   print data
```

### 5.2.2 Amazon Kinesis

Amazon Kinesis adalah layanan komersial yang dikelola sepenuhnya untuk menyerap data streaming real-time. Kinesis menskalakan secara otomatis untuk menangani data streaming volume tinggi yang berasal dari banyak sumber. Data streaming yang dikumpulkan oleh Kinesis dapat diproses oleh aplikasi yang berjalan pada instans Amazon EC2 atau instans komputasi lainnya yang dapat terhubung ke Kinesis.

Kinesis memungkinkan pemasukan data yang cepat dan berkelanjutan serta mendukung blob data dengan ukuran hingga 50Kb. Produsen data menulis catatan data ke aliran Kinesis. Rekaman data terdiri dari nomor urut, kunci partisi, dan blob data. Catatan data di aliran Kinesis didistribusikan dalam pecahan. Setiap pecahan menyediakan unit kapasitas tetap, dan aliran dapat memiliki beberapa pecahan. Satu keping throughput memungkinkan pengambilan data 1MB per detik, hingga 1.000 transaksi PUT per detik dan memungkinkan aplikasi membaca data hingga 2 MB per detik.




Gambar 5.5: *Rancangan Amazon Kinesis*

Box 5.7 menunjukkan program Python untuk menulis ke aliran Kinesis. Dalam contoh ini, koneksi ke layanan Kinesis pertama kali dibuat dan kemudian aliran Kinesis baru dibuat (jika tidak ada) atau dijelaskan. Data ditulis ke aliran Kinesis menggunakan fungsi kinesis.put_record.

**Box 5.7: Python program for writing to a Kinesis stream**

```
from random import randrange
import time
import datetime
import boto
import json
from boto.kinesis.exceptions import ResourceNotFoundException
```

```
ACCESS_KEY = "<enter>"
SECRET_KEY = "<enter>"

kinesis = boto.connect_kinesis(aws_access_key_id=ACCESS_KEY,
   aws_secret_access_key=SECRET_KEY)

#Send some synthetic data to AWS Kinesis
while True:
 ts=time.time()

 data = str(ts) + ',' + str(randrange(0,60)) + ',' +
   str(randrange(0,100)) + ',' + str(randrange(5000,12000)) +
    ',' + str(randrange(0,100))

 print data
 response = kinesis.put_record('forestfire', data, data)
 print response
 time.sleep(1)

 data = str(ts) + ',' + str(randrange(0,60)) + ',' +
   str(randrange(0,100)) + ',' + str(randrange(5000,12000)) +
    ',' + str(randrange(0,100))

 print data
 response = kinesis.put_record('forestfire', data, data)
 print response
 time.sleep(1)
```

Box 5.8 menunjukkan program Python untuk membaca dari aliran Kinesis. Dalam contoh ini, iterator shard diperoleh menggunakan fungsi kinesis.get_shard_iterator. Iterator shard menentukan posisi dalam pecahan tempat Anda ingin mulai membaca rekaman data secara berurutan. Data dibaca menggunakan fungsi kinesis.get_records yang mengembalikan satu atau lebih record data dari sebuah shard.

**■ Box 5.8: Python program for reading from a Kinesis stream**

```
from random import randrange
import time
import datetime
import boto
import json
from boto.kinesis.exceptions import ResourceNotFoundException

ACCESS_KEY = "<enter>"
SECRET_KEY = "<enter>"
```

```
kinesis = boto.connect_kinesis(aws_access_key_id=ACCESS_KEY,
   aws_secret_access_key=SECRET_KEY)

response = kinesis.describe_stream('forestfire')
```

```
if response['StreamDescription']['StreamStatus'] == 'ACTIVE':
 shard_id = response['StreamDescription']['Shards'][0]['ShardId']

response = kinesis.get_shard_iterator('forestfire', shard_id,
'TRIM_HORIZON')
shard_iterator = response['ShardIterator']

response = kinesis.get_records(shard_iterator)
shard_iterator = response['NextShardIterator']

for record in response['Records']:
 print record
```

## 5.3 Sistem Pengumpulan Big Data

Sistem pengumpulan data memungkinkan pengumpulan, agregasi, dan pemindahan data dari berbagai sumber (seperti log server, database, media sosial, data sensor streaming dari perangkat Internet of Things dan sumber lain) ke dalam penyimpanan data terpusat (seperti sistem file terdistribusi atau Database NoSQL).

### 5.3.1 Apache Flume

Apache Flume adalah sistem terdistribusi, andal, dan tersedia untuk mengumpulkan, menggabungkan, dan memindahkan sejumlah besar data dari sumber data yang berbeda ke dalam penyimpanan data terpusat.

#### Rancangan Flume

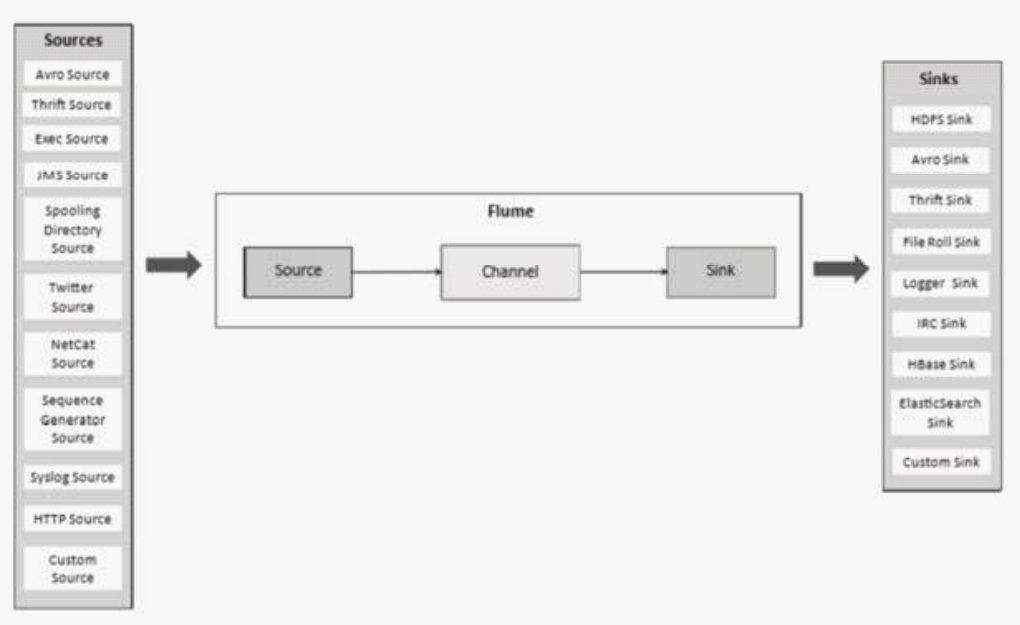
Arsitektur Flume didasarkan pada aliran data dan mencakup komponen berikut:

- **Sumber**
  Sumber adalah komponen yang menerima atau mengumpulkan data dari sumber eksternal. Aliran data Flume dimulai dari sumber. Misalnya, sumber Flume dapat menerima data dari jaringan media sosial (menggunakan API streaming).
- **Channel**
  Setelah data diterima oleh sumber Flume, data tersebut dikirim ke channel. Setiap channel dalam aliran data terhubung ke satu bak tempat data dikeringkan. Aliran data dapat terdiri dari beberapa channel, di mana sumber menulis data ke beberapa channel.
- **Sink**
  Sink adalah komponen yang mengalirkan data dari channel ke penyimpanan data (seperti sistem file terdistribusi atau ke Agent lain). Tiap sink dalam aliran data

dihubungkan ke channel. Sinks mengirimkan data ke tujuan akhirnya atau dirantai ke Agent lain.

- **Agent**
  Agent Flume adalah kumpulan sumber, channel, dan sink. Agent adalah proses yang menampung sumber, channel, dan penyerap tempat data berpindah dari sumber eksternal ke tujuan akhirnya.
- **Peristiwa**
  Peristiwa adalah unit aliran data yang memiliki muatan dan sekumpulan atribut opsional. Sumber flume mengkonsumsi peristiwa yang dihasilkan oleh sumber eksternal.

Flume menggunakan model aliran data yang mencakup sumber, channel, dan sink, yang dikemas menjadi Agent. Gambar 5.7 menunjukkan beberapa contoh aliran data Flume. Aliran data yang paling sederhana memiliki satu sumber, satu channel, dan satu sink. Sumber dapat membuat multipleks data ke beberapa channel untuk tujuan load balancing, atau, untuk pemrosesan paralel. Aliran data yang lebih kompleks dapat dibuat dengan merangkai beberapa Agent di mana sink dari satu Agent mengirimkan data ke sumber Agent lain.

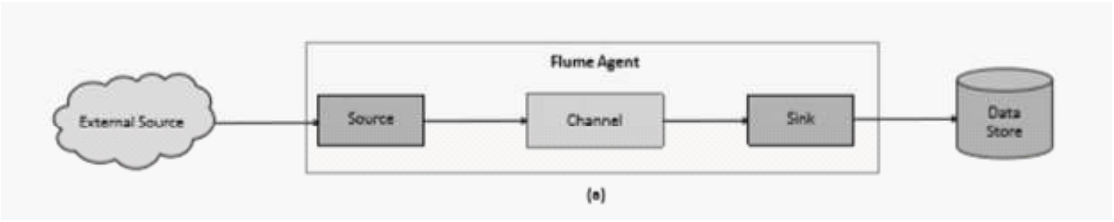



Gambar 5.6: *Arsitektur Apache Flume*

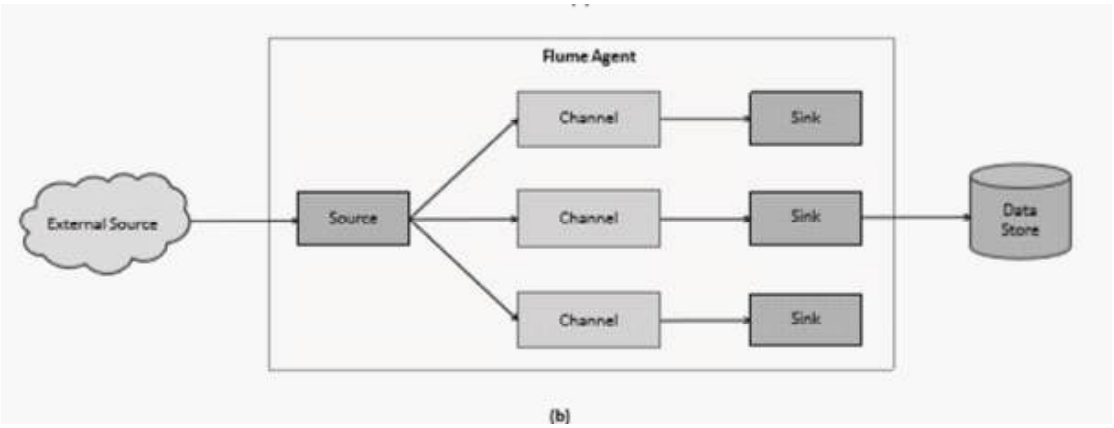
Agent flume didefinisikan dalam file konfigurasi. Box 5.9 menunjukkan definisi umum dari Agent Flume. Dalam file konfigurasi, pertama-tama sumber, channel, dan sink untuk Agent dicantumkan, lalu masing-masing sumber, channel, dan sink didefinisikan. Terakhir, binding antara sumber, channel, dan sink didefinisikan.

**Box 5.9: Generic definition of a Flume agent**

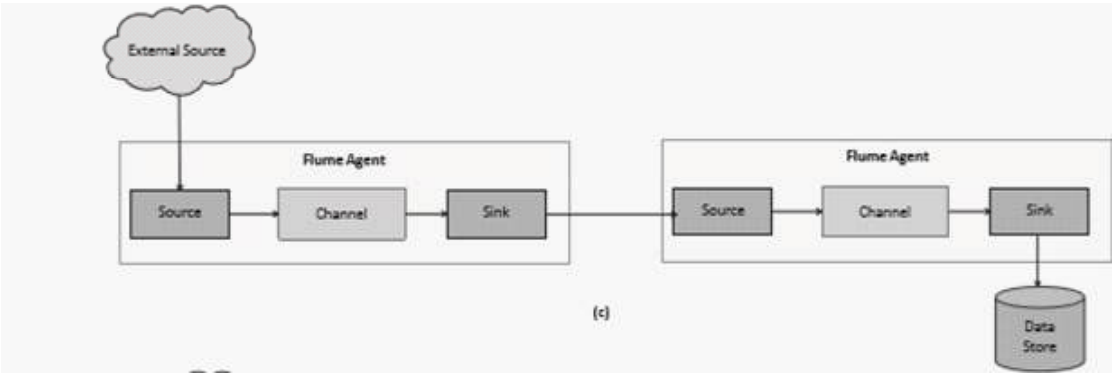
```
<agent name>.sources = <source-1> <source-2> ...  <source-N>
<agent name>.channels = <channel-1> <channel-2> ...  <channel-N>
<agent name>.sinks = <sink-1> <sink-2> ...  <sink-N>

# Define sources
<agent name>.sources.<source-1>.type = <source type>
:
<agent name>.sources.<source-N>.type = <source type>

# Define sinks
<agent name>.sinks.<sink-1>.type = <sink type>
:
<agent name>.sinks.<sink-1>.type = <sink type>


# Define channels
```



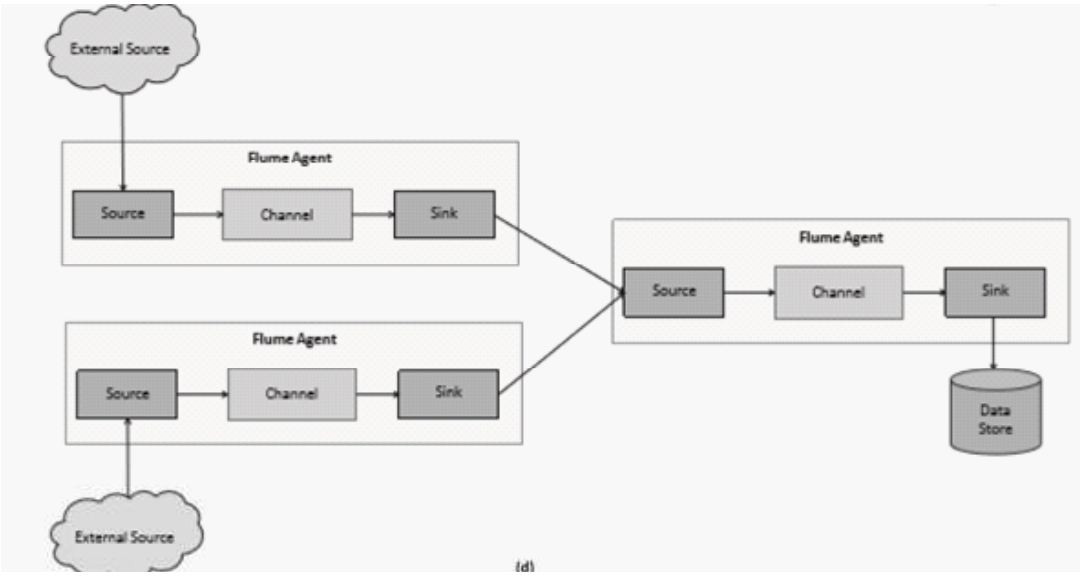
(a)



(b)



(c)




Gambar 5.7: *Contoh Flume data flow*

```
myagent.channels.<channel-1>.type = <channel type>
:
myagent.channels.<channel-N>.type = <channel type>

# Bind the sources and sinks to the channels
myagent.sources.<source-1>.channels = <channel-1>
myagent.sinks.<sink-1>.channel = <channel-1
:
myagent.sources.<source-N>.channels = <channel-1> ...  <channel-N>
myagent.sinks.<sink-N>.channel = <channel-N
```

```
■ #Format of command to run a Flume agent
#sudo flume-ng agent -c <conf file path> -f <conf file> -n <agent name>

#Example
sudo flume-ng agent -c /etc/flume/conf -f
     /etc/flume/conf/flume.conf -n myagent
```

### Flume Source

Flume hadir dengan beberapa sumber bawaan yang memungkinkan pengumpulan dan penggabungan data dari berbagai sistem eksternal. Flume juga memberikan fleksibilitas untuk menambahkan sumber kustom.

- **Avro Source**

  Apache Avro adalah sistem serialisasi data yang menyediakan format data biner yang ringkas dan cepat. Avro menggunakan Interface Definition Language (IDL) untuk mendefinisikan struktur data dalam bentuk skema. Avro didefinisikan dengan JSON,

dan skema selalu disimpan dengan data, yang memungkinkan program membaca data untuk menafsirkan data. Avro juga dapat digunakan dengan Panggilan Prosedur Jarak Jauh (RPC) di mana klien dan server bertukar skema dalam proses berjabat tangan. Avro menyediakan fungsionalitas serialisasi yang mirip dengan sistem lain seperti Thrift dan Protocol Buffer. Sumber Flume Avro menerima peristiwa dari aliran klien Avro eksternal. Sumber Avro dapat diatur menggunakan properti berikut di file konfigurasi Flume untuk Agent:

```
■ myagent.sources = sourcel
myagent.sources.sourcel.type = avro
myagent.sources.sourcel.bind = 0.0.0.0
myagent.sources.sourcel.port = 4141
```

Properti bind *dan port menentukan nama host klien Avro eksternal dan port Avro.*

- **Thrift Source**

  Apache Thrift adalah framework serialisasi yang mirip dengan Avro. Thrift menyediakan tumpukan perangkat lunak dan mesin pembuat kode untuk membangun layanan yang bekerja secara transparan dan efisien dengan berbagai bahasa pemrograman. Seperti Avro, Thrift juga menyediakan tumpukan untuk Panggilan Prosedur Jarak Jauh (RPC). Sumber Flume Thrift menerima peristiwa dari stream klien Thrift eksternal. Sumber barang bekas dapat diatur menggunakan properti berikut di file konfigurasi Flume untuk Agent:

```
■ myagent.sources = sourcel
myagent.sources.sourcel.type = thrift
```

```
myagent.sources.sourcel.bind = 0.0.0.0
myagent.sources.sourcel.port = 4141
```

- **Exec Source**

  Exec source dapat digunakan untuk menyerap data dari output standar. Ketika Agent dengan sumber Exec dimulai, ia menjalankan perintah Unix (ditentukan dalam definisi sumber Exec) dan terus menerima data dari output standar selama proses berjalan. Kasus penggunaan tipikal untuk sumber Exec adalah perintah tail yang memancarkan beberapa baris dari setiap file teks yang diberikan kepadanya sebagai masukan dan menulisnya ke output standar. Tail saat digunakan dengan opsi -F akan mengeluarkan data yang ditambahkan saat file bertambah. Box di bawah ini menunjukkan contoh pengaturan sumber Exec dengan perintah tail.

```
■ myagent.sources = source1
myagent.sources.source1.type = exec
myagent.sources.source1.command = tail -F /var/log/eventlog.log
```

- **JMS Source**

  Java Message Service (JMS) adalah layanan pengiriman pesan yang dapat digunakan oleh aplikasi Java untuk membuat, mengirim, menerima, dan membaca pesan. Sumber JMS menerima pesan dari antrian atau topik JMS. Box di bawah ini menunjukkan contoh penyiapan sumber JMS:

```
■ myagent.sources = s1
myagent.sources.s1.type = jms
myagent.sources.s1.initialContextFactory =
      org.apache.activemq.jndi.ActiveMQInitialContextFactory
myagent.sources.s1.connectionFactory = GenericConnectionFactory
myagent.sources.s1.providerURL = tcp://mqserver:61616
myagent.sources.s1.destinationName = DATA
myagent.sources.s1.destinationType = QUEUE
```

  Untuk terhubung dengan tujuan JMS, diperlukan nama pabrik konteks awal, pabrik koneksi, dan URL penyedia. Jenis tujuan bisa berupa antrian atau topik.

- ***Spooling Directory Source***

  *Spooling Directory Source* berguna untuk menelan file log. Direktori spool disiapkan pada disk dari mana *Spooling Directory Source* mencerna file. Untuk menggunakan sumber untuk menelan log, sistem pembuatan log diatur sedemikian rupa sehingga ketika file-file log digulung, mereka akan dipindahkan ke direktori spool. Spooling Directory Source mengurai file dan membuat acara. Logika parsing dapat dikonfigurasi untuk sumbernya. Logika defaultnya adalah mengurai setiap baris sebagai peristiwa. Meskipun pendekatan alternatif untuk *Spooling Directory Source* adalah dengan menggunakan sumber Exec dengan perintah tail, bagaimanapun, itu tidak dapat diandalkan. Box di bawah ini menunjukkan contoh menyiapkan *Spooling Directory Source* :

```
■ myagent.sources = source1
myagent.sources.source1.type = spooldir
myagent.sources.source1.spoolDir = /var/log/apache/flumeSpool
myagent.sources.source1.fileHeader = true
```

- **Twitter Source**

  Sumber Twitter Flume terhubung ke Twitter streaming API dan menerima tweet secara real-time. Sumber Twitter mengonversi objek tweet ke format Avro sebelum

mengirimnya ke channel hilir. Box di bawah ini menunjukkan contoh pengaturan sumber Twitter:

```
■ myagent.sources = source1
myagent.sources.source1.type =
      org.apache.flume.source.twitter.TwitterSource
myagent.sources.source1.consumerKey = CONSUMER_KEY
myagent.sources.source1.consumerSecret = CONSUMER_SECRET
myagent.sources.source1.accessToken = ACCESS_TOKEN
myagent.sources.source1.accessTokenSecret = ACCESS_TOKEN_SECRET
myagent.sources.source1.maxBatchSize = 10
myagent.sources.source1.maxBatchDurationMillis = 200
```

Sebelum menyiapkan sumber Twitter, Anda perlu membuat aplikasi Twitter dari akun pengembang Twitter dan mendapatkan konsumen serta mengakses token dan rahasia untuk aplikasi tersebut.

- **NetCat Source**

  NetCat adalah utilitas Unix sederhana yang membaca dan menulis data melalui koneksi jaringan, menggunakan protokol TCP atau UDP. Sumber NetCat mendengarkan port spesifik yang datanya ditulis oleh klien NetCat dan mengubah setiap baris teks yang diterima menjadi peristiwa Flume. Box di bawah ini menunjukkan contoh pengaturan sumber NetCat:

```
■ myagent.sources = source1
myagent.sources.source1.type = netcat
myagent.sources.source1.bind = 0.0.0.0
myagent.sources.source1.port = 6666
```

- ***Sequence Generator Source***

  *Sequence Generator Source* menghasilkan peristiwa dengan urutan angka mulai dari 0 dan bertambah 1. Sumber ini terutama digunakan untuk tujuan pengujian. Box di bawah ini menunjukkan contoh menyiapkan sumber Generator Urutan:

```
■ myagent.sources = source1
myagent.sources.source1.type = seq
```

- ***Syslog Source***

  *Sumber syslog* digunakan untuk menelan data syslog. Box di bawah ini menunjukkan contoh penyiapan sumber Syslog TCP.

```
■ myagent.sources = source1
myagent.sources.source1.type = syslogtcp
myagent.sources.source1.host = localhost
myagent.sources.source1.port = 5140
```

- **HTTP Source**

  Sumber HTTP menerima peristiwa HTTP (permintaan POST atau GET) dan mengubahnya menjadi peristiwa Flume. Meskipun sumber dapat menerima peristiwa dalam bentuk permintaan HTTP POST dan GET, perintah GET hanya digunakan untuk eksperimen. Untuk mengubah permintaan HTTP menjadi acara, penangan yang dapat dicolok digunakan. Penangan defaultnya adalah JSONHandler, yang mengharapkan larik objek JSON. Box di bawah ini menunjukkan contoh penyiapan sumber HTTP:

```
myagent.sources = source1
myagent.sources.source1.type = http
myagent.sources.source1.bind = localhost
myagent.sources.source1.port = 81
myagent.sources.source1.handler =
      org.apache.flume.source.http.JSONHandler
```

  Properti bind dan port menentukan nama host dan port tempat sumber harus mendengarkan.

- **Custome Source**

  Flume memungkinkan sumber bea cukai diintegrasikan ke dalam sistem. Sumber khusus diterapkan di Java. File kelas Java dari sumber kustom bersama dengan dependensi disertakan dalam classpath Agent Flume dan juga ditentukan dalam file konfigurasi Agent yang ditunjukkan di bawah ini:

```
myagent.sources = source1
myagent.sources.source1.type = org.example.MySource
```

***Flum Sink***

Flume dilengkapi dengan beberapa bak cuci built-in. Setiap sink di Agent Flume terhubung ke channel dan mengalirkan data dari channel ke penyimpanan data.

- **HDFS Sink**

  Hadoop Distributed File System (HDFS) Sink mengalirkan peristiwa dari channel ke HDFS. Data ditulis ke HDFS dalam bentuk tipe file yang dapat dikonfigurasi. HDFS sink mendukung jenis file SequenceFile, DataStream dan CompressedStream. HDFSsinkallowsthefile untuk mengumpulkan apakah ukuran file melebihi batas tertentu, atau setelah interval yang ditentukan, atau setelah sejumlah peristiwa telah ditulis ke file. Box di bawah ini menunjukkan contoh menyiapkan sink HDFS:

```
■ myagent.sinks = sink1
myagent.sinks.sink1.type = hdfs
myagent.sinks.sink1.hdfs.fileType = DataStream
myagent.sinks.sink1.hdfs.path = /flume/events
myagent.sinks.sink1.hdfs.filePrefix = eventlog
myagent.sinks.sink1.hdfs.fileSuffix = .log
myagent.sinks.sink1.hdfs.batchSize = 1000
```

- **Avro Sink**

  Avro sink mengambil peristiwa dari channel dan mengalirkan peristiwa ke host hilir. Box di bawah ini menunjukkan contoh menyiapkan Avro sink:

```
■ myagent.sinks = sink1
myagent.sinks.sink1.type = avro
myagent.sinks.sink1.hostname = 10.10.10.10
myagent.sinks.sink1.port = 4545
```

- **Thrift Sink**

  Thrift sink mengambil peristiwa dari channel dan mengosongkan peristiwa ke host hilir. Box di bawah ini menunjukkan contoh penyiapan Thrift sink:

```
■ myagent.sinks = sink1
myagent.sinks.sink1.type = thrift
myagent.sinks.sink1.hostname = 10.10.10.10
myagent.sinks.sink1.port = 4545
```

- **File Roll Sink**

  File Roll Sink mengalirkan kejadian ke file di sistem file lokal. Box di bawah ini menunjukkan contoh menyiapkan sink File Roll:

```
■ myagent.sinks = sink1
myagent.sinks.sink1.type = file_roll
myagent.sinks.sink1.sink.directory = /var/log/flume
```

- **Logger Sink**

  Logger Sink mengambil peristiwa dari channel dan mencatat peristiwa. Box di bawah ini menunjukkan contoh menyiapkan Logger sink:

```
■ myagent.sinks = sink1
myagent.sinks.sink1.type = logger
```

- **IRC Sink**

  IRC sink mengambil peristiwa dari channel dan mengalirkan peristiwa ke IRChost. Box di bawah ini menunjukkan contoh pengaturan IRC Sink :

```
■ myagent.sinks = sink1
myagent.sinks.sink1.type = irc
myagent.sinks.sink1.hostname = irc.example.com
myagent.sinks.sink1.nick = flume
myagent.sinks.sink1.chan = #flume
```

- **Hbase Sink**

  Hbase Sink mengambil peristiwa dari channel dan mengalirkan peristiwa ke tabel HBase. Box di bawah ini menunjukkan contoh pemasangan wastafel HBase:

```
■ myagent.sinks = sink1
myagent.sinks.sink1.type = hbase
myagent.sinks.sink1.table = mytable
myagent.sinks.sink1.columnFamily = myfam
myagent.sinks.sink1.serializer =
     org.apache.flume.sink.hbase.RegexHbaseEventSerializer
```

- **Custom Sink**

  Flume memungkinkan sink pabean diintegrasikan ke dalam sistem. Sink kustom diterapkan di Java. File kelas Java dari custom sink bersama dengan dependensi termasuk dalam classpath Agent Flume dan juga ditentukan dalam file konfigurasi Agent yang ditunjukkan di bawah ini:

```
■ myagent.sinks = sink1
myagent.sinks.sink1.type = org.example.MySink
```

***Flum Channel***

Channel menyimpan acara saat dipindahkan dari sumber ke wastafel.

- **Channel Memory**

  Channel Memory menyimpan peristiwa dalam memori dan memberikan throughput yang tinggi. Namun, jika terjadi kegagalan Agent, peristiwa tersebut bisa hilang. Box di bawah ini menunjukkan contoh pengaturan channel memori.

```
■ myagent.channels = channel1
myagent.channels.channel1.type = memory
myagent.channels.channel1.capacity = 10000
myagent.channels.channel1.transactionCapacity = 10000
myagent.channels.channel1.byteCapacityBufferPercentage = 20
myagent.channels.channel1.byteCapacity = 800000
```

- **File Channel**

  File channel menyimpan file kejadian di sistem file lokal. Acara disimpan dalam file pos pemeriksaan di direktori data yang ditentukan dalam konfigurasi channel. Ukuran file maksimum untuk file check point dapat ditentukan. Box di bawah ini menunjukkan contoh pengaturan channel file

```
■ myagent.channels = channel1
myagent.channels.channel1.type = file
myagent.channels.channel1.checkpointDir = /mnt/flume/checkpoint
myagent.channels.channel1.dataDirs = /mnt/flume/data
```

- **JDBC Channel**

  JDBC Channel menyimpan acara dalam database Derby tertanam. Channel ini menyediakan penyimpanan yang tahan lama untuk acara, dan acara dapat dipulihkan dengan mudah jika terjadi kegagalan Agent. Box di bawah ini menunjukkan contoh pengaturan channel JDBC.

```
■ myagent.channels = channel1
myagent.channels.channel1.type = jdbc
```

- **Spillable Memory Channel**

  Spillable Memory Channel menyimpan peristiwa dalam antrian dalam memori dan ketika antrian terisi, peristiwa tersebut ditumpahkan ke disk. Channel ini memberikan throughput yang tinggi dan toleransi kesalahan. Box di bawah ini menunjukkan contoh pengaturan channel Spillable Memory

```
■ myagent.channels = channel1
myagent.channels.channel1.type = SPILLABLEMEMORY
myagent.channels.channel1.memoryCapacity = 10000
myagent.channels.channel1.overflowCapacity = 1000000
myagent.channels.channel1.byteCapacity = 800000
myagent.channels.channel1.checkpointDir = /mnt/flume/checkpoint
myagent.channels.channel1.dataDirs = /mnt/flume/data
```

  Jumlah maksimum kejadian yang disimpan dalam antrian memori ditentukan menggunakan properti memoryCapacity dan ukuran maksimum antrian memori ditentukan menggunakan properti byteCapacity. Antrian dalam memori dianggap penuh, dan peristiwa ditumpahkan ke disk saat batas memoryCapacity atau byteCapacity tercapai.

- **Custom Channel**

  Flume memungkinkan channel bea cukai diintegrasikan ke dalam sistem. Channel khusus diterapkan di Java. File kelas Java dari custom channel bersama dengan

dependensi termasuk dalam classpath Agent Flume dan juga ditentukan dalam file konfigurasi Agent yang ditunjukkan di bawah ini:

```
myagent.channels = channel1
myagent.channels.channel1.type = org.example.MyChannel
```

**Pemilihan Channel**

Agent flume dapat memiliki satu sumber yang terhubung ke beberapa channel. Dalam kasus seperti itu, pemilih channel menentukan kebijakan tentang mendistribusikan peristiwa di antara channel yang terhubung ke satu sumber.

- ***ReplicatingChannelSelector*** adalah pemilih berulang, yang mereplikasi acara yang diterima dari sumber ke semua channel yang terhubung. Box di bawah ini menunjukkan contoh konfigurasi Agent yang memiliki satu sumber yang terhubung ke tiga channel dan menggunakan pemilih channel yang mereplikasi.

```
myagent.sources = source1
myagent.channels = channel1 channel2 channel3
myagent.source.source1.selector.type = replicating
myagent.source.source1.channels = channel1 channel2 channel3
myagent.source.source1.selector.optional = channel3
```

- ***Multiplexing Channel Selecto***r mendistribusikan acara dari sumber ke semua channel yang terhubung. Box di bawah ini menunjukkan contoh konfigurasi Agent yang memiliki satu sumber terhubung ke tiga channel dan menggunakan pemilih channel multiplexing.

```
myagent.sources = source1
myagent.channels = channel1 channel2 channel3
myagent.sources.source1.selector.type = multiplexing
myagent.sources.source1.selector.header = country
myagent.sources.source1.selector.mapping.IN = channel1
myagent.sources.source1.selector.mapping.US = channel2
myagent.sources.source1.selector.default = channel3
```

Properti tajuk menentukan nama atribut untuk memeriksa pendistribusian kejadian di antara channel dan properti pemetaan menentukan pemetaan antara nilai atribut dan channel. Misalnya, dalam konfigurasi di atas, properti header disetel ke atribut negara. Semua acara yang memiliki nilai atribut negara sebagai IN dikirim ke channel1 sementara semua acara dengan nilai atribut negara seperti AS dikirim ke channel2. Channel default ditetapkan sebagai channel3.

- Custom Channel Selector: Flume memungkinkan pemilih channel bea cukai diintegrasikan ke dalam sistem. Pemilih channel khusus diterapkan di Java. File kelas Java dari pemilih channel khusus bersama dengan dependensi termasuk dalam jalur kelas Agent Flume dan juga ditentukan dalam file konfigurasi Agent yang ditunjukkan di bawah ini:

```
■ myagent.sources = sourcel
myagent.channels = channell
myagent.sources.sourcel.selector.type =
      org.example.MyChannelSelector
```

**Sink Processors**

Flume memungkinkan pembuatan grup sink di mana channel dapat dilampirkan ke grup sink tempat kejadian dikeringkan. Prosesor wastafel mendefinisikan bagaimana peristiwa dikeringkan dari channel ke wastafel. Prosesor sink memungkinkan paralelisme, prioritas, dan failover otomatis.

- ***Load balancing Sink Processor*** memungkinkan load balancing kejadian yang dikuras dari channel antara sink dalam grup sink terpasang. Beban didistribusikan di antara daftar sink yang ditentukan menggunakan mekanisme round robin atau pemilihan acak. Box di bawah ini menunjukkan contoh Agent dengan grup sink dan prosesor sink load balancing.

```
■ myagent.sinkgroups = group1
myagent.sinkgroups.group1.sinks = sink1 sink2
myagent.sinkgroups.group1.processor.type = load_balance
myagent.sinkgroups.group1.processor.backoff = true
myagent.sinkgroups.group1.processor.selector = random
```

- ***Failover Sink Processor***: Dengan Failover Sink Processor, prioritas dapat ditetapkan ke sink di antara grup sink. Channel terlampir kemudian mengalirkan acara ke sink prioritas tertinggi. Jika sink prioritas tertinggi gagal, peristiwa dikosongkan ke sink dengan satu prioritas lebih rendah, yang menyediakan failover otomatis. Box di bawah ini menunjukkan contoh Agent dengan grup sink dan prosesor sink failover.

```
■ myagent.sinkgroups = group1
myagent.sinkgroups.group1.sinks = sink1 sink2
myagent.sinkgroups.group1.processor.type = failover
myagent.sinkgroups.group1.processor.priority.sink1 = 2
myagent.sinkgroups.group1.processor.priority.sink2 = 4
myagent.sinkgroups.group1.processor.maxpenalty = 10000
```

### Flume Interceptors

Flume interseptor memungkinkan kejadian untuk dimodifikasi, disaring atau dijatuhkan saat mereka mengalir dari sumber ke channel. Interceptor terhubung ke sumber. Interceptor juga bisa dirantai satu sama lain.

- ***Timestamp interseptor*** menambahkan stempel waktu saat ini ke header peristiwa yang diproses. Interceptor timestamp dapat dikonfigurasi sebagai berikut:

```
■ myagent.sources = source1
myagent.sources.source1.interceptors = i1
myagent.sources.source1.interceptors.i1.type = timestamp
```

- ***Host interceptor*** menambahkan nama host dari Agent Flume ke header dari kejadian yang diproses. Pencegat host dapat dikonfigurasi sebagai berikut:

```
■ myagent.sources = source1
myagent.sources.source1.interceptors = i1
```

```
myagent.sources.source1.interceptors.i1.type = host
myagent.sources.source1.interceptors.i1.hostHeader = hostname
myagent.sources.source1.interceptors.i1.useIP = false
```

- ***Static Interceptor*** menambahkan header statis ke peristiwa yang diproses. Box di bawah ini menunjukkan contoh menambahkan header statis, negara, dengan nilai yang disetel ke AS.

```
■ myagent.sources = source1
myagent.sources.source1.interceptors = i1
myagent.sources.source1.interceptors.i1.type = static
myagent.sources.source1.interceptors.i1.key = country
myagent.sources.source1.interceptors.i1.value = US
```

- ***UUID Interceptor:*** UUID menambahkan pengenal unik universal ke header peristiwa yang diproses. Interceptor UUID dapat dikonfigurasi sebagai berikut:

```
■ myagent.sources = source1
myagent.sources.source1.interceptors = i1
myagent.sources.source1.interceptors.i1.type = uuid
myagent.sources.source1.interceptors.i1.headerName=id
```

- ***Regex Filtering interseptor*** menerapkan ekspresi reguler ke badan peristiwa dan memfilter peristiwa yang cocok. Peristiwa yang cocok dengan ekspresi reguler dapat

disertakan atau dikecualikan. Interceptor Filtering Regex dapat dikonfigurasi sebagai berikut:

```
■ myagent.sources = sourcel
myagent.sources.sourcel.interceptors = il
myagent.sources.sourcel.interceptors.il.type = regex_filter
myagent.sources.sourcel.interceptors.il.regex = .*
myagent.sources.sourcel.interceptors.il.excludeEvents = false
```

**Contoh Flume**

Box 5.10 menunjukkan contoh penyiapan Agent Flume dengan Sumber NetCat & File Roll Sink.

**■ Box 5.10: Flume agent with NetCat Source & File Roll Sink**

```
myagent.sources = r1
myagent.channels = c1
myagent.sinks = k1

# Define source
myagent.sources.r1.type = netcat
myagent.sources.r1.bind = 0.0.0.0
myagent.sources.r1.port = 6666

#Define Sink
myagent.sinks.k1.type = file_roll
myagent.sinks.k1.sink.directory = /var/log/flume
```

```
#Define Channel
myagent.channels.c1.type = file
myagent.channels.c1.checkpointDir = /var/flume/checkpoint
myagent.channels.c1.dataDirs = /var/flume/data

# Bind the source and sink to the channel
myagent.sources.r1.channels = c1
myagent.sinks.k1.channel = c1
```

Untuk menguji Agent, jalankan Agent Flume dan kemudian buka terminal baru dan jalankan perintah berikut:

```
■ nc localhost 6666
```

Ketikkan beberapa teks. Teks yang sama akan dikirim ke sinkfile.

```
■ sudo flume-ng agent -c /etc/flume/conf -f /etc/flume/conf/flume.conf -n
myagent
```

Box 5.11 menunjukkan contoh menyiapkan Agent Flume dengan Sumber Twitter & HDFS Sink.

**Box 5.11: Flume agent with Twitter Source & HDFS Sink**

```
myagent.sources = r1
myagent.channels = c1
myagent.sinks = k1

# Define source
myagent.sources.r1.type = org.apache.flume.source.twitter.TwitterSource
myagent.sources.r1.consumerKey = <enter key here>
myagent.sources.r1.consumerSecret = <enter secret here>
myagent.sources.r1.accessToken = <enter token here>
myagent.sources.r1.accessTokenSecret = <enter token secret here>
myagent.sources.r1.maxBatchSize = 10
myagent.sources.r1.maxBatchDurationMillis = 200

#Define sink
myagent.sinks.k1.type = hdfs
myagent.sinks.k1.hdfs.fileType = DataStream
myagent.sinks.k1.hdfs.path = /flume/events
myagent.sinks.k1.hdfs.filePrefix = eventlog
myagent.sinks.k1.hdfs.fileSuffix = .log
myagent.sinks.k1.hdfs.batchSize = 1000

#Define Channel
myagent.channels.c1.type = file
myagent.channels.c1.checkpointDir = /var/flume/checkpoint
myagent.channels.c1.dataDirs = /var/flume/data

# Bind the source and sink to the channel
myagent.sources.r1.channels = c1
myagent.sinks.k1.channel = c1
```

Box 5.12 menunjukkan contoh penyiapan Agent Flume dengan Sumber HTTP & File Roll Sink.

**Box 5.12: Flume agent with HTTP Source & File Roll Sink**

```
myagent.sources = r1
myagent.channels = c1
myagent.sinks = k1

# Define source
myagent.sources.r1.type = http
myagent.sources.r1.bind = 0.0.0.0
myagent.sources.r1.port = 8000
myagent.sources.r1.handler = org.apache.flume.source.http.JSONHandler
myagent.sources.r1.handler.nickname = randomprops
```

```
#Define sink
myagent.sinks.k1.type = file_roll
myagent.sinks.k1.sink.directory = /var/log/flume

#Define Channel
myagent.channels.c1.type = file
myagent.channels.c1.checkpointDir = /var/flume/checkpoint
myagent.channels.c1.dataDirs = /var/flume/data

# Bind the source and sink to the channel
myagent.sources.r1.channels = c1
myagent.sinks.k1.channel = c1
```

### 5.3.2 Apache Sqoop

Apache Sqoop adalah alat yang memungkinkan mengimpor data dari sistem manajemen database relasional (RDBMS) ke dalam tabel Hadoop Distributed File System (HDFS), Hive atau HBase. Sqoop juga memungkinkan mengekspor data dari HDFS ke RDBMS. Tabel 5.1 mencantumkan berbagai perintah Sqoop.

| Tool | Fungsi |
|---|---|
| Import | Import tabel dari database ke HDFS |
| import-all-tables | Import tabel dari database ke HDFS |
| export | Eksport sebuah direktori HDFS ke sebuah tabel database |
| codegen | Menghasilkan kode untuk berinteraksi dengan record database |
| Create-hive-table | import sebuah definisi tabel pada Hive |
| Eval | Mengevaluasi statement SQL dan hasil display |
| List-database | List ketersediaan database pada server |
| List-tabel | List ketersediaan tabel pada sebuah database |

Tabel 5.1: *Sqoop Tools*

### 5.3.3 Mengimpor Data dengan Sqoop

Gambar 5.8 menunjukkan proses mengimpor data dari RDBMS menggunakan Sqoop. Proses impor dimulai dengan pengguna mengirimkan perintah impor Sqoop. Format perintah impor ditunjukkan di bawah ini:

```
■ sqoop import --connect jdbc:mysql://<IP Address>/<Database Name>
--username <Username> --password <Password> --table <Table Name>
```

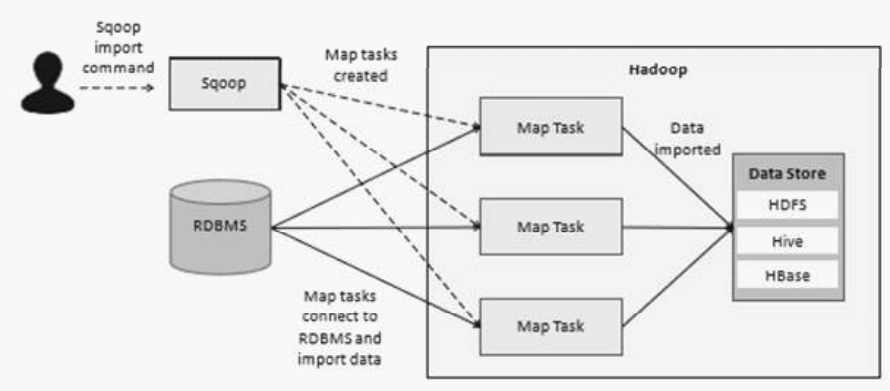
Perintah import menyertakan string koneksi yang menentukan tipe database, nama host server database (atau alamat IP) dan nama database. Sqoop dapat terhubung ke database yang mendukung JDBC.

Contoh perintah impor untuk mengimpor data dari tabel bernama Kursus dari database MySQL bernama Departemen ditunjukkan di bawah ini:

```
■ sqoop import --connect jdbc:mysql://localhost/Department
--username admin --password admin123 --table Courses
```

Perintah impor Sqoop meluncurkan beberapa tugas Map (standarnya adalah empat tugas) yang terhubung ke database dan mengimpor baris dalam tabel secara paralel ke HDFS sebagai file teks yang dibatasi, Avrofil biner atau Hadoop SequenceFiles. Jumlah tugas Map yang digunakan untuk mengimpor data (dan karenanya paralelisme) dapat dikontrol menggunakan opsi —m seperti yang ditunjukkan pada contoh di bawah ini:

```
■ #Use 8 map tasks to import sqoop import --connect
jdbc:mysql://localhost/Department
--username admin --password admin123 --table Courses --m 8
```




Gambar 5.8: *Mengimpor data menggunakan Apache Sqoop*

### 5.3.4 Memilih Data untuk Diimpor

Sementara di contoh sebelumnya, kami mengimpor semua data dari tabel, Sqoop juga memungkinkan mengimpor data yang dipilih. Dengan impor Sqoop, dimungkinkan untuk memilih subset kolom (menggunakan opsi kolom) untuk diimpor dari tabel seperti yang ditunjukkan pada contoh di bawah ini:

```
■ sqoop import --connect jdbc:mysql://localhost/Department
--username admin --password admin123 --table Courses
--columns "name,semester,year"
```

Anda juga dapat menggunakan kueri SQL dengan impor Sqoop untuk memilih data yang akan diimpor seperti yang ditunjukkan pada contoh di bawah ini:

```
■ sqoop import --connect jdbc:mysql://localhost/myDB --username admin
--password admin123 --query 'SELECT a.*, b.* FROM a JOIN b on
(a.id == b.id) WHERE $CONDITIONS'
--split-by a.id --target-dir /user/admin/joinresults
```

Dalam contoh di atas, Sqoop akan mengimpor hasil kueri secara paralel. Karena setiap tugas Map akan menjalankan kueri yang sama, kondisi tertentu diperlukan untuk memisahkan data yang diimpor oleh setiap tugas Map. Token $ CONDITIONS diganti dengan ketentuan oleh alat impor Sqoop pada saat berjalan. Opsi split-by menentukan, di kolom mana pemisahan data harus dilakukan untuk mengimpor data secara paralel. Saat menggunakan kueri SQL untuk menentukan data apa yang akan diimpor, opsi target-dir diperlukan untuk memberikan lokasi target untuk data yang akan diimpor.

### 5.3.5 Konektor Khusus

Meskipun Sqoop dikirimkan dengan konektor JDBC generik, mungkin lebih baik menggunakan konektor JDBC khusus vendor karena dapat memberikan kinerja yang lebih tinggi. Selain itu, beberapa database menyediakan alat pemindahan data yang dapat memindahkan data dengan kinerja yang lebih tinggi. Misalnya, MySQL menyediakan alat mysqldump yang dapat digunakan untuk mengekspor data dari database MySQL. Sqoop memungkinkan alat khusus database untuk digunakan dengan perintah impor Sqoop menggunakan opsi langsung. Box di bawah ini menunjukkan contoh mengimpor data dari MySQL dengan Sqoop menggunakan alat mysqldump

```
■ sqoop import --connect jdbc:mysql://localhost/Department
--username admin --password admin123 --table Courses --direct

#Passing additional arguments to database-specific tool
sqoop import --connect jdbc:mysql://localhost/Department
--username admin --password admin123
--table Courses --direct -- --default-character-set=latin1
```

### 5.3.6 Mengimpor Data ke Hive

Sqoop memungkinkan mengimpor data ke dalam hive menggunakan opsi impor-hive seperti yang ditunjukkan pada contoh di bawah ini. Jika opsi ini disetel, Sqoop akan secara otomatis membuat tabel Hive dan mengimpor data ke dalam tabel.

```
■ sqoop import --connect jdbc:mysql://localhost/Department
--username admin --password admin123 --table Courses --hive-import
```

### 5.3.7 Mengimpor Data ke HBase

Sqoop memungkinkan mengimpor data ke HBase menggunakan opsi tabel-hbase bersama dengan nama tabel target HBase, seperti yang ditunjukkan pada contoh di bawah ini. Sqoop juga mendukung pemuatan massal data ke HBase menggunakan opsi hbase-bulkload.

```
■ sqoop import --connect jdbc:mysql://localhost/Department
--username admin --password admin123 --table Courses
--hbase-table Courses
```

### 5.3.8 Incremental Import

Incremental Import berguna saat Anda sebelumnya telah mengimpor beberapa baris dari tabel, dan Anda ingin mengimpor baris yang lebih baru. Sqoop menyediakan opsi inkremental untuk impor inkremental. Jika opsi ini digunakan, mode juga diperlukan, yang bisa ditambahkan atau diubah terakhir. Mode penambahan digunakan saat tabel diperbarui dengan baris baru dengan nilai ID baris yang meningkat. Kolom untuk memeriksa ID baris ditentukan menggunakan opsi kolom centang.

Modus lastmodified digunakan ketika baris tabel diperbarui dan stempel waktu saat baris terakhir diubah diatur dalam kolom yang dilastmodified. Kolom untuk memeriksa stempel waktu yang terakhir diubah ditentukan menggunakan opsi kolom centang. Opsi nilai terakhir digunakan dalam mode lastmodified untuk menentukan stempel waktu. Ketika proses impor Sqoop selesai, ia mencetak nilai terakhir. Dalam impor berikutnya, nilai terakhir ini ditentukan, sehingga Sqoop hanya dapat mengimpor baris yang memiliki stempel waktu yang dilastmodified lebih besar dari nilai terakhir.
Box di bawah ini menunjukkan contoh impor inkremental:

```
■ sqoop import --connect jdbc:mysql://localhost/Department
--username admin --password admin123 --table Students
--check-column id --incremental append

#Import last modified rows
sqoop import --connect jdbc:mysql://localhost/Department
--username admin --password admin123 --table Students
--check-column last-modified --incremental
lastmodified --last-value "2015-04-03 15:08:45.66"
```

### 5.3.9 Mengimpor Semua Tabel

Perintah Sqoop import-all-tables dapat digunakan untuk mengimpor semua tabel dari database ke HDFS, seperti yang ditunjukkan pada contoh berikut:

```
■ sqoop import-all-tables --connect jdbc:mysql://localhost/Department
```

### 5.3.10 Mengekspor Data dengan Sqoop

Perintah ekspor Sqoop dapat digunakan untuk mengekspor file dari HDFS ke RDBMS, seperti yang ditunjukkan pada contoh berikut:

```
■ sqoop export --connect jdbc:mysql://localhost/Department -table Courses
-export-dir /user/admin/courses
```

Tabel target harus ada di database. Sqoop menerjemahkan perintah ekspor ke dalam satu set pernyataan INSERT untuk menambahkan baris baru ke tabel. Data dari file masukan diurai dan dimasukkan ke dalam tabel target. Alih-alih mode "sisipkan" default, Anda juga dapat menentukan mode "pembaruan", di mana Sqoop akan menggunakan pernyataan UPDATE untuk menggantikan catatan yang ada di dapat ditawar. Opsi mode pembaruan dapat digunakan untuk menentukan mode "pembaruan". Dengan opsi mode pembaruan, mode perlu ditentukan yang dapat diperbarui hanya atau diizinkan untuk dimasukkan. Ketika mode updateonly ditentukan, baris dalam tabel diperbarui dalam proses ekspor hanya jika ada. Dengan mode allowinsert, baris diperbarui jika sudah ada di tabel atau disisipkan jika tidak ada.

## 5.4 Antrian Perpesanan

Antrian pesan berguna untuk pesan push-pull di mana produsen mendorong data ke antrian, dan konsumen menarik data dari antrian. Produsen dan konsumen tidak perlu saling waspada. Antrian pesan memungkinkan pemisahan produsen data dari konsumen. Pada bagian ini, kami akan menjelaskan beberapa sistem antrian pesan berdasarkan protokol seperti Advanced Message Queuing Protocol (AMQP) dan ZeroMQ Message Transfer Protocol (ZMTP).

### 5.4.1 RabbitMQ

RabbitMQ mengimplementasikan Advanced Message Queuing Protocol (AMQP), yang merupakan standar terbuka yang mendefinisikan protokol untuk pertukaran pesan antar sistem. Klien AMQP dapat menjadi produsen atau konsumen. Klien yang memenuhi

standar dapat berkomunikasi satu sama lain melalui broker. Broker adalah aplikasi middleware yang menerima pesan dari produsen dan mengarahkannya ke konsumen. Produsen mempublikasikan pesan ke bursa, yang kemudian mendistribusikan pesan ke antrian berdasarkan aturan perutean yang ditentukan (atau pengikatan). Broker AMQP menyediakan empat jenis pertukaran: pertukaran langsung (untuk perpesanan point-to-point), pertukaran fanout (untuk perpesanan multicast), pertukaran topik (untuk perpesanan langganan-publikasi) dan pertukaran header (yang menggunakan atribut header untuk membuat keputusan perutean). Pertukaran menggunakan binding yang merupakan aturan untuk merutekan pesan ke antrean. Konsumen mengkonsumsi pesan dari antrian. AMQP adalah protokol tingkat aplikasi yang menggunakan TCP untuk pengiriman yang andal. Hubungan logis antara produsen atau konsumen dan broker disebut Channel. Untuk aplikasi yang perlu membuat banyak koneksi dengan broker, tidak diinginkan untuk memiliki banyak koneksi TCP. Untuk aplikasi semacam itu, beberapa Channeldapat diatur melalui satu koneksi.

RabbitMQ adalah Broker AMQP yang diimplementasikan di Erlang dan dirancang untuk menjadi sangat terukur dan andal. Perintah untuk menyiapkan RabbitMQ diberikan di Box 5.13.

**■ Box 5.13: Setting up RabbitMQ**

```
echo 'deb http://www.rabbitmq.com/debian/ testing main' |
sudo tee /etc/apt/sources.list
```

```
wget https://www.rabbitmq.com/rabbitmq-signing-key-public.asc
sudo apt-key add rabbitmq-signing-key-public.asc
sudo apt-get install rabbitmq-server
sudo pip install pika
```

Box 5.14 menunjukkan contoh produsen yang mengirim data ke antrian RabbitMQ. Contoh ini menggunakan library pika yang merupakan implementasi AMQP murni Python. Produser mengirimkan data sintetis bersama dengan stempel waktu ke antrean RabbitMQ.

**■ Box 5.14: Example of a Producer that sends data to RabbitMQ**

```
import pika
from time import time
import json
import pickle, re, os, urllib, urllib2
from datetime import datetime
```

```
from random import randrange
import time
import datetime

connection = pika.BlockingConnection(pika.ConnectionParameters(
host='localhost'))
channel = connection.channel()

channel.queue_declare(queue='test')

while True:
   data = str(randrange(0,60)) + ',' +
       str(randrange(0,100)) + ',' + str(randrange(5000,12000)) +
       ',' + str(randrange(50,350))

   ts=time.time()
   timestamp =
   datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-%d %H:%M:%S')

   msg='timestamp': timestamp, 'data': data
   print msg

   channel.basic_publish(exchange='',
routing_key='test',
body=json.dumps(msg))

   print data
   time.sleep(1)
```

Box 5.15 menunjukkan contoh konsumen yang menggunakan data dari antrian RabbitMQ.

**■ Box 5.15: Example of a Consumer that consumes data from RabbitMQ**

```
import pika

connection = pika.BlockingConnection(pika.ConnectionParameters(
host='localhost'))

channel = connection.channel()

channel.queue_declare(queue='hello')

def callback(ch, method, properties, body):
print "Received %r" % (body,)

channel.basic_consume(callback,
queue='hello',
no_ack=True)

channel.start_consuming()
```

### 5.4.2 ZeroMQ

ZeroMQ adalah pustaka perpesanan berkinerja tinggi yang menyediakan alat untuk membangun sistem perpesanan. Tidak seperti sistem antrian pesan lainnya, ZeroMQ dapat bekerja tanpa perantara pesan. ZeroMQ menyediakan berbagai pola pesan seperti Request-Reply, Publish-Subscribe, Push-Pull dan Exclusive Pair.
Perintah untuk menyiapkan ZeroMQ diberikan di Box 5.16.

**Box 5.16: Setting up ZeroMQ**

```
sudo apt-get install libtool
autoconf automake uuid-dev build-essential
wget http://download.zeromq.org/zeromq-4.0.4.tar.gz
tar zxvf zeromq-4.0.4.tar.gz && cd zeromq-4.0.4
./configure
make
sudo make install
sudo apt-get install python-zmq
```

Box 5.17 memperlihatkan contoh produsen yang mengirim data ke antrian ZeroMQ

**Box 5.17: Example of Producer that sends data to ZeroMQ**

```
import zmq
from time import time
import json
from random import randrange
import time
import datetime

context = zmq.Context()

socket = context.socket(zmq.PUSH)
socket.bind('tcp://127.0.0.1:5555')

while True:
#Generate some synthetic data
   data = str(randrange(0,60)) + ',' +
       str(randrange(0,100)) + ',' + str(randrange(5000,12000)) +
       ',' + str(randrange(50,350))

   ts=time.time()
   timestamp =
   datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-%d %H:%M:%S')

   msg='timestamp': timestamp, 'data': data
   print msg
```

```
    data = zmq.Message(json.dumps(msg))
    socket.send(data)

    print data
    time.sleep(1)
```

Box 5.18 menunjukkan contoh konsumen yang menggunakan data dari antrian ZeroMQ

**Box 5.18: Example of a Consumer that consumes data from ZeroMQ**

```
import zmq
context = zmq.Context()

socket = context.socket(zmq.PULL)
socket.connect('tcp://127.0.0.1:5555')

while True:
    data = socket.recv()
    print data
```

### 5.4.3 RestMQ

RESTMQ adalah antrian pesan yang didasarkan pada protokol berbasis JSON sederhana dan menggunakan HTTP sebagai transport. Antrian diatur sebagai sumber daya REST. RESTMQ dapat digunakan oleh semua klien yang dapat melakukan panggilan HTTP. Perintah untuk menyiapkan RestMQ diberikan di Box 5.19.

**Box 5.19: Setting up RESTMQ**

```
#Install RESTMQ
sudo apt-get install build-essential curl python-pip redis-server
    libffi-dev python-dev -y libssl-dev python-setuptools
git clone https://github.com/gleicon/restmq.git
cd restmq
sudo pip install -r requirements.txt
sudo python setup.py install

# Start RESTMQ
cd restmq/start_scripts
touch acl.conf
bash restmq_server -acl=acl.conf -listen=0.0.0.0 &
```

Box 5.20 menunjukkan contoh produsen yang mengirim data ke antrian RESTMQ.

**Box 5.20: Example of a Producer that sends data to RESTMQ**

```
import requests
import json
import urllib2
from random import randrange
import time
import datetime

while True:
#Generate some synthetic data
   data = str(randrange(0,60)) + ',' +
       str(randrange(0,100)) + ',' + str(randrange(5000,12000)) +
          ',' + str(randrange(50,350))

   ts=time.time()
   timestamp =
      datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-%d %H:%M:%S')

   msg='timestamp': timestamp, 'data': data

   data = urllib.urlencode('queue':'test', 'value':json.dumps(msg))
   r = urllib2.Request('http://localhost:8888/', data)
   f = urllib2.urlopen(r)
   data = f.read()
   f.close()
```

Box 5.21 menunjukkan contoh konsumen yang menggunakan data dari antrian RESTMQ.

**Box 5.21: Example of a Consumer that consumes data from RESTMQ**

```
import json
from twisted.web import client
from twisted.python import log
from twisted.internet import reactor

class CometClient(object):
   def write(self, content):
      try:
         data = json.loads(content)
      except Exception, e:
         log.err("cannot decode json: %s" % str(e))
         log.err("json is: %s" % content)
      else:
         log.msg("got data: %s" % repr(data))

   def close(self):
      pass

if __name__ == "__main__":
   log.startLogging(sys.stdout)
   client.downloadPage("http://localhost:8888/c/test", CometClient())
   reactor.run()
```

```
#Post data to RESTMQ
curl -X POST -d "value=data" http://localhost:8888/q/test

#Get data from RESTMQ
curl http://localhost:8888/c/test
```

### 5.4.4 Amazon SQS

Amazon SQS menawarkan antrean host yang sangat dapat diskalakan dan andal untuk menyimpan pesan saat pesan berpindah di antara komponen aplikasi yang berbeda. SQS hanya menjamin bahwa pesan akan sampai, bukan dalam urutan yang sama seperti saat dimasukkan ke dalam antrian. Meskipun, pada pandangan pertama, AmazonSQS mungkin tampak mirip dengan Amazon Kinesis, namun keduanya ditujukan untuk jenis aplikasi yang sangat berbeda. Sementara Kinesis dimaksudkan untuk aplikasi real-time yang melibatkan tingkat masuk dan keluar data tinggi, SQS hanyalah sistem antrian yang menyimpan dan merilis pesan dengan cara yang dapat diskalakan.

SQS dapat digunakan dalam aplikasi terdistribusi di mana berbagai komponen aplikasi perlu bertukar pesan. Mari kita lihat beberapa contoh penggunaan SQS. Box 5.22 menunjukkan kode Python untuk membuat antrian SQS. Dalam contoh ini, koneksi ke layanan SQS pertama kali dibuat dengan memanggil boto.sqs.connect_to_region. Wilayah AWS, kunci akses, dan kunci rahasia diteruskan ke fungsi ini. Setelah terhubung ke layanan SQS, conn.create_queue dipanggil untuk membuat antrian baru dengan nama antrian sebagai parameter input. Fungsi conn.get_all_queues digunakan untuk mengambil semua antrian SQS.

**Box 5.22: Python program for creating an SQS queue**

```
import boto.sqs

ACCESS_KEY="<enter access key>"
SECRET_KEY="<enter secret key>"
REGION="us-east-1"

print "Connecting to SQS"

conn = boto.sqs.connect_to_region(
      REGION,
      aws_access_key_id=ACCESS_KEY,
      aws_secret_access_key=SECRET_KEY)
```

```
queue_name = 'mytestqueue'

print "Creating queue with name:  " + queue_name
q = conn.create_queue(queue_name)

print "Created queue with name:  " + queue_name

print " \n Getting all queues"

rs = conn.get_all_queues()
for item in rs:
   print item
```

Box 5.23 menunjukkan kode Python untuk menulis ke antrian SQS. Setelah menghubungkan ke antrian SQS, metode queue.write dipanggil dengan pesan sebagai parameter input.

**Box 5.23: Python program for writing to an SQS queue**

```
import boto.sqs
from boto.sqs.message import Message
import time

ACCESS_KEY="<enter access key>"
SECRET_KEY="<enter secret key>"

REGION="us-east-1"

print "Connecting to SQS"

conn = boto.sqs.connect_to_region(
      REGION,
      aws_access_key_id=ACCESS_KEY,
      aws_secret_access_key=SECRET_KEY)

queue_name = 'mytestqueue'

print "Connecting to queue:  " + queue_name
q = conn.get_all_queues(prefix=queue_name)

msg_datetime = time.asctime(time.localtime(time.time()))

msg = "Test message generated on:  " + msg_datetime
print "Writing to queue:  " + msg

m = Message()
m.set_body(msg)
status = q[0].write(m)

print "Message written to queue"

count = q[0].count()

print "Total messages in queue:  " + str(count)
```

Box 5.24 menunjukkan kode Python untuk membaca dari antrian SQS. Setelah menghubungkan ke antrian SQS, metode queue.read dipanggil untuk membaca pesan dari antrian.

**Box 5.24: Python program for reading from an SQS queue**

```
import boto.sqs
from boto.sqs.message import Message

ACCESS_KEY="<enter access key>"
SECRET_KEY="<enter secret key>"

REGION="us-east-1"

print "Connecting to SQS"

conn = boto.sqs.connect_to_region(
      REGION,
      aws_access_key_id=ACCESS_KEY,
      aws_secret_access_key=SECRET_KEY)

queue_name = 'mytestqueue'

print "Connecting to queue:  " + queue_name
q = conn.get_all_queues(prefix=queue_name)

count = q[0].count()

print "Total messages in queue:  " + str(count)

print "Reading message from queue"

for i in range(count):
   m = q[0].read()
   print "Message %d:  %s" % (i+1,str(m.get_body()))
   q[0].delete_message(m)

print "Read %d messages from queue" % (count)
```

## 5.5 Konektor Kustom

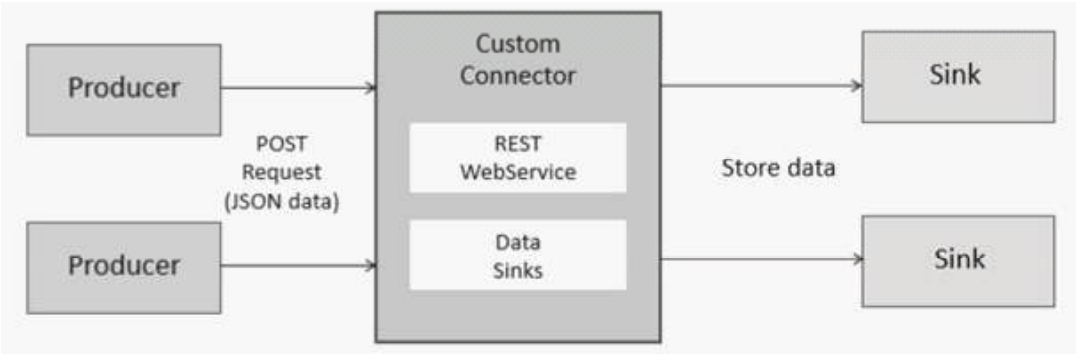
Konektor khusus dan layanan web untuk memperoleh data dari produsen data dapat dikembangkan untuk memenuhi persyaratan aplikasi.

### 5.5.1 Konektor berbasis REST

Gambar 5.9 menunjukkan arsitektur konektor kustom berbasis REST. Konektor memperlihatkan layanan web REST. Produsen data dapat mempublikasikan data

ke konektor menggunakan permintaan HTTP POST yang berisi payload data. Data permintaan yang diterima oleh konektor disimpan ke sink (seperti sistem file lokal, sistem file terdistribusi, atau penyimpanan cloud). Data sink di konektor menyediakan fungsionalitas untuk memproses permintaan HTTP dan menyimpan data ke sink. Manfaat menggunakan konektor berbasis REST adalah bahwa setiap klien yang dapat membuat permintaan HTTP dapat mengirim data ke konektor. Permintaan bersifat kewarganegaraan, dan setiap permintaan membawa semua informasi yang diperlukan untuk memproses permintaan. Header HTTP menambah overhead permintaan sehingga metode ini tidak cocok untuk aplikasi throughput tinggi dan real-time.

Menerapkan Konektor Kustom berbasis REST Mari kita lihat contoh penerapan konektor kustom berbasis REST seperti yang ditunjukkan pada Gambar 5.9. Box 5.25 menunjukkan implementasi Python dari konektor berbasis REST. Dalam contoh ini, kami menggunakan framework web Flask Python untuk mengimplementasikan layanan web. Konektor ini memublikasikan satu titik akhir (seperti 'http: // public-ip / api / data'), tempat aplikasi klien dapat mengirim permintaan HTTP POST bersama dengan payload data. Box 5.26 menunjukkan contoh klien yang mengirimkan beberapa data sensor sintetis ke layanan web. Layanan web menerima data dari payload permintaan POST dan kemudian menerbitkan data ke antrian Amazon SQS dan juga menulis data ke tabel Amazon DynamoDB.




Gambar 5.9: *Konektor kustom berbasis REST*

Manfaat memiliki konektor khusus seperti itu adalah klien dan server menjadi independen satu sama lain. Layanan web memisahkan klien dari server. Server dapat menambah atau mengubah tindakan (seperti menerbitkan data ke antrian atau menyimpan data dalam database) tanpa klien harus menyadari perubahan tersebut. Klien dapat menggunakan alat atau bahasa pemrograman apa pun yang dapat digunakan untuk membuat permintaan HTTP POST. (Catatan: Meskipun kami menyebutnya sebagai konektor REST, ini tidak

sepenuhnya sesuai dengan REST karena kami hanya menerapkan fungsionalitas POST. Metode lain seperti GET, PUT, DELETE mungkin tidak diperlukan jika konektor hanya mengizinkan data untuk diserap.)

**Box 5.25: Python implementation of a REST-based custom connector**

```
import boto.sqs
from boto.sqs.message import Message
import boto.dynamodb2
from boto.dynamodb2.table import Table
import cPickle as pickle
import time
import datetime
import json
from flask import Flask, jsonify, abort,
from flask import request, make_response, url_for

app = Flask(__name__, static_url_path="")


ACCESS_KEY = <Enter AWS Access Key>
SECRET_KEY = <Enter Secret Key>
REGION="us-east-1"
queue_name = 'sensordata'
table_name = 'sensordata'

#Connect to AWS SQS
conn = boto.sqs.connect_to_region(REGION,aws_access_key_id=ACCESS_KEY,
aws_secret_access_key=SECRET_KEY)

q = conn.get_all_queues(prefix=queue_name)

#Connect to AWS DynamoDB
conn_dynamo = boto.dynamodb2.connect_to_region(REGION,
 aws_access_key_id=ACCESS_KEY,
 aws_secret_access_key=SECRET_KEY)

table=Table(table_name,connection=conn_dynamo)

#Publishes data to SQS
def publish_to_sqs(data):
   m = Message()
   m.set_body(data)
   status = q[0].write(m)
   return status

#Writes data to DynamoDB table
def publish_to_dynamo(datadir):
   item = table.put_item(data=datadir)

@app.errorhandler(400)
def bad_request(error):
   return make_response(jsonify({'error':  'Bad request'}), 400)
```

```
@app.errorhandler(404)
def not_found(error):
   return make_response(jsonify({'error':  'Not found'}), 404)


@app.route('/api/data', methods=['POST'])
def post_data():
   data = json.loads(request.data)

   publish_to_sqs(pickle.dumps(data))

   publish_to_dynamo(data)

   return jsonify({'result':  'true'}), 201


if __name__ == '__main__':
   app.run(debug=True)
```

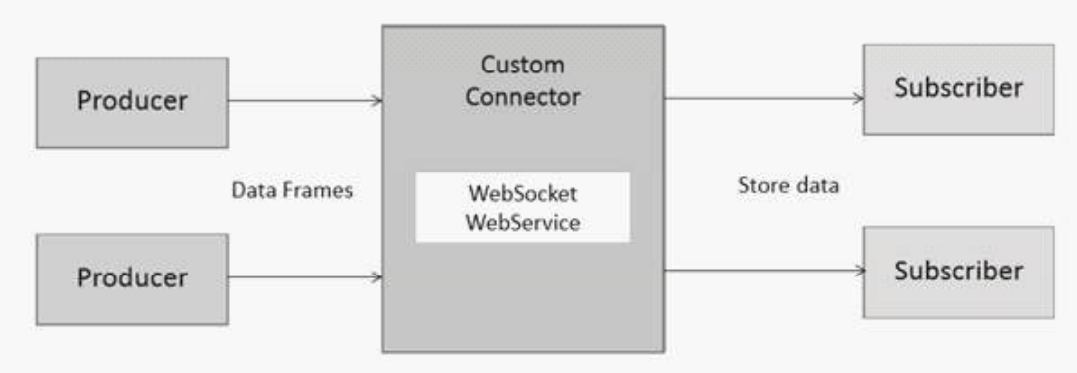
**■ Box 5.26: Python implementation of client program that publishes data to a custom connector**

```
from random import randint
import time
import datetime
import requests
import json

def getData():
   ts=time.time()
   timestamp = datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-%d %H:%M:%S')
   temp = str(randint(0,100))
   humidity = str(randint(0,100))
   co2 = str(randint(50,500))
   light = str(randint(0,10000))
   data = {"timestamp":  timestamp, "temperature":  temp,
         "humidity":  humidity , "co2":  co2, "light":  light}
   return data

def publish(datadir):
   r = requests.post("http://localhost:5000/api/data",
         data = json.dumps(datadir),
         headers={"Content-Type":  "application/json"})

while True:
   data = getData()
   print data
   publish(data)
   time.sleep(1)
```

### 5.5.2 Berbasis WebSocket

Konektor Gambar 5.10 menunjukkan arsitektur konektor kustom berbasis WebSocket. Konektor memperlihatkan layanan web WebSocket. Protokol Perpesanan Aplikasi Web (WAMP) yang merupakan sub-protokol WebSocket dapat digunakan untuk membuat konektor berbasis WebSocket. WAMP menyediakan pola pesan publish-subscribe dan remote procedure call (RPC). Klien (atau produsen data) membuat koneksi TCP dengan konektor dan mengirim bingkai data. Koneksi WebSocket bersifat stateful dan memungkinkan komunikasi dupleks penuh melalui koneksi TCP tunggal. Produsen data memublikasikan data ke titik akhir WebSocket yang diterbitkan oleh konektor. Pelanggan berlangganan ke titik akhir WebSocket dan menerima data dari layanan web WebSocket.

Tidak seperti komunikasi respons-permintaan dengan REST, WebSockets memungkinkan komunikasi dupleks penuh dan tidak memerlukan koneksi baru untuk diatur untuk setiap pesan yang akan dikirim. Komunikasi WebSocket dimulai dengan permintaan penyiapan koneksi yang dikirim oleh klien ke server. Permintaan ini (disebut handshake WebSocket) dikirim melalui HTTP dan server menafsirkannya sebagai permintaan peningkatan. Jika server mendukung protokol WebSocket, server merespons respons handshake WebSocket. Setelah koneksi diatur, klien dan server dapat saling mengirim data / pesan dalam mode dupleks penuh. Tidak ada biaya tambahan untuk penyiapan koneksi dan permintaan penghentian untuk setiap pesan. Komunikasi WebSocket cocok untuk aplikasi yang memiliki persyaratan latensi rendah atau throughput tinggi.




Gambar 5.10: *Konektor kustom berbasis WebSocket*

### 5.5.3 Konektor berbasis MQTT

MQTT (MQ Telemetry Transport) adalah protokol pengiriman pesan langganan ringan yang dirancang untuk perangkat terbatas. MQTT cocok untuk aplikasi Internet of Things (IoT) yang melibatkan perangkat yang mengirimkan data sensor ke server atau backend analisis berbasis cloud untuk diproses dan dianalisis.

Entitas yang terlibat dalam MQTT meliputi:

- Penerbit: Penerbit adalah komponen yang mempublikasikan data ke topik yang dikelola oleh Pialang.
- Broker / Server: Broker mengelola topik dan meneruskan data yang diterima pada suatu topik ke semua pelanggan yang berlangganan topik tersebut.
- Pelanggan: Pelanggan adalah komponen yang berlangganan topik dan menerima data yang diterbitkan tentang topik oleh penerbit.

**Menerapkan Konektor Khusus berbasis MQTT**

Mari kita lihat contoh penerapan konektor berbasis MQTT kustom. Box 5.27 dan 5.28 menunjukkan implementasi Python dari komponen pelanggan dan penerbit MQTT. Komponen pelanggan dalam contoh ini berjalan di server, yang juga menjalankan Broker MQTT. Komponen penerbit berjalan di perangkat yang perlu mempublikasikan data ke server. Perangkat mempublikasikan data ke topik MQTT (misalnya $ iot / test). Pelanggan yang berlangganan topik menerima data dan memproses atau meneruskan data. Tindakan penerusan mungkin termasuk meneruskan data ke antrian pesan, menulis data ke database NoSQL atau menyimpan data ke sistem file terdistribusi.

Manfaat menggunakan koneksi khusus berbasis MQTT atau adalah memisahkan klien dan server (dalam dimensi ruang, waktu dan sinkronisasi). Dengan pemisahan spasi, yang kami maksud adalah klien dan server tidak perlu mengetahui satu sama lain. Pemisahan waktu berarti klien dan server tidak perlu dijalankan secara bersamaan. Synchronization decoupling berarti bahwa komunikasi antara klien dan server dapat terjadi secara asynchronous. Klien dan server tidak perlu menunggu sementara pesan sedang diproses.

**■ Box 5.27: Python implementation of a MQTT-based custom connector (subscriber)**

```
import paho.mqtt.client as mqtt
import json
import boto.sqs
from boto.sqs.message import Message

REGION="us-east-1"
queue_name = 'sensordata'
ACCESS_KEY = <Enter AWS Access Key>
SECRET_KEY = <Enter Secret Key>

conn = boto.sqs.connect_to_region(REGION,aws_access_key_id=ACCESS_KEY,
                        aws_secret_access_key=SECRET_KEY)

q = conn.get_all_queues(prefix=queue_name)


def publish_to_sqs(data):
   m = Message()
   m.set_body(data)
   status = q[0].write(m)
   return status


def on_connect(client, userdata, flags, rc):
   print("Connected with result code "+str(rc))
   client.subscribe("$iot/test")

def on_message(client, userdata, msg):
   data = json.loads(msg.payload)
   publish_to_sqs(data)


client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message

client.connect("localhost", 1883, 60)

client.loop_forever()
```

**■ Box 5.28: Python implementation of a MQTT-based custom connector (publisher)**

```
import paho.mqtt.client as mqtt
import paho.mqtt.publish as publish
import time
from random import randint
import datetime
import requests
```

```
import json

def getData():
   ts=time.time()
   timestamp = datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-%d %H:%M:%S')
   temp = str(randint(0,100))
   humidity = str(randint(0,100))
   co2 = str(randint(50,500))
   light = str(randint(0,10000))

   data = {"timestamp":  timestamp,
      "temperature":  temp, "humidity":  humidity ,
      "co2":  co2, "light":  light}

   return data


def publish_to_topic(data):
   publish.single("$iot/test", payload=json.dumps(data),
           hostname="localhost")

while True:
   data = getData()
   print data
   publish_to_topic(data)
   time.sleep(1)
```

### 5.5.4 Amazon IoT

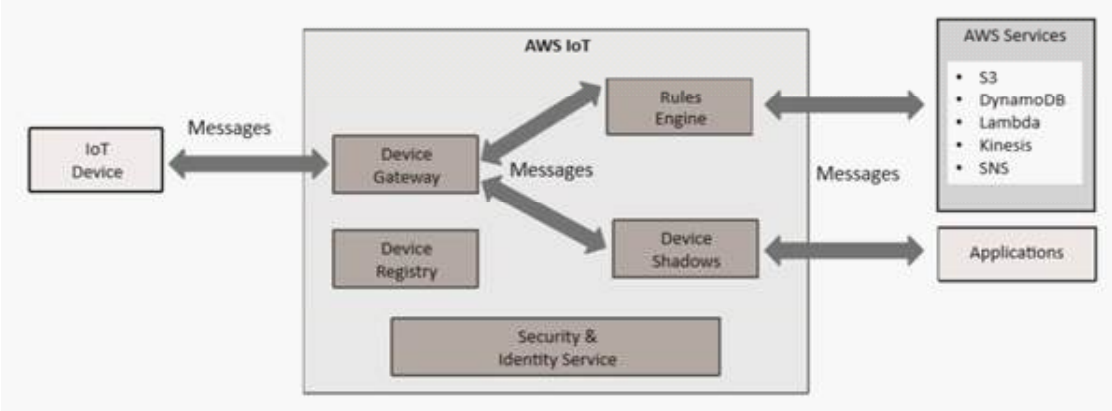
Amazon IoT adalah layanan untuk mengumpulkan data dari perangkat Internet of Things (IoT) (seperti sensor dan peralatan pintar) ke dalam cloud AWS. Data yang dikumpulkan dapat dikirim ke berbagai layanan AWS, mis. disimpan dalam database Amazon DynamoDB, disimpan dalam file di S3, dikirim ke aliran data Amazon Kinesis, dikirim ke Amazon SNS sebagai pemberitahuan push dan dimasukkan ke dalam kode untuk mengeksekusinya dengan layanan Amazon Lambda.

Gambar 5.11 menunjukkan berbagai komponen layanan AWS IoT.

- Device Gateway: Device Gateway memungkinkan perangkat untuk berkomunikasi dengan AWS IoT menggunakan protokol MQTT atau HTTP. Perangkat dapat menerbitkan atau berlangganan topik.
- Device Registry: Device Registry (juga disebut registri hal-hal) memelihara sumber daya yang terkait dengan setiap perangkat termasuk atribut, sertifikat, dan meta-data.
- Device Shadow: Device shadow mempertahankan status perangkat sebagai dokumen JSON. Aplikasi dapat mengambil atau memperbarui status perangkat menggunakan AWS IoT REST API. Bayangan perangkat mempertahankan status perangkat bahkan

saat perangkat mati. Saat perangkat online, status disinkronkan dengan bayangan perangkat.

- Rule Engine: Rule Engine memungkinkan Anda untuk menentukan aturan untuk memproses pesan yang diterima dari perangkat. Dengan menggunakan bahasa seperti SQL, Anda dapat menentukan aturan untuk memilih data, memproses data, dan mengirim data ke layanan AWS lainnya seperti DynamoDB, S3, Kinesis, SNS, dan Lambda.
- Security and Identity Service: Layanan ini memungkinkan perangkat untuk bertukar data secara aman dengan layanan AWS IoT. Untuk perangkat yang berkomunikasi melalui MQTT, otentikasi berbasis sertifikat digunakan. Sertifikat memiliki kebijakan yang terkait dengannya yang memberi otorisasi perangkat untuk mengakses sumber daya tertentu.




Gambar 5.11: *Komponen Amazon IoT*

Mari kita lihat beberapa contoh penggunaan layanan AWS IoT. Langkah pertama adalah membuat sesuatu dari dasbor AWS IoT seperti yang ditunjukkan pada Gambar 5.12. Benda mewakili perangkat di layanan AWS IoT. Saat sesuatu dibuat, entri dibuat di registri perangkat untuk perangkat dan bayangan perangkat juga dibuat. Pada langkah ini, Anda juga dapat menambahkan atribut opsional untuk mendeskripsikan kemampuan perangkat.


Gambar 5.12: *Membuat Sesuatu dari dasbor Amazon IoT*

Pada langkah berikutnya, kami membuat sertifikat yang digunakan oleh perangkat untuk menghubungkan ke AWS IoT. Sertifikat melekat pada sesuatu. Tiga file dibuat dalam langkah ini - file sertifikat, file kunci publik, dan file kunci pribadi. Selanjutnya, kami membuat kebijakan dan melampirkan kebijakan tersebut ke sertifikat untuk menetapkan izin.

Box 5.29 menunjukkan contoh Python untuk menerbitkan pesan ke AWS IoT. Contoh ini menggunakan klien Paho Python MQTT. Untuk menghubungkan ke AWS IoT, diperlukan sertifikat Root Certi fi cate Authority (CA), sertifikat klien dan file kunci pribadi. Contoh ini menyimulasikan perangkat termostat yang mengirimkan status saat ini (suhu) ke AWS IoT, yang menyimpan status tersebut dalam bayangan perangkat. Untuk melaporkan status melalui MQTT, sebuah pesan diterbitkan dengan topik $ aws / things / thingName / shadow / update.

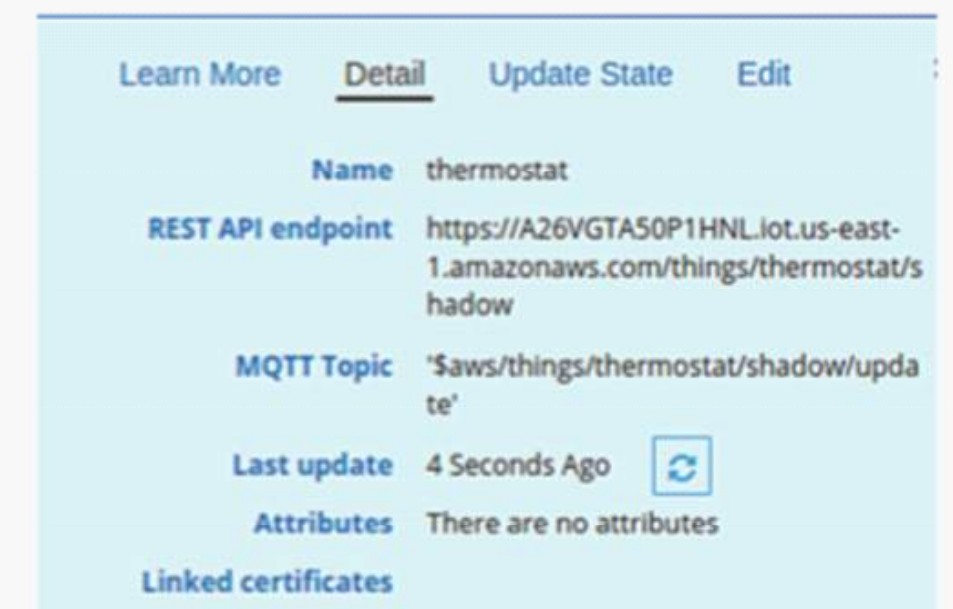
**■ Box 5.29: Python code for publishing messages to AWS IoT**

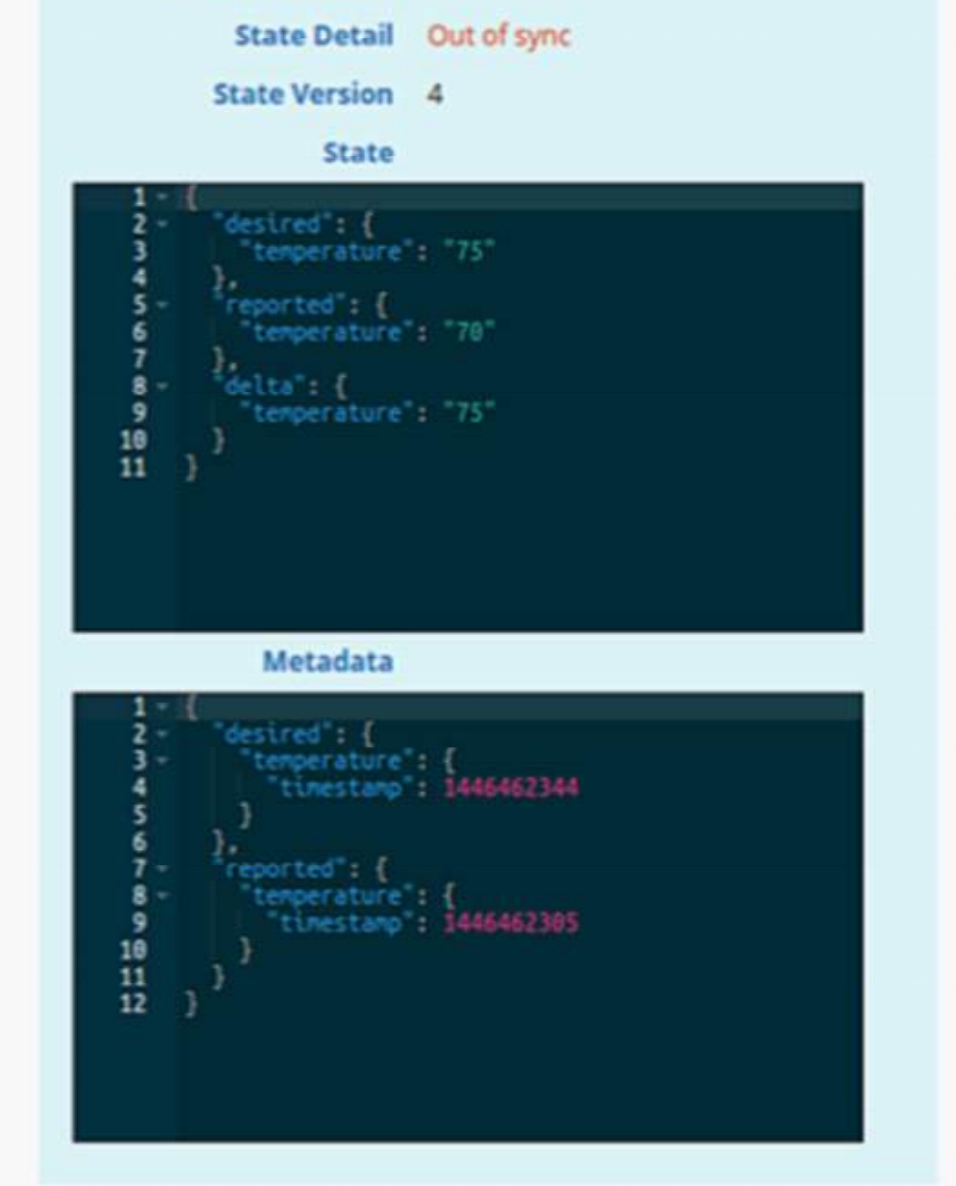
```
import paho.mqtt.client as mqtt
import ssl
import paho.mqtt.publish as publish

connection={
"host":  "A26VGTA50P1HNL.iot.us-east-1.amazonaws.com",
"port":  8883,
"clientId":  "thermostat",
"thingName":  "thermostat",
"caCert":  "root-CA.crt",
"clientCert":  "9795072c41-certificate.pem.crt",
"privateKey":  "9795072c41-private.pem.key"
}

tlsdict= {'ca_certs':connection['caCert'],
  'certfile':connection['clientCert'],
  'keyfile':connection['privateKey'],
```

```
  'tls_version':ssl.PROTOCOL_SSLv23, 'ciphers':None}

state="{ \"state\":  {\"reported\":  { \"temperature\":  \"70\" } } }"

publish.single("$aws/things/thermostat/shadow/update", payload=str(state)
 qos=1, retain=False, hostname=connection['host'],
 port=8883, client_id=connection['clientId'], keepalive=60,
 will=None, auth=None, tls=tlsdict,
 protocol=mqtt.MQTTv311)
```

Status perangkat saat ini dapat dilihat dari dasbor AWS IoT seperti yang ditunjukkan pada Gambar 5.13. Box 5.30 menunjukkan contoh Python untuk berlangganan pembaruan status suatu perangkat. Untuk menerima update dari device shadow over MQTT, perangkat / aplikasi dapat berlangganan topik $ aws / things / thingName / shadow / update / accept.

Aplikasi juga dapat menggunakan AWS IoT REST API untuk menanyakan status terakhir perangkat yang dilaporkan atau memperbarui status perangkat. Misalnya, aplikasi seluler yang mengontrol pengaturan suhu untuk termostat cerdas dapat dibuat. Termostat melaporkan keadaan saat ini (suhu) ke AWS IoT, dan keadaan disimpan dalam bayangan perangkat. Aplikasi seluler dapat memperbarui status yang diinginkan dalam bayangan perangkat alih-alih berkomunikasi langsung dengan termostat. Status yang diinginkan disinkronkan dengan perangkat, saat berikutnya terhubung ke layanan AWS IoT. Status perangkat juga dapat diperbarui dari dasbor AWS IoT seperti yang ditunjukkan pada Gambar 5.14.





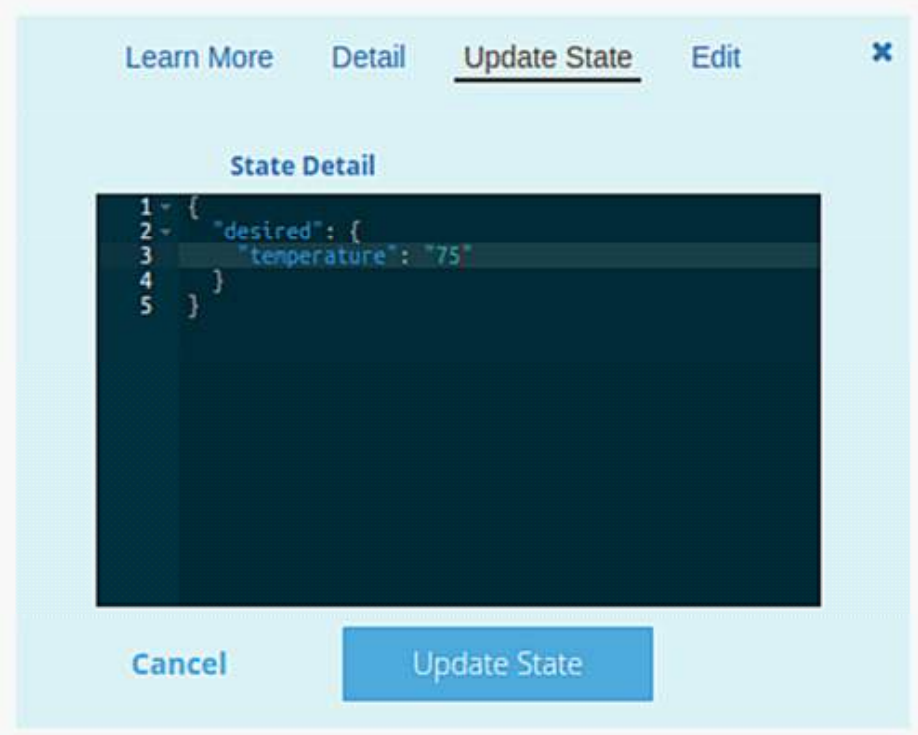

Gambar 5.13: *Melihat status sesuatu dari dasbor Amazon IoT*

**Box 5.30: Python code for subscribing to a topic in AWS IoT**

```
import paho.mqtt.client as mqtt
import ssl
```

```
connection={
"host": "A26VGTA50P1HNL.iot.us-east-1.amazonaws.com",
"port": 8883,
"clientId": "thermostat",
"thingName": "thermostat",
"caCert": "root-CA.crt",
"clientCert": "9795072c41-certificate.pem.crt",
"privateKey": "9795072c41-private.pem.key"
```

```
}

def on_connect(client, userdata, flags, rc):
 print("Connected with result code "+str(rc))
 client.subscribe("$aws/things/rpi/shadow/update/accepted")

def on_message(client, userdata, msg):
 print(msg.topic+" "+str(msg.payload))

client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message

client.tls_set(ca_certs=connection['caCert'],
    certfile=connection['clientCert'],
    keyfile=connection['privateKey'], cert_reqs=ssl.CERT_REQUIRED,
    tls_version=ssl.PROTOCOL_SSLv23, ciphers=None)

client.connect(connection['host'], connection['port'])

client.loop_forever()
```




Gambar 5.14: *Memperbarui status sesuatu dari dasbor Amazon IoT*

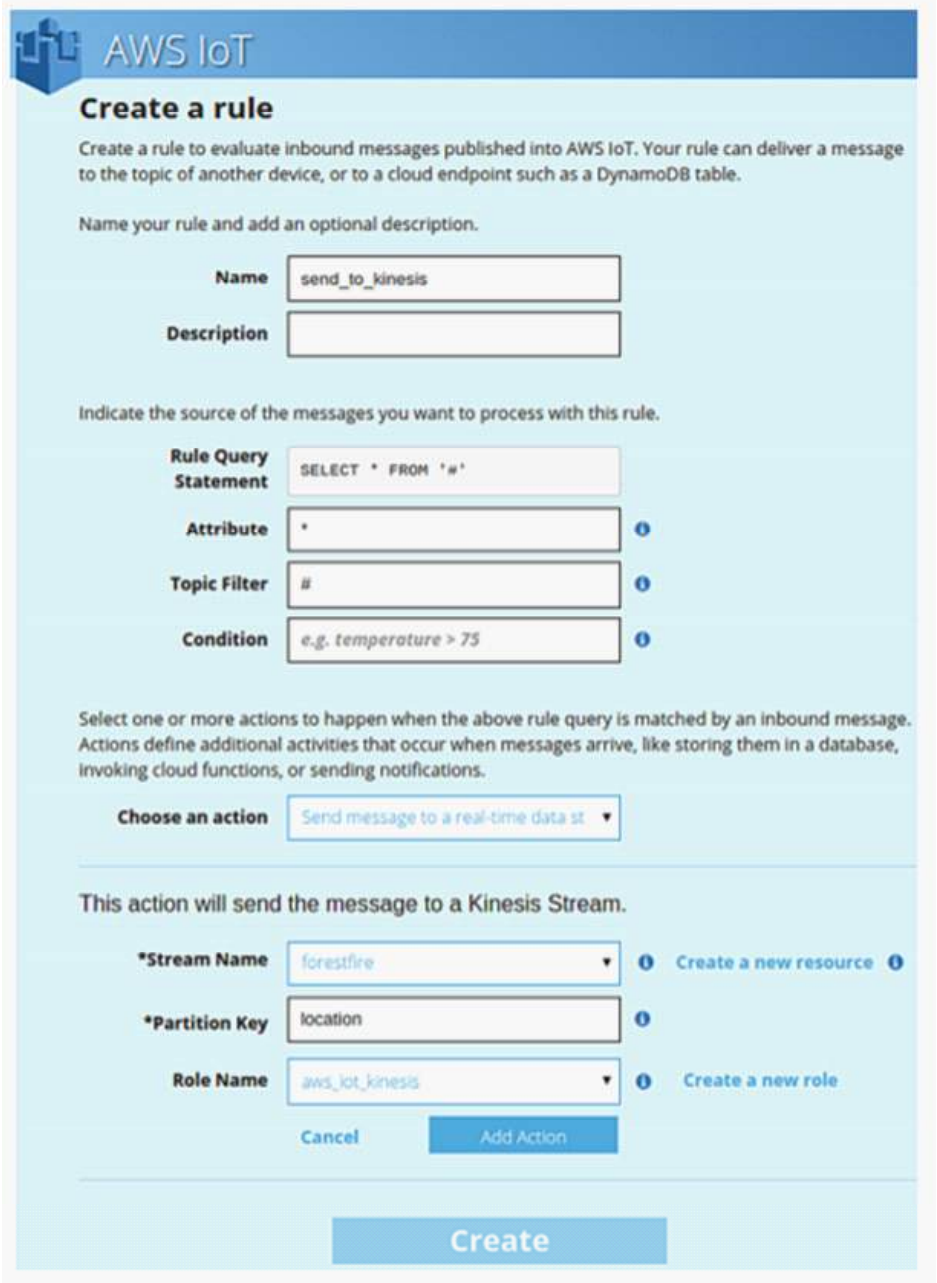
Sekarang mari kita lihat contoh yang lebih maju di mana kita menggunakan mesin aturan untuk mengirim data yang dikumpulkan dari perangkat ke layanan AWS yang berbeda. Untuk contoh ini, kami akan membuat hal baru yang disebut 'hutan' yang mewakili perangkat yang digunakan di hutan untuk melaporkan data yang dikumpulkan dari

berbagai sensor (suhu, kelembaban, cahaya, CO2). Data yang dikumpulkan dari berbagai perangkat yang ditempatkan di hutan dapat dianalisis untuk mendeteksi kebakaran hutan. Meskipun Anda dapat menggunakan AWS IoT Starter Kits untuk membangun perangkat fisik dengan sensor nyata yang terhubung dengannya, untuk kesederhanaan kami akan menggunakan program yang menghasilkan dan mengirimkan data sintetis ke AWS IoT. Box 5.31 menunjukkan program Python untuk mengirim data sensor sintetis ke AWS IoT.

**Box 5.31: Python program for sending synthetic sensor data to AWS IoT**

```
from random import randrange
import time
import datetime
import paho.mqtt.client as mqtt
import ssl
import paho.mqtt.publish as publish

connection={
"host":  "A26VGTA50P1HNL.iot.us-east-1.amazonaws.com",
"port":  8883,
"clientId":  "forest",
"thingName":  "forest",
"caCert":  "root-CA.crt",
"clientCert":  "9795072c41-certificate.pem.crt",
"privateKey":  "9795072c41-private.pem.key"
}

tlsdict= {'ca_certs':connection['caCert'],
   'certfile':connection['clientCert'],
   'keyfile':connection['privateKey'],
   'tls_version':ssl.PROTOCOL_SSLv23, 'ciphers':None}

#Send some synthetic data to AWS IoT
while True:
 ts=time.time()

 data = "{ \"state\":  { \"location\":  \"123\",
  \"timestamp\":  \""+str(ts)+"\",
  \"temperature\":  "+str(randrange(0,60))+",
 \"humidity\":  "+str(randrange(0,60))+",
  \"light\":  "+str(randrange(0,60))+", \"co2\":  "+str(randrange(0,60))+"
}}"
 print data

 publish.single("$aws/things/forest/test", payload=str(data),
  qos=1, retain=False, hostname=connection['host'],
  port=8883, client_id=connection['clientId'],
  keepalive=60, will=None, auth=None, tls=tlsdict,
  protocol=mqtt.MQTTv311)

 time.sleep(1)
```

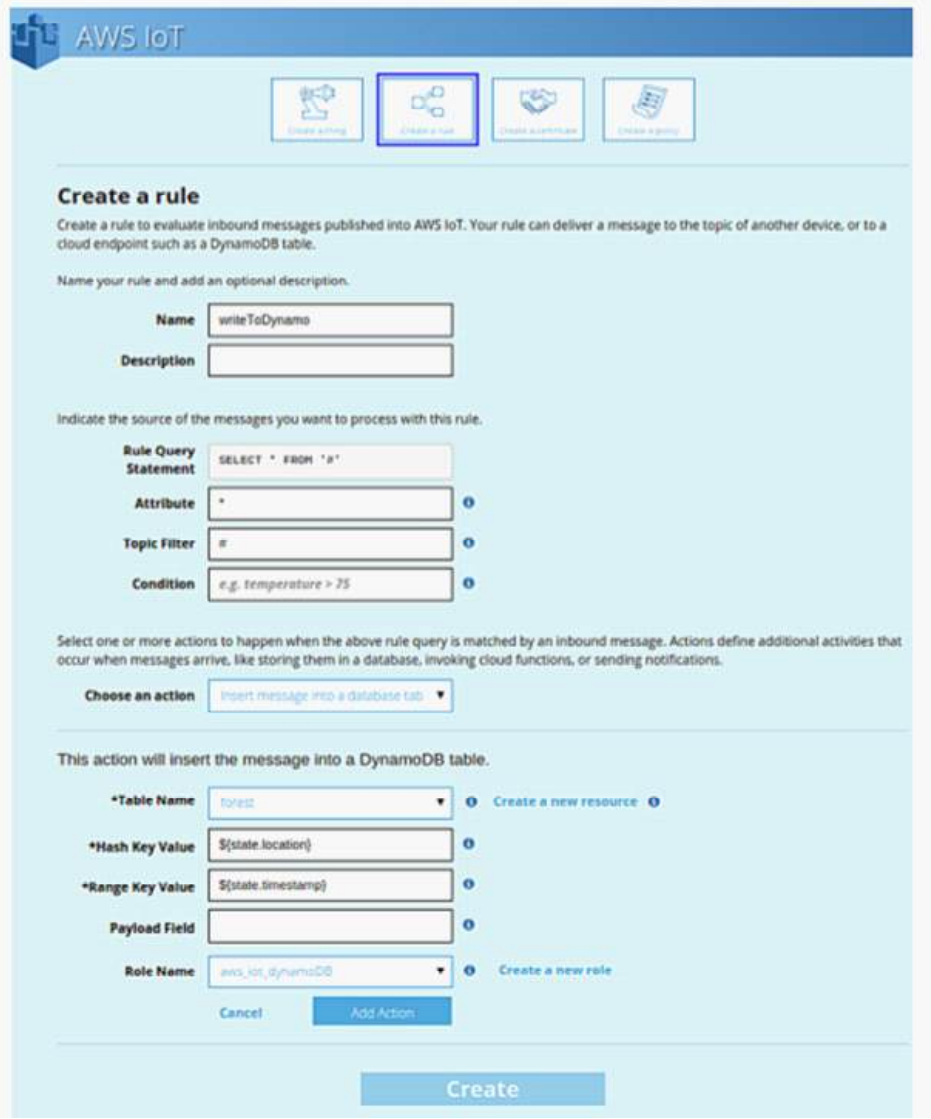
Selanjutnya, kami membuat dua aturan berbeda di AWS IoT untuk menganalisis data ini lebih lanjut. Aturan pertama seperti yang ditunjukkan pada Gambar 5.15 mengirimkan data ke aliran data Amazon Kinesis. Aturan kedua seperti yang ditunjukkan pada Gambar 5.16 menulis data ke tabel Amazon DynamoDB.



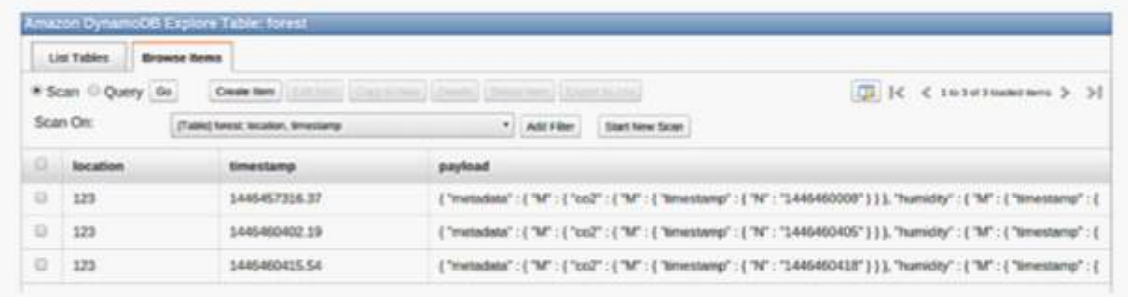

Gambar 5.15: *Membuat aturan dari dasbor Amazon IoT*

Dengan aturan yang sudah ditentukan, jalankan program Python di Box 5.31. Data sintetis yang dihasilkan oleh program ini akan dipublikasikan ke topik $ aws / things / forest / test di AWS IoT. Aturan akan mengirimkan data ke Amazon Kinesis dan Amazon DynamoDB.

Gambar 5.17 menunjukkan screenshot tabel Amazon DynamoDB dengan data yang diterbitkan oleh perangkat. Untuk membaca data dari aliran data Kinesis, Anda dapat menggunakan program yang ditunjukkan pada Box 5.8.



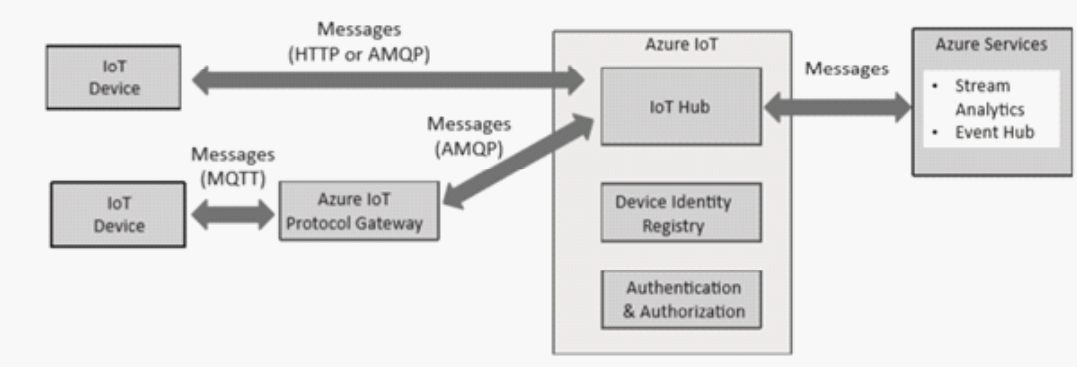

Gambar 5.16: *Membuat aturan dari dasbor Amazon IoT*




Gambar 5.17: *Melihat data yang disimpan oleh aturan AWS IoT ke dalam tabel DynamoDB*

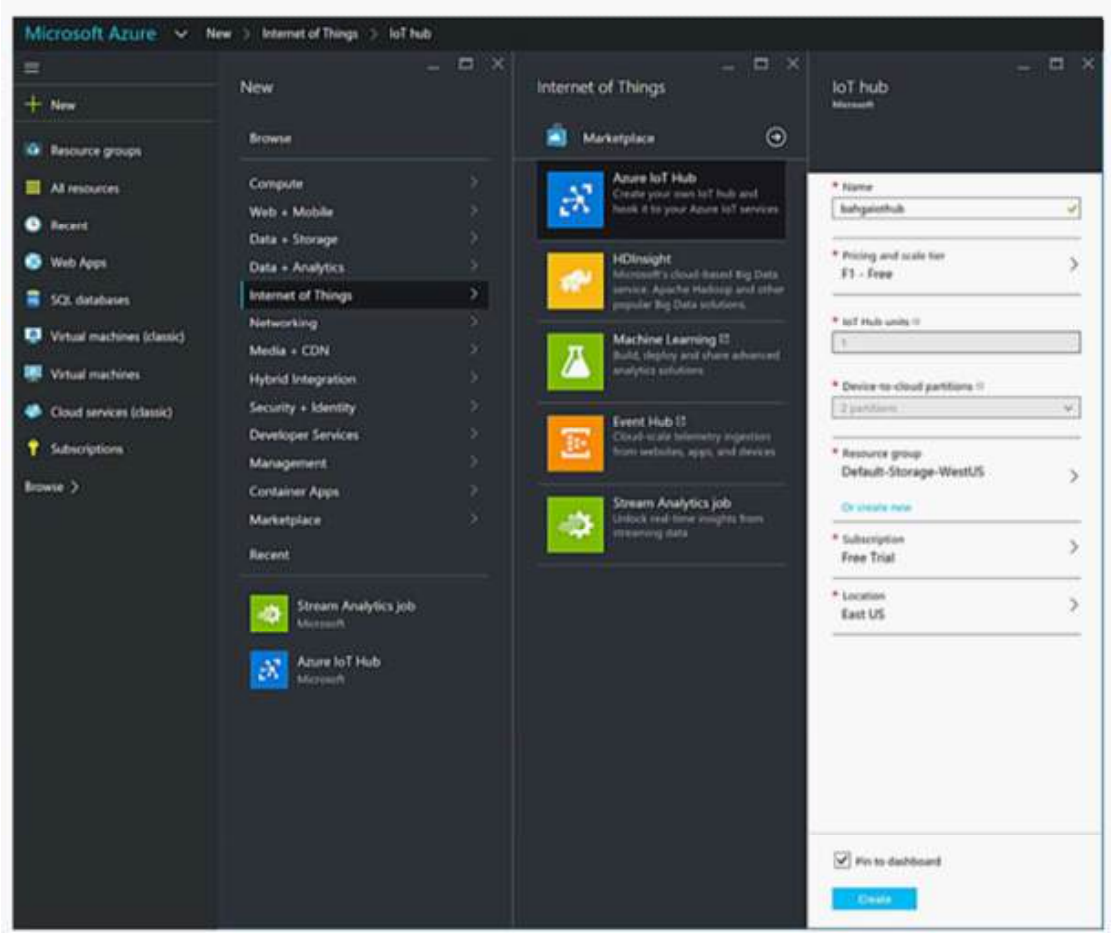
### 5.5.5 Azure IoT Hub

Azure IoT Hub adalah layanan terkelola sepenuhnya untuk komunikasi dua arah antara perangkat IoT dan cloud Azure. Azure IoT Hub menerima pesan dari perangkat IoT dan mengirimkannya ke berbagai layanan Azure (seperti Azure Stream Analytics) untuk pemrosesan pesan lebih lanjut. IoT Hub dapat menyimpan data hingga 7 hari. Aplikasi dapat menggunakan IoT Hub untuk mengirim pesan ke perangkat. Azure menyediakan perpustakaan perangkat untuk menghubungkan berbagai perangkat ke IoT Hub. Protokol yang didukung termasuk HTTP 1.1 dan AMQP 1.0. Dukungan untuk MQTT dapat ditambahkan dengan menjalankan Azure IoT Protocol Gateway, komponen sumber terbuka, yang dapat dijalankan secara lokal atau di cloud. IoT Hub menyertakan registri identitas perangkat yang digunakan untuk menyediakan perangkat dengan kunci keamanannya sendiri untuk menyambungkan ke IoT Hub dengan aman. Gambar 5.18 menunjukkan berbagai komponen Azure IoT Hub.




Gambar 5.18: *Komponen Azure IoT*

Pada bagian sebelumnya, kami menjelaskan contoh perangkat IoT yang digunakan di hutan untuk melaporkan data yang dikumpulkan dari berbagai sensor (suhu, kelembaban, cahaya, CO2) untuk mendeteksi kebakaran hutan. Mari kita ulangi contoh yang sama menggunakan Azure IoT Hub. Langkah pertama adalah membuat IoT Hub yang akan menerima data dari perangkat. Masuk ke Azure Preview Portal dan buat IoT Hub baru seperti yang ditunjukkan pada Gambar 5.19. Setelah IoT Hub dibuat, buka ubin hub IoT di Portal Pratinjau, dan catat Nama Host IoT Hub. Selanjutnya, pilih ikon Key di IoT Hub dan klik kebijakan akses bersama pemilik iothubowner seperti yang ditunjukkan pada Gambar 5.20. Catat string koneksi dan kunci utama.



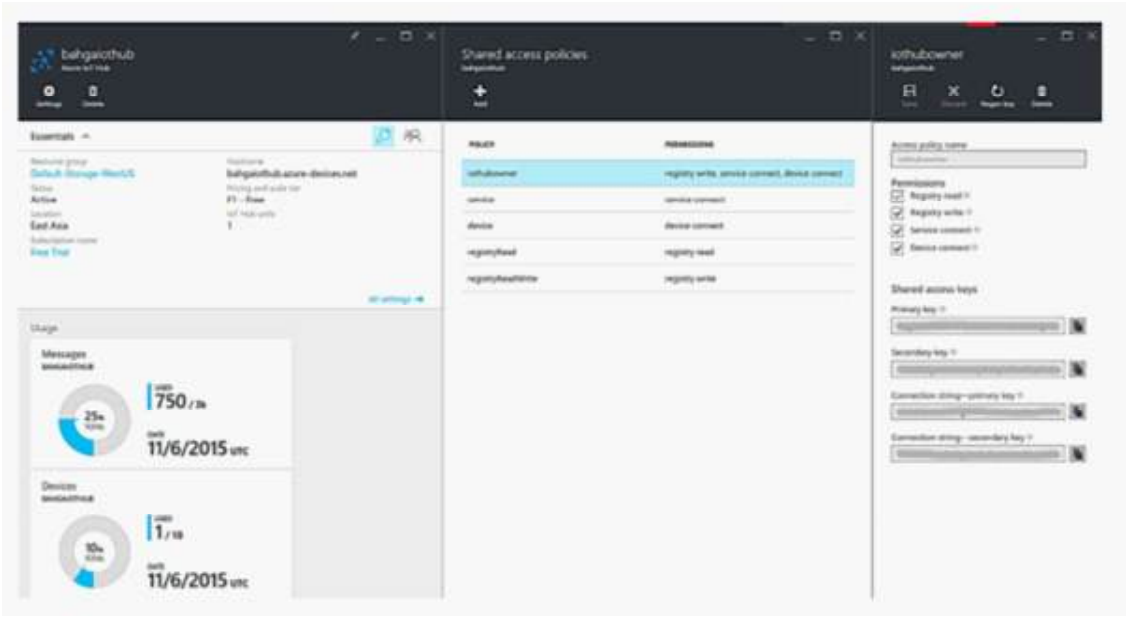

Sekarang IoT Hub sudah beroperasi, mari kita daftarkan perangkat dengan Hub. Untuk membuat identitas perangkat baru, Anda dapat menggunakan alat mandiri yang disebut Device Explorer (yang berjalan di Windows) atau alat NodeJS yang disebut iothub-explorer. NodeJS dapat diinstal sebagai berikut:

```
■ #Installing NodeJS
curl -sL https://deb.nodesource.com/setup_4.x | sudo -E bash -
sudo apt-get install -y nodejs
```

Selanjutnya, instal alat iothub-explorer dan kemudian buat identitas unik dan string koneksi sebagai berikut:

```
■ #Create a new device identity in the IoT Hub
npm install -g iothub-explorer
node iothub-explorer "<enter iothubowner connection string>"
create mydevice -connection-string
```

Catat string koneksi perangkat yang dihasilkan oleh iothub-explorer. Saat menulis buku ini, pustaka dukungan Python untuk IoT Hub belum dirilis. Oleh karena itu, kami akan memberikan contoh menggunakan NodeJS. Box 5.32 menunjukkan contoh sederhana pengiriman data ke IoT Hub menggunakan NodeJS. Dalam contoh ini, gunakan string koneksi perangkat yang dibuat oleh alat iothub-explorer. Program ini menghasilkan beberapa data sintetis acak dan mengirimkannya ke IoT Hub setiap detik.




Gambar 5.20: *Melihat detail hub Azure IoT*

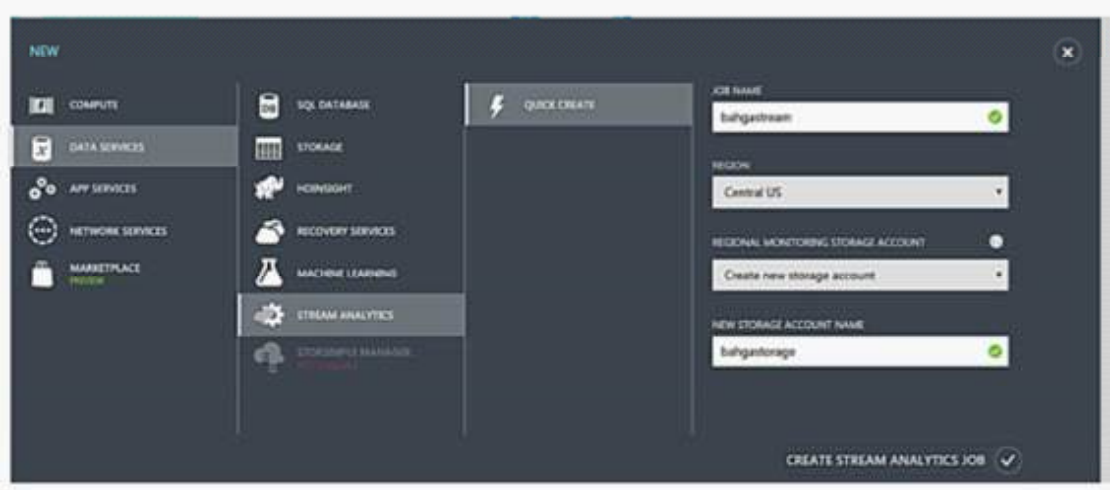
**Box 5.32: NodeJS code for sending data to Azure IoT Hub**

```
var device = require('azure-iot-device');
var connectionString = '<enter>';
var client = new device.Client(connectionString, new device.Https());

// Send some synthetic data to IoT Hub every second
setInterval(function(){
 var temperature = Math.random() * 100 ;
 var humidity = Math.random() * 100 ;
 var light = Math.random() * 10000 ;
 var co2 = Math.random() * 300 ;

 var data = "{deviceid:" + "mydevice" + ",temperature:" +
   String(temperature) + ", humidity:" + String(humidity) +
   ", light:" + String(light) + ", co2:" + String(co2) + " }";

 var message = new device.Message(data);
 console.log("Sending message:  " + message.getData());
 client.sendEvent(message);
}, 1000);
```

Untuk menjalankan program gunakan perintah yang ditampilkan di box di bawah ini:

```
■ #Running NodeJS program shown in Box 5.32
npm install node .
```

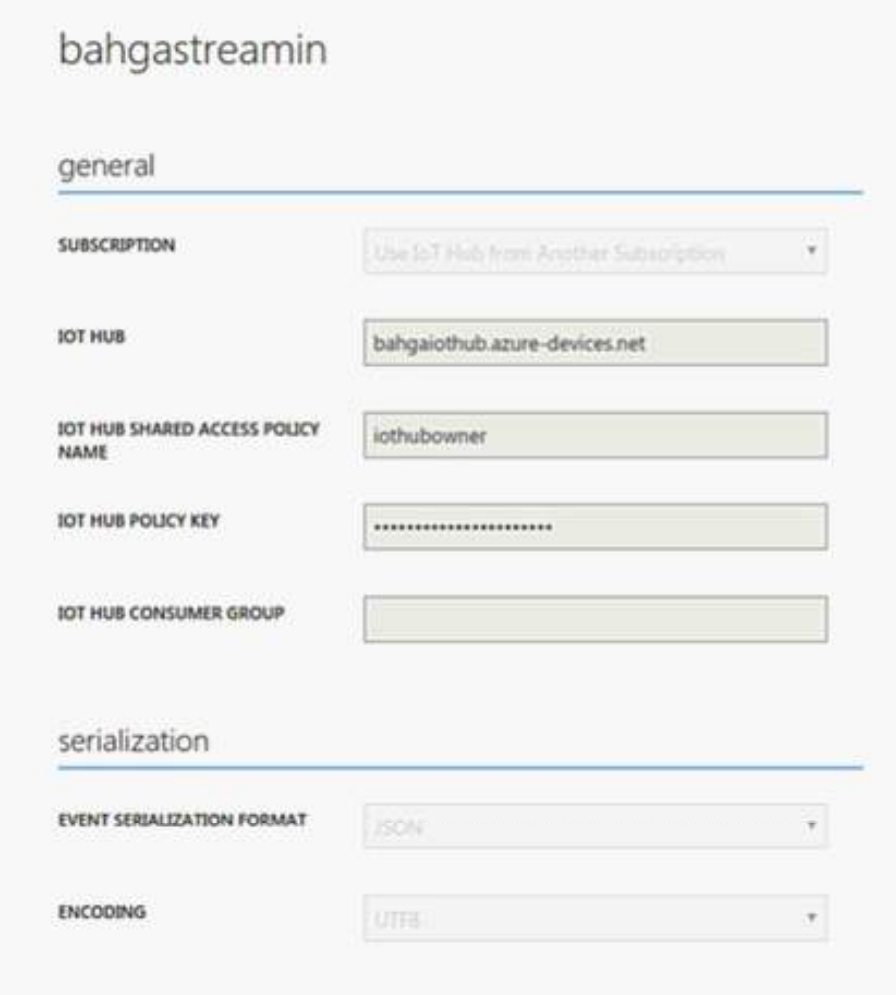
Anda akan dapat melihat jumlah pesan yang diterima dan perangkat yang terhubung di dasbor IoT Hub seperti yang terlihat pada Gambar 5.20. Sekarang kita dapat mengirim pesan ke IoT Hub, mari kita tentukan beberapa aturan untuk pemrosesan pesan lebih lanjut. Dalam kasus AWS IoT, kami menggunakan Mesin Aturan untuk mendefinisikan aturan menggunakan bahasa mirip SQL. Azure menyediakan mesin pemroses kejadian waktu-sareal yang disebut Azure Stream Analytics. Azure Stream Analytics memungkinkan penentuan perhitungan analisis real-time pada streaming data menggunakan bahasa seperti SQL (disebut bahasa kueri Stream Analytics). Mari kita buat pekerjaan Stream Analytics seperti yang ditunjukkan pada Gambar 5.21 dari dasbor Azure. Tugas Stream Analytics mencakup sumber input data streaming, kueri yang diekspresikan dalam bahasa seperti SQL, dan sink output yang menjadi tujuan pengiriman hasil. Gambar 5.22, 5.23 dan 5.24 menunjukkan setelan untuk masukan, kueri, dan output dari tugas Stream Analytics. Sumber input, dalam hal ini, adalah IoT Hub yang kami buat sebelumnya dan output sink adalah Azure Peristiwa Hub. Peristiwa Hub adalah layanan terkelola untuk mengumpulkan dan memproses data dalam jumlah besar dengan latensi rendah secara andal. Peristiwa Hub menyediakan fungsionalitas yang mirip dengan Amazon Kinesis.



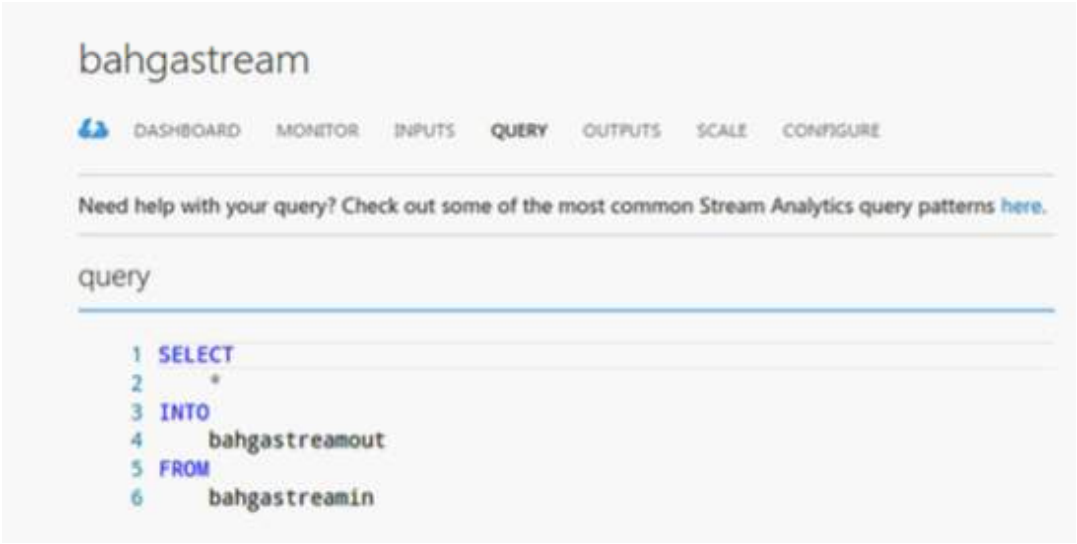

Gambar 5.21: *Membuat tugas analisis aliran Azure*

Gambar 5.25 menunjukkan cara membuat Peristiwa Hub baru dari dasbor Azure. Setelah Peristiwa Hub dibuat, buka tab konfigurasi dan tambahkan kebijakan akses bersama baru (dengan Name = "read-write" dan Permissions = Send, Listen). Salin Kunci Utama untuk kebijakan baca-tulis. Nama kebijakan dan kunci utama ini digunakan saat membuat sink output untuk tugas Stream Analytics seperti yang ditunjukkan pada Gambar 5.24.
Selanjutnya, jalankan acara program di Box 5.32 dan pantau tugas Analisis Aliran dan Pusat Acara dari dasbor Azure. Anda akan dapat melihat pesan yang sedang diproses oleh tugas Stream Analytics dan outputnya diposkan ke Pusat Peristiwa seperti yang terlihat pada Gambar 5.26 dan 5.27.

```
■ #Running the Javascript code in Box 5.32
npm install node .
```



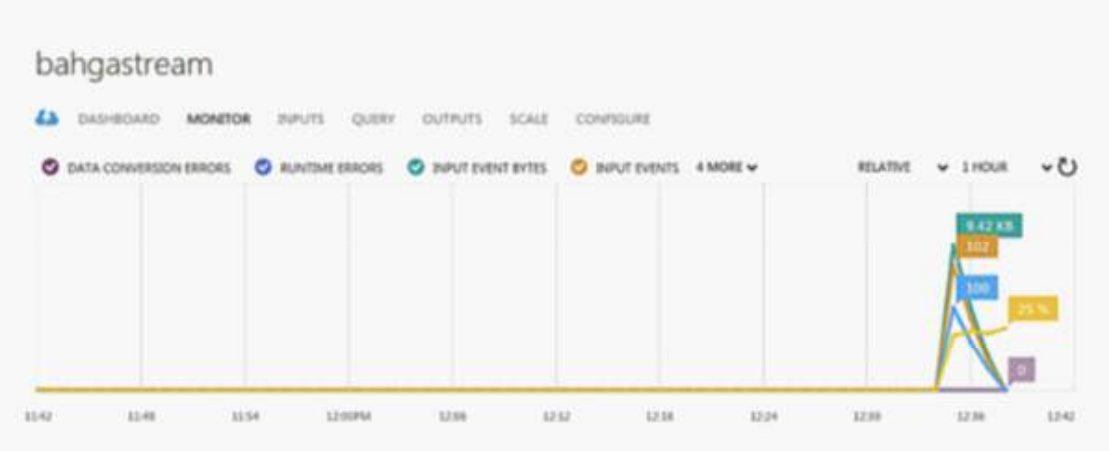

Gambar 5.22: *Pengaturan input untuk pekerjaan analisis aliran*




Gambar 5.23: *Pengaturan kueri untuk tugas analisis aliran*


Gambar 5.24: *Pengaturan output untuk tugas analisis aliran*


Gambar 5.25: *Membuat Pusat Acara dari dasbor Azure*




Gambar 5.26: *Output Azure Peristiwas Hub*


Gambar 5.27: *Output Azure Peristiwas Hub*

## Ringkasan

Dalam bab ini, kami menjelaskan berbagai konektor data yang memungkinkan pengumpulan data dari sumber raw data untuk diserap ke dalam sistem file terdistribusi atau database NoSQL, untuk analisis batch data, atau yang menghubungkan sumber data ke streaming atau kerangka kerja pemrosesan dalam memori untuk analisis data secara real-time. Kami menjelaskan model perpesanan publikasikan-berlangganan dan dorong-tarik. Publish-Subscribe adalah model komunikasi yang melibatkan penerbit, broker dan konsumen. Penerbit mengirimkan data ke topik yang dikelola oleh broker. Ketika broker menerima data untuk suatu topik dari penerbit, ia mengirimkan data tersebut ke semua konsumen yang berlangganan. Kami menjelaskan kerangka kerja perpesanan langganan-publikasi Apache Kafka dan Amazon Kinesis. Selanjutnya, kami menjelaskan kerangka pengumpulan data sumber-sink yang disebut Apache Flume. Apache Flume adalah sistem terdistribusi, andal, dan tersedia untuk mengumpulkan, menggabungkan, dan memindahkan sejumlah besar data dari sumber data yang berbeda ke dalam penyimpanan data terpusat. Selanjutnya, kami menjelaskan Apache Sqoop, yang merupakan alat yang memungkinkan mengimpor data dari sistem manajemen database relasional (RDBMS) ke dalam tabel HDFS, Hive atau HBase. Kami menjelaskan berbagai antrean perpesanan seperti RabbitMQ, ZeroMQ, RestMQ, dan Amazon SQS. Contoh membangun konektor kustom berbasis REST dan berbasis MQTT disediakan. Terakhir, kami menjelaskan layanan IoT dari Amazon dan Azure yang memungkinkan pengumpulan data dari perangkat Internet of Things (IoT) ke cloud, di mana data tersebut dapat diproses lebih lanjut.

# Big Data Storage

Bab 6

Bab ini mencakup

- Hadoop Distributed File System (HDFS)

Di bab sebelumnya, kami menjelaskan alat dan kerangka kerja untuk akuisisi data dari berbagai jenis sumber dan menelan data ke dalam big data stack. Opsi untuk penyimpanan data dalam big data stack mencakup sistem file terdistribusi atau database NoSQL. Dalam bab ini, kami akan menjelaskan Hadoop Distributed File System (HDFS) untuk penyimpanan big data. Setelah data dipindahkan dari sumber data ke HDFS, kita dapat menggunakan kerangka kerja khusus untuk analisis batch atau kueri interaktif untuk menganalisis data.

## 6.1. HDFS

HDFS adalah sistem file terdistribusi (DFS) yang berjalan pada cluster besar dan menyediakan akses throughput tinggi ke data. HDFS adalah sistem yang sangat toleran terhadap kesalahan dan dirancang untuk bekerja dengan perangkat keras komoditas. HDFS menyimpan setiap file sebagai urutan blok. Blok dari setiap file direplikasi pada beberapa mesin dalam sebuah cluster untuk memberikan toleransi kesalahan. Mari kita lihat ciri-ciri HDFS:

- Scalable Storage for Large Files: HDFS telah dirancang untuk menyimpan file besar (biasanya berukuran dari gigabyte hingga terabyte). File besar dipecah menjadi potongan atau blok dan setiap blok direplikasi di beberapa mesin dalam cluster. HDFS telah dirancang untuk menskalakan cluster yang terdiri dari ribuan node.
- Replikasi: HDFS mereplikasi blok data ke beberapa mesin dalam cluster yang membuat sistem dapat diandalkan dan toleran terhadap kesalahan. Ukuran blok default yang digunakan adalah 64MB dan faktor replikasi default adalah 3.
- Streaming Data Access: HDFS telah dirancang untuk pola akses data streaming dan menyediakan streaming baca dan tulis dengan output yang tinggi. Desain HDFS melonggarkan beberapa persyaratan POSIX untuk mengaktifkan akses data streaming dan membuatnya sesuai untuk operasi batch sehingga mengurangi kemampuan akses interaktif. Pilihan desain ini telah dibuat untuk memenuhi persyaratan aplikasi

yang melibatkan pola akses data sekali tulis, baca berkali-kali. HDFS tidak cocok untuk aplikasi yang membutuhkan akses latensi rendah ke data. Sebaliknya, HDFS menyediakan akses data throughput yang tinggi.

- File Appends : HDFS pada awalnya dirancang untuk memiliki file yang tidak dapat diubah. File yang pernah ditulis ke HDFS tidak dapat dimodifikasi dengan menulis di lokasi sewenang-wenang dalam file atau menambahkannya ke file. Versi terbaru HDFS telah memperkenalkan kemampuan append. Proses penambahan file akan dibahas nanti di bab ini.

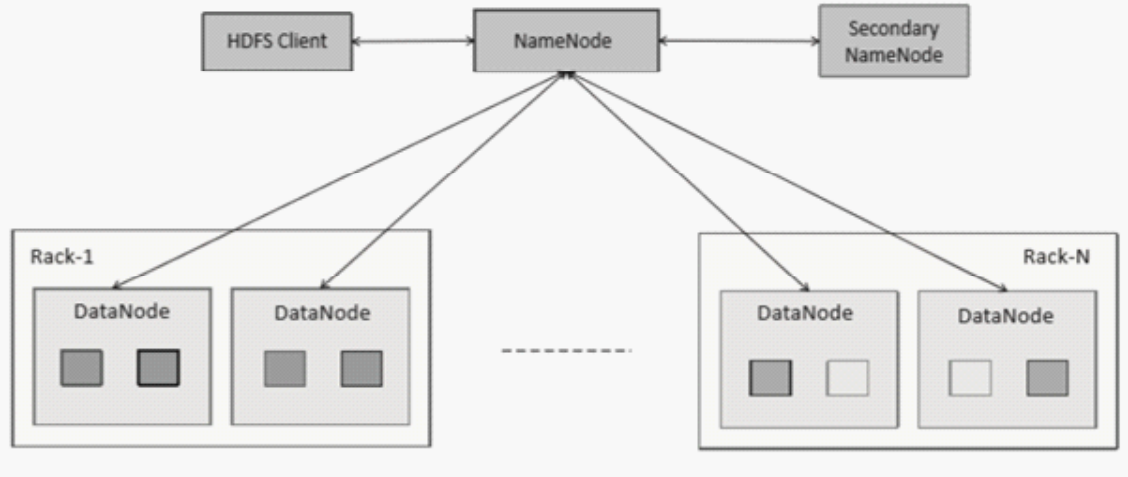
### 6.1.2 Arsitektur HDFS

Gambar 6.1 menunjukkan arsitektur HDFS. HDFS memiliki dua jenis node: Namenode dan Datanode.

#### 6.1.2.1 Namenode

Namenode mengelola namespace sistem file. Semua meta-data sistem file disimpan di Namenode. Meskipun Namenode bertanggung jawab untuk menjalankan operasi seperti membuka dan menutup file, tidak ada data yang benar-benar mengalir melalui Namenode. Namenode menjalankan operasi baca dan tulis sementara data ditransfer langsung ke / dari Datanode. HDFS membagi file menjadi blok, dan blok disimpan di Datanodes. Untuk setiap blok, beberapa replika disimpan. Namenode secara persisten menyimpan meta-data sistem file dan pemetaan blok ke datanoda, pada disk sebagai dua file: fsimage dan file edit. Fsimage berisi snapshot lengkap dari meta-data sistem file. File pengeditan menyimpan pembaruan tambahan ke meta-data.

Ketika Name nodest art, ini memuat file fsimage ke dalam memori dan menerapkan file edit untuk membawa tampilan dalam memori dari sistem file yang up-to-date. Namenode kemudian menulis file fsimage baru ke disk.




Gambar 6.1: *Arsitektur HDFS*

#### 6.1.2.2 Namenode Sekunder

File pengeditan terus bertambah ukurannya, seiring waktu, karena pembaruan tambahan disimpan. Tanggung jawab untuk menerapkan pembaruan ke file fsimage didelegasikan ke Secondary Namenode, karena Namenode mungkin tidak memiliki sumber daya yang cukup, karena menjalankan operasi lain. Proses ini disebut checkpointing. Proses checkpointing dilakukan baik secara berkala (default 1 jam) atau setelah sejumlah transaksi uncheckpointed tercapai di Namenode.

Ketika proses check pointing dimulai, node Secondary Name mendownload image dan mengedit file dari Namenode ke direktori checkpoint di Secondary Namenode. Secondary Namenode kemudian menerapkan pengeditan pada file fsimage dan membuat file fsimage baru. Fsimage baru diunggah oleh Secondary Namenode ke Namenode tersebut.

#### 6.1.2.3 Datanode

Sementara node Name menyimpan meta-data sistem file, Datanode menyimpan blok data dan melayani permintaan baca dan tulis. Datanode secara berkala mengirim pesan detak jantung dan memblokir laporan ke Namenode. Sementara pesan detak jantung memberi tahu Namenode bahwa Datanode masih hidup, laporan blok berisi informasi tentang blok di Datanode.

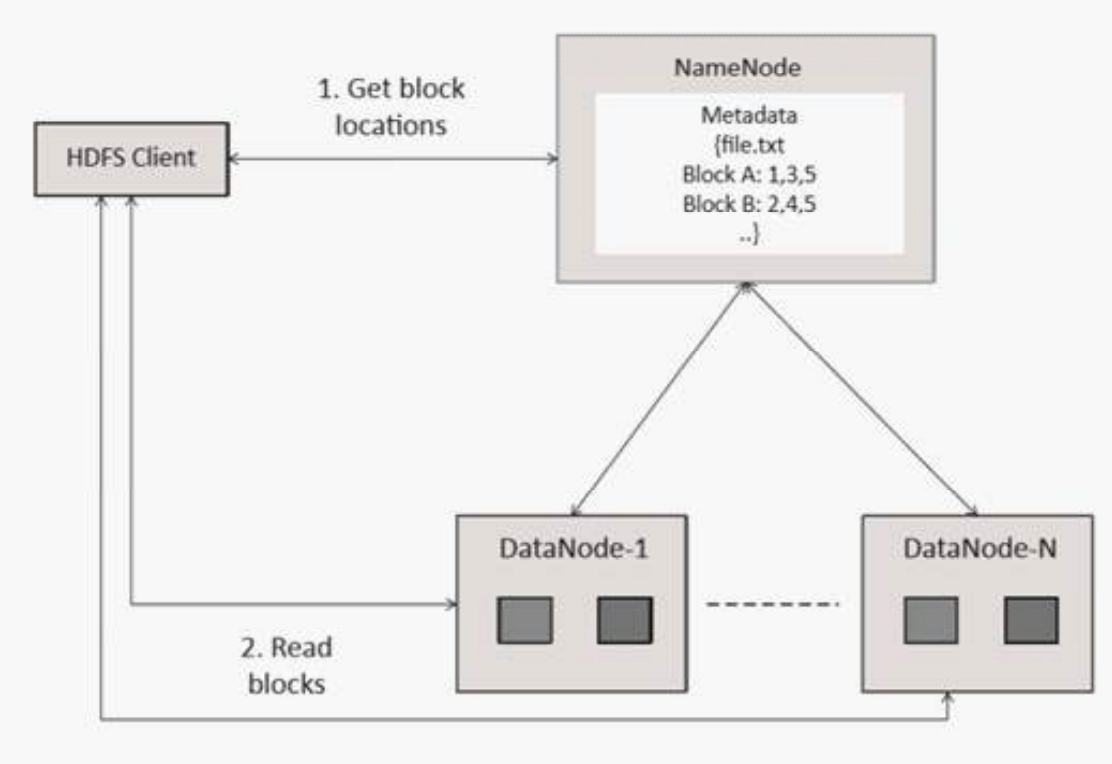
#### 6.1.2.4 Pemblokiran & Replikasi Data

Blok direplikasi di Datanode dan secara default tiga replika dibuat. Penempatan replika di node Data ditentukan oleh kebijakan penempatan yang sadar. Kebijakan penempatan

ini memastikan keandalan dan ketersediaan blok. Untuk faktor replikasi tiga, satu replika ditempatkan pada node pada rak lokal, replika kedua ditempatkan pada node yang berbeda pada rak jarak jauh dan replika ketiga ditempatkan pada node yang berbeda pada rak jarak jauh yang sama. Ini memastikan bahwa meskipun rak tidak tersedia, setidaknya satu replika akan tetap tersedia. Penempatan replika pada node yang berbeda di rak yang sama meminimalkan lalu lintas jaringan di antara rak.

#### 6.1.2.5 Jalur Baca HDFS

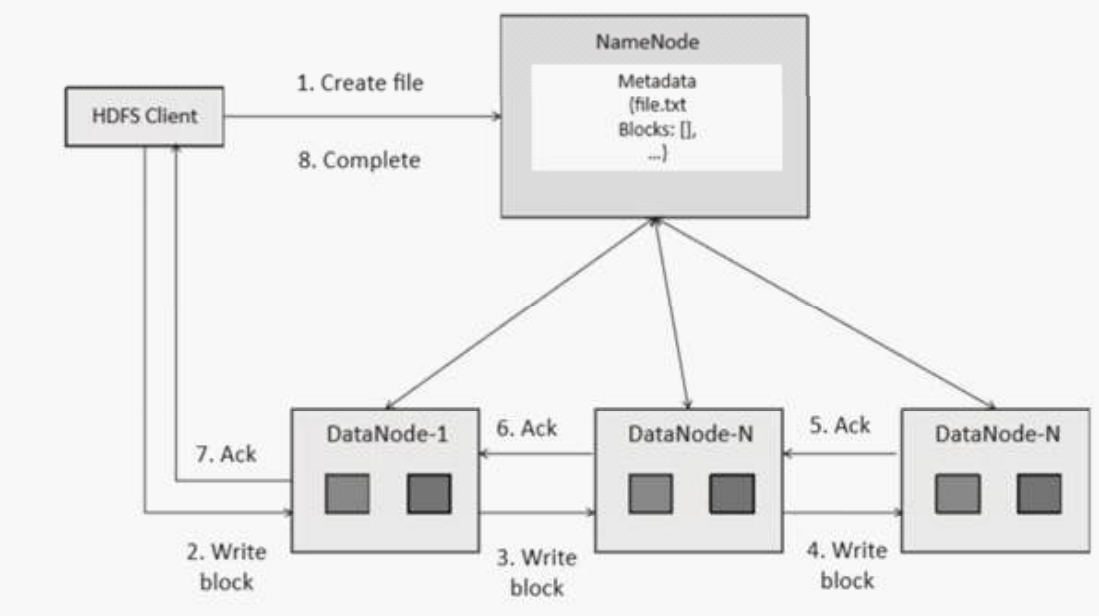
Gambar 6.2 menunjukkan jalur baca HDFS. Proses membaca dimulai dengan klien mengirimkan permintaan ke Namenode untuk mendapatkan lokasi blok data untuk sebuah file. Namenode memeriksa apakah file tersebut ada dan apakah klien memiliki izin yang memadai untuk membaca file tersebut. Namenode merespons dengan lokasi blok data yang diurutkan berdasarkan jarak ke klien. Ini membantu dalam meminimalkan lalu lintas antar node karena klien dapat membaca blok dari node terdekat. Misalnya, jika klien berada di node yang sama dengan blok data, ia dapat membaca blok data secara lokal. Klien membaca blok data langsung dari Datanode secara berurutan, sampai semua blok telah dibaca. Datanodes mengalirkan data ke klien. Selama proses membaca, jika replika menjadi tidak tersedia, klien dapat membaca replika lain di Datanode yang berbeda.




Gambar 6.2: *Jalur baca HDFS*

### 6.1.2.6 Jalur Tulis HDFS

Gambar 6.3 menunjukkan jalur tulis HDFS. Proses tulis dimulai dengan klien mengirimkan permintaan ke Namenode untuk membuat file baru di namespace sistem file. Namenode memeriksa apakah pengguna memiliki izin yang memadai untuk membuat file dan apakah file tersebut belum ada di sistem file. Namenode menanggapi klien dengan objek aliran output. Klien menulis data ke objek aliran output yang membagi data menjadi paket dan mengantrekannya ke antrean data. Paket dikonsumsi dari antrian data di utas terpisah, yang meminta Namenode untuk mengalokasikan blok baru di Datanode tempat data harus ditulis. Namenode merespons dengan lokasi blok di Datanodes. Klien kemudian membuat koneksi langsung ke Datanode di mana blok akan direplikasi membentuk pipa replikasi. Paket data yang digunakan dari antrian data ditulis ke Datanode pertama pada pipeline replikasi, yang menulis data ke Datanode kedua di pipeline dan seterusnya. Setelah paket berhasil ditulis, setiap Datanode di dalam pipeline mengirimkan pengakuan. Klien melacak semua paket yang diakui oleh Datanodes. Proses penulisan paket data ke Datanodes berlanjut hingga ukuran blok tercapai. Setelah mencapai ukuran blok, klien kembali meminta Namenode untuk mengembalikan satu set blok baru di Datanodes. Klien kemudian mengalirkan paket ke Datanodes. Proses ini berulang sampai semua paket data ditulis dan diakui. Akhirnya, klien menutup aliran output dan mengirimkan permintaan ke Namenode untuk menutup file.




Gambar 6.3: *Jalur tulis HDFS*

### 6.1.3 Contoh Penggunaan HDFS

Alat Baris Perintah HDFS

```
#Copy file to HDFS
#Format of command:
hdfs dfs -put <local source> <destination on HDFS>

#Example:
hdfs dfs -put file /user/hadoop/file
```

```
#Get file from HDFS
#Format of command:
hdfs dfs -get <source on hdfs> <local destination>

#Example:
hdfs dfs -get /user/hadoop/file file
```

```
#List files on HDFS
#Format of command:
hdfs dfs -ls <args>

#Example:
hdfs dfs -ls /user/hadoop/
```

```
#Remove a file on HDFS
#Format of command:
hdfs dfs -rm <HDFS Path>

#Example:
hdfs dfs -rm /user/hadoop/file
```

```
#Create a directory on HDFS
#Format of command:
hdfs dfs -mkdir <paths>

#Example:
hdfs dfs -mkdir /user/hadoop/dir
```

#### 6.1.3.2 Mengakses HDFS dengan Python

Di bagian ini kami memberikan contoh Python untuk mengakses HDFS menggunakan paket python Snakebite.

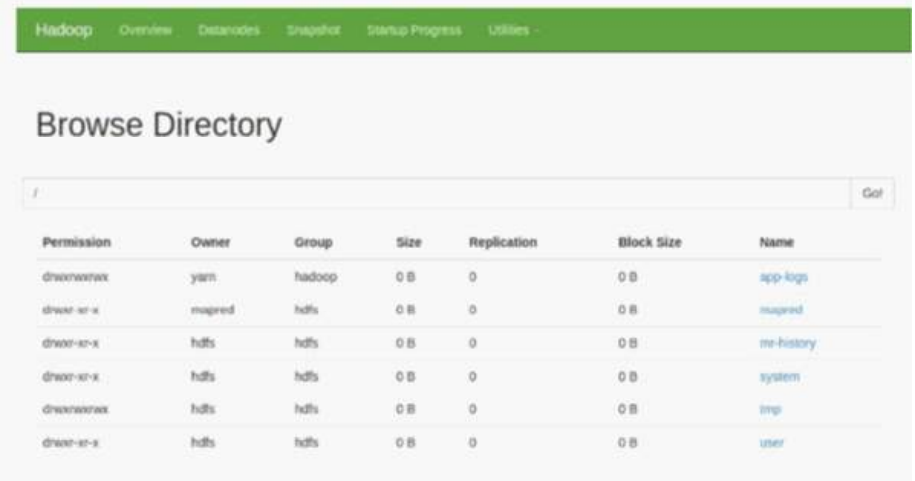
```
#Listing files on HDFS with Python
from snakebite.client import Client
client = Client("localhost", 8020, use_trash=False)
list(client.ls(["/"]))
```

```
#Reading a file from HDFS with Python
from snakebite.client import Client
client = Client("localhost", 8020, use_trash=False)
list(client.text(["/user/input.txt"]))
```

```
■ #Copying a file from HDFS with Python
from snakebite.client import Client
client = Client("localhost", 8020, use_trash=False)
list(client.copyToLocal(["/user/input.txt"], '/home/ubuntu/'))
```

### 6.1.3.3 Interface Web HDFS

HDFS menyediakan interface web dari mana Anda dapat menelusuri sistem file dan juga mendownload file tertentu seperti yang ditunjukkan pada Gambar 6.4 dan 6.5.




Gambar 6.4: *Menelusuri file di HDFS menggunakan interface web*


Gambar 6.5: *Download file dari HDFS menggunakan interface web*

## Ringkasan

HDFS adalah sistem file terdistribusi yang berjalan pada cluster besar dan menyediakan akses throughput tinggi ke data. HDFS menyediakan penyimpanan yang dapat diskalakan untuk file besar yang dipecah menjadi blok. Blok direplikasi untuk membuat sistem yang dapat diandalkan menjadi toleran terhadap default. HDFS Namenode menyimpan meta-data sistem file dan bertanggung jawab untuk menjalankan operasi seperti membuka dan menutup file. Secondary Namenode membantu dalam proses check pointing dengan menerapkan update dalam file edit ke file fsimage yang berisi snapshot lengkap dari meta-data sistem file. Datanodes menyimpan blok data yang direplikasi. Penempatan replika di Datanode ditentukan oleh kebijakan penempatan yang sadar rak. Kami menjelaskan contoh mengakses HDFS menggunakan alat baris perintah, pustaka Python untuk HDFS dan interface web HDFS.

# Analisis Batch

Bab 7

Bab ini mencakup

- Kerangka Analisis Batch
- Hadoop dan MapReduce
- Pig
- Apache Oozie
- Apache Spark
- Apache Solr

Dalam bab ini kami menjelaskan alat dan kerangka kerja untuk "pemrosesan batch" data termasuk: Hadoop-MapReduce , Pig, Oozie, Spark, dan Solr.

## 7.1 Hadoop dan MapReduce

Apache Hadoop [64] adalah kerangka kerja sumber terbuka untuk pemrosesan batch terdistribusi dari big data. Demikian pula, MapReduce adalah model pemrograman paralel [22] yang cocok untuk analisis big data. Algoritme MapReduce memungkinkan penghitungan skala besar diparalelkan secara otomatis di sejumlah besar server.

### 7.1.1 Model Pemrograman MapReduce

MapReduce adalah model pemrosesan data paralel untuk pemrosesan dan analisis data skala besar [22]. Model MapReduce memiliki dua fase: Map dan Reduce. Program MapReduce ditulis dalam gaya pemrograman fungsional untuk membuat fungsi Map dan Reduce. Data yang dimasukkan ke map dan fase reduksi berupa key-value pair. Sistem run-time untuk MapReduce biasanya merupakan kelompok besar yang dibangun dari perangkat keras komoditas. Sistem run-time MapReduce menangani tugas-tugas seperti mempartisi data, Schedulleran pekerjaan dan komunikasi antara node dalam cluster. Ini mempermudah pemrogram untuk menganalisis data berskala besar tanpa mengkhawatirkan tugas-tugas seperti partisi dan Schedulleran data.

Dalam fase Map, data dibaca dari sistem file terdistribusi, dipartisi di antara sekumpulan node komputasi dalam kluster, dan dikirim ke node sebagai sekumpulan key-value

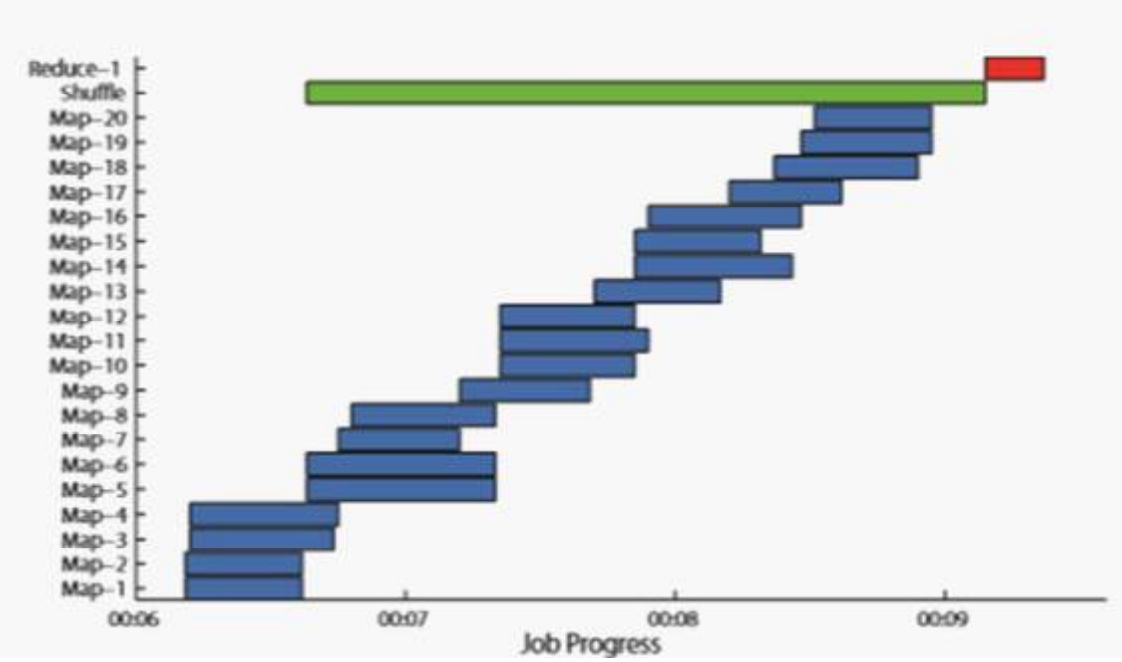
pair . Tugas Map memproses catatan masukan secara independen satu sama lain dan menghasilkan hasil antara sebagai key-value pair . Hasil antara disimpan di disk lokal dari node yang menjalankan tugas Map. Ketika semua tugas Map selesai, fase Kurangi dimulai di mana data perantara dengan kunci yang sama dikumpulkan. Tugas Gabungkan opsional dapat digunakan untuk melakukan agregasi data pada data perantara dari kunci yang sama untuk output mapper sebelum mentransfer output ke tugas Kurangi.

Program MapReduce memanfaatkan lokalitas data dan pemrosesan data berlangsung di node tempat data berada. Dalam pendekatan tradisional untuk analisis data, data dipindahkan ke node komputasi yang mengakibatkan penundaan transmisi data antara node dalam cluster. Namun, model pemrograman MapReduce memindahkan komputasi ke tempat data berada sehingga mengurangi transmisi data dan meningkatkan efisiensi. Model pemrograman MapReduce sangat cocok untuk pemrosesan paralel data skala besar di mana tugas analisis data dapat diselesaikan dengan map independen dan operasi pengurangan.

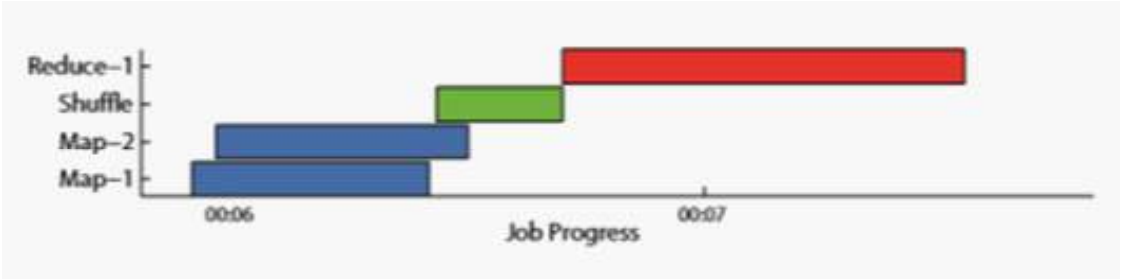
Gambar 7.1 dan 7.2 menunjukkan aliran eksekusi jumlah kata dan indeks pekerjaan MapReduce indeks terbalik. Seperti yang terlihat dari gambar ini, fase sortir dan shuf e dimulai segera setelah tugas map selesai dan fase pengurangan dimulai setelah pasangan kunci-nilai antara dari semua tugas map dikirim ke reducer.

## 7.2 Hadoop YARN

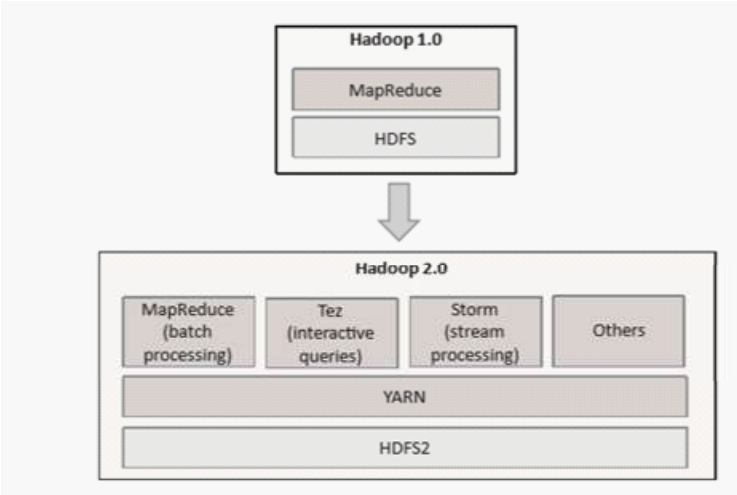
Hadoop YARN adalah arsitektur generasi penerus Hadoop (versi 2.x). Dalam arsitektur YARN, mesin pengolah asli Hadoop (MapReduce ) telah dipisahkan dari komponen manajemen sumber daya (yang sekarang menjadi bagian dari YARN) seperti yang ditunjukkan pada Gambar 7.3. Hal ini membuat YARN secara efektif menjadi sistem operasi untuk Hadoop yang mendukung mesin pemrosesan yang berbeda pada kluster Hadoop seperti MapReduce untuk pemrosesan batch, Apache Tez [60] untuk kueri interaktif, Apache Storm [65] untuk pemrosesan aliran, misalnya.




Gambar 7.1: *Map / Reduce penempatan slot untuk pekerjaan MapReduce jumlah kata*




Gambar 7.2: *Map / Reduce penempatan slot untuk indeks pekerjaan MapReduce indeks terbalik*




Gambar 7.3: *Perbandingan arsitektur Hadoop 1.x dan 2.x.*

Gambar 7.4 menunjukkan alur kerja eksekusi pekerjaan MapReduce untuk kerangka kerja Hadoop MapReduce (MR2) generasi berikutnya. Arsitektur MapReduce generasi berikutnya membagi dua fungsi utama JobTracker di Hadoop 1.x - manajemen sumber daya dan manajemen siklus hidup pekerjaan - menjadi beberapa komponen terpisah - Manajer Sumber Daya dan Master Aplikasi. Komponen utama YARN dijelaskan sebagai berikut:

- Resource Manager (RM): RM mengelola penugasan global sumber daya komputasi ke aplikasi. RM terdiri dari dua layanan utama:
- Scheduller: Scheduller adalah layanan siap pakai yang mengelola dan memberlakukan kebijakan Schedulleran sumber daya di kluster.
- Manajer Aplikasi (AsM): AsM mengelola Master Aplikasi yang sedang berjalan di cluster. Sebagai Bertanggung jawab untuk memulai master aplikasi dan untuk memantau serta memulai ulang mereka di node yang berbeda jika terjadi kegagalan.
- Application Master (AM): AM per aplikasi mengelola siklus hidup aplikasi. AM bertanggung jawab untuk merundingkan sumber daya dari RM dan bekerja dengan NM untuk melaksanakan dan memantau tugas.
- Node Manager (NM): NM per mesin mengelola proses pengguna pada mesin itu.
- Containers: Container adalah kumpulan resource yang dialokasikan oleh RM (memori, CPU dan jaringan). Container adalah entitas konseptual yang memberi aplikasi hak istimewa untuk menggunakan sejumlah sumber daya pada mesin tertentu untuk menjalankan tugas. Setiap node memiliki NM yang memunculkan banyak kontainer berdasarkan alokasi sumber daya yang dibuat oleh RM.

Gambar 7.4 menunjukkan cluster YARN dengan node Resource Manager dan tiga node Node Manager. Jumlah Master Aplikasi yang berjalan sebanyak lamaran (pekerjaan). AM setiap aplikasi mengelola tugas aplikasi seperti memulai, memantau, dan memulai kembali tugas jika terjadi kegagalan. Setiap aplikasi memiliki banyak tugas. Setiap tugas dijalankan dalam wadah terpisah. Kontainer dalam arsitektur YARN mirip dengan slot tugas di Hadoop MapReduce 1.x (MR1). Namun, tidak seperti MR1 yang membedakan antara memetakan dan mengurangi slot, setiap container di YARN dapat digunakan untuk memetakan dan mengurangi tugas. Ada model alokasi sumber di MR1 terdiri dari sejumlah slot map yang telah ditentukan dan slot yang dikurangi. Alokasi slot statis ini menghasilkan pemanfaatan cluster yang rendah. Model alokasi sumber daya YARN lebih fleksibel dengan pengenalan wadah sumber daya yang meningkatkan pemanfaatan cluster.

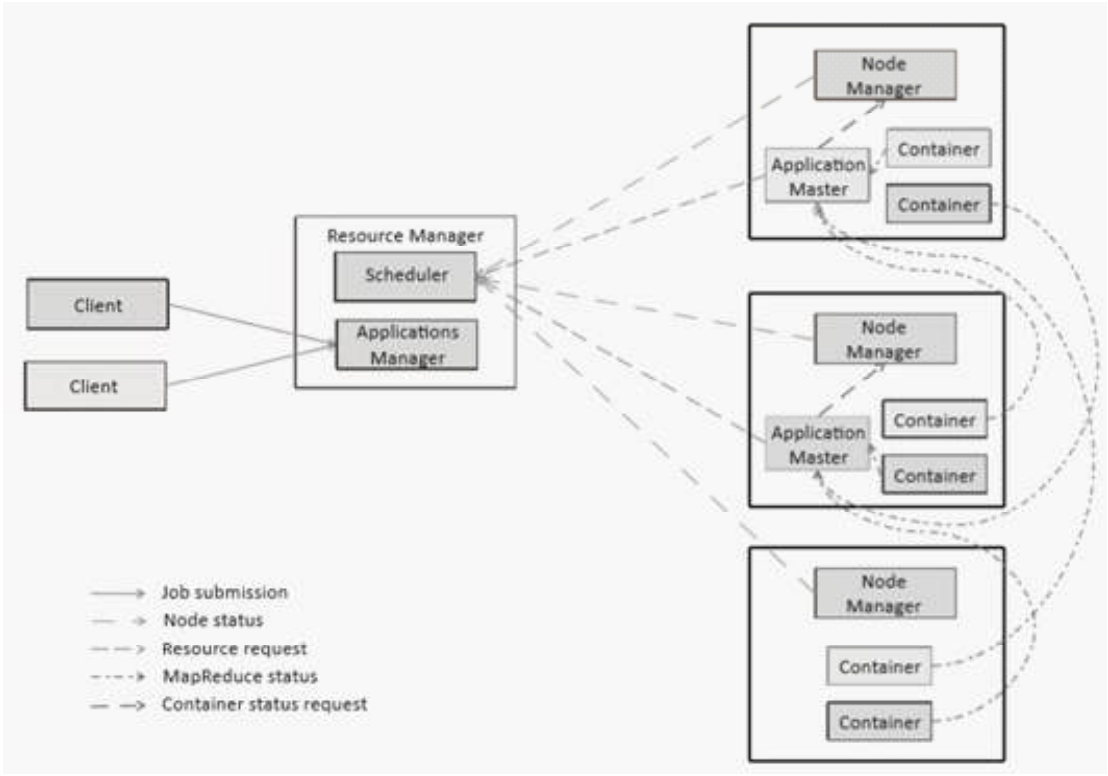
Untuk lebih memahami alur kerja eksekusi tugas YARN, mari kita menganalisis interaksi antara komponen utama di YARN. Gambar 7.5 menunjukkan interaksi antara Klien dan Manajer Sumber Daya. Eksekusi pekerjaan dimulai dengan pengajuan permintaan lamaran baru oleh klien ke RM. RM kemudian menanggapi dengan ID aplikasi unik dan informasi tentang kemampuan sumber daya cluster yang akan dibutuhkan klien dalam meminta sumber daya untuk menjalankan aplikasi AM. Menggunakan informasi yang diterima dari RM, klien membangun dan mengirimkan Konteks Pengiriman Aplikasi yang berisi informasi seperti antrian Scheduller, prioritas dan informasi pengguna. Application Submission Context juga berisi Container Launch Context yang berisi jar aplikasi, file pekerjaan, token keamanan, dan kebutuhan resource apa pun. Klien dapat meminta RM untuk laporan aplikasi. Klien juga dapat "memaksa membunuh" aplikasi dengan mengirimkan permintaan ke RM.

Gambar 7.6 menunjukkan interaksi antara Manajer Sumber Daya dan Master Aplikasi. Setelah menerima konteks pengiriman aplikasi dari klien, RM menemukan wadah yang tersedia yang memenuhi persyaratan sumber daya untuk menjalankan AM untuk aplikasi tersebut. Saat menemukan wadah yang sesuai, RM menghubungi NM agar wadah tersebut memulai proses AM pada node. Saat AM diluncurkan, AM mendaftarkan dirinya dengan RM. Proses pendaftaran terdiri dari jabat tangan yang menyampaikan informasi seperti port RPC yang akan didengarkan oleh AM, URL pelacakan untuk memantau status dan kemajuan aplikasi, dll. Pendaftaran respon dari RM berisi informasi untuk AM yang digunakan dalam menghitung dan meminta permintaan sumber daya untuk tugas individu aplikasi (seperti kemampuan sumber daya minimum dan maksimum untuk cluster). AM menyampaikan detak jantung dan informasi kemajuan ke RM. AM mengirimkan permintaan alokasi sumber daya ke RM yang berisi daftar container yang diminta, dan mungkin juga berisi daftar container yang dirilis oleh AM. Setelah menerima permintaan alokasi, komponen Scheduller RM menghitung daftar kontainer yang memenuhi permintaan dan mengirimkan kembali respon alokasi. Setelah menerima daftar sumber daya, AM menghubungi NMS terkait untuk memulai kontainer. Saat pekerjaan selesai, AM mengirimkan pesan Aplikasi Selesai ke RM.

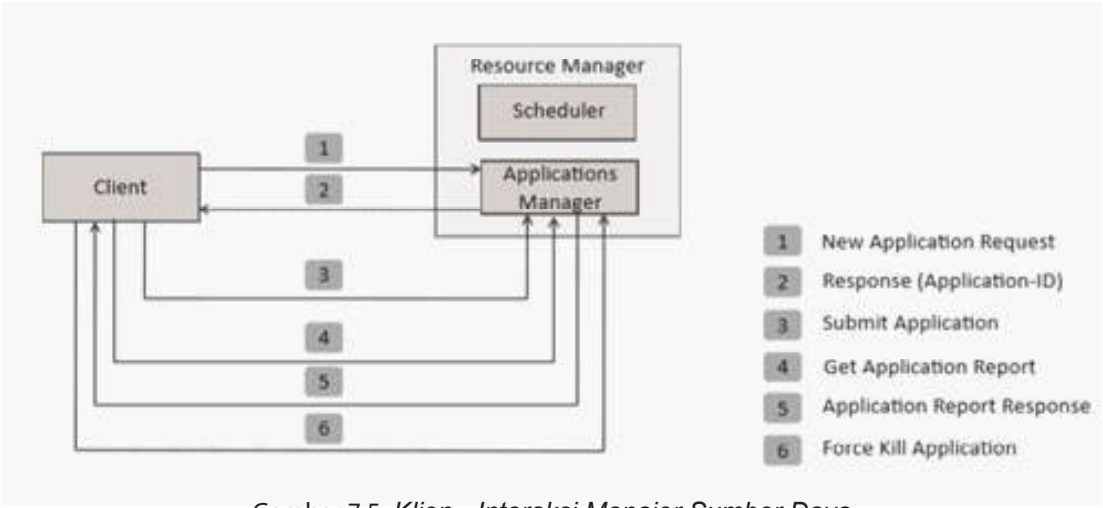
Gambar 7.7 menunjukkan interaksi antara Master Aplikasi dan Node Manager. Berdasarkan daftar sumber daya yang diterima dari RM, AM meminta NM hosting untuk setiap penampung untuk memulai penampung. AM dapat meminta dan menerima laporan status kontainer dari Node Manager. Gambar 7.8 menunjukkan eksekusi pekerjaan MapReduce dalam cluster YARN.

## 7.3 Scheduller Hadoop

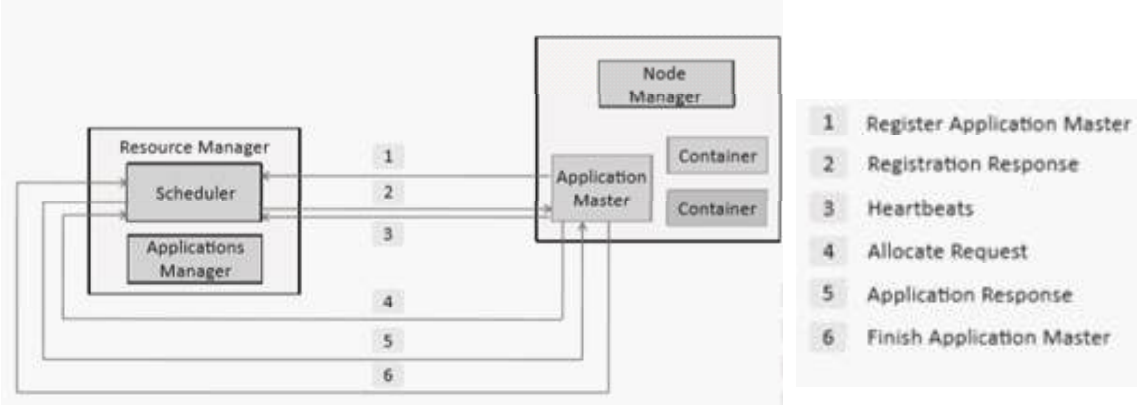
Scheduller adalah komponen yang dapat dicolokkan di Hadoop yang memungkinkannya untuk mendukung algoritma Scheduller yang berbeda. Kerangka Scheduller yang dapat dicolokkan memberikan fleksibilitas untuk mendukung berbagai beban kerja dengan berbagai prioritas dan batasan kinerja. Algoritme Schedulleran Hadoop dijelaskan sebagai berikut:.



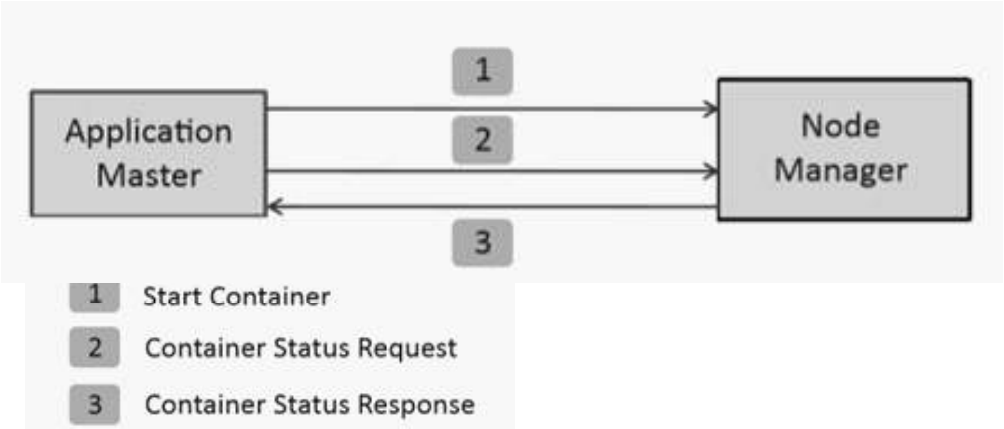

Gambar 7.4: *Eksekusi tugas Hadoop MapReduce Next Generation (YARN)*



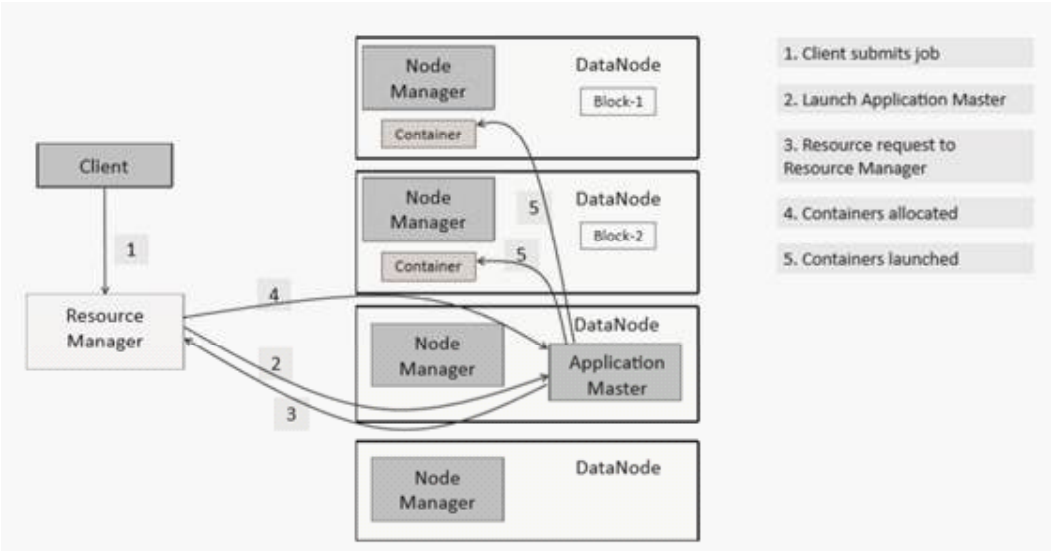

Gambar 7.5: *Klien - Interaksi Manajer Sumber Daya*




Gambar 7.6: *Manajer Sumber Daya - Interaksi Master Aplikasi*




Gambar 7.7: *Master Aplikasi - interaksi Node Manager*




Gambar 7.8: *Eksekusi pekerjaan MapReduce dalam cluster YARN*

### 7.3.1 FIFO

Scheduller FIFO memelihara antrian pekerjaan di mana pekerjaan antri. Scheduller menarik pekerjaan dengan cara pertama-masuk-keluar (pekerjaan terlama lebih dulu) untuk Schedulleran. Tidak ada konsep prioritas atau ukuran pekerjaan di Scheduller FIFO.

***Fair Scheduller***

Fair Scheduller [50] pada awalnya dikembangkan oleh Facebook. Facebook menggunakan Hadoop untuk mengelola konten besar dan data log yang dikumpulkannya setiap hari. Kami memahami bahwa kebutuhan akan Fair Scheduller muncul saat Facebook ingin berbagi infrastruktur pergudangan data di antara banyak pengguna. Fair Scheduller mengalokasikan sumber daya secara merata di antara banyak pekerjaan dan juga memberikan jaminan kapasitas. Fair Scheduller menugaskan sumber daya ke pekerjaan sedemikian rupa sehingga setiap pekerjaan mendapatkan bagian yang sama dari sumber daya yang tersedia dengan rata-rata lembur. Tidak seperti Scheduller FIFO, yang membentuk antrian pekerjaan, Fair Scheduller memungkinkan pekerjaan pendek selesai dalam waktu yang wajar sementara tidak membiarkan pekerjaan yang lama. Slot tugas yang gratis ditetapkan ke tugas baru, sehingga setiap tugas mendapatkan waktu CPU yang kira-kira sama. Fair Scheduller mempertahankan satu set kumpulan tempat pekerjaan ditempatkan. Setiap kolam memiliki kapasitas yang terjamin. Saat ada satu pekerjaan yang berjalan, semua sumber daya ditetapkan ke pekerjaan itu. Jika ada beberapa pekerjaan di kumpulan, setiap kumpulan mendapatkan setidaknya slot tugas sebanyak yang dijamin. Setiap pangkalan menerima setidaknya bagian minimum. Jika sebuah kumpulan tidak memerlukan bagian yang dijamin, kelebihan kapasitas dibagi di antara pekerjaan lain. Ini memungkinkan Scheduller menjamin kapasitas untuk kumpulan sambil menggunakan sumber daya secara efisien saat kumpulan tersebut tidak berisi tugas. Fair Scheduller melacak waktu komputasi yang diterima oleh setiap pekerjaan. Scheduller menghitung secara berkala perbedaan antara waktu komputasi yang diterima oleh setiap pekerjaan dan waktu yang seharusnya diterima dalam Schedulleran yang ideal. Pekerjaan yang memiliki defisit tertinggi dari waktu komputasi yang diterima dijadwalkan berikutnya. Ini memastikan bahwa seiring waktu, setiap pekerjaan mendapatkan bagian waktu komputasi yang adil.

Fair Scheduller berguna ketika cluster *Hadoop* kecil atau besar dibagikan di antara beberapa grup pengguna dalam suatu organisasi. Meskipun Fair Scheduller memastikan keadilan dengan mempertahankan satu set kumpulan dan memberikan jaminan kapasitas

untuk setiap kumpulan, itu tidak memberikan jaminan waktu apa pun dan karenanya tidak dilengkapi untuk pekerjaan real-time.

### 7.3.2 Capasity Scheduller

The Capacity Scheduler [51] dikembangkan oleh Yahoo. Capacity Scheduller kapasitas memiliki fungsi yang mirip dengan Fair Scheduller tetapi mengadopsi filosofi Schedulleran yang berbeda. Dalam Capacity Schedulle, beberapa antrian bernama ditentukan, masing-masing dengan sejumlah map yang dapat dikonfigurasi dan mengurangi slot. Setiap antrian juga diberikan kapasitas yang terjamin. Capacity Schedulle memberikan setiap antrian kapasitasnya ketika itu berisi pekerjaan, dan berbagi kapasitas yang tidak digunakan di antara antrian. Dalam setiap antrian Scheduller FIFO dengan prioritas digunakan. Untuk keadilan, dimungkinkan untuk membatasi persentase tugas yang berjalan per pengguna, sehingga pengguna berbagi cluster secara merata. Waktu tunggu untuk setiap antrian dapat dikonfigurasi. Ketika antrian tidak dijadwalkan lebih dari waktu tunggu, ia dapat mendahului tugas antrian lain untuk mendapatkan bagian yang adil. Saat TaskTracker memiliki slot kosong, Capacity Schedulle memilih antrean dengan rasio jumlah slot yang berjalan terhadap kapasitas paling rendah. Scheduller kemudian mengambil pekerjaan dari antrian yang dipilih untuk dijalankan. Pekerjaan diurutkan berdasarkan waktu pengiriman dan prioritasnya. Pekerjaan dipertimbangkan secara berurutan, dan pekerjaan dipilih jika penggunanya berada dalam kuota pengguna untuk antrian, yaitu, pengguna belum menggunakan sumber daya antrian di atas batas yang ditentukan.

Capacity Schedulle berguna saat cluster Hadoop besar dibagikan dengan beberapa klien dan jenis serta prioritas pekerjaan yang berbeda. Meskipun Capacity Schedulle memastikan keadilan dengan mempertahankan satu set antrian dan memberikan jaminan kapasitas untuk setiap antrian, itu tidak memberikan jaminan waktu apa pun dan, oleh karena itu, mungkin tidak dilengkapi dengan baik untuk pekerjaan real-time.

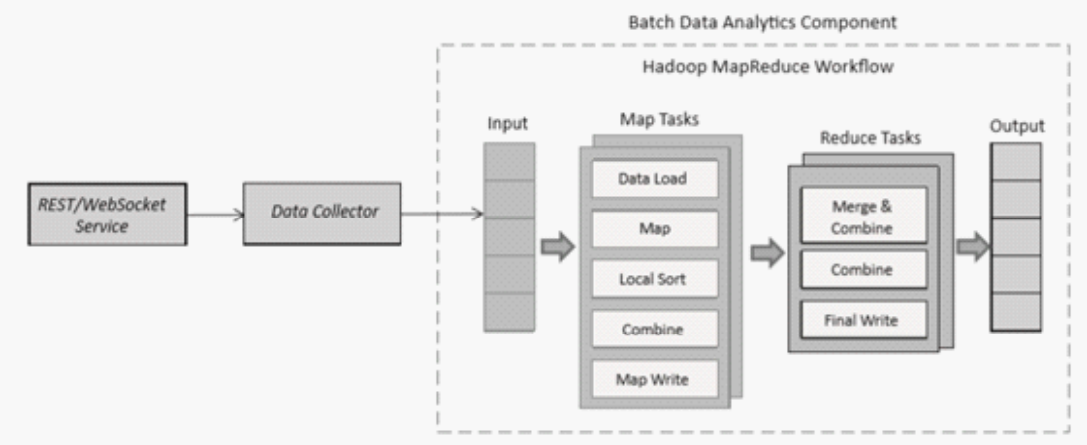
## 7.4 Contoh *Hadoop – MapReduce*

### 7.4.1 Analisis *Batch* pada Sensor Data

Gambar 7.9 menunjukkan aliran kerja Hadoop MapReduce untuk analisis batch data cuaca. Analisis batch dilakukan untuk menggabungkan data (seperti menghitung rata-rata, maksimum, dan minimum) pada berbagai skala waktu. Untuk contoh ini, kami akan mengasumsikan bahwa kami memiliki pengumpul data yang mengambil data sensor yang dikumpulkan dalam database cloud dan membuat file raw data dalam bentuk yang sesuai untuk diproses oleh Hadoop.

File data mentah terdiri dari mahasiswaan sensor mentah bersama dengan stempel waktu seperti yang ditunjukkan di bawah ini:
"2015-04-29 10:15:32", 38,42,34,5:
"2015-04-30 10:15:32 ", 87,48,21,4

Box 7.1 menunjukkan program map untuk analisis batch data sensor. Program map membaca data dari input standar (stdin) dan membagi data menjadi cap waktu dan mahasiswaan sensor individu. Program map memancarkan *key-value pair* yang kuncinya adalah bagian dari stempel waktu (yang bergantung pada skala waktu tempat data akan digabungkan), dan nilainya adalah string mahasiswaan sensor yang dipisahkan koma.




Gambar 7.9: *Menggunakan Hadoop MapReduce untuk analisis batch data sensor*

**Box 7.1: Map program - mapper.py**

```
#!/usr/bin/env python
import sys

#Calculates mean temperature, humidity, light and CO2
# Input data format:
#"2014-04-29 10:15:32",37,44,31,6
#Output:
#"2014-04-29 10:15 [48.75, 31.25, 29.0, 16.5]"

#Input comes from STDIN (standard input)
for line in sys.stdin:
   # remove leading and trailing whitespace
   line = line.strip()
   data = line.split(',')
   l=len(data)
   #For aggregation by minute
   key=str(data[0][0:17])

   value=data[1]+','+data[2]+','+data[3]+','+data[4]
   print '%s \t%s' % (key, value)
```

Box 7.2 menunjukkan program pengurangan untuk analisis batch data sensor. Key-value pair yang dipancarkan oleh program map diubah menjadi reducer dan dikelompokkan berdasarkan kuncinya. Peredam membaca key-value pair yang dikelompokkan berdasarkan kunci yang sama dari input standar dan menghitung alat mahasiswaan suhu, kelembapan, cahaya, dan CO. Box 7.3 menunjukkan perintah untuk mengirimkan pekerjaan MapReduce pada cluster Hadoop dan melihat file output pada HDFS.

**Box 7.2: Reduce program - reducer.py**

```
#!/usr/bin/env python
from operator import itemgetter
import sys
import numpy as np

current_key = None
current_vals_list = []
word = None

#Input comes from STDIN
for line in sys.stdin:
   # remove leading and trailing whitespace
   line = line.strip()

#Parse the input from mapper
   key, values = line.split('\t', 1)
   list_of_values = values.split(',')

#Convert to list of strings to list of int
   list_of_values = [int(i) for i in list_of_values]

   if current_key == key:
      current_vals_list.append(list_of_values)
   else:
      if current_key:
         l = len(current_vals_list)+ 1
         b = np.array(current_vals_list)
         meanval = [np.mean(b[0:l,0]),np.mean(b[0:l,1]),
         np.mean(b[0:l,2]), np.mean(b[0:l,3])]
         print '%s\t%s' % (current_key, str(meanval))

      current_vals_list = []
      current_vals_list.append(list_of_values)
      current_key = key
#Output the last key if needed
if current_key == key:
   l = len(current_vals_list)+ 1
   b = np.array(current_vals_list)
   meanval = [np.mean(b[0:l,0]),np.mean(b[0:l,1]),
      np.mean(b[0:l,2]), np.mean(b[0:l,3])]
   print '%s\t%s' % (current_key, str(meanval))
```

**Box 7.3: Running MapReduce program on Hadoop cluster**

```
#Testing locally
$cat data.txt | python mapper.py | python reducer.py

#Running on Hadoop cluster
#Copy data file to HDFS sudo -u user1 bin/hadoop dfs -copyFromLocal
data.txt input

#Run MapReduce job bin/hadoop jar
contrib/streaming/hadoop-*streaming*.jar
-mapper mapper.py -reducer reducer.py
-file /home/ubuntu/hadoop/mapper.py
-file /home/ubuntu/hadoop/reducer.py
-input input/* -output output

#View output bin/hadoop dfs -ls output
bin/hadoop dfs -cat output/part-00000
```

### 7.4.2 Analisis Batch pada N-Gram Dataset

Mari kita lihat contoh lain dari MapReduce untuk menganalisis Google N-Gramdataset [29], yang merupakan kumpulan n-gram (tupel ukuran tetap kata-kata) yang tersedia secara bebas yang diekstrak dari korpus Google Books. N menentukan jumlah elemen dalam tupel, jadi misalnya, 5 gram berisi lima kata. N-gram dalam kumpulan data ini diproduksi dengan melewatkan window geser di atas teks buku dan mengeluarkan catatan untuk setiap token baru. Misalnya, untuk baris - 'Python adalah bahasa tingkat tinggi', The 2-gram (atau bigram) akan menjadi:

(Python, is)
(is, a)
(a, high)
(high, level)
(level, language)
Setiap baris data berisi:

- n-gram itu sendiri
- tahun di mana n-gram muncul
- berapa kali n-gram muncul di buku dari tahun yang sesuai (hitungan)
- jumlah halaman di mana n-gram muncul pada tahun ini (jumlah halaman)
- jumlah buku berbeda di mana n-gram muncul pada tahun ini (jumlah buku)

Contoh (5-gram): analisis sering dideskripsikan sebagai 1991 1 1 1 Interpretasi dari 5-gram: Pada tahun 1991, frase "analisis sering kali digambarkan sebagai" terjadi satu kali (itu yang pertama 1), dan pada satu halaman ( 1 kedua), dan dalam satu buku (1 ketiga).

Box 7.4 menunjukkan program MapReduce yang menghitung bigram paling populer (2 gram) sepanjang masa dalam dataset. Contoh ini menggunakan pustaka MRJob Python yang memungkinkan Anda menulis pekerjaan MapReduce dengan Python dan menjalankannya di beberapa platform termasuk mesin lokal, klaster Hadoop, dan Amazon Elastic MapReduce (EMR). MRJob dapat diinstal sebagai berikut:

```
■ #Installing MRJob
sudo apt-get install git
git clone https://github.com/Yelp/mrjob.git
cd mrjob
python setup.py install
```

Contoh dalam Box 7.4 mengimplementasikan kelas MyMRJob yang mendefinisikan fungsi mapper dan peredam. Dalam contoh ini, kita memiliki satu pasangan pengurang map dan fungsi reduksi lainnya yang dirantai ke output peredam pertama. Ketika program dijalankan, fungsi mapper dipanggil untuk setiap baris dari file masukan.

**■ Box 7.4: MapReduce program that calculates the most popular bigram of all time - mr.py**

```
from mrjob.job import MRJob

class MyMRJob(MRJob):
   def mapper(self, _, line):
      data=line.split('\t')
      ngram = data[0].strip()
      year = data[1].strip()
      count = data[2].strip()
      pages = data[3].strip()
      books = data[4].strip()

      #Emit key-value pairs where key is ngram+year and value is count
      yield ngram+year, int(count)

   def reducer(self, key, list_of_values):
      # Send all (count, ngram+year) pairs to the same reducer.
      # So we can easily use Python's max() function.
      yield None, (sum(list_of_values),key)
```

```
    def reducer2(self, _, list_of_values):
        # Reducer-2 get input tuples as follows:
        # None, [(212, cloud computing 2006), (156, mobile phones 2003)]
        # max function will yield tuple with max value of the count
        yield max(list_of_values)

    def steps(self):
        return [self.mr(mapper=self.mapper,
        reducer=self.reducer), self.mr(reducer=self.reducer2)]

if __name__ == '__main__':
    MyMRJob.run()
```

Program MapReduce dijalankan sebagai berikut:

```
■ #Running MapReduce program
python mr.py googlebooks-eng-us-all-2gram-20090715-50-subset.csv
```

### 7.4.3 Temukan kata-kata teratas dengan MapReduce

Mari kita lihat contoh MapReduce lain yang menemukan kata-N teratas dalam file teks, di mana N adalah nomor yang dapat dikonfigurasi. Box 7.5 menunjukkan program Python MapReduce untuk menemukan 3 kata teratas dalam sebuah file. Ketika program ini dijalankan, baris dalam file teks masukan diteruskan ke pembuat map. Fungsi mapper membagi garis dan memancarkan key-value pair di mana kunci adalah setiap kata dalam baris dan nilainya adalah 1. Fungsi peredam pertama meringkas daftar nilai untuk setiap kunci, sehingga menghitung jumlah kata. Fungsi peredam memancarkan key-value pair di mana kuncinya adalah Tidak Ada dan nilai adalah tupel yang berisi jumlah dan kata. Mengatur kunci sebagai Tidak Ada memastikan bahwa semua (hitungan, kata) tupel dikirim ke peredam yang sama. Fungsi peredam kedua mengurutkan daftar nilai (di mana setiap nilai adalah tupel (hitungan, kata)) dan memancarkan nilai N teratas.

**■ Box 7.5: MapReduce program that finds the top-N words in a file- topNwords.py**

```
from mrjob.job import MRJob

class MyMRJob(MRJob):
    def mapper(self, _, line):
        line = line.strip()
        words = line.split()
        for word in words:
            yield (word, 1)
```

```
    def reducer(self, key, list_of_values):
        word = key
        total_count = sum(list_of_values)
        yield None, (total_count, word)

    def reducer2(self, _, list_of_values):
        N = 3
        list_of_values = sorted(list(list_of_values), reverse=True)
        return list_of_values[:N]

    def steps(self):
        return [self.mr(mapper=self.mapper,
        reducer=self.reducer), self.mr(reducer=self.reducer2)]

if __name__ == '__main__':
    MyMRJob.run()
```

## 7.5 Pig

Sementara MapReduce adalah model pemrograman yang kuat untuk analisis big data, untuk pekerjaan analisis kompleks tertentu, pengembang mungkin merasa sulit untuk mengidentifikasi key-value pair yang terlibat di setiap langkah dan kemudian mengimplementasikan map dan mengurangi fungsi. Selain itu, pekerjaan analisis yang kompleks mungkin memerlukan beberapa pekerjaan MapReduce untuk dirantai.

Pig adalah bahasa pemrosesan data tingkat tinggi yang memudahkan pengembang untuk menulis skrip analisis data, yang diterjemahkan ke dalam program Map Reduce oleh penyusun Pig. Pig mencakup: (1) bahasa tingkat tinggi (disebut Pig Latin) untuk mengekspresikan program analisis data dan (2) kompiler yang menghasilkan urutan program MapReduce dari skrip pig.

Pig dapat dijalankan baik dalam mode lokal atau mode MapReduce. Dalam mode lokal, Pig berjalan di dalam proses JVM tunggal di mesin lokal. Mode lokal berguna untuk tujuan pengembangan dan menguji skrip dengan file data kecil pada satu mesin. Mode MapReduce membutuhkan cluster Hadoop. Dalam mode MapReduce, Pig dapat menganalisis data yang disimpan dalam HDFS. Kompiler Pig menerjemahkan skrip pig ke dalam program MapReduce yang dijalankan dalam cluster Hadoop. Pig menyediakan cangkang interaktif yang disebut grunt, untuk mengembangkan skrip pig. Grunt dapat diluncurkan sebagai berikut:

```
# Launching Pig Grunt Shell
# Pig local mode
>pig -x local

#Pig MapReduce mode
>pig
```

Mari kita lihat beberapa operator Pig yang umum digunakan dengan contohnya. Kami akan menggunakan kumpulan data cuaca NCDC [52] sebagai contoh. NCDC menyediakan akses ke data harian dari Jaringan Referensi Iklim A.S. / Jaringan Referensi Iklim Regional A.S. (USCRN / USRCRN) melalui FTP.

### 7.5.1 Loading Data

Pig menyediakan operator LOAD untuk memuat data. Operator LOAD memuat data dari file ke dalam relasi. Relasi Pig adalah kumpulan tupel di mana setiap tupel memiliki banyak bidang. Contoh operator LOAD ditunjukkan di bawah ini:

```
# LOAD example
data = LOAD 'data.txt' as (text:chararray);
```

### 7.5.2 Tipe Data dalam Pig

Pig mendukung tipe data sederhana seperti int, long, fl oat, double, chararray, bytearray, boolean, datetime, dan tipe data kompleks seperti tuple, bag dan map. Tipe data sederhana bekerja dengan cara yang sama seperti pada bahasa pemrograman lainnya. Mari kita lihat tipe data kompleks secara detail.

**Tuple**

Tupel adalah kumpulan bidang yang teratur.

```
# Tuple example
(1,10)
```

**Bag**

Bag adalah koleksi tupel yang tidak berurutan. Tas diwakili dengan kawat gigi keriting.

```
■ # Bag example
{(1,10.4),(1,4.9),(1,5.0)})
```

**Map**

Map adalah sekumpulan key-value pair . Map direpresentasikan dengan tanda kurung siku dan # digunakan untuk memisahkan kunci dan nilai.

```
■ # Map example
[temp#20.0, humidity#70]
```

### 7.5.3 Analisis & Data Filtering

Operator FOREACH digunakan untuk memproses setiap baris dalam suatu relasi dan operator GENERATE digunakan untuk mendefinisikan bidang dan menghasilkan baris baru dari aslinya. Misalnya, dengan kumpulan data cuaca yang dimuat sebelumnya, untuk menghasilkan hubungan hanya dengan bulan dan suhu, FOREACH dan GENERATE dapat digunakan sebagai berikut:

```
■ # FOREACH example
monthTemp = FOREACH data GENERATE SUBSTRING(text, 10,12) as month,
(double)SUBSTRING(text, 38,45) as temp;
DUMP monthTemp;
(01,22.9)
:
(12,5.6)
```

Operator FILTER digunakan untuk menyaring tupel dari relasi berdasarkan kondisi yang ditentukan. Misalnya, untuk menyaring semua baris dengan suhu kurang dari 20, operator FILTER dapat digunakan sebagai berikut:

```
■ # FILTER example
low = FILTER monthTemp by temp<20.0;
DUMP low;
(01,10.4)
:
(12,4.8)
```

Operator GROUP dapat digunakan untuk mengelompokkan data dalam satu atau lebih relasi. Misalnya, untuk mengelompokkan relasi monthTemp menurut field bulan, operator GROUP dapat digunakan sebagai berikut:

```
■ #GROUP example
monthTempGroup = GROUP monthTemp by month;
DESCRIBE monthTempGroup;
monthTempGroup: {group: chararray,monthTemp: {(month: chararray,temp:
double)}}
DUMP monthTempGroup;
(1,{(1,11.7),...,(1,9.7)})
(12,{(12,20.3),...,(12,4.8)})
```

Operator UNION dapat digunakan untuk menggabungkan konten dari dua relasi atau lebih. Contoh di bawah ini menunjukkan cara mendapatkan penyatuan dua relasi:

```
■ #UNION example
low = FILTER monthTemp by temp<10.0;
high = FILTER monthTemp by temp>20.0;
lowHigh = UNION low,high;
```

Operator JOIN digunakan untuk menggabungkan dua relasi. Misalnya untuk menggabungkan dua relasi (satu yang menahan suhu maksimum di setiap bulan dan yang lainnya memegang suhu minimum di setiap bulan), JOIN dapat digunakan sebagai berikut:

```
■ # JOIN example
maxTemp = FOREACH monthTempGroup GENERATE group, MAX(monthTemp.temp);
minTemp = FOREACH monthTempGroup GENERATE group, MIN(monthTemp.temp);
maxMinTemp = JOIN maxTemp BY $0, minTemp BY $0;
```

Pig menyediakan berbagai fungsi bawaan seperti AVG, MIN, MAX, SUM, dan COUNT. Pada contoh di atas, MAX dan MIN digunakan untuk mendapatkan suhu maksimum dan minimum setiap bulannya. Ekspresi $ N dalam pernyataan bergabung digunakan untuk menentukan kolom di mana gabungan harus dilakukan. Atau, nama kolom juga dapat diberikan.

### 7.5.4 Storing Result

Untuk menyimpan hasil pada sistem file digunakan operator STORE. Pig menggunakan strategi evaluasi malas dan menunda evaluasi ekspresi sampai operator STORE atau DUMP memicu hasil untuk disimpan atau ditampilkan.

```
grunt# STORE example
low = FILTER monthTemp by temp<20.0;
STORE low;
```

### 7.5.5 Debugging Operator

Operator DUMP digunakan untuk membuang hasil di konsol. DUMP digunakan dalam mode interaktif untuk tujuan debugging. Operator DESCRIBE digunakan untuk melihat skema relasi.

```
grunt# DESCRIBE example
monthTempGroup = GROUP monthTemp by month;
DESCRIBE monthTempGroup;
monthTempGroup: {group: chararray, monthTemp:
{(month: chararray,temp: double)}}
```

Operator EXPLAIN digunakan untuk melihat rencana eksekusi logis, fisik, dan MapReduce untuk menghitung suatu relasi. Contoh berikut menunjukkan rencana eksekusi untuk menghitung relasi monthTemp.

```
grunt# EXPLAIN example
EXPLAIN monthTemp;
#-----------------------------------
# Map Reduce Plan
#-----------------------------------
MapReduce node scope-308
Map Plan
monthTemp: Store(fakefile:org.apache.pig.builtin.PigStorage) - scope-307
|
|--monthTemp: New For Each(false,false)[bag] - scope-306
| |
| POUserFunc(org.apache.pig.builtin.SUBSTRING)[chararray] - scope-298
| |
| |--Project[chararray][0] - scope-295
| |
```

```
| |--Constant(10) - scope-296
| |
| |--Constant(12) - scope-297
| |
| Cast[double] - scope-304
| |
| |--POUserFunc(org.apache.pig.builtin.SUBSTRING)[chararray] - scope-303
| |
| |--Project[chararray][0] - scope-300
| |
| |--Constant(38) - scope-301
| |
| |--Constant(45) - scope-302
|
|--data: New For Each(false)[bag] - scope-294
| |
| Cast[chararray] - scope-292
| |
| |--Project[bytearray][0] - scope-291
|
|--data:
Load(file:///home/ubuntu/pig-0.15.0/data.txt:org.apache.pig.builtin.PigStorage) -
scope-290-----
Global sort: false
----------
```

Operator ILLUSTRATE digunakan untuk menampilkan eksekusi pernyataan selangkah demi selangkah untuk menghitung relasi dengan sampel data yang kecil. Contoh di bawah ini menunjukkan pernyataan ILLUSTRATE untuk relasi monthTemp.

```
grunt> # ILLUSTRATE example
ILLUSTRATE monthTemp;
-----------------------
| data | text:chararray |
-----------------------
| | 03739 20140207 2.422 -75.93 37.29 3.7 -2.8 0.5 0.8 0.0 12.02 C
15.1 -5.0 1.9 89.5 41.2 67.1 0.231 0.214 0.214
0.215 0.214 2.9 3.2 3.4 4.0 5.2 |
-----------------------------
| monthTemp | month:chararray | temp:double |
-------------------------------------
| | 02 | 3.7 |
-------------------------------------
```

### 7.5.6 Contoh Pig

Mari kita lihat beberapa contoh analisis data batch dengan Pig. Box 7.6 menunjukkan contoh penghitungan jumlah kata dengan Pig. Dalam contoh ini, data pertama kali dimuat dari file teks. Garis-garis tersebut diberi token menggunakan fungsi TOKENIZE yang membuat sekantong tupel (dari kata-kata dalam satu baris) untuk setiap baris dalam file teks. Fungsi FLATTEN digunakan untuk mengencangkan tas agar tuple dapat dikelompokkan. Operator GROUP digunakan untuk mengelompokkan kata. Terakhir, fungsi COUNT digunakan untuk menghitung kemunculan untuk setiap kata. Cara terbaik untuk memahami relasi yang terlibat di setiap langkah adalah dengan menggunakan operator debugging seperti DUMP, DESCRIBE, dan ILLUSTRATE.

**■ Box 7.6: Pig script for computing word count**

```
data = LOAD 'input.txt' as (lines:chararray);
words = FOREACH data GENERATE FLATTEN(TOKENIZE(lines)) AS word;
wordGroup = GROUP words BY word;
counts = FOREACH wordGroup GENERATE group, COUNT(words);
store counts into 'counts';
```

Di awal bab ini, kami menjelaskan contoh program MapReduce yang menghitung bigram paling populer sepanjang masa dari dataset Google N-Gram. Mari kita lihat contoh penggunaan Pig untuk menghitung bigram paling umum di setiap tahun dalam kumpulan data. Box 7.7 menunjukkan skrip Pig untuk menghitung bigram paling umum di setiap tahun.

**■ Box 7.7: Pig script for computing the most common bigram in each year in the dataset**

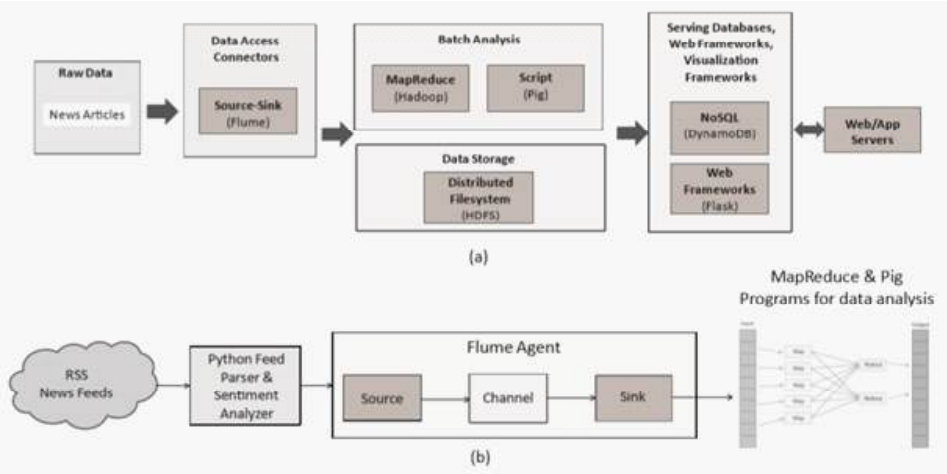
```
#Expted output:  (year, bigram, count)
data = LOAD 'hdfs:///ngraminput' using
PigStorage() AS (ngram:chararray, year:int,count:int);
yearData = GROUP data BY year;
maxYearData = FOREACH yearData GENERATE
group AS groupId, MAX(data.count) AS maxCount;
joinResult = JOIN maxYearData BY
(groupId,maxCount), data by (year,count);
result = FOREACH joinResult GENERATE
$0 AS year, $1 AS count, $2 AS ngram;
STORE result into 'hdfs:///result';
```

## Studi Kasus

### Analisis Batch pada Artikel Berita

Pada bagian ini, kami akan menjelaskan studi kasus tentang sistem untuk analisis batch artikel berita yang dikumpulkan dari beberapa situs berita. Sistem menghitung sentimen setiap artikel berita dan menemukan topik yang sedang tren.

Mengingat persyaratan analisis sistem ini, mari kita mapkan sistem ke salah satu pola analisis yang diusulkan di Bab-1. Karena sistem memproses artikel berita dalam model batch, kami menyarankan penggunaan pola Alpha. Gambar 7.10 (a) menunjukkan realisasi pola Alpha untuk sistem ini, dengan alat dan kerangka kerja khusus yang dapat digunakan. Sistem menggunakan MapReduce dan Pig untuk analisis batch, seperti menghitung kata paling umum dan frasa paling umum di antara semua artikel berita. Gambar 7.10 (b) menunjukkan arsitektur sistem analisis berita.




Gambar 7.10: *(a) Sebuah realisasi pola alpha untuk analisis bathc pada artikel baru, (b) Arsitektur sistem*

Untuk mengumpulkan artikel berita dari situs web yang berbeda, program Python yang ditunjukkan pada Box 7.8 digunakan. Dalam program ini, tersedia tautan umpan RSS (Rich Site Summary) untuk berbagai situs berita. Perpustakaan parsing RSS feed yang disebut feedparser digunakan. Program Python berjalan terus menerus untuk mengumpulkan RSS feed setiap lima menit dan menghitung skor sentimen dari berita utama. Output dari program Python adalah file CSV yang memiliki kolom berikut: skor analisis sentimen, headline berita, URL, sumber, dan timestamp.

Untuk menghitung sentimen, kami menggunakan leksikon sentimen AFINN [18], yang merupakan daftar lebih dari 2400 kata bahasa Inggris yang dinilai untuk sentimen yang merupakan bilangan bulat antara minus lima (negatif) dan plus lima (positif).

File CSV yang dihasilkan oleh program Python kemudian dipindahkan ke HDFS untuk analisis lebih lanjut. Untuk ini, kerangka Apache Flume digunakan. Konfigurasi Flume yang digunakan mencakup sumber direktori spool dan sink HDFS. File CSV yang dihasilkan oleh program Python diserap dari direktori spool ke HDFS menggunakan Flume. Box 7.9 menunjukkan konfigurasi Flume yang digunakan. Dengan kumpulan kumpulan data dalam HDFS, program MapReduce dan Pig digunakan untuk melakukan komputasi untuk semua fitur aplikasi seperti membagi berita ke dalam kategori berdasarkan skor sentimen, menganalisis lalu lintas berita setiap jam, dan menentukan kata dan topik yang sedang tren. Aplikasi web Flask digunakan untuk menampilkan hasil.

**Box 7.8: Python program for aggregating news articles**

```
import feedparser
import threading
import sys
import time
import subprocess
from datetime import datetime

refresh_time = 300 #in seconds

rss_feeds = [
('http://rss.cnn.com/rss/cnn_topstories.rss', 'CNN'),
('http://rss.nytimes.com/services/xml/rss/nyt/HomePage.xml',
'The New York Times'),
('http://www.wsj.com/xml/rss/3_7085.xml', 'Wall Street Journal'),
('http://www.news.gatech.edu/rss/all', 'Georgia Tech News')]

csv_file = open("Output.csv","a")
```

```
sentiments={}
file = open('AFINN-111.txt')
lines = file.readlines()
for line in lines:
s = line.split("\t")
sentiments[s[0]] = s[1].strip()
file.close()
```

```
def get_sentiment_score(phrase):
   phrase = phrase.strip()
   total = 0.0
   for word in phrase.split():
      word = word.lower()
      for char in '[!@#$)(*<>=+/:;&%#|{},.? ']-':
         word = word.replace(char,'')
      if word in list(sentiments):
         total = total + float(sentiments[word])
   return total
```

```
last_round_headlines = []
this_round_headlines = []
```

```
def get_headlines(rss_feed):
   feed = feedparser.parse(rss_feed[0])
   for entry in feed.entries:
      title = entry.title.encode('utf-8')
      this_round_headlines.append(title)
      if title not in last_round_headlines:
         score = get_sentiment_score(title)
         title = title.replace(',','')
         #Format:  # score, title, link, source, timestamp
         to_print = "" + str(score) + ", " + title + ", "
          + entry.link.encode('utf-8') + ", " + rss_feed[1] +
          ", " + str(datetime.now())
         csv_file.write(to_print)
```

**Box 7.9: Flume configuration**

```
agent.sources = pstream
agent.channels = memoryChannel
agent.channels.memoryChannel.type = memory
agent.channels.memoryChannel.capacity = 10000000
agent.channels.memoryChannel.transactionCapacity = 10000000
agent.sources.pstream.channels = memoryChannel
agent.sources.pstream.type = spooldir
agent.sources.pstream.spoolDir = test
agent.sinks = hdfsSink
agent.sinks.hdfsSink.type = hdfs
agent.sinks.hdfsSink.channel = memoryChannel
agent.sinks.hdfsSink.hdfs.path = /spooldirtest
```

```
agent.sinks.hdfsSink.hdfs.fileType = DataStream
agent.sinks.hdfsSink.hdfs.writeFormat = Text
agent.sinks.hdfsSink.hdfs.batchSize = 10000000
agent.sinks.hdfsSink.hdfs.rollCount = 0
agent.sinks.hdfsSink.hdfs.rollInterval = 0
agent.sinks.hdfsSink.hdfs.rollSize = 0
agent.sinks.hdfsSink.hdfs.fileSuffix = .csv
```

Box 7.10 menunjukkan program MapReduce untuk menemukan frase dua kata yang paling umum dari input.

**■ Box 7.10: MapReduce program for finding the most common two-word phrase from the input**

```
from mrjob.job import MRJob
import string

stop_words = ["for","of","the","in","a",
      "to","is","news", "breaking", "and",
      "you", "by", "your", "on", "at",
      "as", "this", "it", "with", "from"]

exclude = set(string.punctuation)

#----- Load Ignore Words Dict ---
stopFile = open('StopWords.txt')
lines = stopFile.readlines()
for line in lines:
   s = line.split("\t")
   stop_words.append(s)
stopFile.close()
```

**■ Box 7.10: MapReduce program for finding the most common two-word phrase from the input**

```
from mrjob.job import MRJob
import string

stop_words = ["for","of","the","in","a",
      "to","is","news", "breaking", "and",
      "you", "by", "your", "on", "at",
      "as", "this", "it", "with", "from"]

exclude = set(string.punctuation)

#----- Load Ignore Words Dict ---
stopFile = open('StopWords.txt')
lines = stopFile.readlines()
for line in lines:
   s = line.split("\t")
   stop_words.append(s)
stopFile.close()
```

```
        phrase = '%s %s' %(word1, word2)
        phrases.append(phrase)

      link = data[2].strip()
      for phrase in phrases:
         yield phrase, (1, sentiment, link)

   def reducer(self, key, list_of_values):
      totalCount = 0.0
      totalSentiment = 0.0
      list_of_links = []
      for item in list_of_values:
```

```
            #Skip if the link has already been processed
            if item[2] in list_of_links:
               continue

            #Add count
            totalCount = totalCount + item[0]
            #Add sentiment
            totalSentiment = totalSentiment + float(item[1])
            #Append links
            list_of_links.append(item[2])

         avgSentiment = totalSentiment / totalCount
         yield None, (totalCount, key, avgSentiment, list_of_links)

      def reducer2(self, _, list_of_values):
         #Print top 25
         count = 0
         for item in sorted(list_of_values, reverse = True):
            if count > 25:
               break
            print item
            count = count + 1

      def steps(self):
         return [self.mr(mapper=self.mapper,
         reducer=self.reducer), self.mr(reducer=self.reducer2)]

if __name__ == '__main__':
   MyMRJob.run()
```

Box 7.11 menunjukkan program Pig untuk menemukan kata paling umum dari berita.

**■ Box 7.11: Pig program for finding the most common word from the news**

```
data = LOAD 'data.csv' USING PigStorage(',') as
(sentiment:float, headline:chararray, link:chararray);
/* Remove any duplicates */
data = DISTINCT data;

/* Group by each headline*/
headlines = group data BY headline;
words = FOREACH data GENERATE flatten(TOKENIZE(headline)) as wordTuple;
C = group words by wordTuple;

/* Get count, organize by count, limit to the top 50 */
wordCount = foreach C generate COUNT(words) as count, group as word;
OUT = ORDER wordCount by count DESC;
OUT = LIMIT OUT 50;
STORE OUT into 'mostCommonWords.txt';
```

## 7.6 Apache Oozie

Banyak aplikasi analisis batch memerlukan lebih dari satu tugas MapReduce untuk dirantai untuk melakukan analisis data. Ini dapat dilakukan dengan menggunakan sistem Apache Oozie. Oozie adalah sistem Scheduller aliran kerja yang memungkinkan pengelolaan pekerjaan Hadoop. Dengan Oozie, Anda dapat membuat alur kerja yang merupakan kumpulan tindakan (seperti pekerjaan MapReduce) yang disusun sebagai Grafik Asiklik Langsung (DAG). Ketergantungan kontrol ada di antara tindakan-tindakan dalam alur kerja. Jadi, sebuah aksi dijalankan hanya ketika aksi sebelumnya diselesaikan. Alur kerja Oozie menetapkan urutan tindakan yang perlu dijalankan menggunakan bahasa definisi proses berbasis XML yang disebut bahasa definisi proses Hadoop (hPDL). Oozie mendukung berbagai jenis tindakan seperti Hadoop MapReduce, sistem file Hadoop, Pig, Java, Email, Shell, Hive, Sqoop, SSH, dan tindakan kustom.

### 7.6.1 Alur Kerja Oozie untuk Analisis Data

Mari kita lihat contoh menganalisis data log. Dengan asumsi bahwa data yang diterima memiliki struktur sebagai berikut (termasuk cap waktu dan kode status / kesalahan):

```
■ #timestamp, status/error "2014-07-01 20:03:18",115
"2014-07-01 20:04:15",106
:
"2014-07-01 20:10:15",110
```

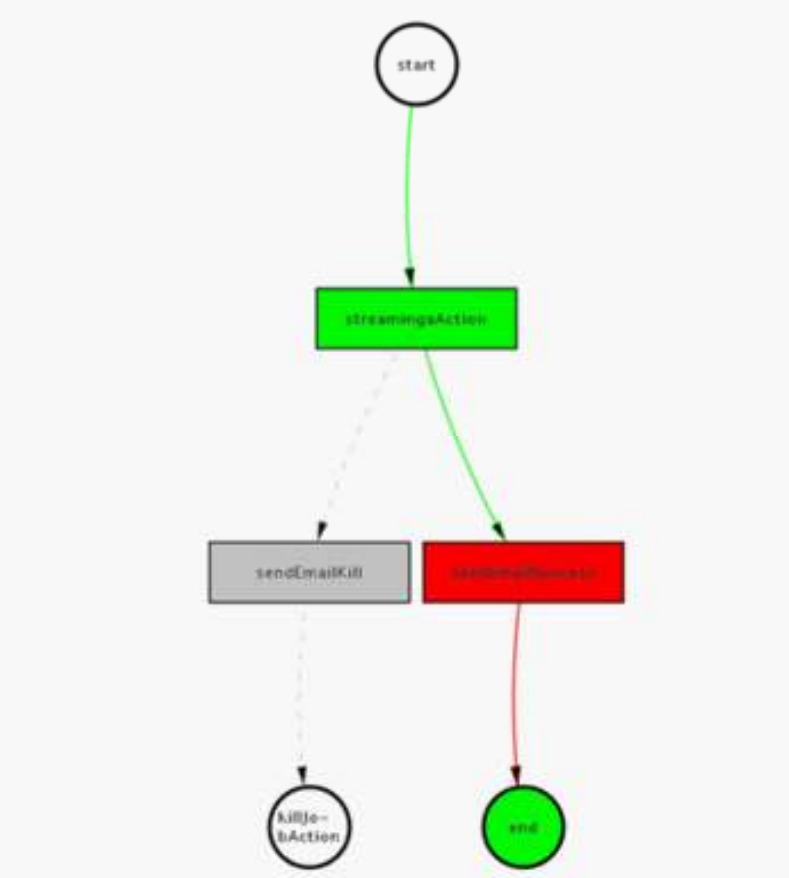
Tujuan dari tugas analisis adalah untuk menemukan hitungan setiap kode status / kesalahan dan menghasilkan output dengan struktur seperti yang ditunjukkan di bawah ini:

```
■ #status/error, count 111, 6
112, 7
113, 12
```

Gambar 7.11 menunjukkan presentasi alur kerja Oozie yang terdiri dari tindakan pekerjaan Hadoopstreaming MapReduce dan tindakan Email yang memberitahukan keberhasilan atau kegagalan pekerjaan.

Box 7.12 dan 7.13 menunjukkan map dan program pengurangan yang dijalankan dalam alur kerja. Program map mem-parsing kode status / kesalahan dari setiap baris di input dan memancarkan key-value pair di mana kuncinya adalah kode status / kesalahan dan

nilainya adalah 1. Program pengurangan menerima key-value pair yang dipancarkan oleh program map yang dikumpulkan dengan kunci yang sama. Untuk setiap kunci, program pengurangan menghitung hitungan dan memancarkan key-value pair di mana kuncinya adalah kode status / kesalahan dan nilainya adalah hitungan.




Gambar 7.11: *Oozie workflow*

**Box 7.13: Reduce program for computing counts of status/error codes**

```
#!/usr/bin/env python
from operator import itemgetter
import sys

current_key = None
current_count = 0
key = None

# input comes from STDIN
for line in sys.stdin:
   line = line.strip()

   key, count = line.split('\t', 1)
```

```
    count = int(count)

    if current_key == key:
       current_count += count
    else:
       if current_key:
          unpackedKey = current_key.split(',')
          print '%s%s' % (current_key, current_count)
       current_count = count
          current_key = key

if current_key == key:
    unpackedKey = current_key.split(',')
    print '%s\t%s' % (current_key, current_count)
```

Box 7.14 menunjukkan spesifikasi aliran kerja Oozie yang ditunjukkan pada Gambar 7.11. Aliran kerja Oozie telah diparameterisasi dengan variabel-variabel dalam definisi alur kerja. Nilai dari variabel-variabel ini disediakan dalam file properti pekerjaan yang ditunjukkan pada Box 7.15

**■ Box 7.14: Oozie workflow for computing counts of status/error codes**

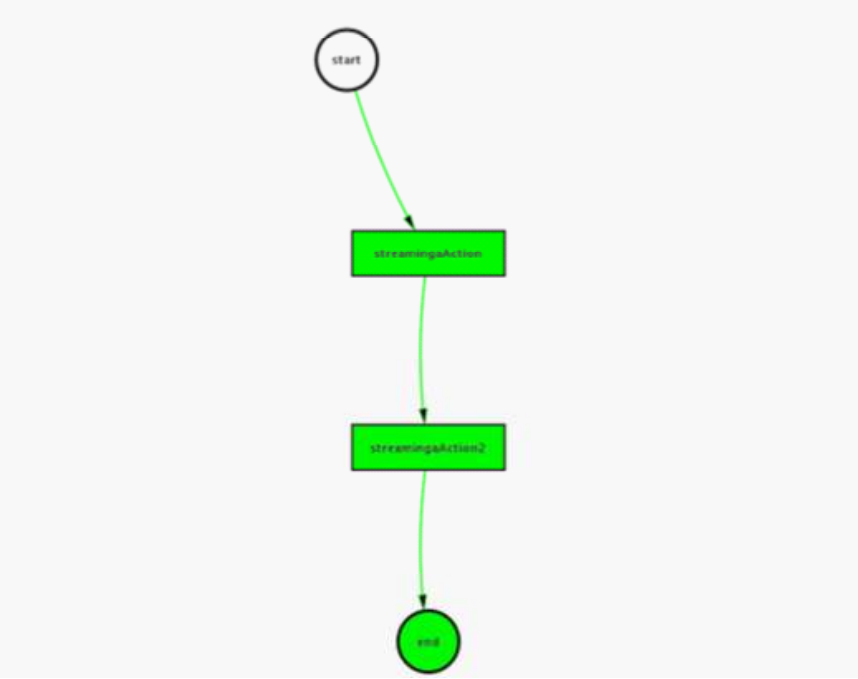
```
<workflow-app name="PythonOozieApp" xmlns="uri:oozie:workflow:0.1">
 <start to="streamingaAction"/>
  <action name="streamingaAction">
<map-reduce>
    <job-tracker>${jobTracker}</job-tracker>
    <name-node>${nameNode}</name-node>
    <prepare>
     <delete path="${outputDir}"/>
    </prepare>
    <streaming>
     <mapper>python Mapper.py</mapper>
     <reducer>python Reducer.py</reducer>
    </streaming>
    <configuration>
<property>
<name>oozie.libpath</name>
    <value>${oozieLibPath}/mapreduce-streaming</value>
   </property>
     <property>
      <name>mapred.input.dir</name>
      <value>${inputDir}</value>
     </property>
     <property>
      <name>mapred.output.dir</name>
      <value>${outputDir}</value>
     </property>
<property>
      <name>mapred.reduce.tasks</name>
      <value>1</value>
```

```
      </property>
     </configuration>
<file>${appPath}/Mapper.py#Mapper.py</file>
<file>${appPath}/Reducer.py#Reducer.py</file>
  </map-reduce>
<ok to="sendEmailSuccess"/>
<error to="sendEmailKill"/>
  </action>

<action name="sendEmailSuccess">
  <email xmlns="uri:oozie:email-action:0.1">
    <to>${emailToAddress}</to>
    <subject>Status of workflow ${wf:id()}</subject>
    <body>The workflow ${wf:id()} completed successfully</body>
  </email>
  <ok to="end"/>
  <error to="end"/>
 </action>
 <action name="sendEmailKill">
  <email xmlns="uri:oozie:email-action:0.1">
   <to>${emailToAddress}</to>
   <subject>Status of workflow ${wf:id()}</subject>
   <body>The workflow ${wf:id()} had issues and was killed.
The error message is:  ${wf:errorMessage(wf:lastErrorNode())}</body>
  </email>
  <ok to="killJobAction"/>
  <error to="killJobAction"/>
 </action>

  <kill name="killJobAction">
   <message>"Killed job due to error:
${wf:errorMessage(wf:lastErrorNode())}"</message>
  </kill>
 <end name="end" />
</workflow-app>
      <value>${oozieLibPath}/mapreduce-streaming</value>
    </property>
      <property>
       <name>mapred.input.dir</name>
       <value>${inputDir}</value>
      </property>
      <property>
       <name>mapred.output.dir</name>
       <value>${outputDir}</value>
      </property>
<property>
       <name>mapred.reduce.tasks</name>
       <value>1</value>
      </property>
     </configuration>
<file>${appPath}/Mapper.py#Mapper.py</file>
<file>${appPath}/Reducer.py#Reducer.py</file>
  </map-reduce>
<ok to="sendEmailSuccess"/>
<error to="sendEmailKill"/>
  </action>
```

```
queueName=default

oozie.libpath=${nameNode}/user/hduser/share/lib
oozie.use.system.libpath=true
oozie.wf.rerun.failnodes=true

oozieProjectRoot=${nameNode}/user/hduser/oozieProject
appPath=${oozieProjectRoot}/pythonApplication
oozie.wf.application.path=${appPath}
oozieLibPath=${oozie.libpath}

inputDir=${oozieProjectRoot}/pythonApplication/data/
outputDir=${appPath}/output
```

**■ Box 7.15: Job properties file for Oozie workflow**

```
nameNode=hdfs://master:54310
jobTracker=master:54311
```




Gambar 7.12: *Alur kerja Oozie untuk menghitung status / kode kesalahan dengan jumlah maksimum*

Sekarang mari kita lihat aliran kerja yang lebih rumit yang memiliki dua pekerjaan MapReduce. Memperluas contoh yang dijelaskan sebelumnya di bagian ini, katakanlah kita ingin menemukan kode status / kesalahan dengan jumlah maksimum. Pekerjaan MapReduce di alur kerja sebelumnya menghitung jumlah untuk setiap kode status / kesalahan. Pekerjaan MapReduce kedua, yang menggunakan output dari pekerjaan MapReduce pertama menghitung jumlah maksimum. Map dan program pengurangan untuk pekerjaan MapReduce kedua ditunjukkan pada Box 7.16 dan 7.17.

Gambar 7.12 menunjukkan representasi aliran Kerja untuk status komputasi / kode kesalahan dengan jumlah maksimum. Spesi fi kasi aliran kerja ditunjukkan pada Box 7.18.

**■ Box 7.16: Map program for computing status/error code with maximum count**

```
#!/usr/bin/env python
import sys

#Data format
#"2014-07-01 20:03:18",115

# input comes from STDIN (standard input)
for line in sys.stdin:
   # remove leading and trailing whitespace
   line = line.strip()
   # split the line into words
   data = line.split('\t')

   #For aggregation by minute
   print '%s\t%s' % (data[0], data[1])
```

**■ Box 7.17: Reduce program for computing status/error code with maximum count**

```
#!/usr/bin/env python
from operator import itemgetter
import sys

current_key = None
current_count = 0
key = None
maxcount=0
maxcountkey=None
```

```
<property>
      <name>mapred.reduce.tasks</name>
      <value>1</value>
     </property>
    </configuration>
<file>${appPath}/Mapper.py#Mapper.py</file>
<file>${appPath}/Reducer.py#Reducer.py</file>
  </map-reduce>
<ok to="sendEmailSuccess"/>
<error to="sendEmailKill"/>
  </action>

<action name="sendEmailSuccess">
  <email xmlns="uri:oozie:email-action:0.1">
    <to>${emailToAddress}</to>
    <subject>Status of workflow ${wf:id()}</subject>
    <body>The workflow ${wf:id()} completed successfully</body>
  </email>
  <ok to="end"/>
  <error to="end"/>
 </action>
 <action name="sendEmailKill">
  <email xmlns="uri:oozie:email-action:0.1">
   <to>${emailToAddress}</to>
   <subject>Status of workflow ${wf:id()}</subject>
   <body>The workflow ${wf:id()} had issues and was killed.
```

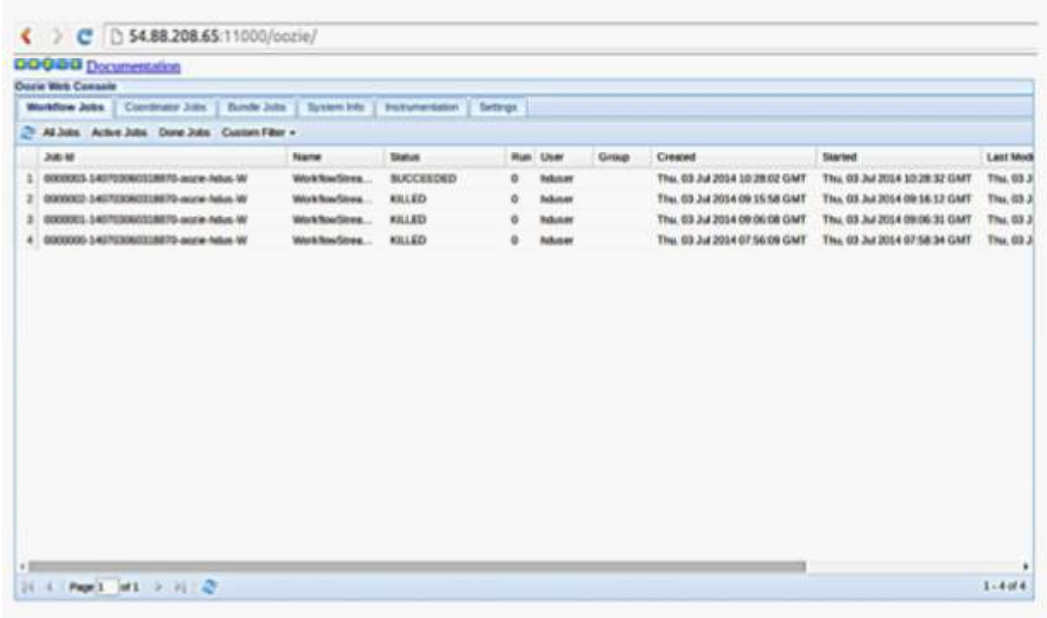
**■ Box 7.18: Oozie workflow for computing status/error code with maximum count**

```
<workflow-app name="PythonOozieApp" xmlns="uri:oozie:workflow:0.1">
 <start to="streamingaAction"/>
  <action name="streamingaAction">
<map-reduce>
    <job-tracker>${jobTracker}</job-tracker>
    <name-node>${nameNode}</name-node>
    <prepare>
    <streaming>
     <mapper>python Mapper.py</mapper>
     <reducer>python Reducer.py</reducer>
    </streaming>
    <configuration>
<property>
<name>oozie.libpath</name>
    <value>${oozieLibPath}/mapreduce-streaming</value>
   </property>
     <property>
      <name>mapred.input.dir</name>
      <value>${inputDir}</value>
     </property>
     <property>
      <name>mapred.output.dir</name>
      <value>${outputDir}</value>
     </property>
```

```
<property>
      <name>mapred.reduce.tasks</name>
      <value>1</value>
     </property>
    </configuration>
<file>${appPath}/Mapper.py#Mapper.py</file>
<file>${appPath}/Reducer.py#Reducer.py</file>
  </map-reduce>
    <ok to="streamingaAction2"/>
    <error to="killJobAction"/>
  </action>

<action name="streamingaAction2">
<map-reduce>
    <job-tracker>${jobTracker}</job-tracker>
    <name-node>${nameNode}</name-node>
    <streaming>
     <mapper>python Mapper1.py</mapper>
     <reducer>python Reducer1.py</reducer>
    </streaming>
    <configuration>
<property>
<name>oozie.libpath</name>
    <value>${oozieLibPath}/mapreduce-streaming</value>
   </property>
```

```
     <property>
      <name>mapred.input.dir</name>
      <value>${outputDir}</value>
     </property>
     <property>
      <name>mapred.output.dir</name>
      <value>${outputDir}/output2</value>
     </property>
<property>
      <name>mapred.reduce.tasks</name>
      <value>1</value>
     </property>
    </configuration>
<file>${appPath}/Reducer1.py#Reducer1.py</file>
  </map-reduce>
    <ok to="end"/>
    <error to="killJobAction"/>
  </action>

  <kill name="killJobAction">
   <message>"Killed job due to error:
${wf:errorMessage(wf:lastErrorNode())}"</message>
  </kill>
 <end name="end" />
</workflow-app>
```

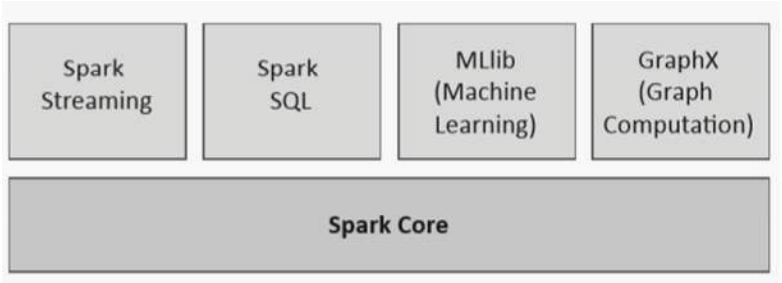
Gambar 7.13 menunjukkan screenshot dari konsol web Oozie yang dapat digunakan untuk memantau status aliran kerja Oozie.




Gambar 7.13: *Screenshoot Oozie web console*

## 7.7 Apache Spark

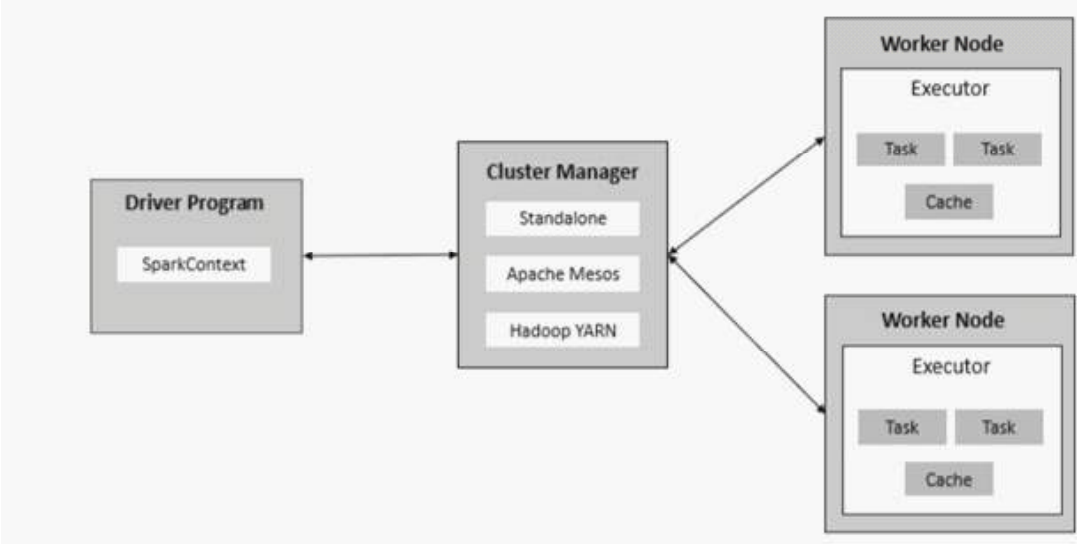
Apache Spark adalah kerangka kerja komputasi cluster open source untuk analisis data [62]. Spark mendukung komputasi cluster dalam memori dan berjanji untuk lebih cepat dari Hadoop. Spark mendukung berbagai alat tingkat tinggi untuk analisis data seperti Streaming Spark untuk pekerjaan streaming, Spark SQL untuk analisis data terstruktur, perpustakaan pembelajaran mesin MLlib untuk Spark, dan pemrosesan forgraf GraphX. Spark memungkinkan kueri real-time, batch, dan interaktif serta menyediakan API untuk bahasa Scala, Java, dan Python.




Gambar 7.14: *Spark tools*

- Spark Core: Spark Core menyediakan fungsionalitas umum (seperti Schedulleran tugas dan input / output), yang digunakan oleh komponen Spark lainnya. Spark menyediakan abstraksi data yang disebut dataset terdistribusi tangguh (RDD) yang merupakan kumpulan elemen yang dipartisi di seluruh node dalam kluster Spark. Elemen RDD dapat dioperasikan secara paralel di cluster. RDD adalah kumpulan objek yang tidak berubah dan terdistribusi.
- Spark Streaming: Spark Streaming adalah komponen Spark untuk analisis data streaming seperti data sensor, data aliran klik, log server web, dll.
- Spark SQL: Spark SQL adalah komponen Spark yang memungkinkan pembuatan kueri interaktif data menggunakan kueri SQL. Spark SQL dijelaskan secara rinci di Bab-9 di mana kami mendeskripsikan alat untuk kueri interaktif.
- Spark MLlib: Spark MLlib adalah pustaka pembelajaran mesin Spark yang menyediakan implementasi algoritma pembelajaran mesin yang umum digunakan untuk pengelompokan, klasifikasi, regresi, pemfilteran kolaboratif, dan pengurangan dimensi.
- Spark GraphX: Spark GraphX adalah komponen untuk melakukan komputasi grafik. GraphX menyediakan implementasi algoritma grafik umum seperti PageRank, komponen terhubung, dan penghitungan segitiga.

Gambar 7.15 menunjukkan komponen cluster Spark. Setiap aplikasi Spark terdiri dari program driver dan dikoordinasikan oleh objek SparkContext. Spark mendukung berbagai manajer cluster termasuk manajer cluster mandiri Spark, Apache Mesos dan Hadoop YARN. Manajer klaster mengalokasikan sumber daya untuk aplikasi di node pekerja. Eksekutor yang dialokasikan pada node pekerja menjalankan kode aplikasi sebagai beberapa tugas. Aplikasi diisolasi satu sama lain dan dijalankan dalam proses eksekutornya sendiri di node pekerja.




Gambar 7.15: *Komponen Spark cluster*

Spark hadir dengan skrip spark-ec2 (di direktori spark / ec2) yang memudahkan pengaturan klaster Spark di Amazon EC2. Dengan skrip spark-ec2, Anda dapat dengan mudah meluncurkan, mengelola, dan mematikan klaster Spark di Amazon EC2. Untuk memulai cluster Spark, gunakan perintah berikut:

```
■ ./spark-ec2 -k <keypair> -i <key-file> -s <num-slaves>
launch <cluster-name> -instance-type=<INSTANCE_TYPE>
```

Setup cluster spark pada EC2 dikonfigurasi untuk menggunakan HDFS sebagai sistem file default-nya. Untuk menganalisis konten file, file harus terlebih dahulu disalin ke HDFS menggunakan perintah berikut:

```
■ bin/hadoop fs -put file.txt file.txt
```

Spark mendukung mode shell yang dengannya Anda dapat menjalankan perintah secara interaktif untuk menganalisis data. Untuk meluncurkan shell Spark Python, jalankan perintah berikut:

```
■ ./bin/pyspark
```

Saat Anda meluncurkan shell PySpark, SparkContext dibuat dengan nama variabel sc.

**Membuat RDD**

RDD dapat dibuat dengan memparalelkan koleksi yang ada atau dengan memuat dataset eksternal seperti yang ditunjukkan pada box di bawah ini:

```
#Create RDD from a local file
lines = sc.textFile("file:///root/spark/README.md")

#Create RDD by parallelizing an existing collection
data = sc.parallelize([1, 2, 2, 3, 3, 4, 5])
```

### 7.6.1 Spark Operation

Spark RDD mendukung dua jenis operasi:
Transformasi: Transformasi digunakan untuk membuat kumpulan data baru dari yang sudah ada.
Tindakan: Tindakan mengembalikan nilai ke program driver setelah menjalankan komputasi pada kumpulan data.

**Transformasi**

Mari kita lihat beberapa transformasi yang umum digunakan dengan contoh. Sebagai contoh, kami akan menggunakan tiga dataset seperti di bawah ini:

```
lines = sc.textFile("file:///root/spark/README.md")

data1 = sc.parallelize([1, 2, 2, 3, 3, 4, 5])
data2 = sc.parallelize([3, 4, 5, 6, 7, 8])
```

**Map**

Transformasi Map mengambil sebagai masukan fungsi yang diterapkan ke setiap elemen dari dataset dan memetakan setiap item masukan ke item lain.

```
#map transformation example
lineLengths = lines.map(lambda s:  len(s))
lineLengths.take(5)
[14, 0, 78, 72, 73]
```

### Filter

Transformasi filter menghasilkan kumpulan data baru dengan memfilter kumpulan data sumber menggunakan fungsi yang ditentukan

```
■ #filter transformation example
filteredLines = lines.filter(lambda line:  line.find('Spark')>0)
filteredLines.take(3)
[u'# Apache Spark', u'rich set of higher-level tools including Spark SQL
for SQL and structured', u'and Spark Streaming for stream processing.']
```

### reduceByKey

Transformasi reduceByKey bila diterapkan pada kumpulan data yang berisi key-value pair , nilai agregat dari setiap kunci menggunakan fungsi yang ditentukan.

```
■ # reduceByKey transformation example
splitLines = lines.flatMap(lambda line:  line.split())
words=splitLines.map(lambda word:  (word, 1))

counts=words.reduceByKey(lambda a, b:  a+b)
counts.take(5)
[(u'all', 1), (u'when', 1), (u'"local"', 1),
(u'including', 3), (u'computation', 1)]
```

### flatMap

Transformasi flatMap transformasi mengambil input fungsi yang diterapkan ke setiap elemen dari dataset. Transformasi flatMap dapat memetakan setiap item masukan ke nol atau lebih item output.

```
■ #flatMap transformation example
splitLines = lines.flatMap(lambda line:  line.split())
splitLines.take(10)
[u'#', u'Apache', u'Spark', u'Spark', u'is',
u'a', u'fast', u'and', u'general', u'cluster']
```

### Sample

Transformasi Sample mengambil sampel data dengan atau tanpa penggantian.

```
■ #sample transformation example
datasample = data1.sample(False, 0.5)
datasample.collect()
[3, 3, 4]
```

### Union

Transformasi Union menghasilkan kumpulan data baru dari gabungan dua kumpulan data

```
#union transformation example
data = data1.union(data2)
data.collect()
[1, 2, 2, 3, 3, 4, 5, 3, 4, 5, 6, 7, 8]
```

### Intersection

Transformasi intersection menghasilkan kumpulan data baru dari perpotongan dua kumpulan data.

```
#intersection transformation example
data = data1.intersection(data2)
data.collect()
[4, 5, 3]
```

### Join

Transformasi join menghasilkan kumpulan data baru dengan menggabungkan dua kumpulan data yang berisi key-value pair .

```
#join transformation example
a=sc.parallelize([('John', 1), ('Tom', 2), ('Ben', 3)])
b=sc.parallelize([('John', 'CA'), ('Tom', 'GA'), ('Ben', 'VA')])
c=a.join(b)
c.collect()
[('Ben', (3, 'VA')), ('John', (1, 'CA')), ('Tom', (2, 'GA'))]
```

Transformasi lazy dan tidak dihitung sampai suatu tindakan memerlukan hasil untuk dikembalikan ke program driver. Dengan menghitung transformasi secara malas, Spark dapat melakukan operasi dengan cara yang lebih efisien karena operasi dapat dikelompokkan bersama. Spark API memungkinkan menggabungkan transformasi dan tindakan.

### *Actions*

Mari kita lihat beberapa action yang umum digunakan dengan contoh:

### *Reduce*

Tindakan reduce menggabungkan elemen-elemen dalam sebuah dataset menggunakan fungsi yang ditentukan.

```
■ #reduce transformation example
lineLengths = lines.map(lambda s:  len(s))
totalLength = lineLengths.reduce(lambda a, b:  a + b)
3526
```

### *Collect*

Tindakan collect digunakan untuk mengembalikan semua elemen hasil sebagai array.

```
■ #Map transformation example
lineLengths = lines.map(lambda s:  len(s))
lineLengths.collect()
[14, 0, 78, 72, 73, ...  , 70]
```

### *First*

First action mengembalikan elemen pertama dalam kumpulan data

```
■ #Map transformation example
lines.first()
u'# Apache Spark'
```

### *Count*

Count action mengembalikan jumlah elemen dalam dataset.

```
■ #count transformation example
lines.count()
98
```

### *Take*

Take action mengembalikan n elemen pertama dalam sebuah dataset.

```
■ #Map transformation example
lines.take(3)
[u'# Apache Spark', u'', u'Spark is a fast and general
cluster computing system for Big Data.  It provides']
```

***takeSample***

takeSample action mengembalikan sampel yang berisi sejumlah elemen tertentu dari kumpulan data dengan atau tanpa penggantian.

```
#takeSample transformation example
data1.takeSample(False, 2)
[2, 2]
```

***saveAsTextFile***

saveAsTextFile action menulis elemen dalam dataset ke file teks baik di sistem file lokal atau HDFS.

```
#saveAsTextFile transformation example
lines.saveAsTextFile('/path/to/file')
```

***saveAsSequenceFile***

saveAsSequenceFile action menulis elemen dalam dataset ke Hadoop SequenceFile pada sistem file lokal atau HDFS.

```
#saveAsSequenceFile transformation example
data.saveAsSequenceFile('/path/to/file')
```

Sekarang mari kita lihat aplikasi Spark mandiri yang menghitung jumlah kata dalam sebuah file. Box 7.19 menunjukkan program Python untuk menghitung jumlah kata. Program ini menggunakan fungsi map dan pengurangan. Transformasi flatMap dan map mengambil sebagai masukan fungsi yang diterapkan ke setiap elemen dari dataset. Sementara fungsi flatMap dapat memetakan setiap item masukan ke nol atau lebih item output, fungsi map memetakan setiap item masukan ke item lain. Transformasi mengambil sebagai masukan, fungsi yang diterapkan ke elemen data. Fungsi masukan bisa dalam bentuk ekspresi lambda Python atau fungsi lokal. Dalam contoh jumlah kata, flatMap menerima ekspresi lambda yang membagi setiap baris file menjadi kata-kata. Transformasi map mengeluarkan key-value pair di mana kuncinya adalah kata dan nilainya adalah 1. Transformasi reduceByKey menggabungkan nilai dari setiap kunci menggunakan fungsi yang ditentukan (tambahkan fungsi dalam contoh ini). Terakhir, tindakan kumpulkan digunakan untuk mengembalikan semua elemen hasil sebagai larik.

**Box 7.19: Apache Spark Python program for computing word count**

```
from operator import add
from pyspark import SparkContext

sc = SparkContext(appName="WordCountApp")
lines = sc.textFile("file.txt")
counts = lines.flatMap(lambda x:
x.split(' ')).map(lambda x:  (x, 1)).reduceByKey(add)

output = counts.collect()

for (word, count) in output:
   print "%s:  %i" % (word, count)
```

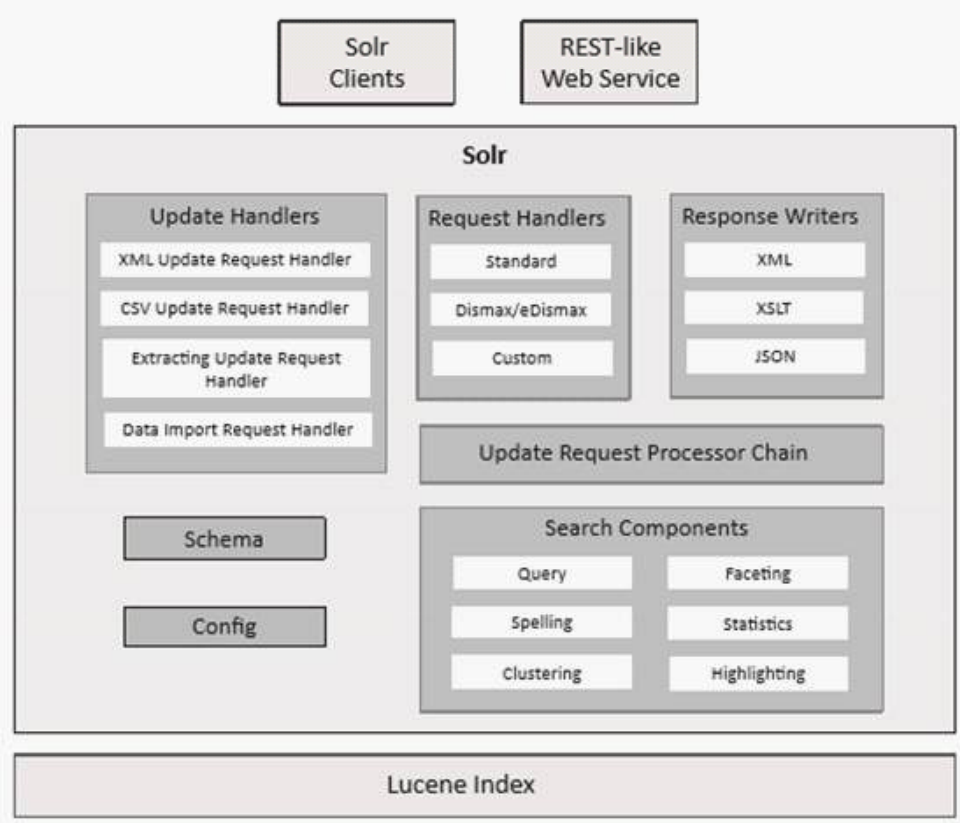
## 7.8 Search

### 7.8.1 Apache Solr

Apache Solr adalah kerangka kerja sumber terbuka dan skalabel untuk mencari data. Solr dibangun di atas Apache Lucene, yang merupakan pustaka sumber terbuka untuk pengindeksan dan pencarian. Untuk mengaktifkan pencarian dokumen, Solr membuat indeks dokumen. Solr dapat mengindeks dokumen dalam format XML, JSON, CSV, dan biner. Solr menyediakan layanan web seperti REST yang dapat digunakan untuk pengindeksan dan kueri.

Gambar 7.16 menunjukkan komponen Solr. Solr menggunakan Lucene untuk membangun dan memelihara indeks terbalik yang berisi pemetaan dari istilah pencarian ke dokumen. Sementara Lucene mengelola struktur indeks dan menjalankan kueri, Solr digunakan untuk mendefinisikan struktur indeks. Dengan Solr, struktur indeks dapat didefinisikan dalam file XML (schema.xml), yang berisi definisi berbagai bidang dan tipe data yang digunakan dalam indeks. Solr menyediakan field dinamis yang bisa digunakan untuk mendefinisikan tipe field secara otomatis tanpa secara eksplisit mendefinisikannya dalam skema indeks. Solr berjalan sebagai aplikasi Javaweb dan menyediakan layanan web seperti REST berdasarkan HTTP. Solr juga dapat diakses menggunakan klien Solr yang tersedia untuk bahasa pemrograman yang berbeda seperti Python, Java, PHP dan Ruby. Penangan Pembaruan Solr memproses permintaan dan memperbarui indeks. Jenis permintaan yang didukung oleh penangan pembaruan termasuk tambahkan, hapus, komit, dan optimalkan. Ketika dokumen baru ditambahkan ke dalam indeks, dokumen tersebut hanya terlihat di pencarian setelah dokumen tersebut dimasukkan ke indeks.

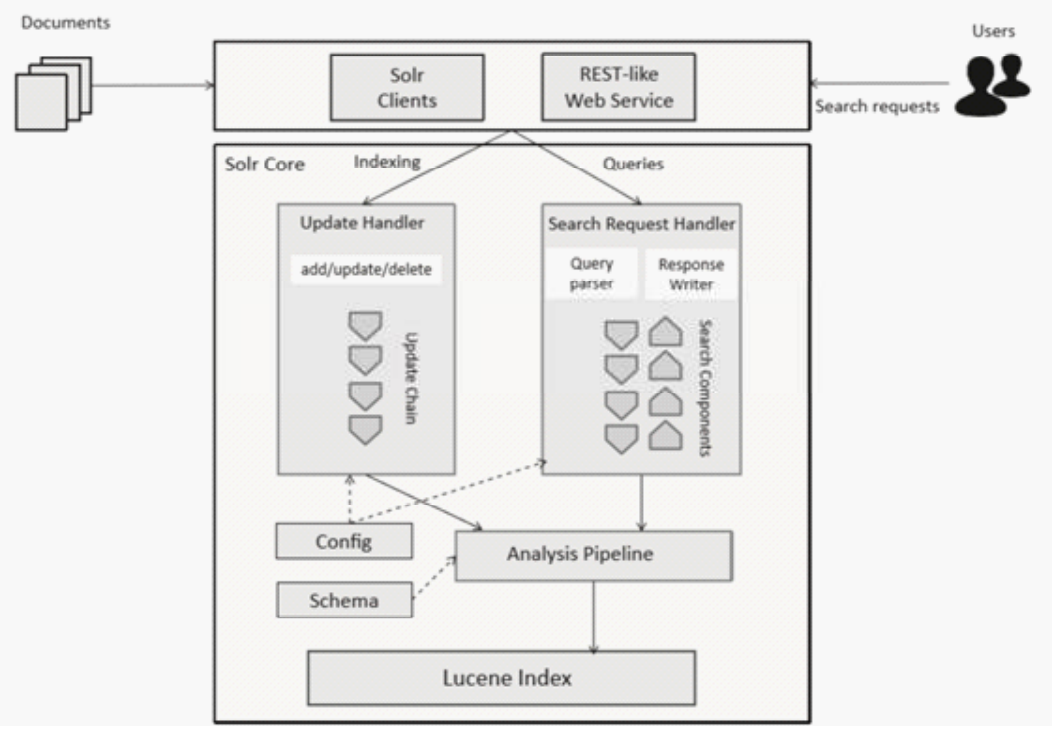
Solr menyediakan opsi komit normal (atau komit keras), komit lunak, dan komit otomatis untuk melakukan perubahan ke indeks. Komit normal, melakukan semua perubahan ke file indeks Lucene ke disk. Dalam soft-commit, semua perubahan dilakukan ke struktur data Lucene tetapi perubahan tidak dilakukan ke disk. Soft-commit digunakan untuk fitur pencarian hampir real-time, yang akan dibahas nanti di bagian ini. Opsi komit otomatis memungkinkan perubahan dilakukan secara teratur atau ketika jumlah dokumen yang tidak terikat menjadi lebih besar dari ambang batas. Penangan Permintaan Solr memproses permintaan masuk yang bisa berupa permintaan kueri atau permintaan pembaruan indeks. Penulis Respon Solr menghasilkan respon dalam format yang diinginkan (seperti XML, CSV, JSON). Solr menyediakan berbagai Komponen Pencarian, yang menyediakan implementasi fitur pencarian seperti faceting, penyorotan, dll., Dijelaskan nanti di bagian ini. Semua permintaan pembaruan dijalankan melalui rangkaian plugin yang disebutUpdateRequestProcessorChain. Gambar7.16 menunjukkan komponen-komponen yang terlibat dalam proses pengindeksan dan kueri.




Gambar 7.16: *Komponen Apache Solr*

Solr menyediakan fungsionalitas penyebaran yang disebut SolrCloud yang membuatnya lebih mudah untuk mengatur cluster server Solr. SolrCloud menyediakan kemampuan

pengindeksan dan pencarian terdistribusi. SolrCloud menggunakan Zookeeper untuk konfigurasi dan koordinasi terpusat. Solr Cloud memungkinkan load balancing dan fail-over untuk kueri dan penerapan Solr yang sangat tersedia dan toleran terhadap kesalahan. Setiap instance Solr dapat memiliki beberapa indeks. Solr dapat mengukur untuk mengindeks jutaan dokumen dan menangani kueri dari jutaan pengguna.




Gambar 7.17: *Indexing dan Querying pada Apache Sol*

Mari kita lihat beberapa fitur utama Solr:

- Faceting: Faceting adalah fitur dalam Solr yang memungkinkan hasil dikelompokkan berdasarkan bidang tertentu atau kriteria yang ditentukan. Dengan faceting, pengguna bisa memperhalus hasil pencarian. Misalnya, faceting dapat digunakan dalam sistem pencarian untuk aplikasi eCommerce untuk mengelompokkan produk berdasarkan harga, vendor, warna, ukuran, dll.
- Auto-suggest: Fitur saran otomatis, juga auto-complete atau type-ahead search memberikan saran kepada pengguna saat mereka mengetik dalam kueri. Saran otomatis memberikan daftar kueri yang disarankan berdasarkan dokumen yang diindeks.

- Pemeriksaan Ejaan: Fitur pemeriksaan ejaan di Solr memungkinkan koreksi ejaan dalam kueri dengan memberikan saran kueri berdasarkan istilah serupa dalam indeks.
- Highlighting: Fitur highlighting klik memungkinkan penyorotan bagian tertentu dari dokumen yang cocok dengan kueri pengguna. Bagian yang cocok disertakan dalam respons kueri dan petunjuk pemformatan disertakan untuk menyoroti bagian yang cocok.
- Grouping: Pengelompokan mengelompokkan dokumen terkait dalam hasil pencarian dan memberikan label ke grup. Pengelompokan mencegah beberapa dokumen dengan konten serupa dikembalikan secara terpisah dalam hasil pencarian. Dokumen serupa dalam hasil pencarian dikelompokkan dalam proses yang disebut clustering online. Solr juga mendukung fl ine-clustering dokumen dalam indeks.
- Spasial searching: Solr mendukung pencarian spasial yang memungkinkan penyaringan hasil pencarian berdasarkan lokasi. Solr dapat mengindeks data spasial seperti garis lintang / bujur,
- Penomoran halaman dan Peringkat: Solr menyediakan penomoran halaman hasil pencarian. Saat pengguna mencari istilah, hanya hasil N teratas yang dikembalikan untuk halaman pertama. Hasilnya diberi peringkat berdasarkan skor relevansi. Manfaat penomoran halaman adalah alih-alih mengembalikan semua dokumen yang cocok, hanya sebagian dokumen yang dikembalikan pada halaman tertentu, yang mempercepat respons.
- Pengelompokan Hasil: Solr mendukung pengelompokan hasil berdasarkan bidang pengelompokan sehingga alih-alih mengembalikan dokumen terpisah, dokumen dikelompokkan bersama dan dokumen teratas di setiap grup dikembalikan. Misalnya, saat mencari koleksi buku, hasilnya dapat dikelompokkan berdasarkan bidang genre. Hasilnya kemudian akan berisi semua genre unik dan buku teratas di setiap genre.
- Near Real-time Search : Fitur pencarian hampir real-time di Solr memungkinkan pencarian dokumen segera setelah diindeks. Fitur ini berguna untuk aplikasi dengan konten yang berubah secara dinamis seperti aplikasi berita dan media sosial.
- More Like This: Fitur ini memungkinkan pengguna untuk menanyakan dokumen yang mirip dengan dokumen yang dikembalikan untuk permintaan pencarian.

### Contoh Solr

Di bagian ini, kami akan menjelaskan contoh pengindeksan dan kueri data dengan Solr. Untuk mengatur Solr, dapatkan rilis terbaru dari situs web Solr. Sebagai contoh, kami akan

membuat penerapan SolrCloud di mesin lokal menggunakan perintah yang ditunjukkan di bawah ini:

```
wget http://apache.arvixe.com/lucene/solr/5.2.1/solr-5.2.1.tgz
tar -xzf solr-5.2.1.tgz
cd solr-5.2.1/

bin/solr start -e cloud -noprompt
```

Ketika Solr dimulai, Anda dapat mengakses interface pengguna admin di http://localhost:8983/solr/.

**Box 7.20: Sample JSON file to index**

```
[
{
   "id" :  "1",
   "name" :  "HTC Desire 816",
   "brand" :  "HTC",
   "display_t" :  "5.5 inch S-LCD Display",
   "memory_s" :  "8 GB",
   "price_f" :  150.00,
   "color" :  "black",
   "features" :  ["Android 4.4 Kit Kat OS","13 MP Rear Facing BSI Camera",
"5 MP Front Facing"]
},

{
   "id" :  "2",
   "name" :  "LG Tribute",
   "brand" :  "LG",
   "display_t" :  "4.5 inch Capacitive Display",
   "memory_s" :  "8 GB",
   "price_f" :  65.00,
   "color" :  "black",
   "features" :  ["Android 4.4 Kit Kat OS",
"5 MP Rear Facing FSI Camera"]
},

{
   "id" :  "3",
   "name" :  "Samsung Galaxy S5 White",
   "brand" :  "Samsung",
   "display_t" :  "5.1inch Full HD Super AMOLED",
   "memory_s" :  "16 GB",
   "price_f" :  499.99,
   "color" :  "white",
   "features" :  ["Android 4.4 Kit Kat OS","16 megapixel camera",
"2.5 GHz quad-core processor"]
},
```

```
{
   "id" :  "4",
   "name" :  "Kyocera Hydro Vibe",
   "brand" :  "Kyocera",
   "display_t" :  "4.5 inch qHD IPS screen",
   "memory_s" :  "8 GB",
   "price_f" :  49.90,
   "color" :  "black",
   "features" :  ["Andriod 4.3 Jelly Bean OS",
"8 MP Rear Facing BSI Camera", "2 MP Front Facing"]
}

]
```

Sebelum kita bisa mulai menggunakan Solr untuk melakukan kueri data, kita perlu mengindeks beberapa data. Box 7.20 menunjukkan contoh file JSON yang akan kita indeks. File ini berisi detail produk tentang empat ponsel.

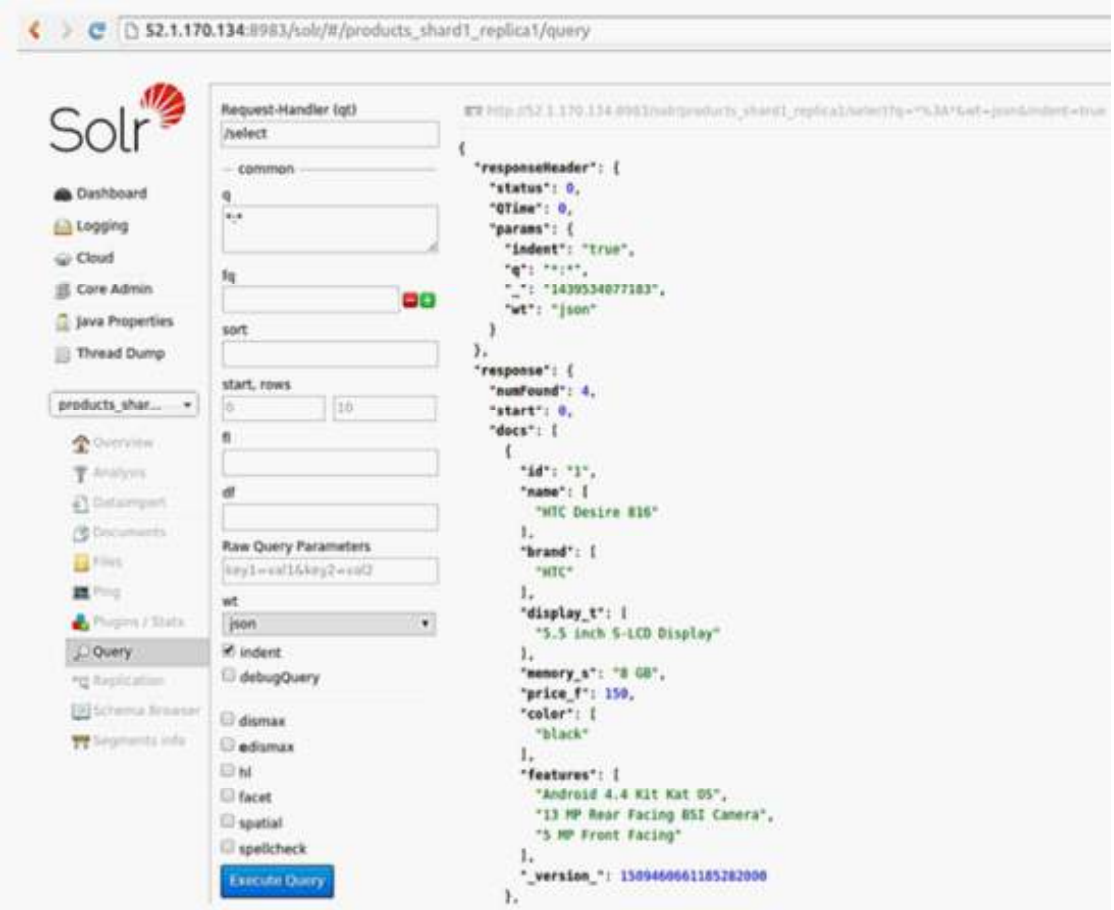
Solr menyediakan utilitas yang disebut post untuk pengindeksan. Unit dasar dari data yang diindeks disebut dokumen. Dokumen adalah kumpulan bidang. Setiap bidang dalam dokumen diindeks atau disimpan atau keduanya. Bidang yang diindeks adalah salah satu yang dapat dicari dan diurutkan, tetapi bidang tersebut tidak dikembalikan dalam hasil pencarian. Selama proses pengindeksan, bidang yang diindeks menjalani fase analisis di mana berbagai transformasi diterapkan. Bidang yang disimpan adalah yang dikembalikan dalam hasil pencarian. Bidang yang disimpan disimpan apa adanya tanpa menjalani analisis.

Solr membutuhkan skema (didefinisikan dalam schema.xml), yang mencakup definisi bidang dan informasi tentang bagaimana bidang harus dianalisis dalam proses pengindeksan. Meskipun direkomendasikan untuk mendefinisikan skema secara eksplisit untuk kontrol halus atas proses pengindeksan, Solr juga menyediakan opsi bidang Dinamis yang memungkinkan Solr mengindeks bidang yang tidak didefinisikan secara eksplisit dalam skema. Bidang dinamis memiliki kartu bebas dalam namanya, misalnya, bidang yang diakhiri dengan '_i' diperlakukan sebagai bidang bilangan bulat, bidang yang diakhiri dengan '_s' diperlakukan sebagai bidang string, bidang yang diakhiri dengan '_f' diperlakukan sebagai bidang oat , dan seterusnya.

Box di bawah ini menunjukkan cara mengindeks file JSON. Koleksi baru bernama produk dibuat dengan file JSON yang diindeks.

```
■  /solr-5.2.1$ bin/post -c products products.json

java -classpath /home/ubuntu/solr-5.2.1/dist/solr-core-5.2.1.jar
-Dauto=yes -Dc=products -Ddata=files org.apache.solr.util.SimplePostTool
products.json SimplePostTool version 5.0.0
Posting files to [base] url http://localhost:8983/solr/products/update..
Entering auto mode.  File endings considered are xml,json,csv,
pdf,doc,docx, ppt,pptx,xls,xlsx,odt,odp,ods,ott,otp,ots,rtf,htm,
html,txt,log POSTing file products.json (application/json) to [base]
1 files indexed.
COMMITting Solr index changes to
http://localhost:8983/solr/products/update...
Time spent:  0:00:00.071
```

Setelah dokumen diindeks, Solr dapat ditanyai baik menggunakan dasbor admin Solr atau melalui klien REST. Gambar 7.18 menunjukkan contoh query Solr dari dashboard admin.




Gambar 7.18: *Querying dari dasbor Apache Solr*

```
■ $ curl "http://localhost:8983/solr/products/select?q=*%3A*&wt=json
&indent=true"
{
"responseHeader":{
   "status":0,
   "QTime":1,
   "params":{
    "indent":"true",
    "q":"*:*",
    "wt":"json"}},
"response":{"numFound":4,"start":0,"docs":[
    {
      "id":"1",
      "name":["HTC Desire 816"],
      "brand":["HTC"],
      "display_t":["5.5 inch S-LCD Display"],
      "memory_s":"8 GB",
      "price_f":150.0,
      "color":["black"],
      "features":["Android 4.4 Kit Kat OS",
       "13 MP Rear Facing BSI Camera",
       "5 MP Front Facing"],
      "_version_":1509468909469696000},
    {
      "id":"2",
      "name":["LG Tribute"],
      "brand":["LG"],
      "display_t":["4.5 inch Capacitive Display"],
      "memory_s":"8 GB",
      "price_f":65.0,
      "color":["black"],
      "features":["Android 4.4 Kit Kat OS",
       "5 MP Rear Facing FSI Camera"],
      "_version_":1509468909473890304},
    {
      "id":"3",
      "name":["Samsung Galaxy S5 White"],
      "brand":["Samsung"],
      "display_t":["5.1inch Full HD Super AMOLED"],
      "memory_s":"16 GB",
      "price_f":499.99,
      "color":["white"],
      "features":["Android 4.4 Kit Kat OS",
       "16 megapixel camera",
       "2.5 GHz quad-core processor"],
      "_version_":1509468909474938880},
    {
      "id":"4",
      "name":["Kyocera Hydro Vibe"],
      "brand":["Kyocera"],
      "display_t":["4.5 inch qHD IPS screen"],
      "memory_s":"8 GB",
      "price_f":49.9,
      "color":["black"],
```

```
      "features":["Andriod 4.3 Jelly Bean OS",
       "8 MP Rear Facing BSI Camera",
       "2 MP Front Facing"],
      "_version_":1509468909475987456}]
}}
```

Box di bawah ini menunjukkan contoh query untuk istilah tertentu.

```
■ $ curl "http://localhost:8983/solr/products/select?q=htc&wt=json
&indent=true"
{
"responseHeader":{
   "status":0,
   "QTime":2,
   "params":{
    "indent":"true",
    "q":"htc",
    "wt":"json"}},
"response":{"numFound":1,"start":0,"docs":[
    {
      "id":"1",
      "name":["HTC Desire 816"],
      "brand":["HTC"],
      "display_t":["5.5 inch S-LCD Display"],
      "memory_s":"8 GB",
      "price_f":150.0,
      "color":["black"],
      "features":["Android 4.4 Kit Kat OS",
       "13 MP Rear Facing BSI Camera",
       "5 MP Front Facing"],
      "_version_":1509468909469696000}]
}}
```

Mari kita lihat contoh Faceting, yang memungkinkan hasil dikelompokkan berdasarkan bidang tertentu atau kriteria yang ditentukan. Box di bawah ini menunjukkan contoh pengaturan hasil ke dalam subset menggunakan bidang harga. Respon tersebut menunjukkan jumlah produk untuk setiap harga.

```
■ $ curl "http://localhost:8983/solr/products/select?q=*:*&wt=json
&indent=on&rows=0&facet=true&facet.field=price_f"
{
"responseHeader":{
   "status":0,
   "QTime":1,
   "params":{
    "facet":"true",
    "indent":"on",
    "q":"*:*",
    "facet.field":"price_f",
    "wt":"json",
```

```
    "rows":"0"}},
"response":{"numFound":4,"start":0,"docs":[]
},
"facet_counts":{
   "facet_queries":{},
   "facet_fields":{
    "price_f":[
      "49.9",1,
      "65.0",1,
      "150.0",1,
      "499.99",1]},
   "facet_dates":{},
   "facet_ranges":{},
   "facet_intervals":{},
   "facet_heatmaps":{}}}
```

Solr juga memungkinkan mempartisi hasil menggunakan berbagai segi. Box di bawah ini menunjukkan contoh segi jangkauan di mana nilai awal jangkauan, nilai akhir dan celah jangkauan ditentukan. Responsnya menunjukkan jumlah produk di setiap rentang.

```
■ $ curl "http://localhost:8983/solr/products/select?q=*:*&wt=json
&indent=on&rows=0&facet=true&facet.range=price_f
&f.price_f.facet.range.start=0.0&f.price_f.facet.range.end=500.0
&f.price_f.facet.range.gap=100"
{
"responseHeader":{
   "status":0,
   "QTime":2,
   "params":{
    "facet":"true",
    "indent":"on",
    "q":"*:*",
    "f.price_f.facet.range.end":"500.0",
    "facet.range":"price_f",
    "wt":"json",
    "f.price_f.facet.range.gap":"100",
    "f.price_f.facet.range.start":"0.0",
    "rows":"0"}},
"response":{"numFound":4,"start":0,"docs":[]
},
"facet_counts":{
   "facet_queries":{},
   "facet_fields":{},
   "facet_dates":{},
   "facet_ranges":{
    "price_f":{
      "counts":[
       "0.0",2,
       "100.0",1,
       "200.0",0,
       "300.0",0,
```

```
        "400.0",1],
      "gap":100.0,
      "start":0.0,
      "end":500.0}},
   "facet_intervals":{},
   "facet_heatmaps":{}}}
```

Box di bawah ini memperlihatkan contoh menambahkan dokumen baru ke indeks menggunakan layanan web Solr.

```
■ curl http://localhost:8983/solr/products/update/json?commit=true -d
'[{"id" :  "5",
"name" :  "Samsung Galaxy S4",
"brand" :  "Samsung",
"display_t" :  "5 inch Super AMOLED",
"memory_s" :  "16 GB",
"price_f" :  329.90,
"color" :  "white",
"features" :  ["Andriod 4.3 Jelly Bean OS","13 MP Rear Facing Camera"]}]'

{"responseHeader":{"status":0,"QTime":4}}
```

Anda dapat mengambil dokumen menggunakan ID seperti yang ditunjukkan pada box di bawah ini.

```
■ curl http://localhost:8983/solr/products/get?id=5
{
"doc":
{
   "id":"5",
   "name":["Samsung Galaxy S4"],
   "brand":["Samsung"],
   "display_t":["5 inch Super AMOLED"],
   "memory_s":"16 GB",
   "price_f":329.9,
   "color":["white"],
   "features":["Andriod 4.3 Jelly Bean OS",
    "13 MP Rear Facing Camera"],
   "_version_":1509469873669931008}}
```

## Ringkasan

Dalam bab ini Anda telah mempelajari cara menggunakan alat dan kerangka kerja untuk pemrosesan batch data termasuk: Hadoop-MapReduce, Pig, Oozie, Spark, dan Solr. Apache Hadoop adalah kerangka kerja sumber terbuka untuk pemrosesan batch terdistribusi dari big data. MapReduce adalah model pemrosesan data paralel untuk

pemrosesan dan analisis data skala besar. Model MapReduce memiliki dua fase: Map dan Reduce. Dalam fase Map, data dibaca dari sistem file terdistribusi, dipartisi di antara sekumpulan node komputasi dalam kluster, dan dikirim ke node sebagai sekumpulan key-value pair . Tugas map memproses catatan masukan secara independen satu sama lain dan menghasilkan hasil antara sebagai key-value pair . Hasil antara disimpan di disk lokal dari node yang menjalankan tugas Map. Ketika semua tugas Map selesai, fase Kurangi dimulai di mana data antara dengan kunci yang sama dikumpulkan. Kami menjelaskan arsitektur Hadoop generasi berikutnya dan manajer cluster YARN. Selanjutnya, kami menjelaskan Scheduller Hadoop FIFO, Adil dan Kapasitas. Kami memperkenalkan Pig, yang merupakan bahasa pemrosesan data tingkat tinggi. Pig memudahkan pengembang untuk menulis skrip analisis data, yang diterjemahkan ke dalam program MapReduce oleh penyusun Pig. Sistem Scheduller aliran kerja Apache Oozie telah dijelaskan. Selanjutnya, kami memperkenalkan kerangka kerja Apache Spark dan menjelaskan berbagai transformasi dan tindakan yang dapat dilakukan dengan Spark. Terakhir, kami menjelaskan kerangka kerja Apache Solr untuk mencari data.

# Analisis Real-Time

Bab 8

Bab ini membahas :

- Analisis Framework Real-time
- Apache Storm
- Apache Spark

Dalam bab ini kami menjelaskan kerangka kerja Apache Storm dan Spark Streaming untuk mengimplementasikan aplikasi analisis data real-time.

## 8.1 Stream Processing

### 8.1.1 Apache Storm

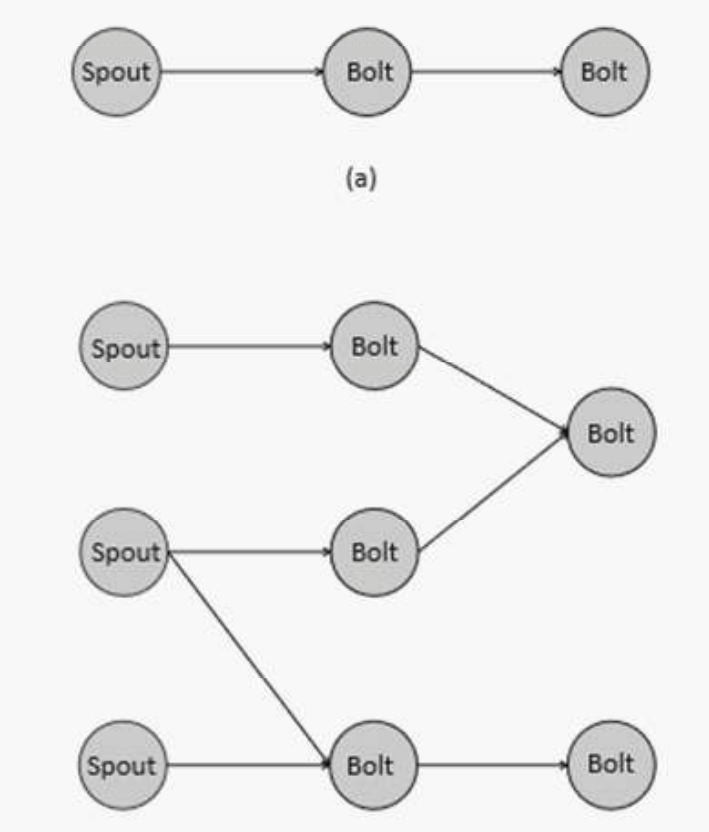
Apache Storm adalah kerangka kerja untuk komputasi real-time terdistribusi dan toleran terhadap kesalahan [65]. Storm dapat digunakan untuk pemrosesan aliran data secara real-time. Storm dapat menyerap data dari berbagai sumber seperti framework perpesanan langganan-publikasi (seperti Kafka atau Kinesis), antrean pesan (seperti RabbitMQ, ZeroMQ, atau SQS) dan konektor kustom lainnya yang dijelaskan di Bab-5. Storm adalah kerangka kerja yang skalabel dan terdistribusi, serta menawarkan pemrosesan pesan yang andal. Storm telah dirancang untuk berjalan tanpa batas dan memproses aliran data secara real-time. Latensi pemrosesan dengan Storm berada dalam urutan milidetik.

#### 8.1.1.1 Konsep

- Topologi: Pekerjaan komputasi pada cluster Storm, yang disebut "topologi", adalah grafik komputasi. Topologi Storm terdiri dari beberapa proses pekerja yang didistribusikan di cluster. Setiap proses pekerja menjalankan subset topologi. Topologi terdiri dari dua jenis node; Cerat dan Baut. Gambar 8.1 menunjukkan beberapa contoh topologi Storm ini. Node dalam topologi dihubungkan oleh tepi yang diarahkan. Setiap node menerima aliran data dari node lain dan memancarkan aliran baru.
- Tuple: Node dalam topologi mengkonsumsi data dalam bentuk tupel. Setiap node menerima tupel data dari node sebelumnya dan memancarkan tupel yang diproses lebih lanjut oleh node downstream. Tupel adalah daftar nilai yang diurutkan. Tuple

dapat berisi tipe data primitif seperti integer, fl oat, doubles, string, boolean, short, long, dll, dan juga tipe kustom (dengan serializer kustom yang disediakan).

- Stream: Stream adalah urutan tupel yang tidak terbatas. Node dalam topologi menerima aliran, memprosesnya, dan memancarkan aliran baru. Aliran output dapat dikonsumsi dan diproses oleh node hilir mana pun di topologi. Dalam topologi yang kompleks, seperti yang ditunjukkan pada Gambar 8.1 (b), node dapat memancarkan atau menelan banyak aliran.
- Spout: Spout adalah jenis node dalam topologi, yang merupakan sumber aliran. Spouts menerima data dari sumber eksternal dan memancarkannya ke topologi sebagai aliran tupel. Cerat tidak memproses tupel; mereka hanya memancarkan tupel yang dikonsumsi oleh baut di topologi




Gambar 8.1: *Contoh tipologi Storm*

- Bolt: Bolt adalah jenis node dalam topologi yang memproses tupel. Baut menerima aliran tupel, memprosesnya, dan memancarkan aliran output. Baut dapat menerima aliran baik dari cerat atau baut lainnya. Baut dapat melakukan berbagai jenis operasi pemrosesan data seperti pemfilteran, agregasi, penggabungan, fungsi kustom, dll. Topologi badai dirancang sedemikian rupa sehingga setiap baut melakukan

transformasi sederhana pada aliran data. Transformasi kompleks dipecah menjadi transformasi yang lebih sederhana, yang dilakukan dengan banyak baut. Karena baut yang berbeda memproses data secara paralel, Storm dapat mencapai latensi rendah untuk pemrosesan data.

- Pekerja: Cerat dan baut memiliki banyak proses pekerja. Setiap proses pekerja itu sendiri memiliki banyak rangkaian eksekusi (disebut tugas). Tugas-tugas ini memproses data secara paralel.

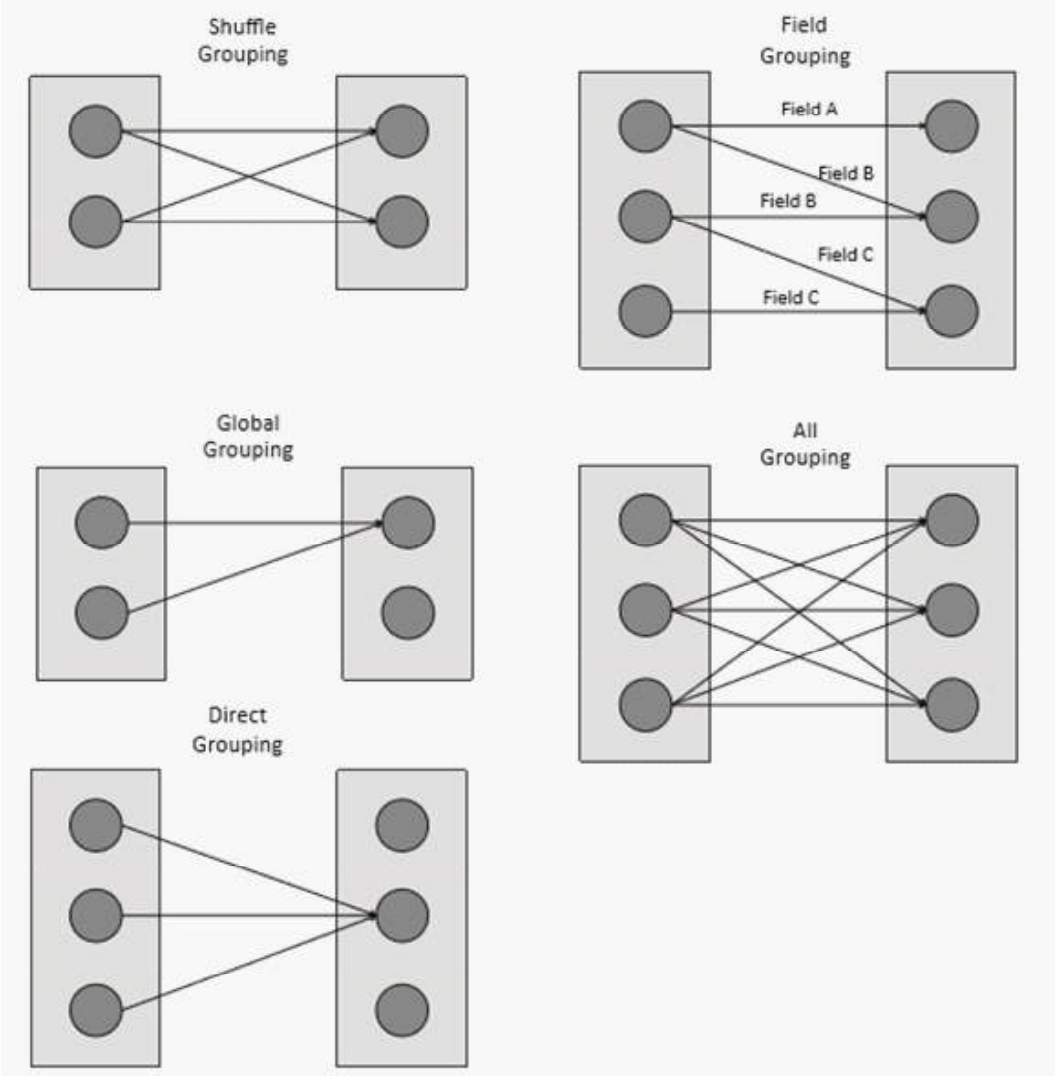
#### 8.1.1.2 Stream Groupings

Karena baut dalam topologi dapat memiliki banyak tugas (rangkaian eksekusi), beberapa mekanisme yang diperlukan untuk kode halus bagaimana aliran harus menjadi bagian yang rusak. Partisi ini didefinisikan dalam istilah pengelompokan aliran. Pengelompokan aliran menentukan bagaimana tupel yang dipancarkan oleh cerat atau baut didistribusikan di antara tugas-tugas baut hilir.

Storm mendukung jenis pengelompokan aliran berikut:

- Shuffle Grouping: Dalam pengelompokan acak, tupel didistribusikan secara acak di seluruh tugas sehingga setiap tugas mendapatkan jumlah tupel yang sama.
- Field Grouping: Dalam pengelompokan field, field pengelompokan ditentukan dimana tupel dalam aliran dikelompokkan. Tupel dengan nilai yang sama dari bidang pengelompokan selalu dikirim ke tugas yang sama.
- All Grouping: Dalam semua pengelompokan, aliran disiarkan ke semua tugas di baut. Jenis pengelompokan ini digunakan di mana aliran akan direplikasi ke semua tugas di baut tujuan.
- Global Grouping: Dalam pengelompokan global, seluruh aliran dikirim ke tugas tertentu dari baut tujuan (tugas dengan ID terendah).
- Direct Grouping: Dalam pengelompokan langsung, simpul pengirim (cerat atau baut) memutuskan tugas mana di baut tujuan yang harus menerima aliran.

Gambar 8.2 menunjukkan berbagai jenis pengelompokan aliran.



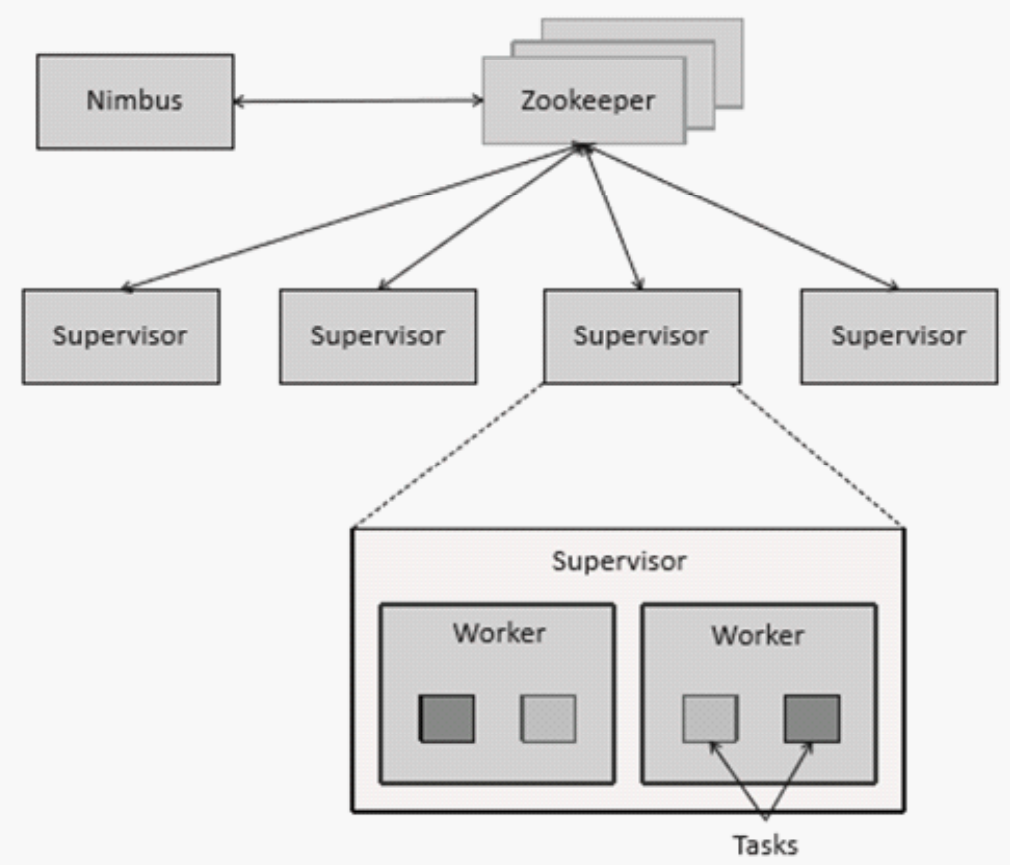

Gambar 8.2: *Stream grouping pada Storm*

### 8.1.1.3 Perancangan

Gambar 8.3 menunjukkan komponen cluster Storm. Cluster Storm terdiri dari komponen Nimbus, Supervisor, dan Zookeeper. Nimbus bertanggung jawab untuk mendistribusikan kode topologi dan tugas di sekitar cluster, meluncurkan pekerja di seluruh cluster, dan memantau pelaksanaan topologi. Cluster Storm memiliki satu atau lebih node Supervisor tempat proses pekerja dijalankan. Nimbus mengirimkan sinyal kepada supervisor untuk memulai atau menghentikan proses.

Node supervisor berkomunikasi dengan Nimbus melalui Zookeeper. Zookeeper adalah layanan koordinasi terdistribusi berkinerja tinggi untuk memelihara informasi konfigurasi, penamaan, menyediakan sinkronisasi terdistribusi dan layanan grup [66]. Zookeeper diperlukan untuk koordinasi cluster Storm. Zookeeper mempertahankan status operasional cluster.

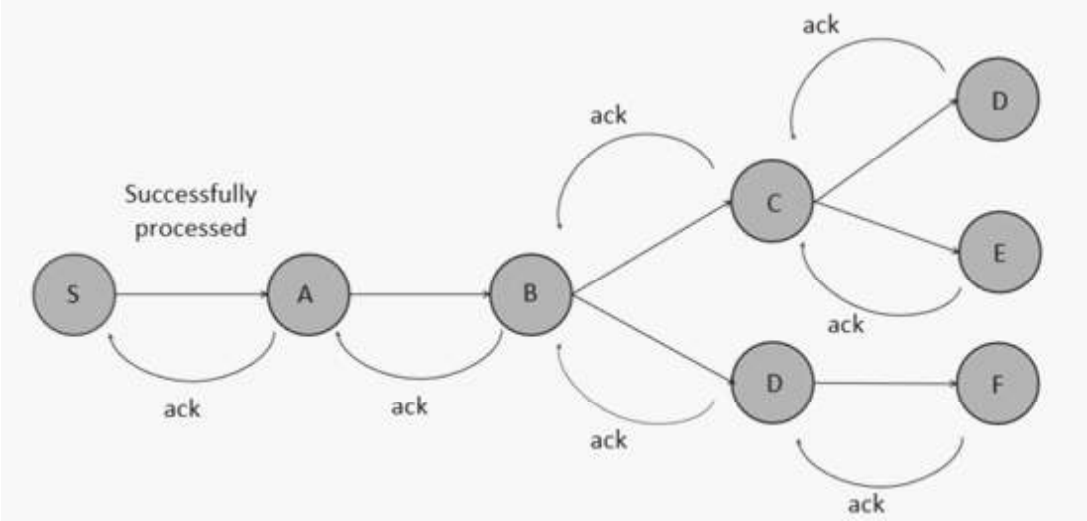
Topologi badai mencakup implementasi cerat dan baut serta definisi topologi. Topologi dikemas sebagai file JAR dan dikirimkan ke node Nimbus untuk dieksekusi. Nimbus mengunggah topologi ke semua supervisor dan memberi sinyal kepada supervisor untuk meluncurkan proses pekerja. Tugas cerat dan baut (utas eksekusi) ditugaskan ke proses pekerja di node supervisor. Topologi dipantau oleh node Nimbus. Jika seorang pekerja pada supervisor gagal, supervisor memulai ulang. Jika seorang supervisor gagal, Nimbus menugaskan kembali tugas tersebut kepada supervisor lainnya. Jika Nimbus mati, proses pekerja tidak terpengaruh karena informasi negara dipertahankan oleh Zookeeper. Daemon Nimbus dan Supervisor dijalankan di bawah pengawasan (menggunakan alat seperti monit, supervisord), sehingga dapat dimulai ulang jika mati.




Gambar 8.3: *Komponen Storm cluster*

#### 8.1.1.4 Reliable Processing

Storm menyediakan pemrosesan tupel yang andal. Storm menjamin bahwa setiap tupel yang dipancarkan oleh cerat diproses. Dalam topologi, sebuah tupel yang dipancarkan oleh cerat diproses oleh baut-baut tersebut sehingga menghasilkan beberapa tupel yang didasarkan pada tupel asli. Ini menghasilkan pohon tupel seperti yang ditunjukkan pada Gambar 8.4. Baut dalam topologi mengakui pemrosesan tupel ke baut atau cerat hulu. Jika semua baut dalam pohon tupel menyatakan bahwa tupel telah berhasil diproses, cerat menandai pemrosesan tupel yang akan diselesaikan, melakukan pembersihan, dan mengirimkan pemberitahuan ke sumber data eksternal. Jika ada baut di pohon tupel yang menunjukkan bahwa pemrosesan tupel gagal (atau habis waktunya), spout menandai pemrosesan tupel sebagai gagal. Ketika pemrosesan tupel gagal, cerat memancarkan kembali tupel.




Gambar 8.4 *: Pohon Tuple*

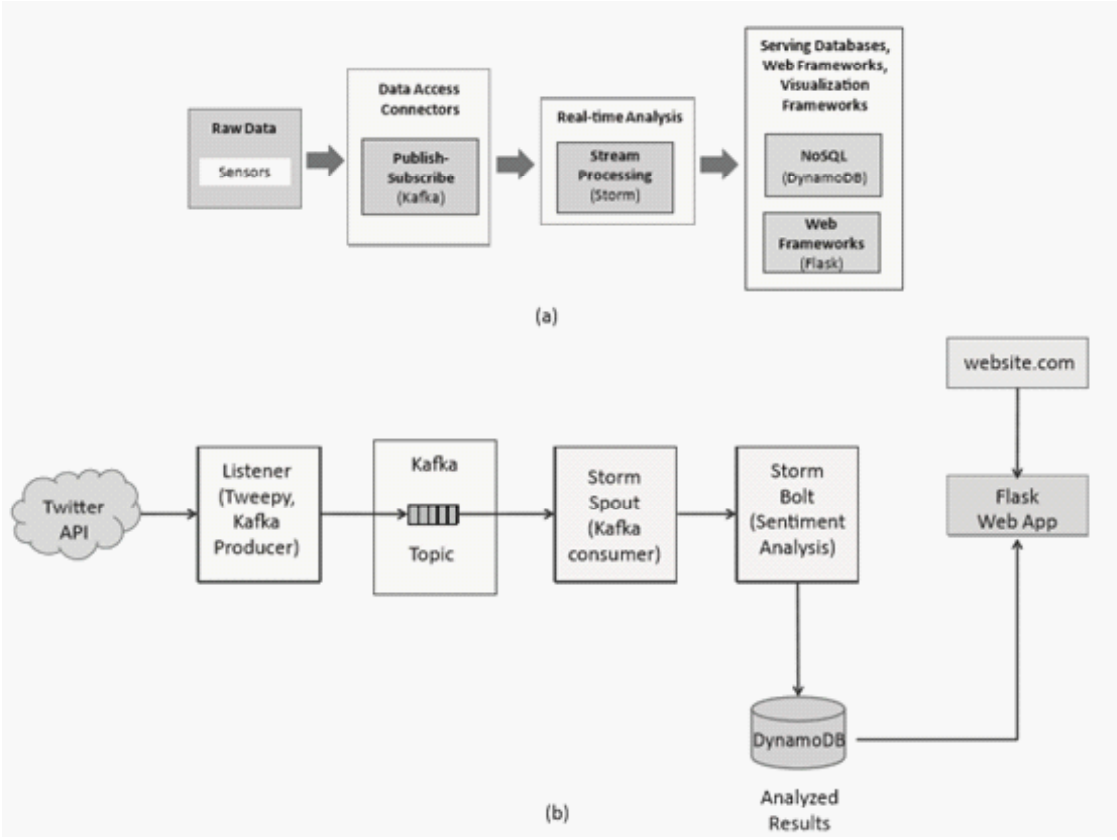
## 8.2 Studi Kasus Storm

### 8.2.1 Analisis Sentimen Twitter Real-time

Di bagian ini, kami akan menjelaskan studi kasus pada sistem untuk analisis sentimen real-time feed Twitter. Mengingat persyaratan analisis sistem ini, mari kita mapkan sistem ke salah satu pola analisis yang diusulkan di Bab-1. Karena sistem memproses umpan media sosial secara real-time, kita dapat menggunakan beta pattern. Gambar8.5 (a) menunjukkanarealisasi pola Bet untuk sistem ini, dengan alat dan kerangka kerja khusus yang dapat digunakan. Untuk sistem ini, kami akan menggunakan kerangka kerja Apache

Storm dan Apache Kafka. Gambar 8.5 (b) menunjukkan komponen-komponen sistem analisis sentimen real-time termasuk:

- Listener: Komponen listener terhubung ke Twitter dengan API streaming dan mengambil tweet secara real-time. Pustaka Python yang disebut tweepy digunakan untuk mengambil tweet yang berisi kata kunci tertentu. Listener menerbitkan tweet ke topik Kafka yang dikelola oleh Kafka Broker.
- Storm Spout: Storm Spout berisi seorang konsumen Kafka yang mengambil tweet dari topik Kafka dan mengeluarkan tupel (berisi tweet) untuk diproses oleh Storm Bolt.
- Storm Bolt: Storm Bolt menganalisis tweet dan menghitung sentimen mereka menggunakan leksikon sentimen.
- DynamoDB: Tweet dengan stempel waktu dan sentimennya disimpan oleh Storm Bolt ke tabel Amazon DynamoDB.
- Flask Web App: FlaskWebApp mengambil tweet dan sentimennya dari DynamoDB dan menampilkannya.
- Mari kita lihat langkah-langkah yang terlibat dalam membangun sistem analisis sentimen real-time bersama dengan implementasi berbagai komponen.

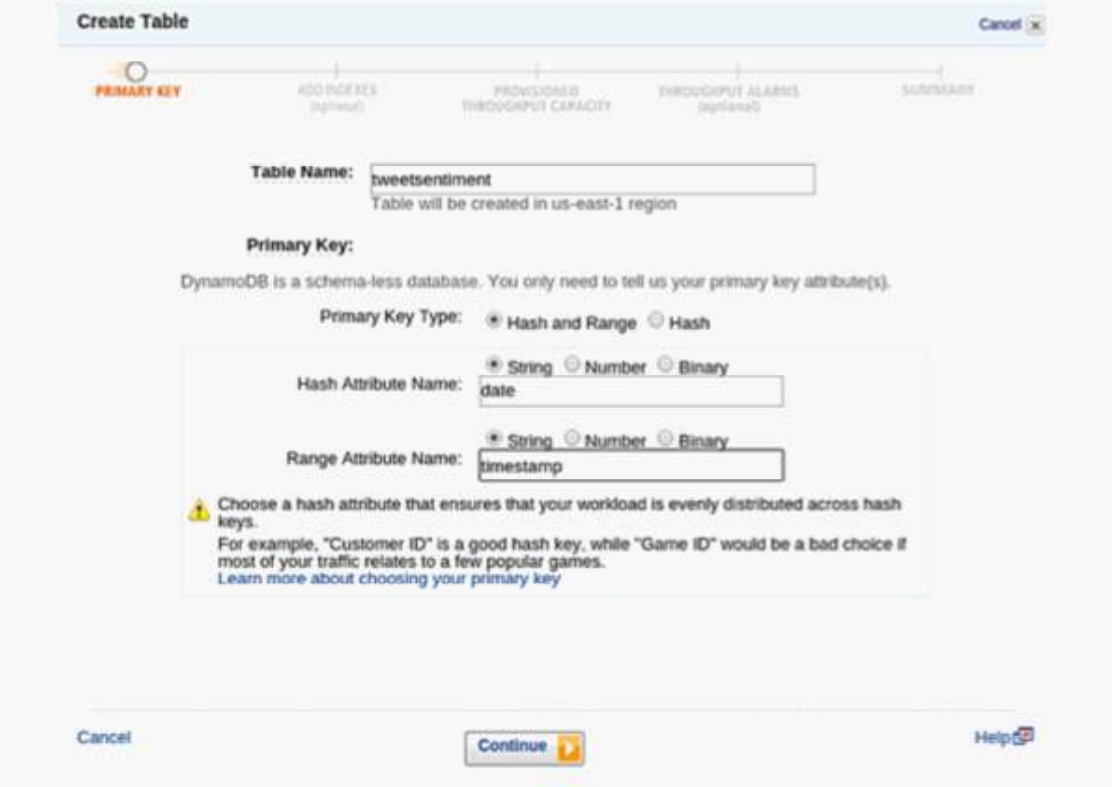



Gambar 8.5: *(a) Realisasi beta pattern untuk analisis sentimen real-time dari umpan Twitter (b) Arsitektur Sistem*

### *Membuat Kafka Topic*

Komponen listener menerbitkan tweet ke sebuah topik Kafka. Untuk membuat topik Kafka, ikuti perintah di bawah ini:

```
■ #Creating a Kafka Topic
cd /usr/hdp/2.2.0.0-2041/kafka
bin/kafka-topics.sh -create -zookeeper
   ip-10-179-181-24.ec2.internal:2181 -replication-factor 1
   -partitions 1 -topic mytopic

bin/kafka-topics.sh -list -zookeeper ip-10-179-181-24.ec2.internal:2181

(Change the DNS in the above commands to the
DNS of the instance on which Kafka is setup)
```




Gambar 8.6 *: Membuat tabel DynamoDB untuk menyimpan tweet dan sentimennya*

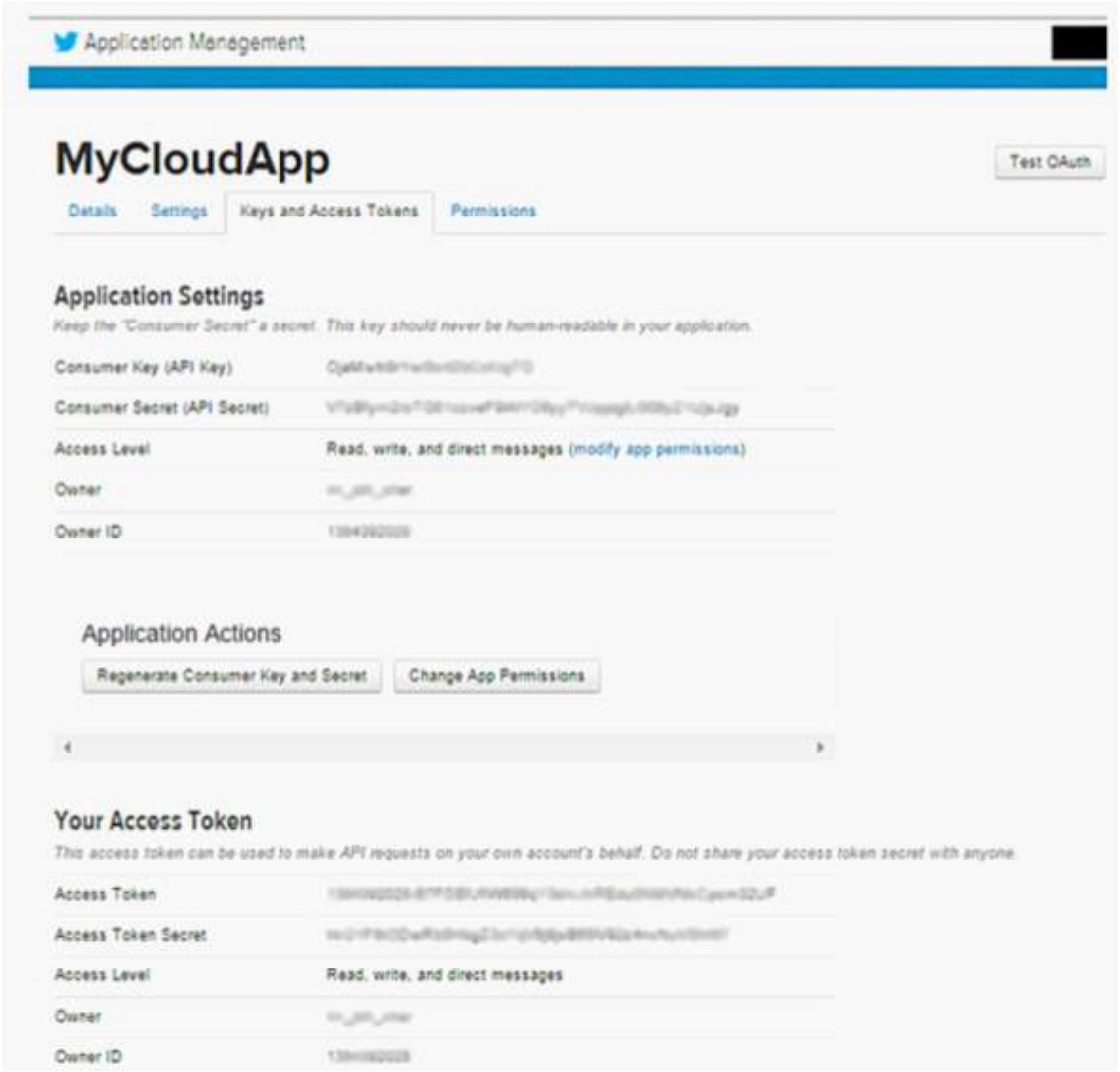
### *Membuat Tabel DynamoDB*

Dalam studi kasus ini, kami akan menggunakan database Amazon DynamoDB untuk menyimpan hasil yang dianalisis. Untuk menggunakan DynamoDB kita perlu membuat tabel baru seperti yang ditunjukkan pada Gambar 8.6.

### Mendapatkan Tombol Aplikasi Twitter

Untuk terhubung ke Twitter streaming API, listener memerlukan kunci aplikasi Twitter. Anda dapat mendaftar untuk akun pengembang Twitter dan membuat aplikasi baru di: https://dev.twitter.com/apps

Catat token akses aplikasi, rahasia token akses, kunci konsumen dan rahasia konsumen dari halaman detail aplikasi seperti yang ditunjukkan pada Gambar 8.7.




Gambar 8.7 *: Mendapatkan kredensial aplikasi Twitter dari akun pengembang Twitter*

### *Implement Listener*

Komponen listener terhubung ke Twitter dengan API streaming, mengambil tweet secara real-time dan menerbitkan tweet ke topik Kafka. Box 8.1 menunjukkan implementasi Python dari komponen listener.

**■ Box 8.1: Python implementation of listener that receives tweets**

```
from tweepy.streaming import StreamListener
from tweepy import OAuthHandler
from tweepy import Stream
```

```
import time
import datetime
from kafka.client import KafkaClient
from kafka.producer import SimpleProducer

#Connect to Kafka
client = KafkaClient("ip-172-31-60-0.ec2.internal:6667")
producer = SimpleProducer(client)


#Get Twitter API keys from Twitter developer account
access_token = "<Enter>"
access_token_secret = "<Enter>"
consumer_key = "<Enter>"
consumer_secret = "<Enter>"


def publish(data):
   producer.send_messages('mytopic', data.encode())


class StdOutListener(StreamListener):

   def on_data(self, data):
      publish(data)

   def on_error(self, status):
      print status

if __name__ == '__main__':
   print 'Listening...'
   #This handles Twitter authetification and
   #the connection to Twitter Streaming API
   #sets callback on data
   l = StdOutListener()
   auth = OAuthHandler(consumer_key, consumer_secret)
   auth.set_access_token(access_token, access_token_secret)
   stream = Stream(auth, l)

   #This line filter Twitter Streams to capture data by the keywords
   stream.filter(track=['basketball'])
```

### *Implement Storm Spout dan Bolt*

Box 8.2 menunjukkan implementasi Storm Spout. Spout mengambil tweet dari topik Kafka dan memancarkannya sebagai tupel ke dalam topologi.

**Box 8.2: Python implementation of Spout**

```
from storm import Spout, emit, log
from kafka.client import KafkaClient
from kafka.consumer import KafkaConsumer
from kafka.producer import SimpleProducer

client = KafkaClient("ip-172-31-60-0.ec2.internal:6667")
consumer = KafkaConsumer("tweetsent",
   metadata_broker_list=['ip-172-31-60-0.ec2.internal:6667'])

#gets the tweet
def getData():
data = consumer.next().value
return data

class MySpout(Spout):
   def nextTuple(self):
      data = getData()
      emit([data])


MySpout().run()
```

Box 8.3 menunjukkan implementasi Storm Bolt dari Python, yang menghitung sentimen tweet dan menyimpan hasilnya dalam tabel DynamoDB.

**Box 8.3: Python implementation of Bolt**

```
import storm
from datetime import date
import time
import datetime
import boto.dynamodb2
from boto.dynamodb2.table import Table
import json
import re

ACCESS_KEY="<Enter>"
SECRET_KEY="<Enter>"
REGION="us-east-1"


#Connect to DynamoDB
conn = boto.dynamodb2.connect_to_region(REGION,
```

```
        aws_access_key_id=ACCESS_KEY,
        aws_secret_access_key=SECRET_KEY)

TERMS={}
#----- Load Sentiments Dict ---
sent_file = open('AFINN-111.txt')
sent_lines = sent_file.readlines()
for line in sent_lines:
   s = line.split("")
   TERMS[s[0]] = s[1]

sent_file.close()



#----- Find Sentiment -------
```

Meskipun Storm Spouts and Bolts dapat diimplementasikan dengan Python (dan bahasa lain), tidak ada cara langsung untuk mengimplementasikan topologi Storm dengan Python. Box 8.4 menunjukkan implementasi Java dari topologi Storm.

**■ Box 8.4: Storm topology implementation**

```
package com.mycompany.app;
import backtype.storm.Config;
import backtype.storm.LocalCluster;
import backtype.storm.StormSubmitter;
import backtype.storm.task.ShellBolt;
import backtype.storm.spout.ShellSpout;
import backtype.storm.topology.IRichBolt;
import backtype.storm.topology.IRichSpout;
import backtype.storm.topology.OutputFieldsDeclarer;
import backtype.storm.topology.TopologyBuilder;
import backtype.storm.tuple.Fields;
import java.util.Map;


public class App {

public static class SensorSpout extends ShellSpout implements IRichSpout
{
   public SensorSpout() {
      super("python", "myspout.py");
   }

   @Override
   public void declareOutputFields(OutputFieldsDeclarer declarer) {
      declarer.declare(new Fields("word"));
   }
```

```
    @Override
    public Map<String, Object> getComponentConfiguration() {
        return null;
    }
}

public static class SensorBolt extends ShellBolt implements IRichBolt {
    public SensorBolt() {
        super("python", "mybolt.py");
    }

    @Override
    public void declareOutputFields(OutputFieldsDeclarer declarer) {
        declarer.declare(new Fields("word"));
    }

    @Override
    public Map<String, Object> getComponentConfiguration() {
        return null;
    }
}


public static void main(String[] args) throws Exception {

    TopologyBuilder builder = new TopologyBuilder();

    builder.setSpout("spout", new SensorSpout(), 5);
    builder.setBolt("analysis",
          new SensorBolt(), 8).shuffleGrouping("spout");

    Config conf = new Config();
    conf.setDebug(true);
```

### *Bangun Proyek Badai*

Proyek badai dikemas sebagai file JAR. Box di bawah ini menunjukkan pohon direktori dari proyek Storm. File Spout dan Bolt Python disertakan dalam folder multilang / resources, dan implementasi topologi ada di file App.java.

```
■ #Storm Project Directory Tree
/home/ubuntu/storm-project
|- multilang
|   `- resources
|      |- mybolt.py
|      |- myspout.py
|      `- storm.py
|- pom.xml
|- src
|   |- main
|   |   `- java
```

```
|   |     `- com
|   |        `- mycompany
|   |           `- app
|   |              |- App.java
`- target
   `- storm-project-jar-with-dependencies.jar
```

```
■ #Build Storm Project
cd /home/ubuntu/storm-project
mvn clean package
```

File JAR project Storm (storm-project-jar-with-dependencies.jar) akan dibuat dalam folder storm-project / target.

### *Menerapkan Aplikasi Web*

Terakhir, kami menerapkan aplikasi web untuk menyajikan hasil analisis sentimen. Box 8.5 menunjukkan implementasi Python dari aplikasi web menggunakan kerangka kerja web Python Flask

**■ Box 8.5: Flask web application for visualizing tweet sentiments**

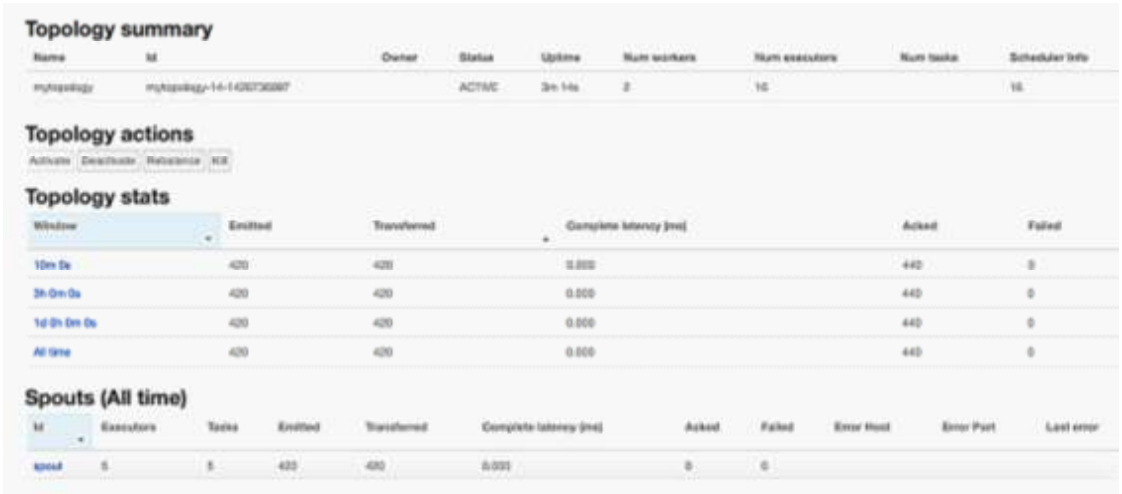
```
from flask import Flask
import urllib2
import boto.dynamodb2
from boto.dynamodb2.table import Table

app = Flask(__name__)
from datetime import date
today = date.today()

#-----Connect to DynamoDB------
ACCESS_KEY="<Enter>"
SECRET_KEY="<Enter>"
REGION="us-east-1"


conn_db = boto.dynamodb.connect_to_region(REGION,
aws_access_key_id=ACCESS_KEY,
aws_secret_access_key=SECRET_KEY)

table = conn_db.get_table('twittersentiment')
#----------------------

@app.route('/')
def tweet_home():
```

```
    #Scan DynamoDB table
    results=table.scan()

    html = '<html><body><table width=80% border=1 align="center">'+
          '<tr><td><strong>Timestamp</strong></td>
          <td><strong>Date</strong></td>
          <td><strong>Tweet</strong></td>
          <td><strong>Sentiment</strong></td>
          </tr>'


    for result in results:
       html+='<tr><td>'+result['timestamp']+'</td><td>'+
       result['date']+'</td><td>'+result['tweet']+'</td><td>'+
       result['sentiment']+'</td></tr>'

    html+='</table></body></html>'

    return html

if __name__ == '__main__':
    app.run(host='0.0.0.0')
```

### *Sumit Topologi Storm*

Untuk mengirim dan menjalankan topologi Storm, jalankan perintah berikut di cluster Storm:

```
■ #Submit Storm Topology
storm jar
/home/ubuntu/storm-project/target/storm-project-jar-with-dependencies.jar
com.mycompany.app.App mytopology
```

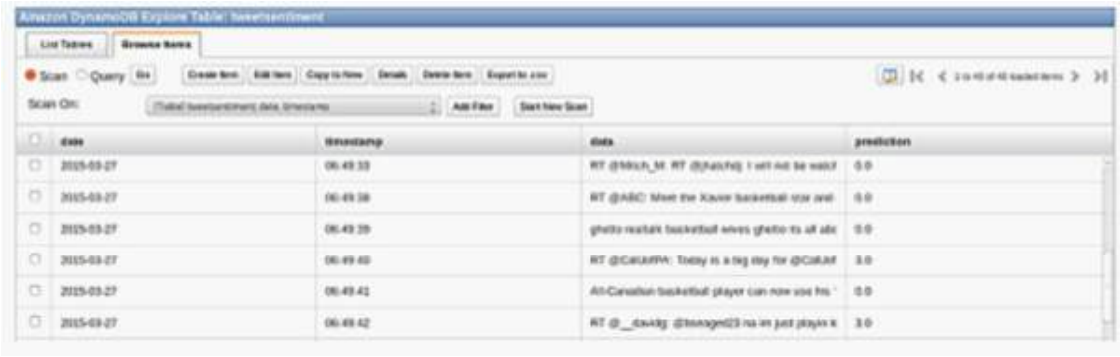



Gambar 8.8: *Screenshot halaman Tipologi Storm*


Gambar 8.9: *Screenshot halaman Storm Spout*

Setelah mengirimkan topologi, jalankan listener dan aplikasi web Flask. Anda dapat melihat status topologi Storm, Spout dan Bolt dari halaman Storm UI seperti yang ditunjukkan pada Gambar 8.8, 8.9 dan 8.10. Anda akan dapat melihat tweet bersama dengan sentimen yang dihitung di DynamoDBdashboardasshowninFigures8.11dan aplikasi web seperti yang ditunjukkan pada Gambar 8.12.


Gambar 8.10: *Screenshot halaman Storm Bolt*



Gambar 8.11: *tabel DynamoDB yang menunjukkan analisis tweets*

| Timestamp | Date | Data | Prediction |
|---|---|---|---|
| 06:49:16 | 2015-03-27 | Under Armour Men's Micro G Torch Basketball Shoes in Blue in Sizes 6.5 to 14 http://t.co/H3lByAulXV http://t.co/GBo28vKuKH | 0.0 |
| 06:49:17 | 2015-03-27 | Under Armour Men's Micro G Torch Basketball Shoes in Blue in Sizes 6.5 to 14 http://t.co/H3lByAulXV http://t.co/XYmt9JXcFJ | 0.0 |
| 06:49:18 | 2015-03-27 | Under Armour Men's Micro G Torch Basketball Shoes in Blue in Sizes 6.5 to 14 http://t.co/H3lByAulXV http://t.co/FUW9iYRnhp | 0.0 |
| 06:49:19 | 2015-03-27 | Under Armour Men's Micro G Torch Basketball Shoes in Blue in Sizes 6.5 to 14 http://t.co/H3lByAulXV http://t.co/FUW9iYRnhp | 0.0 |
| 06:49:20 | 2015-03-27 | Great Range of Baseball and Basketball Shirts at: http://t.co/TYLIXEvrrb | 0.0 |
| 06:49:23 | 2015-03-27 | 'Happy' dancing kid steals the spotlight at high school basketball tournament - http://t.co/PLSijc50t6 http://t.co/A2EQ2K5FKW | -2.0 |
| 06:49:27 | 2015-03-27 | "@iamannaceline: Pag nandito si Bugoy Cariño sa lugar namin. normal lang sya, nag lalaro pa nga ng basketball kasama mga tambay eh hahahaha" | 0.0 |
| 06:49:28 | 2015-03-27 | "@CardinMorgane: Dans l'indifférence générale @EsbvaLm remporte L'eurocup #BasketBall bravo à elles" @bbarbusse @mc_naves @generalifrance | 0.0 |
| 06:49:29 | 2015-03-27 | RT @FoxNews: Late UNC basketball coach Smith left former lettermen $200 each http://t.co/aOSYTajOJ7 http://t.co/bEzdygMOpD | 0.0 |

Gambar 8.12: *Screenshot dari aplikasi web Flask yang menampilkan tweet dan sentimen yang dihitung*

### 8.2.2 Analisis Data Cuaca Real-time

Mari kita lihat studi kasus lain pada sistem untuk analisis data cuaca secara real-time untuk membuat prediksi tentang terjadinya kabut atau kabut di lokasi tertentu. Sistem menentukan apakah kabut atau kabut diharapkan berdasarkan berbagai parameter termasuk suhu saat ini, kelembaban, angin, dan tekanan. Dataset yang digunakan dalam studi kasus ini diperoleh dari National Oceanic and Atmospheric Administration (NOAA). Dataset NOAA digunakan untuk melatih kelas pembelajaran mesin berbasis Support Vector Machine (SVM). Untuk menguji sistem, data historis mengenai suhu, kelembaban, angin, dan tekanan untuk lokasi tertentu diumpankan ke sistem (sebagai data sintetik real-time).

Mengingat persyaratan analisis sistem ini, mari kita mapkan sistem ke salah satu pola analisis yang diusulkan di Bab-1. Karena sistem memproses data cuaca secara real-time, kita dapat menggunakan beta pattern. Gambar 8.13 (a) menunjukkan realisasi beta pattern untuk sistem ini, dengan alat dan kerangka kerja khusus yang dapat digunakan. Untuk sistem ini, kami akan menggunakan kerangka kerja Apache Storm dan Apache Kafka. Gambar 8.13 (b) menunjukkan arsitektur sistem. Sistem ini mencakup komponen-komponen berikut:

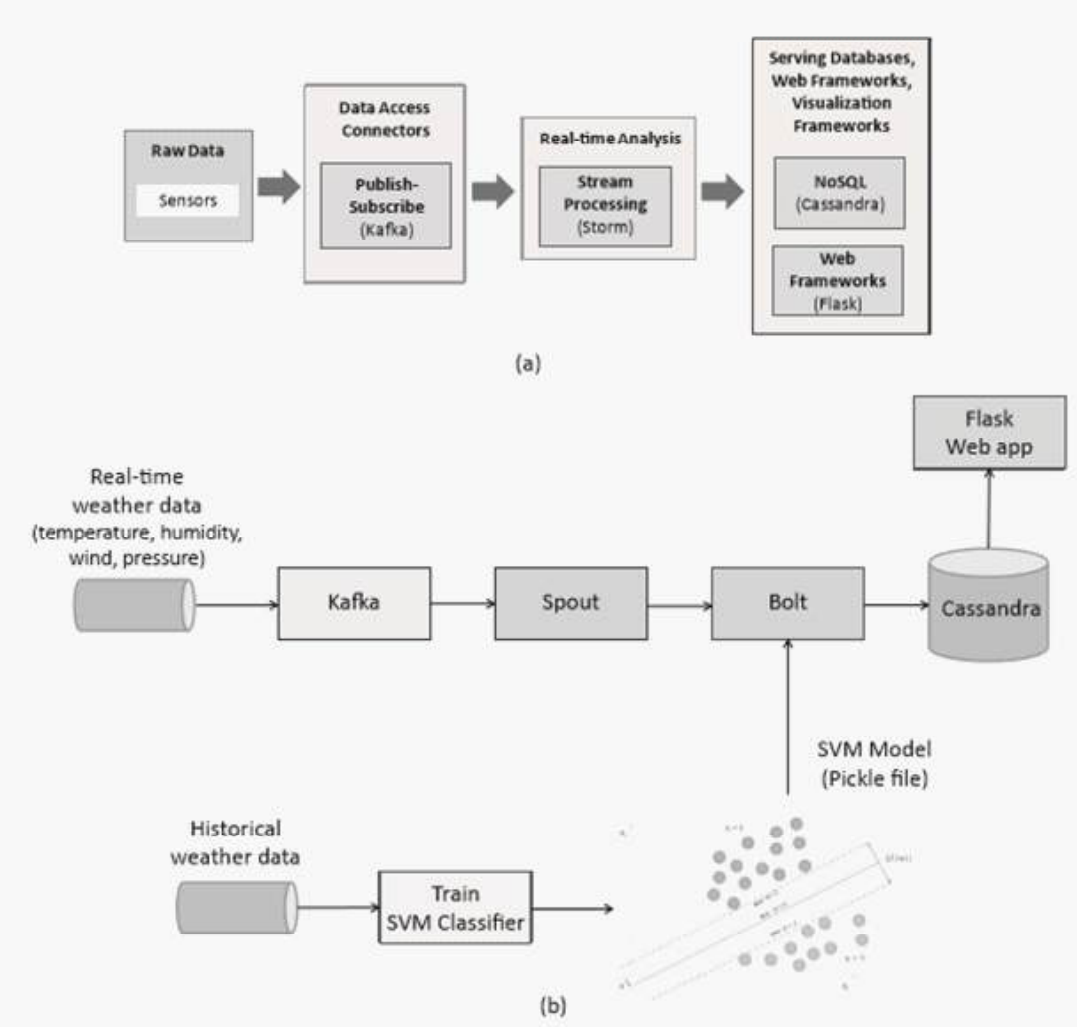
- Listener: Komponen listener terhubung ke sumber data cuaca dan mendapatkan data suhu, kelembaban, angin, dan tekanan saat ini.
- Kafka: Listener mempublikasikan data ke topik Kafka.
- Storm Spout: Storm Spout berlangganan topik Kafka dan menerima semua pesan yang diterbitkan dari listener. Cerat menerima data ini dan memancarkan tupel yang berisi pengamatan setiap jam.
- Storm Bolt: Storm Bolt menerima data dan memuat model SVM dari file acar. Model SVM digunakan untuk memprediksi kabut dan kabut asap.
- Cassandra Database: Prediksi bersama dengan timestamp disimpan dalam database Cassandra.
- Flask Web App: Flask Web App digunakan untuk memvisualisasikan hasil.

***Dataset***

Data yang digunakan untuk pengujian dan pelatihan kelas pembelajaran mesin serta data real-time berasal dari Data Klimatologi Lokal AS NOAA. Dataset khusus yang digunakan adalah Quality Controlled Local Climatological Data (QCLCD).

***Listener***

Box 8.6 menunjukkan implementasi python dari komponen listener. Listener terhubung ke sumber data cuaca. Untuk studi kasus ini, listener menggunakan data dari NOAA. Listener menggunakan data dari bulan yang sama tetapi untuk tahun yang berbeda dari data yang digunakan untuk melatih kelas. Listener memutar ulang data dari tiga file CSV: PositiveFog.csv, PositiveHaze.csv, dan NegativeFogHaze.csv. Listener menerbitkan entri data baru ke Kafka setiap 5 detik untuk mensimulasikan penerimaan data dari sensor nyata secara berkala.




Gambar 8.13: *(a) Realisasi beta pattern untuk analisis real-time data cuaca (oleh) Arsitektur sistem*

**Box 8.6: Python implementation of Listener**

```
from random import randrange
import time
import datetime
from kafka.client import KafkaClient
from kafka.consumer import SimpleConsumer
from kafka.producer import SimpleProducer
import csv
import json
import time

#Connect to Kafka
#change to private DNS
client = KafkaClient("127.0.0.1:6667")
```

```
producer = SimpleProducer(client)

def is_number(s):
   try:
      float(s)    except ValueError:
      return False
   return True

def sysTest(filename):
   f = open(filename)
   csv_f = csv.reader(f)
   tList=[]
   tClass=[]
   counter=0
   for row in csv_f:
      new_list=[]
      if counter < 2:
         counter=counter+1
         continue
      for item in row:
         if not(is_number(item)):
            continue
         new_list.append(float(item))
      #print new_list
      if len(new_list)!=5:
         continue
      dataSend=json.dumps(new_list).encode('utf-8')
      producer.send_messages('weather',dataSend)
      time.sleep(5)
      print dataSend
      print type(dataSend)
   f.close()


if __name__ == '__main__':
   print 'Publishing...'
   sysTest('PositiveFog.csv')
   sysTest('PositiveHaze.csv')
   sysTest('NegativeFogHaze.csv')
```

***Prediksi Model***

Dataset NOAA digunakan untuk melatih dua kelas pembelajaran mesin berbasis SVM, satu untuk kabut dan satu untuk kabut. SVM dilatih tentang data per jam positif (kabut atau kabut memang terjadi) dan data per jam negatif. Data pelatihan disusun dalam tiga file CSV: FogTrain.csv, HazeTrain.csv, danNegativeTrain.csv. FogTrain.csv memiliki data positif untuk observasi per jam offog, HazeTrain.csvhaspositif untuk observasi kabut asap per jam, dan NegativeTrain.csv memiliki data negatif untuk observasi kabut dan kabut per jam. Setelah kedua SVM dilatih, keduanya disimpan dalam file acar: svmHaze.pkl dan svmFog.pkl. File ini digunakan dalam Storm Bolt untuk membuat prediksi. Box 8.7

menunjukkan kode python untuk melatih klasifikasi SVM untuk kabut dan kabut asap. Kami menggunakan pustaka pembelajaran mesin scikit-learn Python untuk membangun model SVM.

**Box 8.7: Python code for training SVM Classifier**

```
from sklearn import svm
import csv
import numpy as np
from sklearn.externals import joblib

def is_number(s):
   try:
      float(s) # for int, long and float
   except ValueError:
      return False

   return True


def svmTrain():
   f = open('PositiveHaze.csv')
   csv_f = csv.reader(f)
   tList=[]
   tClass=[]
   counter=0
   for row in csv_f:
      new_list=[]
      if counter < 2:
         counter=counter+1
         continue
      for item in row:
         if not(is_number(item)):
            continue
         new_list.append(float(item))
      print new_list
      tList.append(new_list)
      tClass.append(1)

   f.close()
```

```
   f=open('NegativeHazeAndNegativeFog.csv')
   csv_f=csv.reader(f)
   counter=0
   for row in csv_f:
      new_list=[]
      if counter < 2:
         counter=counter+1
         continue
      for item in row:
         if not(is_number(item)):
            continue
         new_list.append(float(item))
```

```
    print new_list
    if len(new_list)!=5:
       print "ERROR"
       continue
    tList.append(new_list)
    tClass.append(0)

f.close();
f=open('PositiveFog.csv')
csv_f= csv.reader(f)
tFogList=[]
tFogClass=[]
counter=0

for row in csv_f:
    new_list=[]
    if counter < 2:
       counter=counter+1
       continue
    for item in row:
       if not(is_number(item)):
          continue
       new_list.append(float(item))
    print new_list
    if len(new_list)!=5:
       continue
       exit(2)
    tFogClass.append(1)
    tFogList.append(new_list)

f.close()
f=open('NegativeHazeAndNegativeFog.csv')
csv_f=csv.reader(f)
counter=0
for row in csv_f:
    new_list=[]
    if counter < 2:
       counter=counter+1
       continue
    for item in row:
       if not(is_number(item)):
          continue
       new_list.append(float(item))
    print new_list
    if len(new_list)!=5:
       continue
    tFogClass.append(0)
    tFogList.append(new_list)

svmThunder = svm.SVC()

svmThunder.fit(tList,tClass)

svmFog = svm.SVC()
```

```
    svmFog = svm.SVC()

    svmFog.fit(tFogList,tFogClass)

    joblib.dump(svmThunder, 'svmHaze.pkl')
    joblib.dump(svmFog, 'svmFog.pkl' )


if __name__ == '__main__':
    svmTrain()
```

### *Storm Spout*

Storm Spout berlangganan topik Kafka dan menerima pesan yang diterbitkan oleh listener. Spout memancarkan data ini sebagai aliran tupel ke dalam topologi Storm untuk diproses oleh Bolt. Box 8.8 menunjukkan implementasi Storm Spout dari Python.

**▪ Box 8.8: Python implementation of Storm Spout**

```
from storm import Spout, emit, log
from kafka.client import KafkaClient
from kafka.consumer import KafkaConsumer
from kafka.producer import SimpleProducer
from kafka.consumer import SimpleConsumer

client = KafkaClient("127.0.0.1:6667")
consumer = KafkaConsumer("weather",
    metadata_broker_list=['127.0.0.1:6667'])

def getData():
    data = consumer.next().value
    data = data.replace('[',").replace(']',").split(',')
    data = [ float(x) for x in data ]
    return data

class SensorSpout(Spout):
    def nextTuple(self):
        data = getData()
        emit([data])

SensorSpout().run()
```

```
            new_list.append(float(item))
        print new_list
        if len(new_list)!=5:
            continue
        tFogClass.append(0)
        tFogList.append(new_list)

    svmThunder = svm.SVC()

    svmThunder.fit(tList,tClass)
```

```
    svmFog = svm.SVC()

    svmFog.fit(tFogList,tFogClass)

    joblib.dump(svmThunder, 'svmHaze.pkl')
    joblib.dump(svmFog, 'svmFog.pkl' )


if __name__ == '__main__':
    svmTrain()
```

### *Storm Bolt*

Storm Bolt menerima data yang dipancarkan oleh Spout dan memuat klasifikasi SVM yang terlatih untuk mencari kabut dan kabut dari file acar. Data masukan diteruskan ke klasifikasi ini untuk memprediksi kabut dan kabut. Untuk menyimpan prediksi bersama dengan cap waktu, database Cassandra digunakan. Cassandra adalah sistem manajemen database terdistribusi open source yang dirancang untuk aplikasi big data. Box 8.9 menunjukkan implementasi Python dari baut Storm.

**■ Box 8.9: Python implementation of Bolt**

```
import storm
import json
import re
from cassandra.cluster import Cluster
import blist
from sklearn import svm
from sklearn.externals import joblib
import datetime
#------- Load Clf files -------
svmFog = joblib.load('svmFog.pkl')
svmHaze = joblib.load('svmHaze.pkl')

#----- Connect to Cassandra -----
cluster = Cluster(['127.0.0.1'])
session = cluster.connect('predictions')

def analyzeData(data):
    fog_predict = svmFog.predict(data)
    haze_predict = svmHaze.predict(data)
    return [str(fog_predict[0]), str(haze_predict[0])]

class SensorBolt(storm.BasicBolt):
    def process(self, tup):
        data = tup.values[0]

        output = analyzeData(data)
        result = "Result:  "+ str(output)
        timestamp= datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
```

```
        #Store analyzed results in Cassandra
        session.execute(
        """
        INSERT INTO data(timestamp, fog_prediction, haze_prediction)
        VALUES (%s, %s, %s)
        """,
        (timestamp, output[0], output[1])
        )

        storm.emit([result])

SensorBolt().run()
```

***Web Aplication***

Box 8.10 menunjukkan kode Python untuk aplikasi web Flask yang mengambil data dari database Cassandra dan menampilkannya dalam tabel.

**■ Box 8.10: Flask web application code**

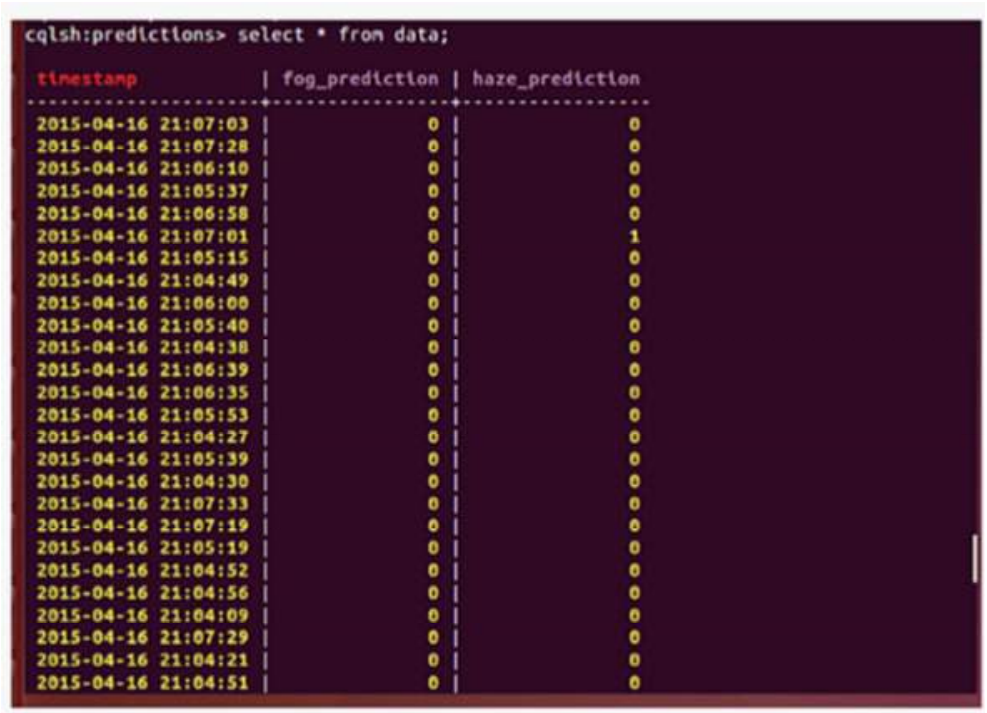
```
from flask import Flask
import urllib2
app = Flask(__name__)
import blist

from cassandra.cluster import Cluster

cluster = Cluster(['127.0.0.1'])
session = cluster.connect('predictions')

@app.route('/')
def tweet_home():


   html = '<html><body><table width=80% border=1 align="center">'+
         '<tr><td><strong>Timestamp</strong></td>'+
         '<td><strong>Fog Prediction</strong></td>'+
         '<td><strong>Haze Prediction</strong></td></tr>'

   data = session.execute('SELECT * FROM data')
   for each in data:
      html+='<td>'+each.timestamp+'</td><td>'+
      each.fog_prediction+'</td><td>'+
      each.haze_prediction+'</td></tr>'

   html+='</table></body></html>'

   return html

if __name__ == '__main__':
   app.run(host='0.0.0.0')
```

Untuk menjalankan sistem, jalankan listener dan aplikasi web program Python dan kirimkan topologi Storm. Anda akan dapat melihat prediksi kabut / kabut dari cangkang Cassandra seperti yang ditunjukkan pada Gambar 8.14 dan aplikasi web seperti yang ditunjukkan pada Gambar 8.15.

## 8.3 Pemrosesan Dalam Memori

### 8.3.1 Apache Spark

Kami menjelaskan arsitektur Spark dan operasi Spark di bab sebelumnya dan bagaimana Spark dapat digunakan untuk pemrosesan batch data. Di bagian ini, kami akan menjelaskan komponen Spark Streaming untuk analisis data streaming seperti data sensor, data clickstream, log server web, dll. Data streaming diserap dan dianalisis dalam batch mikro. Spark Streaming memungkinkan pemrosesan streaming yang dapat diskalakan, throughput tinggi, dan toleransi kesalahan. Spark Streaming menyediakan abstraksi tingkat tinggi yang disebut DStream (aliran diskrit). DStream adalah urutan RDD. Spark dapat menyerap data dari berbagai jenis sumber data seperti kerangka perpesanan langganan-publikasi, antrean perpesanan, sistem file terdistribusi, dan konektor khusus. Data yang diserap diubah menjadi DStreams. Gambar 8.16 menunjukkan komponen Spark Streaming.

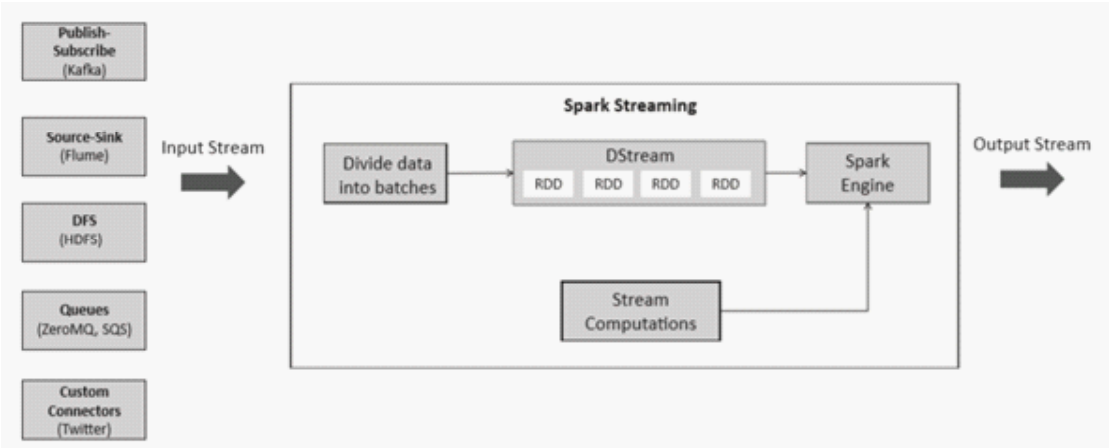


```
cqlsh:predictions> select * from data;

 timestamp           | fog_prediction | haze_prediction
---------------------+----------------+----------------
 2015-04-16 21:07:03 |              0 |               0
 2015-04-16 21:07:28 |              0 |               0
 2015-04-16 21:06:10 |              0 |               0
 2015-04-16 21:05:37 |              0 |               0
 2015-04-16 21:06:58 |              0 |               0
 2015-04-16 21:07:01 |              0 |               1
 2015-04-16 21:05:15 |              0 |               0
 2015-04-16 21:04:49 |              0 |               0
 2015-04-16 21:06:08 |              0 |               0
 2015-04-16 21:05:40 |              0 |               0
 2015-04-16 21:04:38 |              0 |               0
 2015-04-16 21:06:39 |              0 |               0
 2015-04-16 21:06:35 |              0 |               0
 2015-04-16 21:05:53 |              0 |               0
 2015-04-16 21:04:27 |              0 |               0
 2015-04-16 21:05:39 |              0 |               0
 2015-04-16 21:04:30 |              0 |               0
 2015-04-16 21:07:33 |              0 |               0
 2015-04-16 21:07:19 |              0 |               0
 2015-04-16 21:05:19 |              0 |               0
 2015-04-16 21:04:52 |              0 |               0
 2015-04-16 21:04:56 |              0 |               0
 2015-04-16 21:04:09 |              0 |               0
 2015-04-16 21:07:29 |              0 |               0
 2015-04-16 21:04:21 |              0 |               0
 2015-04-16 21:04:51 |              0 |               0
```

Gambar 8.14: *Screenshot Cassandra shell yang menunjukkan prediksi fog/haze*

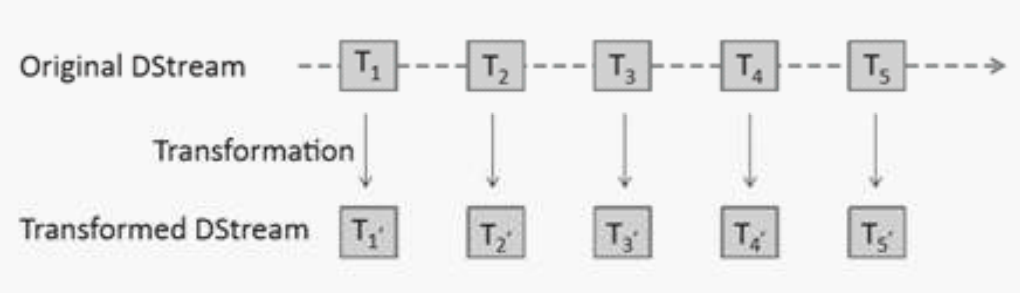
| Timestamp | Fog Prediction | Haze Prediction |
|---|---|---|
| 2015-04-16 21:07:03 | 0 | 0 |
| 2015-04-16 21:07:28 | 0 | 0 |
| 2015-04-16 21:06:10 | 0 | 0 |
| 2015-04-16 21:05:37 | 0 | 0 |
| 2015-04-16 21:06:58 | 0 | 0 |
| 2015-04-16 21:07:52 | 0 | 0 |
| 2015-04-16 21:07:01 | 0 | 1 |
| 2015-04-16 21:05:15 | 0 | 0 |
| 2015-04-16 21:04:49 | 0 | 0 |
| 2015-04-16 21:06:00 | 0 | 0 |
| 2015-04-16 21:07:56 | 0 | 0 |
| 2015-04-16 21:05:40 | 0 | 0 |
| 2015-04-16 21:08:10 | 0 | 0 |
| 2015-04-16 21:04:38 | 0 | 0 |
| 2015-04-16 21:06:39 | 0 | 0 |
| 2015-04-16 21:06:35 | 0 | 0 |
| 2015-04-16 21:05:53 | 0 | 0 |
| 2015-04-16 21:07:47 | 0 | 0 |
| 2015-04-16 21:08:06 | 0 | 0 |
| 2015-04-16 21:04:27 | 0 | 0 |
| 2015-04-16 21:05:39 | 0 | 0 |
| 2015-04-16 21:04:30 | 0 | 0 |
| 2015-04-16 21:07:33 | 0 | 0 |
| 2015-04-16 21:07:19 | 0 | 0 |
| 2015-04-16 21:05:19 | 0 | 0 |
| 2015-04-16 21:04:52 | 0 | 0 |
| 2015-04-16 21:04:56 | 0 | 0 |

Gambr 8.15: *Screenshot Flask web application yang menunjukka prediksi fog/haze*




Gambar 8.16: *Spark Streaming*

Sama seperti operasi untuk RDD yang dijelaskan di bab sebelumnya, Spark menyediakan operasi untuk DStream. Gambar 8.17 menunjukkan DStream, yang terdiri dari RDD, di mana setiap RDD berisi data dari interval waktu tertentu. Operasi DStream diterjemahkan ke dalam operasi pada RDD yang mendasarinya. Transformasi DStream seperti map, fl atMap, filter, reduceByKey tidak memiliki kewarganegaraan saat transformasi diterapkan ke RDD di DStream secara terpisah.

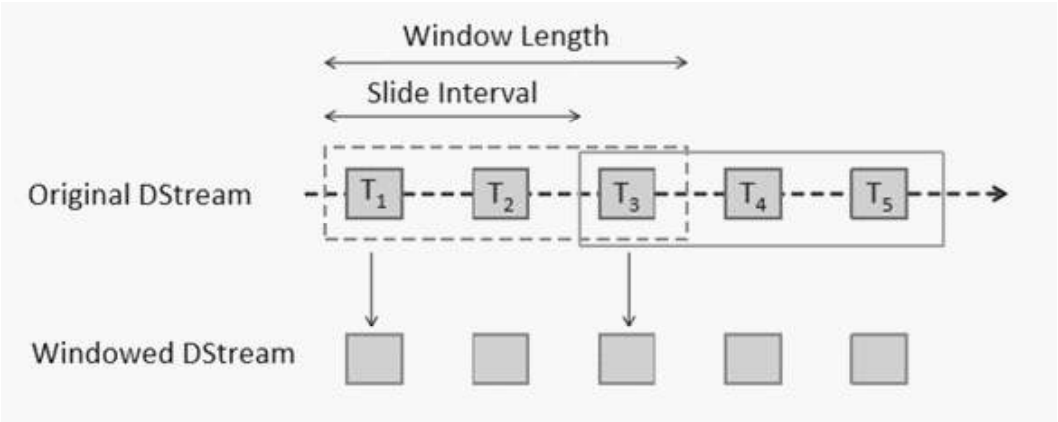



Gambar 8.17: *Transformasi Spark DStream*

Spark juga mendukung operasi stateful seperti operasi berwindow dan memperbarui operasi State By Key. Operasi stateful membutuhkan checkpointing untuk tujuan toleransi kesalahan. Untuk operasi stateful, direktori checkpoint disediakan yang RDD-nya ditentukan secara berkala. Gambar 8.18 menunjukkan contoh operasi window. Operasi window memungkinkan komputasi dilakukan melalui window geser data. Untuk operasi window, panjang window dan interval slide ditentukan. Pada contoh di Gambar 8.18, panjang window adalah 3 dan interval geser adalah 2.

### *Operasi Window*

Mari kita lihat beberapa operasi window yang umum digunakan:




Gambar 8.18: *Transformasi Spark DStream window*

***Window***

Operasi window mengembalikan DStream baru dari window geser di atas sumber DStream.

```
■ #Format:  window(windowLength, slideInterval)
# Example:  Return a new DStream with RDDs containing last
# 8 seconds of data, every 4 seconds
windowStream = sourceStream.window(8,4)
```

***countByWindow***

Operasi countByWindow menghitung jumlah elemen di window di atas DStream.

```
■ #Format:  countByWindow(windowDuration, slideDuration)
# Example:  Count the number of elements in a sliding window with
# window duration of 10 and slide interval of 4 seconds
count = sourceStream.countByWindow(10,4)
```

***reduceByWindow***

Operasi reduceByWindow menggabungkan elemen dalam window geser di atas aliran menggunakan fungsi yang ditentukan.

```
■ #Format:  reduceByWindow(func, windowLength, slideInterval)
# Example:  In a text data stream compute the running line lenghts
# with window duration of 10 and slide interval of 4 seconds
totalLength = lineLengths.reduceByWindow(lambda a, b:  a + b, 10, 4)
```

***reduceByKeyAndWindow***

Operasi reduceByKeyAndWindow saat diterapkan pada DStream yang berisi key-value pair , menggabungkan nilai dari setiap kunci dalam window geser di atas aliran menggunakan fungsi yang ditentukan. Operasi reduceByKeyAndWindow memiliki dua bentuk. Dalam satu bentuk, nilai yang dikurangi melalui window baru dihitung dengan menerapkan fungsi yang ditentukan ke seluruh window. Dalam bentuk lain, pengurangan nilai di window baru dihitung dengan menerapkan fungsi ke nilai baru yang masuk ke window dan fungsi terbalik di atas nilai yang meninggalkan window.

```
■ #Format:  reduceByKeyAndWindow(func, windowLength,
slideInterval, numPartitions)

# Example:  Count the number of words in a text stream
# with a sliding window of duration 10 seconds
and slide interval of 4 seconds

words = textStream.flatMap(lambda line:  line.split(" ")).map(lambda word:  (word,
1))
counts = reduceByKeyAndWindow(lambda a, b:  a + b, 10, 4)

# Alternative and optimized implementation
#Format:  reduceByKeyAndWindow(func, invFunc, windowLength,
slideInterval, numPartitions)

counts = reduceByKeyAndWindow(lambda a, b:  a + b,
lambda a, b:  a - b, 10, 4)
```

***countByValueAndWindow***

Operasi reduceByKeyAndWindow saat diterapkan di DStream yang berisi key-value pair , menampilkan DStream baru dengan key-value pair yang nilainya adalah jumlah untuk setiap kunci (jumlah elemen) di window geser.

```
■ #Format:  countByValueAndWindow(windowLength,
slideInterval, numPartitions)
# Example:  Count the number of elements for each key in a sliding
window with # window duration of 10 and slide interval of 4 seconds
count = sourceStream.countByValueAndWindow(10,4)
```

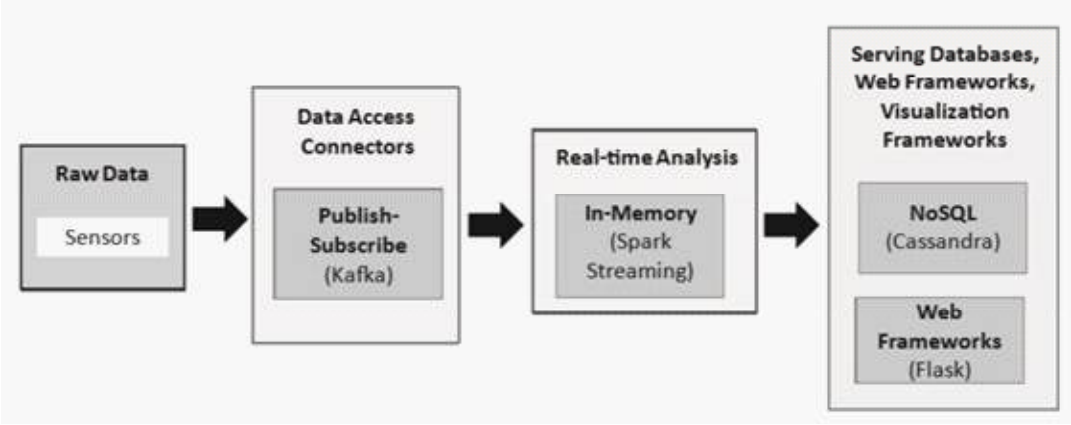
***updateStateByKey***

Tipe lain dari operasi stateful adalah operasi updateStateByKey yang memelihara dan melacak status setiap kunci dalam dataset. Operasi updateStateByKey membutuhkan status yang akan ditentukan dan fungsi pembaruan untuk memperbarui status menggunakan status sebelumnya dan nilai baru.

```
■ #Format:  updateStateByKey(func)
# Example:  Compute a running count of number of words
# in a text stream counts = lines.flatMap(lambda line:  line.split(" "))
.map(lambda word:  (word, 1))
.updateStateByKey(updateFunc)
```

## 8.4 Studi Kasus Spark

Pada bagian ini kami menyajikan dua studi kasus tentang analisis data sensor real-time dan satu studi kasus tentang analisis sentimen real-time tweet Twitter. Studi kasus pertama adalah tentang sistem untuk mendeteksi kebakaran hutan dengan menganalisis data sensor yang dikumpulkan dari sejumlah perangkat IoT yang digunakan di hutan. Studi kasus kedua adalah tentang sistem smart parking yang mendeteksi slot kosong di tempat parkir. Mengingat persyaratan analisis dari kedua sistem, mari kita mapkan sistem ke salah satu pola analisis yang diusulkan di Bab-1. Karena sistem ini memproses data sensor secara real-time, kita dapat menggunakan beta pattern. Gambar 8.19 menunjukkan realisasi beta pattern untuk deteksi kebakaran hutan dan sistem smart parking, dengan alat dan kerangka kerja khusus yang dapat digunakan.



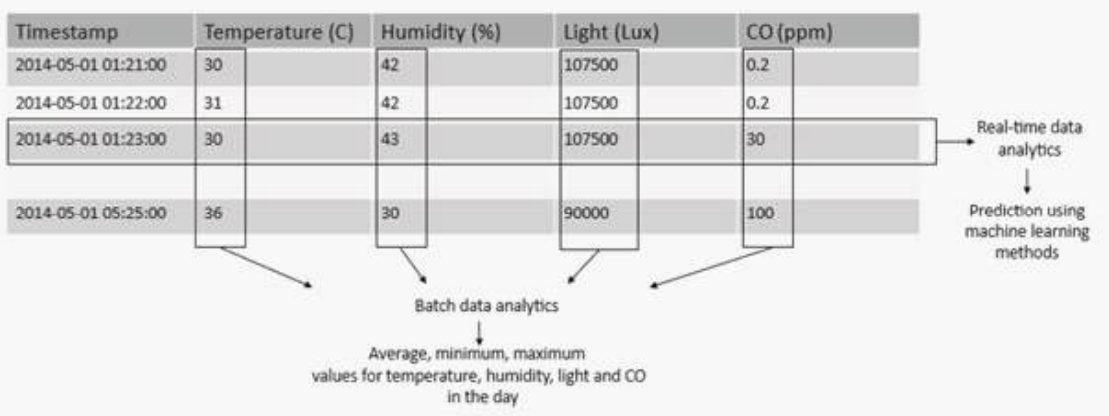

Gambar 8.19: *Realisasi Beta pattern untuk deteksi kebakaran hutan dan sistem smart parking*

### 8.4.1 Real-time Sensor Data Analysis

Studi kasus ini membahas tentang analisis data sensor real-time untuk sistem deteksi kebakaran hutan. Sistem ini memiliki beberapa titik akhir yang ditempatkan di hutan. End-node dilengkapi dengan sensor untuk mengukur suhu, kelembaban, cahaya dan karbon monoksida (CO) di berbagai lokasi di hutan. Setiap node akhir mengirimkan data secara independen ke topik Kafka. Sistem menggunakan Spark untuk analisis data dan membuat prediksi.

Gambar 8.20 menunjukkan contoh data yang dikumpulkan untuk deteksi kebakaran hutan. Setiap baris dalam tabel menunjukkan mahasiswaan suhu, kelembaban, cahaya, dan sensor CO yang diberi label waktu. Dengan menganalisis mahasiswaan sensor secara real-time (setiap baris tabel), prediksi tentang terjadinya kebakaran hutan dapat dibuat.

Mahasiswaan sensor juga dapat dikumpulkan pada berbagai skala waktu (menit, per jam, harian atau bulanan) untuk menentukan mahasiswaan mean, maksimum dan minimum. Data ini dapat membantu dalam mengembangkan model prediksi.




Gambar 8.20: *Format data yang dikumpulkan untuk deteksi kebakaran hutan*

### *Filtering Data*

Mari kita lihat aplikasi Spark Streaming untuk memfilter data sensor. Kode Python untuk aplikasi Spark ditunjukkan di Box 8.11. StreamingContext dibuat dengan menentukan SparkContext yang mendasari dan interval batch untuk membuat batch kecil data untuk analisis. Contoh ini menggunakan utilitas Kafka Spark untuk membuat DStream. DStream diubah dengan operasi map dan filter.

**Box 8.11: Apache Spark Python program for filtering sensor readings**

```
#Data format:
#"2014-06-25 10:47:44",26,36,2860,274

from pyspark import SparkContext
from pyspark.streaming import StreamingContext
from pyspark.streaming.kafka import KafkaUtils

sc = SparkContext(appName="FilterSensorData")
scc = StreamingContext(sc,1)

#Replace with DNS of instance running Zookeeper
zkQuorum = "ip-172-31-33-135.ec2.internal:2181"
topic = "forestfire"

kvs = KafkaUtils.createStream(ssc, zkQuorum,
      "spark-streaming-consumer", topic:1)
lines = kvs.map(lambda x:  x[1])
```

```
splitlines = lines.map(lambda line:  line.split(','))
filteredlines = splitlines.filter(lambda line:  int(line[1])>20 and
     int(line[2])>20 and int(line[3])>6000
     and int(line[4])>200)

filteredlines.pprint()

ssc.start()
ssc.awaitTermination()
```

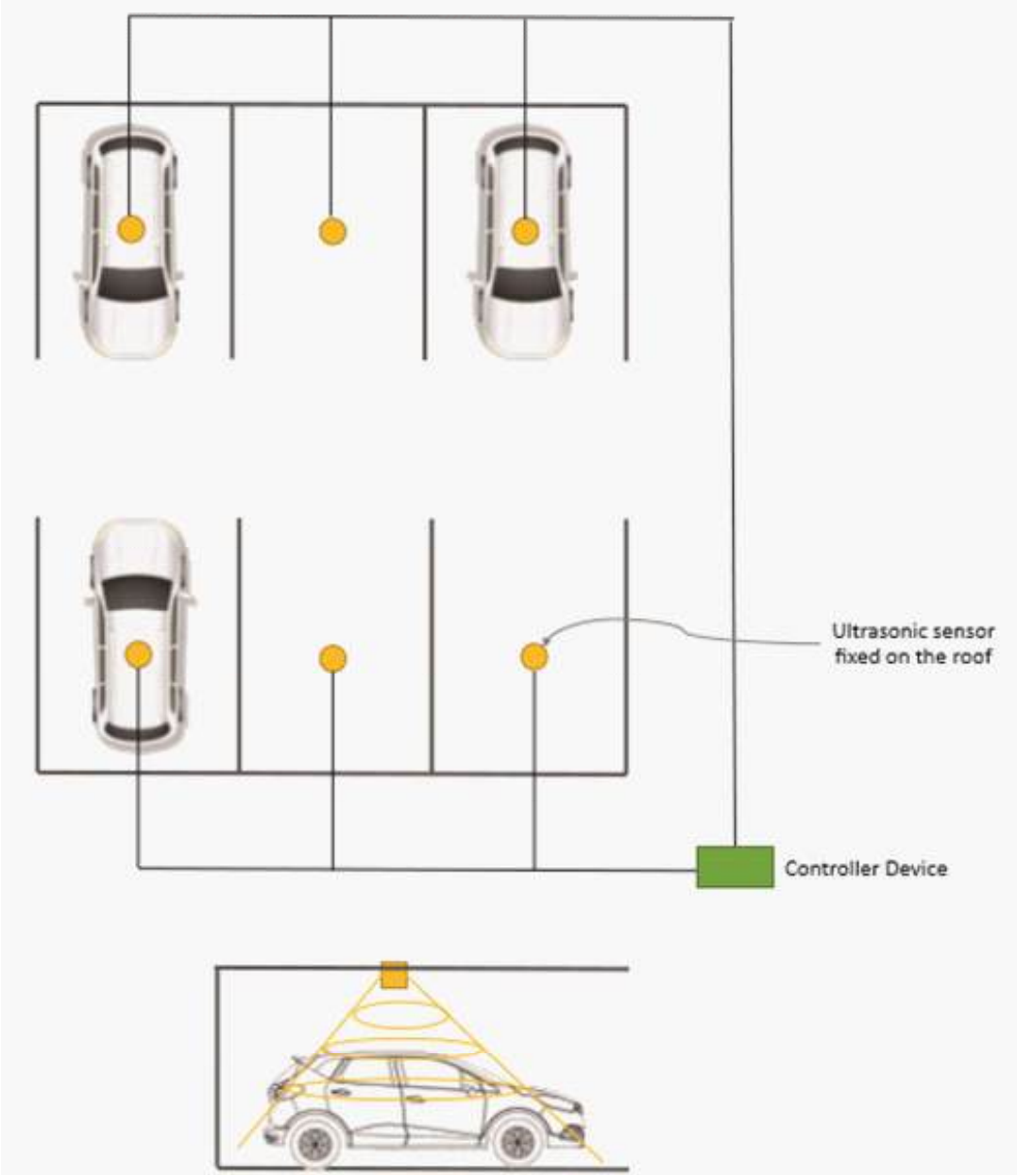
### 8.4.2 Real-Time Parking Sensor Data Analysis untuk Sistem Smart Parking

Dalam studi kasus ini, kami mendeskripsikan sistem untuk analisis real-time data sensor parkir di tempat smart parking untuk penentuan harga tempat parkir yang dinamis. Sistem menganalisis data parkir dari berbagai sensor yang terletak di tempat parkir dan secara dinamis memvariasikan harga untuk setiap tempat parkir berdasarkan tingkat hunian. Tingkat hunian tempat parkir dihitung, dan model penetapan harga dinamis digunakan untuk menghitung harga. Data yang diproses dapat disimpan dalam database untuk analisis pola penggunaan dan sejarah lebih lanjut.

Untuk keperluan studi kasus ini kita akan menggunakan generator data sintetis yang menghasilkan data dengan format seperti di bawah ini:

```
■ Parking Lot ID, Parking Spot ID, Timestamp, Occupied
Example:
5235, 20, 20:30, True
```

Dalam skenario yang ideal, setiap sensor parkir akan mengirimkan data setiap beberapa detik. Untuk studi kasus ini, kami menghasilkan waktu mulai dan berakhir untuk setiap tempat parkir saja. Durasi sesi parkir dipilih dari fungsi distribusi propiglitas. Box 8.12 menunjukkan kode Python untuk menghasilkan data sensor parkir sintetis. Data sensor dikirim ke topik Kafka.




Gambar 8.21: *Sistem Smart parking*

**Box 8.12: Python program for generating synthetic parking sensor data**

```
import signal
import sys
import random
from kafka.client import KafkaClient
from kafka.consumer import SimpleConsumer
from kafka.producer import SimpleProducer
```

```
#Connect to kafka
client = KafkaClient('ip-172-31-39-49.ec2.internal:6667')
producer = SimpleProducer(client)

class sensorMessage(object):
   def __init__(self,lotId,spotId,timeStamp,occupied):
      self.lotId       =    lotId
      self.spotId      =    spotId
      self.timeStamp   =    timeStamp
      self.occupied    =   occupied

   def getKey(self):
      return self.timeStamp

class parkingLot(object):
   def __init__(self,lotId):
      self.emptySpots = range(1,101)
      self.lotId = lotId

   def getLotId(self):
      return self.lotId

   def getEmptySpotId(self):
      if(len(self.emptySpots) != 0):
         return random.choice(self.emptySpots)
      else:
         return 0

class timeStamp(object):
   def __init__(self):
      self.hours       = range(0,24,1)
      self.minutes     = range(0,60,1)
      self.offset    = [1,1,1,1,1,2,2,4,4,8,8,8,8,8,8,8,8,6]

   def getTime(self):
      hour_start    = random.choice(self.hours)
      min_start    = random.choice(self.minutes)
      min_end = min_start
      hour_end = (hour_start + random.choice(self.offset)) % 24
      return (str(hour_start) + ":" + str(min_start),str(hour_end) +
":" + str(min_end))

def startParkingSession(lotId,spotId,timestamp):
   return sensorMessage(lotId,spotId,timestamp,True)
```

```
def endParkingSession(lotId,spotId,timestamp):
   return sensorMessage(lotId,spotId,timestamp,False)

parkingMessages = []

lots = [parkingLot(1),parkingLot(2),parkingLot(3)]

timeObj = timeStamp()
```

```
for i in range(0,200000):
   lot    = random.choice(lots)
   lotId    = lot.getLotId()
   spotId    = lot.getEmptySpotId()

   if spotId == 0:
      continue

   startTime,endTime = timeObj.getTime()

   parkingMessages.append(startParkingSession(lotId,spotId,startTime))
   parkingMessages.append(endParkingSession(lotId,spotId,endTime))

parkingMessages.sort(key = lambda msg:msg.getKey())

for msg in parkingMessages:
   x = str(msg.lotId), str(msg.spotId), str(msg.timeStamp),
     str(msg.occupied)
   producer.send_messages("smartparking",bytes(x))
```

Kami menggunakan Spark Streaming untuk analisis real-time data sensor parkir. Instans Spark Streaming terhubung ke Kafka dengan membuat Aliran Data baru yang disebut DStream. Spark Streaming membuat kumpulan mikro data yang diterima dari Kafka. Interval batch disetel ke 250ms. Jadi, setiap 250ms, kumpulan data sensor digabungkan menjadi satu RDD. Program Spark menghitung jumlah total slot yang ditempati di setiap tempat parkir dan menghitung harga dinamis berdasarkan heuristik gradien sederhana. Hasilnya disimpan dalam database Cassandra. Box 8.13 menunjukkan kode Python untuk aplikasi Spark Streaming yang menganalisis data parkir. Program menghitung rasio hunian untuk setiap tempat parkir.

**■ Box 8.13: Spark Streaming application for parking data analysis**

```
import random
import sys
import uuid
from cassandra.cluster import Cluster
from pyspark import SparkContext
from pyspark.streaming import StreamingContext
from pyspark.streaming.kafka import KafkaUtils
from operator import add
from kafka.client import KafkaClient
from kafka.consumer import SimpleConsumer
from kafka.producer import SimpleProducer
from pyspark.sql import SQLContext
```

```
from pyspark.sql.types import Row, StructField,
   StructType, StringType, IntegerType
import datetime
from datetime import timedelta


timestamp = datetime.datetime.utcnow()
client = KafkaClient('ip-172-31-39-49.ec2.internal:6667')
producer = SimpleProducer(client)

cluster = Cluster()
session = cluster.connect('test')
baseRate = 2
totalSlots = 500.0


def printel(x):
   global timestamp
   l = x.collect()
   if len(l) !=3:
   return
   print l
   for lot,cars in l:
      lotid= lot.strip("'")
   occrate = ( cars / totalSlots) * 100

   if occrate > 100:
      occrate = 100
          price = -1
      else:
         price = 2 + (occrate/100) * 20

      session.execute("INSERT INTO smartpark (key,lotid,
          occrate,time,price) VALUES (%s,%s,%s,%s,%s)", [uuid.uuid4(),
         int(lotid),occrate,str(timestamp)[:-7],price])
      seconds = random.randint(1800,2400)
      timestamp = timestamp + datetime.timedelta(0,seconds)


sc = SparkContext(appName="SmartParking")
sqlContext = SQLContext(sc)
ssc = StreamingContext(sc, 0.250)

# Replace with DNS of instance running Zookeeper
zkQuorum = "ip-172-31-39-49.ec2.internal:2181"
topic = "smartparking"

kvs = KafkaUtils.createStream(ssc, zkQuorum, "spark-streaming-consumer",
topic:  1)
lines = kvs.map(lambda x:  x[1])

lines = lines.map(lambda line:  line.encode('ascii','ignore'))
lines = lines.map(lambda line:  line.split(","))
lines = lines.map(lambda line:  (line[0][1:]  ,1
```

```
      if line[3][2:-2] == "True" else 0)).reduceByKey(add)

lines.foreachRDD(lambda rdd:  printel(rdd))
ssc.start()
ssc.awaitTermination()
```

Box 8.14 menunjukkan kode Python untuk aplikasi web Flask yang menampilkan rasio hunian untuk tempat parkir yang berbeda dan Gambar 8.22 menunjukkan screenshot dari aplikasi web.

**■ Box 8.14: Python code for the Flask web application**

```
from flask import Flask, render_template
from cassandra.cluster import Cluster


cluster = Cluster()
app = Flask(__name__)

@app.route('/')
def index():
   session = cluster.connect('test')
   lot1 = session.execute('Select occrate, price from smartpark where
          lotid = 1 limit 1')

   lot2 = session.execute('Select occrate, price from smartpark where
          lotid = 2 limit 1')

   lot3 = session.execute('Select occrate, price from smartpark where
          lotid = 3 limit 1')

   return render_template('index.html', price1=str(lot1[0].price),
      price2=str(lot2[0].price), price3=str(lot3[0].price),
      occrate1=str(lot1[0].occrate), occrate2=str(lot2[0].occrate),
      occrate3=str(lot3[0].occrate))


@app.route('/ParkingLot/<pid>')
def ParkingLot(pid=1):
   session = cluster.connect('test')
   lot = session.execute('Select occrate,price from smartpark where
      lotid = %s limit 30', [int(pid)])

   return render_template('plot.html', pid=pid,
      price=lot[0].price, occrate= lot[0].occrate, lot=lot)


if __name__ == '__main__':
   app.run(host='0.0.0.0', debug = True)
```

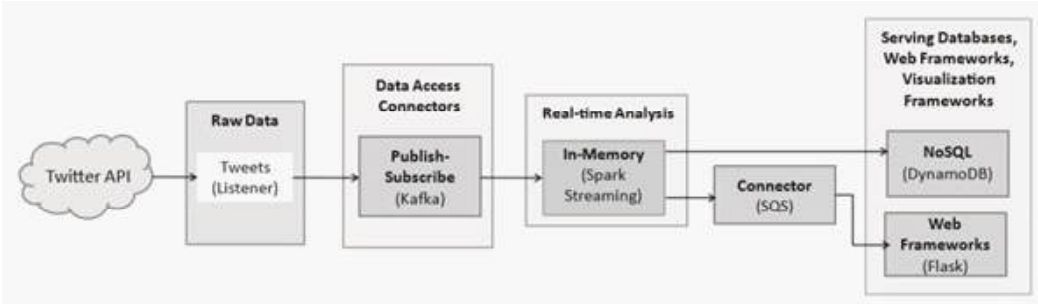

Gambar 8.22: *Aplikasi web smart parking yang menampilkan rasio hunian untuk tempat parkir yang berbeda*

### 8.4.3 Analisis sentimen Twiter Real-time

Di bagian ini, kami akan menjelaskan studi kasus pada sistem untuk analisis sentimen real-time dari feed Twitter menggunakan Spark Streaming. Gambar 8.23 menunjukkan realisasi beta pattern untuk sistem ini, dengan alat dan kerangka kerja khusus yang dapat digunakan.

Komponen sistem ini meliputi:

- Listener: Komponen listener terhubung ke Twitter dengan API streaming dan mengambil tweet secara real-time. Pustaka Python yang disebut tweepy digunakan untuk mengambil tweet yang berisi kata kunci tertentu. Listener menerbitkan tweet ke topik Kafka yang dikelola oleh Kafka Broker.
- Spark Streaming: Komponen Spark Streaming membuat DStream dengan menghubungkan ke topik Kafka dan menghitung sentimen mereka menggunakan leksikon sentimen.
- SQS: Tweet dengan stempel waktu dan sentimennya didorong oleh komponen Streaming Spark ke antrean SQS.
- DynamoDB: Tweet yang dianalisis juga disimpan dalam tabel Amazon DynamoDB.
- Flask Web App: Flask Web App mengambil tweet dari antrian SQS dan menampilkannya.




Gambar 8.23: *Realisasi beta pattern untuk analisis sentimen Twitter real-time*

Box 8.15 menunjukkan kode Python untuk komponen Listener yang mengambil tweet menggunakan API Twitter dan menerbitkan tweet ke topik Kafka. Keywords_list dalam program ini berisi daftar kata kunci yang terkait dengan Tweet yang diambil. Untuk studi kasus ini, kami menggunakan kata kunci – 'cricket'.

**Box 8.15: Listener component for fetching Tweets and publishing to Kafka**

```
from tweepy.streaming import StreamListener
from tweepy import OAuthHandler
from tweepy import Stream
from random import randrange
import time
import datetime
from kafka import KafkaProducer
import httplib
import json

#Variables that contains the user credentials to access Twitter API
access_token = "<enter>"
access_token_secret = "<enter>"
consumer_key = "<enter>"
consumer_secret = "<enter>"
#List of keywords to filter
keywords_list = ['cricket']

#Connect to Kafka
producer = KafkaProducer(bootstrap_servers=['127.0.0.1:6667'])

#This is a function publishes data to a Kafka topic
def publish(data):
 tweet = json.loads(data)
 if("retweeted" in tweet.keys()):
 if(tweet['retweeted'] == False):
  print tweet['text']

  #Send tweet to a topic named Cricket
  producer.send('Cricket', data.encode("utf"))

#This is a basic listener that just prints received tweets to stdout.
class StdOutListener(StreamListener):
 def on_data(self, data):
 publish(data)
 return True

 def on_error(self, status):
  print status

if __name__ == '__main__':
 #This handles Twitter authetification and
 #the connection to Twitter Streaming API
 l = StdOutListener()
 auth = OAuthHandler(consumer_key, consumer_secret)
```

```
except httplib.IncompleteRead:
 print "Incomplete Read!!!!"
 pass
```

Box 8.16 menunjukkan kode Python untuk aplikasi Streaming Spark yang menghitung sentimen tweet. Untuk mengaktifkan integrasi dengan Kafka di aplikasi Spark Streaming kami menggunakan pustaka KafkaUtils. Sebuah DStream diatur untuk menggunakan aliran tweet dari topik Kafka (di mana listener menerbitkan tweet tersebut). Aliran tweet yang dipartisi ulang menjadi RDD terpisah menggunakan fungsi map untuk melakukan transformasi paralel pada elemen masing-masing (masing-masing tweet).

Menggunakan metode foreachRDD, serta metode foreach, pertama-tama program memecah kumpulan RDD menjadi RDD terpisah, kemudian meneruskan setiap elemen RDD terpisah ke fungsi kustom untuk memformat ulang tweet, mengurai tweet, menghitung sentimen, dan terakhir mendorong data yang diproses ke antrean Amazon SQS untuk konsumsi web, serta database DynamoDB untuk penyimpanan.

Untuk menghitung sentimen, kami menggunakan leksikon sentimen AFINN [18], yang merupakan daftar lebih dari 2400 kata bahasa Inggris yang dinilai untuk sentimen yang merupakan bilangan bulat antara minus lima (negatif) dan plus lima (positif).

**■ Box 8.16: Spark Streaming application for sentiment analysis of tweets**

```
import boto.dynamodb2
from boto.dynamodb2.table import Table
import boto.sqs
from boto.sqs.message import Message
import cPickle as pickle
import json
from pyspark import SparkContext
from pyspark.streaming import StreamingContext
from pyspark.streaming.kafka import KafkaUtils
import random
from random import randint
import string
import time
import datetime
from datetime import datetime as newdatetime
import pytz

# Create connection to DynamoDB service
conn_dynamo = boto.dynamodb2.connect_to_region(REGION,
aws_access_key_id=ACCESS_KEY,
aws_secret_access_key=SECRET_KEY)
```

```
# Retrieve handle to DynamoDB table
table1=Table('tweets',connection=conn_dynamo)

# Create connection to SQS service
conn_sqs = boto.sqs.connect_to_region(REGION,
aws_access_key_id=ACCESS_KEY,
aws_secret_access_key=SECRET_KEY)

# Retrieve handle to queue
q1 = conn_sqs.get_all_queues(prefix='cricket')

TERMS={}

#----- Load Sentiments Dict ---
sent_file = open('AFINN-111.txt')
sent_lines = sent_file.readlines()
for line in sent_lines:
s = line.split("")
TERMS[s[0]] = s[1]

sent_file.close()


#----- Find Sentiment -------
def findsentiment(tweet):
 sentiment=0.0

 if tweet.has_key('text'):
  text = tweet['text']
  text=re.sub('[!@#$)(*=+/:;&%#|{},.? ']', '', text)
  splitTweet=text.split()

  for word in splitTweet:
   if TERMS.has_key(word):
    sentiment = sentiment+ float(TERMS[word])

 return sentiment

#---Send analyzed Tweet to SQS and DynamoDB---
def send_results(words):

 p = pickle.dumps(json.loads(json.dumps(datas)))
 m = Message()
 m.set_body(p)
 status = q1[0].write(m)

 item = table1.put_item(data=datas)

# Create a local StreamingContext
sc = SparkContext("local[2]", "NetworkWordCount")
ssc = StreamingContext(sc, 5)

cricket_kvs = KafkaUtils.createStream(ssc, '127.0.0.1',
   "spark-streaming-consumer", {'Cricket':  1})

cricket_lines = cricket_kvs.map(lambda x:  x[1])
```

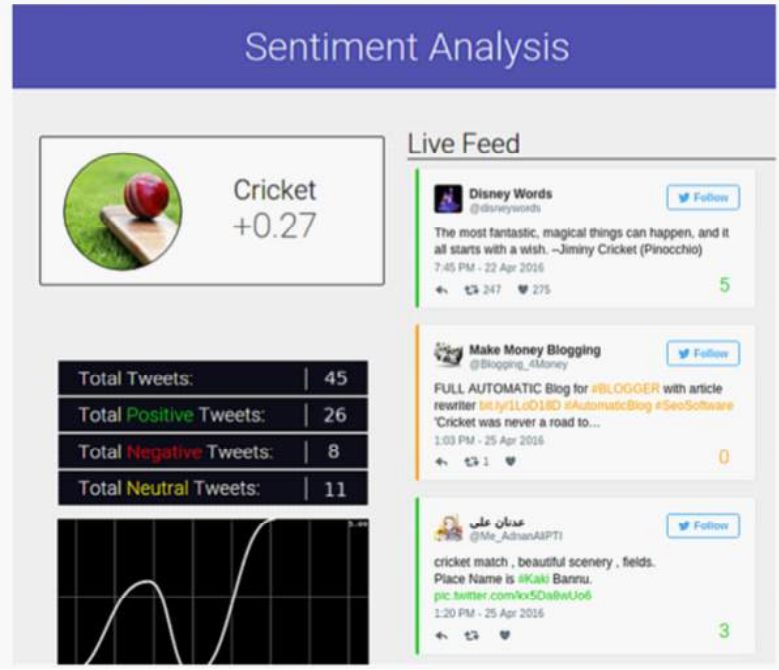
```
cricket_lines.pprint()

cricket_lines.foreachRDD(lambda rdd:  rdd.foreach(send_results))

# Start the computation
ssc.start()

# Wait for the computation to terminate
```

Box 8.17 menunjukkan kode Python untuk aplikasi web Flask. Karena keterbatasan ruang, kami hanya memasukkan kode Python untuk aplikasi web Flask dan menghilangkan kode HTML dan JavaScript untuk interface pengguna. Gambar 8.24 menunjukkan screenshot dari aplikasi web Flask yang menampilkan tweet yang dianalisis.

**■ Box 8.17: Flask web application to displays analyzed Tweets**

```
from flask import Flask, jsonify, abort, request, make_response, url_for
from flask.ext.cors import CORS
import sqlite3
import json
import random
from random import randint
import string
import datetime
import boto.sqs
from boto.sqs.message import Message
import cPickle as pickle
from time import sleep

app = Flask(__name__, static_url_path="")
cors = CORS(app, resources={r"/api/*":  {"origins":  "*"}})

#Create connection to SQS service
conn_sqs = boto.sqs.connect_to_region(REGION,
aws_access_key_id=ACCESS_KEY,
aws_secret_access_key=SECRET_KEY)

# Retrieve handle to queue
q1 = conn_sqs.get_all_queues(prefix='weather')

@app.route('/api/update', methods=['GET'])
def get_updates():
 topics = {}
 tweets = []
 flag = False
 data1 = {'empty':  'null'}

 count = q1[0].count()
 if count > 0:
  m = q1[0].read()
  bod = m.get_body()
```

```
    data1 = pickle.loads(str(bod))
    q1[0].delete_message(m)

      topics['Cricket'] = data1
  print topics
  return jsonify(topics)


if __name__ == '__main__':
  app.run(debug=True, host='0.0.0.0')
```




Gambar 8.24: *Screenshot Flask web application untuk menunjukkan analisis tweets*

### 8.4.4 Analisis Window pada Tweets

Studi kasus ini adalah tentang analisis tweet berwindow. Untuk studi kasus ini, komponen listener yang sama seperti yang dijelaskan di bagian sebelumnya dapat digunakan. Listener mengirim tweet ke topik Kafka. Box 8.18 menunjukkan kode untuk aplikasi streaming Spark untuk analisis tweet. Aplikasi ini menggunakan operasi reduceByKeyAndWindow untuk menemukan jumlah tweet positif, negatif, dan netral yang diterima dalam 30 detik terakhir setiap 10 detik.

**Box 8.18: Spark Streaming application for windowed analysis of tweets**

```
from pyspark import SparkContext
from pyspark.streaming import StreamingContext
from pyspark.streaming.kafka import KafkaUtils

#Load Sentiments Dictionary
sent_file = open('AFINN-111.txt')
sent_lines = sent_file.readlines()
for line in sent_lines:
   s = line.split("")
   TERMS[s[0]] = s[1]
sent_file.close()

def findsentiment(tweet):
   sentiment=0.0
   if tweet.has_key('text'):
      text = tweet['text']
      splitTweet=text.split()

      for word in splitTweet:
         if TERMS.has_key(word):
            sentiment = sentiment+ float(TERMS[word])

   return sentiment

def analyzeData(data):
   tweet = json.loads(data)
   sentiment= findsentiment(tweet)
   if sentiment>0:
      return ("Positive", 1)
   elif sentiment<0:
      return ("Negative", 1)
   else:
      return ("Neutral", 1)

sc = SparkContext(appName="SentimentAnalysis")
scc = StreamingContext(sc,1)

#Replace with DNS of instance running Zookeeper
zkQuorum = "ip-172-31-33-135.ec2.internal:2181"
topic = "tweets"

kvs = KafkaUtils.createStream(ssc, zkQuorum,
"spark-streaming-consumer", topic:1)
tweets = kvs.map(lambda x:  x[1])

sentiments = tweets.map(analyzeData)

windowedSentiments = sentiments.reduceByKeyAndWindow(lambda x,
             y:  x + y, lambda x, y:  x - y, 30, 10)

windowedSentiments.pprint()

ssc.start()
ssc.awaitTermination()
```

## Ringkasan

Dalam bab ini kami menjelaskan kerangka kerja Apache Storm dan Spark Streaming, untuk analisis data real-time. Storm adalah kerangka kerja untuk komputasi real-time terdistribusi dan toleran terhadap kesalahan. Storm adalah kerangka kerja yang skalabel dan terdistribusi, serta menawarkan pemrosesan pesan yang andal. Pekerjaan komputasi pada cluster Storm disebut topologi. Spout adalah jenis node dalam topologi, yang merupakan sumber aliran. Bolt adalah jenis node dalam topologi yang memproses tupel. Cerat dan baut memiliki banyak proses pekerja dan setiap pekerja memiliki banyak tugas. Kami menjelaskan berbagai jenis pengelompokan aliran untuk mempartisi aliran di antara tugas-tugas. Kami menjelaskan peran Nimbus, Supervisor dan Zookeeper, di cluster Storm. Di bagian kedua dari bab ini, kami menjelaskan kerangka kerja Spark Streaming, yang memungkinkan pemrosesan aliran yang dapat diskalakan, throughput tinggi, dan toleran terhadap kesalahan. Spark Streaming menyediakan abstraksi tingkat tinggi yang disebut DStream. Spark dapat menyerap data dari berbagai jenis sumber data dan data yang diserap diubah menjadi DStreams. Spark mendukung transformasi DStream yang stateless, dan juga operasi stateful seperti operasi berwindow. Studi kasus media sosial real-time, cuaca dan analisis data sensor menggunakan Storm dan Spark dijelaskan.

# Interactive Query


Bab ini mencakup

- Interactive Querying frameworks
- SparkSQL
- Hive
- Amazon Redshift
- Google BigQuery

Kueri interaktif berguna saat aplikasi analisis Anda menuntut fleksibilitas untuk membuat kueri data sesuai permintaan. Bab ini menjelaskan alat dan kerangka kerja untuk kueri interaktif daribigdata. Ini termasuk SparkSQL, Hive, Google BigQuery, dan Amazon RedShift. Alat dan kerangka kerja ini memungkinkan pengguna untuk menanyakan data dengan menulis pernyataan dalam bahasa mirip SQL.

## 9.1. Spark SQL

Spark SQL adalah komponen Spark yang memungkinkan pembuatan kueri interaktif. Spark SQL dapat secara interaktif membuat kueri data terstruktur dan semi terstruktur menggunakan kueri seperti SQL. Spark SQL menyediakan abstraksi pemrograman yang disebut DataFrames. DataFrame adalah kumpulan data terdistribusi yang diatur ke dalam kolom bernama. DataFrames dapat dibuat dari RDD yang ada, data terstruktur (file (seperti file teks, Parquet, JSON, Apache Avro), tabel Hive dan juga dari database eksternal. Spark menyediakan API Sumber Data, yang memungkinkan mengakses data terstruktur melalui Spark SQL.
Mari kita lihat contoh penggunaan Spark SQL untuk kueri interaktif data dari shell Spark. Untuk meluncurkan shell Spark Python, gunakan perintah yang ditunjukkan pada box di bawah ini:

```
■ bin/pyspark
```

Spark SQL menyediakan Konteks SQL, yang merupakan titik masuk untuk Spark SQL. Konteks SQL menyediakan fungsionalitas untuk membuat DataFrame, mendaftarkan

DataFrame sebagai tabel dan menjalankan pernyataan SQL di atas tabel. SQLContext dapat dibuat dari SparkContext seperti yang ditunjukkan pada box di bawah ini.

```
from pyspark.sql import SQLContext, Row
sqlContext = SQLContext(sc)
from pyspark.sql.types import *
```

Mari kita lihat contoh di mana kita membuat dan menggunakan DataFrame. Kami akan menggunakan kumpulan data Google N-Gram [29] dalam contoh ini. File dataset dalam format CSV dan berisi data pada bigram dengan kolom berikut: Bigram, Year, Count, Pages, Books. Dalam contoh ini, RDD pertama kali dibuat dengan memuat file dataset. Baris dalam file dipisahkan untuk mendapatkan kolom individu yang kemudian diubah menjadi objek Row dengan meneruskan daftar key-value pair ke kelas Row. SQLContext menyediakan fungsi create DataFrame untuk mengubah RDD objek Row menjadi DataFrame dengan menyimpulkan tipe data.

```
lines = sc.textFile("file:///home/hadoop/
googlebooks-eng-us-all-2gram-20090715-50.csv")

parts = lines.map(lambda l:  l.split(""))

ngrams = parts.map(lambda x:  Row(ngram=x[0],
year=int(x[1]), ngramcount=int(x[2]), pages=int(x[3]), books=int(x[4])))

schemaNGrams = sqlContext.createDataFrame(ngrams)
```

Untuk melihat baris dalam DataFrame yang dibuat, fungsi show dapat digunakan yang mencetak N baris pertama ke konsol (default N = 20).

```
>>> schemaNGrams.show()

+---+----+-------+---+---+
|books| ngram|ngramcount|pages|year|
+---+----+-------+---+---+
|   1|   !  09|       1|   1|1829|
|   3|   !  09|       3|   3|1879|
|   2|   !  09|       2|   2|1911|
|   4|   !  09|       4|   4|1941|
|   4|   !  09|       4|   4|1969|
|  12|   !  09|      17|  17|1994|
|   1|!  13.5|       1|   1|1936|
|   1|!  1430|       1|   1|1861|
|   1|!  1430|       1|   1|1959|
```

```
|   2|!  16th|       3|   3|1854|
|   1|!  16th|       1|   1|1959|
|   2|!  1791|       2|   2|1856|
|   1|!  1791|       1|   1|1968|
|   1|!  1847|       1|   1|1859|
|   2|!  1847|       2|   2|1909|
|   2|!  1847|       2|   2|1962|
|   2|!  1944|       2|   2|1945|
|   2|!  1944|       8|   8|1977|
|   1|!  1944|       1|   1|2007|
|   2|!  23rd|       2|   2|1957|
+----+-----+--------+----+----+
```

Untuk melihat skema DataFrame, fungsi printSchema dapat digunakan seperti yang ditunjukkan di bawah ini:

```
■ »> schemaNGrams.printSchema()
root
|- books:  long (nullable = true)
|- ngram:  string (nullable = true)
|- ngramcount:  long (nullable = true)
|- pages:  long (nullable = true)
|- year:  long (nullable = true)
```

Metode alternatif untuk membuat DataFrame adalah dengan menentukan skema seperti yang ditunjukkan pada contoh di bawah ini. RDD tupel pertama kali dibuat, dan skema didefinisikan menggunakan StructType. Akhirnya, skema diterapkan ke RDD tupel untuk membuat DataFrame.

```
■ ngrams = parts.map(lambda x:  (x[0], x[1], x[2], x[3], x[4]))
schemaString = "ngram year ngramcount pages books"
fields = [StructField(field_name, StringType(), True)
for field_name in schemaString.split()]
schema = StructType(fields)
schemaNGrams = sqlContext.createDataFrame(ngrams, schema)
```

Setelah melihat metode pembuatan DataFrames, sekarang mari kita lihat beberapa contoh kueri data. Box di bawah ini menunjukkan contoh fungsi filter yang memfilter baris menggunakan kondisi yang diberikan. Dalam contoh ini, kami menyaring semua N-gram yang memiliki hitungan lebih dari lima.

```
■ schemaNGrams.filter(schemaNGrams['ngramcount'] > 5).show()

+---+-----+----------+---+---+
|books| ngram|ngramcount|pages|year|
+---+-----+----------+---+---+
|  12|  !  09|   17|  17|1994|
| 2| !  1944|    8|  8|1977|
|  11|  !  28|   15|  15|1866|
|  10|  !  28|   10|  10|1891|
|  32|  !  28|   37|  37|1916|
|  14|  !  28|   14|  14|1941|
|  41|  !  28|   48|  47|1966|
|  57|  !  28|   76|  76|1991|
|  15|  !  56|   15|  15|1979|
|  54|  !  56|   61|  61|2004|
| 3| !  936|   16|  15|1943|
| 6| !  936|    9|  9|1973|
|  14| !  ANNE|   108|  95|1916|
| 4| !  ANNE|   35|  26|1941|
| 6| !  ANNE|   28|  26|1969|
| 6|  !  AS|    6|  6|1892|
| 6|  !  AS|    6|  6|1943|
| 6|  !  AS|    7|  7|1968|
|  10|  !  AS|   17|  15|1993|
|  17|!  Abort|   24|  21|2004|
+---+-----+----------+---+---+
```

Fungsi groupBy dapat digunakan untuk mengelompokkan DataFrame menggunakan kolom yang ditentukan. Agregasi (seperti avg, max, min, sum, count) kemudian dapat diterapkan ke DataFrame yang dikelompokkan. Box di bawah ini menunjukkan contoh pengelompokan N-Gram menurut tahun dan kemudian menerapkan agregasi hitungan untuk menemukan jumlah total N-Gram di setiap tahun.

```
■ schemaNGrams.groupBy("year").count().show()

+---+---+
|year|count|
+---+---+
|1831|  79|
|1832|  57|
|1833|  56|
|1834|  47|
|1835|  71|
|1836|  66|
|1837|  74|
|1838|  56|
|1839|  63|
|1840|  66|
```

```
|1841|  71|
|1842|  54|
|1843|  87|
|1844|  81|
|1845|  91|
|1846|  95|
|1847|  72|
|1848|  76|
|1849| 101|
|1850| 104|
+----+----+
```

Spark SQL memungkinkan mendaftarkan DataFrame sebagai tabel sementara untuk membuat kueri data menggunakan kueri seperti SQL. Dengan DataFrame (skema N-Gram) yang dibuat, tabel sementara (ngram) dibuat menggunakan fungsi register TempTable. Kueri SQL untuk memfilter semua N-gram yang memiliki jumlah lebih dari lima ditampilkan di bawah ini:

```
■ schemaNGrams.registerTempTable("ngrams")

result = sqlContext.sql("SELECT ngram, ngramcount
FROM ngrams WHERE ngramcount >= 5").show()
+-----+--------+
| ngram|ngramcount|
+-----+--------+
| !  09|   17|
|!  1944|    8|
| !  28|   15|
| !  28|   10|
| !  28|   37|
| !  28|   14|
| !  28|   48|
| !  28|   76|
| !  56|    5|
| !  56|    5|
| !  56|   15|
| !  56|   61|
| !  936|   16|
| !  936|    9|
|!  ANNE|   108|
|!  ANNE|   35|
|!  ANNE|   28|
| !  AS|    5|
| !  AS|    6|
| !  AS|    6|
+-----+--------+
```

Box di bawah ini memperlihatkan contoh kueri SQL yang menggunakan klausa GROUP BY untuk mengelompokkan N-Gram menurut kolom tahun dan pernyataan COUNT untuk menghitung jumlah N-Gram di setiap tahun. Hasilnya diurutkan berdasarkan hitungan N-Gram.

```
■ result = sqlContext.sql("SELECT year, COUNT(*) AS cnt FROM
ngrams GROUP BY year ORDER BY cnt DESC").show()

+---+---+
|year|cnt|
+---+---+
|2007|470|
|2002|450|
|2000|447|
|2003|446|
|2006|445|
|2001|445|
|1997|441|
|2004|440|
|1988|437|
|1999|436|
|2005|435|
|1991|432|
|1998|421|
|1995|415|
|1996|410|
|1987|408|
|1994|402|
|1990|397|
|1978|390|
|1986|390|
+---+---+
```

Box di bawah ini menunjukkan contoh menemukan N-Gram dengan jumlah maksimum (N-Gram terpopuler) di setiap tahun. Klausa GROUP BY digunakan untuk mengelompokkan N-Gram menurut tahun, dan klausa MAX digunakan untuk menemukan jumlah maksimum.

```
■ result = sqlContext.sql("SELECT ngram, year, MAX(ngramcount) AS
maxCount FROM ngrams GROUP BY ngram, year").show()

+----------+---+-----+
|   ngram|year|maxCount|
+----------+---+-----+
|    ) î|1900|   2|
|   ) $15|1939|   3|
|    ( HCOa|1989|   1|
| """ THERE'S"|1894|   3|
|""" obliterate"|1969|  10|
```

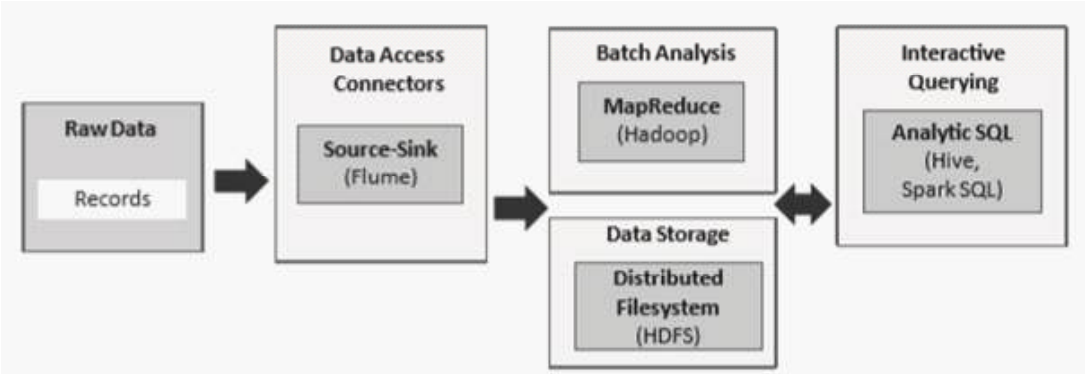
```
|  ' Mulberry|1839|   1|
|  ) refinery|1951|   3|
|    & RH|1982|  13|
| & Covington|1955|   1|
|    "|1942|  208|
|   ( 1260|1887|  13|
| """ Rain's"|1956|   3|
|  """ khaki"|1912|   5|
| """ Miinchner"|1919|   5|
```

## Studi Kasus

### Kueri Interaktif Data Cuaca

Pada bagian ini kami menyajikan studi kasus pada sistem untuk kueri interaktif data cuaca. Kami akan menggunakan dataset cuaca NCDC [52] untuk studi kasus ini. NCDC menyediakan akses ke data harian dari Jaringan Referensi Iklim A.S. / Jaringan Referensi Iklim Regional A.S. (USCRN / USRCRN) melalui FTP.

Mengingat persyaratan analisis sistem ini, mari kita mapkan sistem ke salah satu pola analisis yang diusulkan di Bab-1. Karena sistem memungkinkan kueri interaktif data cuaca, kita dapat menggunakan pola Delta. Gambar 9.1 menunjukkan realisasi pola Delta untuk sistem ini, dengan alat dan kerangka kerja khusus yang dapat digunakan. Di bagian ini kami menjelaskan penggunaan Spark SQL untuk kueri interaktif dari dataset cuaca. Penggunaan Hive untuk membuat kueri dataset yang sama dijelaskan di bagian selanjutnya.




Gambar 9.1: *Realisasi pola Delta untuk kueri interaktif data cuaca*

Untuk memindahkan data cuaca ke HDFS dari sumber eksternal, kita dapat menggunakan konektor sumber-sink seperti Flume. Karena dataset yang digunakan dalam studi kasus ini dalam bentuk file teks tunggal, kita cukup memindahkan file

teks ke dalam HDFS menggunakan alat baris perintah HDFS. Box di bawah ini menunjukkan kode Python untuk membuat DataFrame dari dataset menggunakan fungsi createDataFrame yang mengubah RDD objek Row menjadi DataFrame dengan menyimpulkan tipe datanya.

```
■ from pyspark.sql import SQLContext, Row
sqlContext = SQLContext(sc)
from pyspark.sql.types import *

lines = sc.textFile("file:///home/hadoop/
CRND0103-2014-VA_Cape_Charles_5_ENE_T.txt")
parts = lines.map(lambda l:  l.split(" "))
```

```
weatherdata = parts.map(lambda x:  Row(WBANNO=int(x[0]),
LST_DATE=int(x[1]),
CRX_VN=float(x[2]),
LONGITUDE=float(x[3]),
LATITUDE=float(x[4]),
T_DAILY_MAX=float(x[5]),
T_DAILY_MIN=float(x[6]),
T_DAILY_MEAN=float(x[7]),
T_DAILY_AVG=float(x[8]),
P_DAILY_CALC=float(x[9]),
SOLARAD_DAILY=float(x[10]),
SUR_TEMP_DAILY_TYPE=x[11],
SUR_TEMP_DAILY_MAX=float(x[12]),
SUR_TEMP_DAILY_MIN=float(x[13]),
SUR_TEMP_DAILY_AVG=float(x[14]),
RH_DAILY_MAX=float(x[15]),
RH_DAILY_MIN=float(x[16]),
RH_DAILY_AVG=float(x[17]),
SOIL_MOISTURE_5_DAILY=float(x[18]),
SOIL_MOISTURE_10_DAILY=float(x[19]),
SOIL_MOISTURE_20_DAILY=float(x[20]),
SOIL_MOISTURE_50_DAILY=float(x[21]),
SOIL_MOISTURE_100_DAILY=float(x[22]),
SOIL_TEMP_5_DAILY=float(x[23]),
SOIL_TEMP_10_DAILY=float(x[24]),
SOIL_TEMP_20_DAILY=float(x[25]),
SOIL_TEMP_50_DAILY=float(x[26]),
SOIL_TEMP_100_DAILY=float(x[27]) ))

schemaWeather = sqlContext.createDataFrame(weatherdata)
schemaWeather.registerTempTable("weather")
```

Box di bawah ini menunjukkan skema DataFrame yang dibuat

```
>>> schemaWeather.printSchema()
root
|- WBANNO: string (nullable = true)
|- LST_DATE: string (nullable = true)
|- CRX_VN: string (nullable = true)
|- LONGITUDE: string (nullable = true)
|- LATITUDE: string (nullable = true)
|- T_DAILY_MAX: string (nullable = true)
|- T_DAILY_MIN: string (nullable = true)
|- T_DAILY_MEAN: string (nullable = true)
|- T_DAILY_AVG: string (nullable = true)
|- P_DAILY_CALC: string (nullable = true)
|- SOLARAD_DAILY: string (nullable = true)
|- SUR_TEMP_DAILY_TYPE: string (nullable = true)
|- SUR_TEMP_DAILY_MAX: string (nullable = true)
|- SUR_TEMP_DAILY_MIN: string (nullable = true)
|- SUR_TEMP_DAILY_AVG: string (nullable = true)
|- RH_DAILY_MAX: string (nullable = true)
|- RH_DAILY_MIN: string (nullable = true)
|- RH_DAILY_AVG: string (nullable = true)
|- SOIL_MOISTURE_5_DAILY: string (nullable = true)
|- SOIL_MOISTURE_10_DAILY: string (nullable = true)
|- SOIL_MOISTURE_20_DAILY: string (nullable = true)
|- SOIL_MOISTURE_50_DAILY: string (nullable = true)
|- SOIL_MOISTURE_100_DAILY: string (nullable = true)
|- SOIL_TEMP_5_DAILY: string (nullable = true)
|- SOIL_TEMP_10_DAILY: string (nullable = true)
|- SOIL_TEMP_20_DAILY: string (nullable = true)
|- SOIL_TEMP_50_DAILY: string (nullable = true)
|- SOIL_TEMP_100_DAILY: string (nullable = true)
```

Box di bawah ini menunjukkan contoh kueri SQL untuk mengambil nomor stasiun WBAN, tanggal dan suhu harian maksimum yang diurutkan dalam urutan menurun. Klausa LIMIT digunakan untuk membatasi jumlah baris yang dikembalikan

```
# Sort by maximum temperature
result = sqlContext.sql("SELECT WBANNO, LST_DATE, T_DAILY_MAX FROM
weather ORDER BY T_DAILY_MAX DESC LIMIT 10").show()
+-----+--------+-----------+
|WBANNO|LST_DATE|T_DAILY_MAX|
+-----+--------+-----------+
| 3739|20140618|       33.4|
| 3739|20140708|       33.1|
| 3739|20140902|       32.9|
| 3739|20140702|       32.6|
| 3739|20140901|       32.1|
| 3739|20140709|       32.0|
| 3739|20140619|       31.8|
| 3739|20140906|       31.7|
```

Box di bawah ini menunjukkan contoh kueri SQL untuk menemukan suhu maksimum yang diamati sepanjang tahun.

```
# Max Temp observed in entire year
result = sqlContext.sql("SELECT WBANNO, MAX(T_DAILY_MAX) AS
maxTemp FROM weather GROUP BY WBANNO").show()

+-----+-------+
|WBANNO|maxTemp|
+-----+-------+
| 3739| 33.4|
+-----+-------+
```

Box di bawah ini menunjukkan contoh kueri SQL untuk menemukan suhu minimum yang diamati sepanjang tahun. Perhatikan penggunaan klausa WHERE untuk menyaring nilai yang hilang (setel ke -9999.0).

```
# Min Temp observed in entire year
result = sqlContext.sql("SELECT WBANNO, MIN(T_DAILY_MIN) AS
minTemp FROM weather WHERE T_DAILY_MIN <> -9999.0
GROUP BY WBANNO").show()

+-----+-------+
|WBANNO|minTemp|
+-----+-------+
| 3739| -15.1|
+-----+-------+
```

## 9.2 Hive

Apache Hive adalah kerangka kerja data warehousing yang dibangun di atas Hadoop. Hive menyediakan bahasa kueri seperti SQL yang disebut Hive Query Language, untuk membuat kueri data yang berada di HDFS. Hive mengatur data ke dalam tabel seperti database relasional. Sementara data tabel berada di HDFS, Hive menyertakan Metastore yang menyimpan metadata tabel (seperti skema tabel). Tabel Hive dibuat berseri dan disimpan dalam HDFS. Untuk setiap tabel, Hive memiliki direktori di HDFS. Tabel dibagi menjadi beberapa partisi yang mempercepat kueri. Partisi selanjutnya dibagi menjadi beberapa ember. Hive mengonversi kueri seperti SQL menjadi serangkaian pekerjaan yang dijalankan di kluster Hadoop. Hive dapat menggunakan MapReduce atau Apache Tez sebagai mesin eksekusi.

Hive menyediakan shell untuk membuat tabel dan membuat kueri data. Shell Hive dapat diluncurkan dengan perintah Hive. Box di bawah ini menunjukkan contoh pembuatan tabel Hive dari shell Hive.

```
# Creating Hive table
hive> CREATE TABLE weatherdata
(station INT, country STRING, timestamp INT, temperature FLOAT, humidity
FLOAT);
```

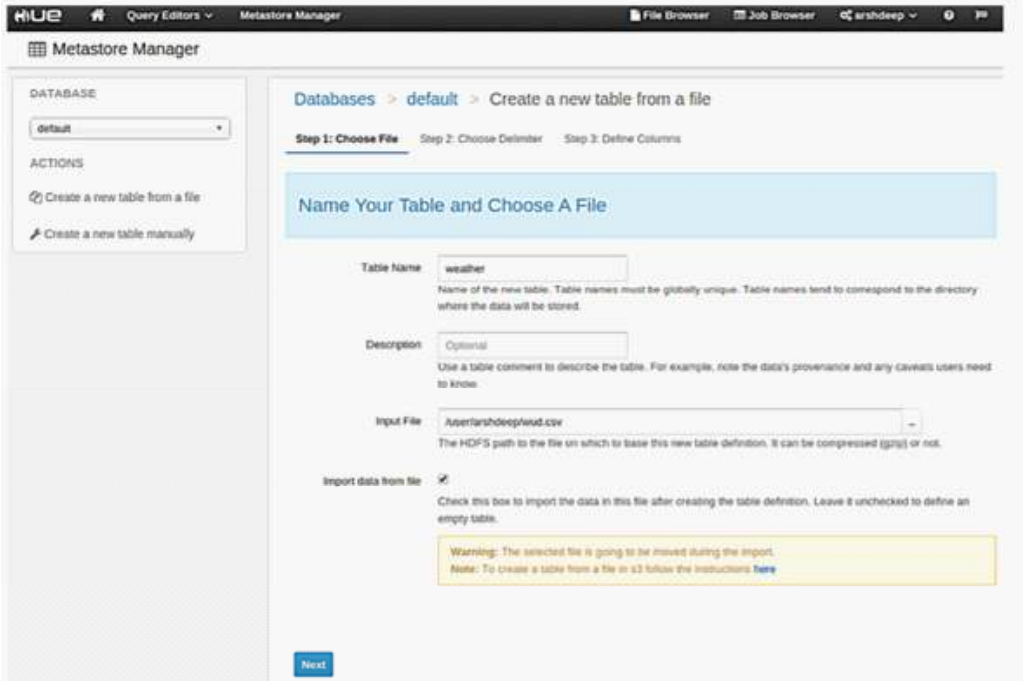
Opsi tambahan seperti format baris, format penyimpanan, partisi, dan keranjang juga dapat ditentukan saat membuat tabel seperti yang ditunjukkan pada contoh di bawah ini:

```
# Creating Hive table
hive> CREATE TABLE weatherdata
(station INT, timestamp INT, temperature FLOAT, humidity FLOAT)
PARTITIONED BY(country STRING)
CLUSTERED BY(station) SORTED BY(timestamp) INTO 4 BUCKETS
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
STORED AS TEXTFILE;
```

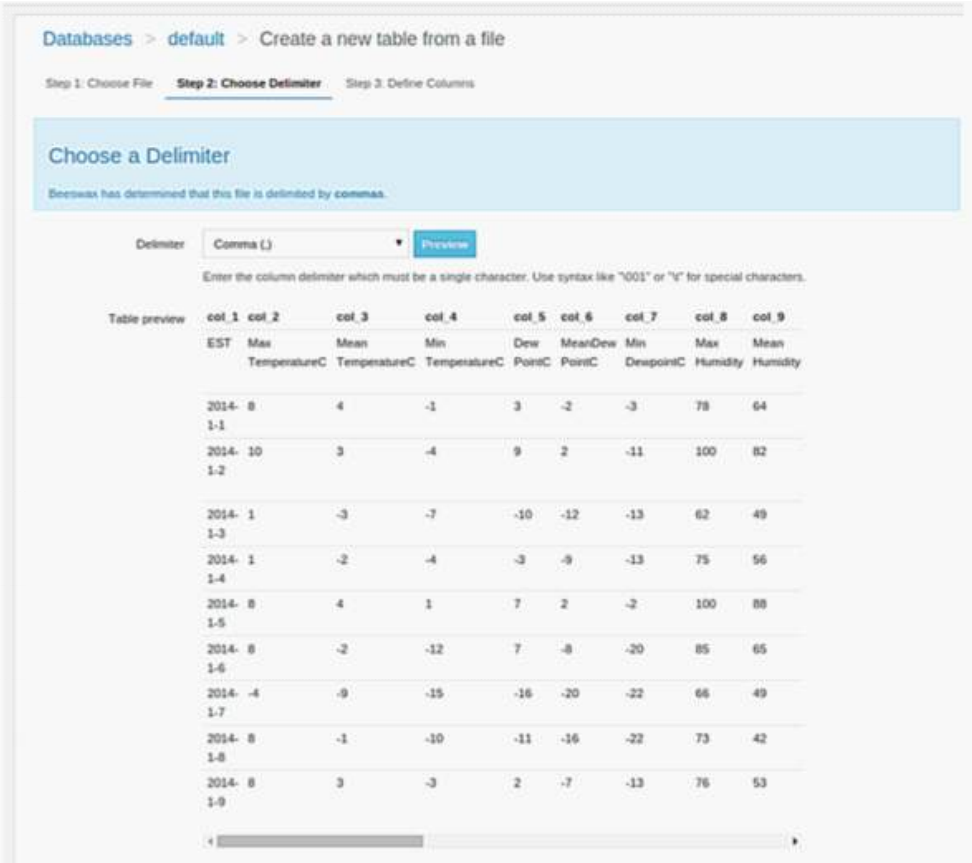
Tabel dapat dipartisi oleh satu atau lebih kolom. Saat tabel dipartisi, Hive membuat direktori data terpisah untuk setiap kombinasi nilai yang berbeda di kolom partisi. Tabel dapat disimpan sebagai file teks biasa, SequenceFiles atau dalam format file ORC. Tabel atau partisi dapat dibagi lagi menjadi beberapa box dengan menentukan kolom CLUSTERED BY. Data dapat diurutkan dalam keranjang dengan menentukan kolom URUTKAN MENURUT.

Untuk contoh di bagian ini, kita akan menggunakan Apache Hue, yang merupakan interface Web open source untuk menganalisis data dengan Hadoop. Dengan Hue, Anda dapat membuat tabel Hive, membuat dan menjalankan kueri Hive dari interface web.

Gambar 9.2 menunjukkan cara membuat tabel Metastore dari interface web Hue. File data bisa diunggah langsung dari wizard atau jika file sudah ada di HDFS, jalur file ditentukan. Untuk contoh di bagian ini, kami akan menggunakan kumpulan data cuaca untuk kota Atlanta untuk tahun 2014, diperoleh dari Weather Underground [31]. Pada langkah berikutnya, pembatas untuk file dataset ditentukan seperti yang ditunjukkan pada Gambar 9.3. Pada langkah berikutnya, nama kolom dan tipe data untuk kolom ditentukan seperti yang ditunjukkan pada Gambar 9.4.

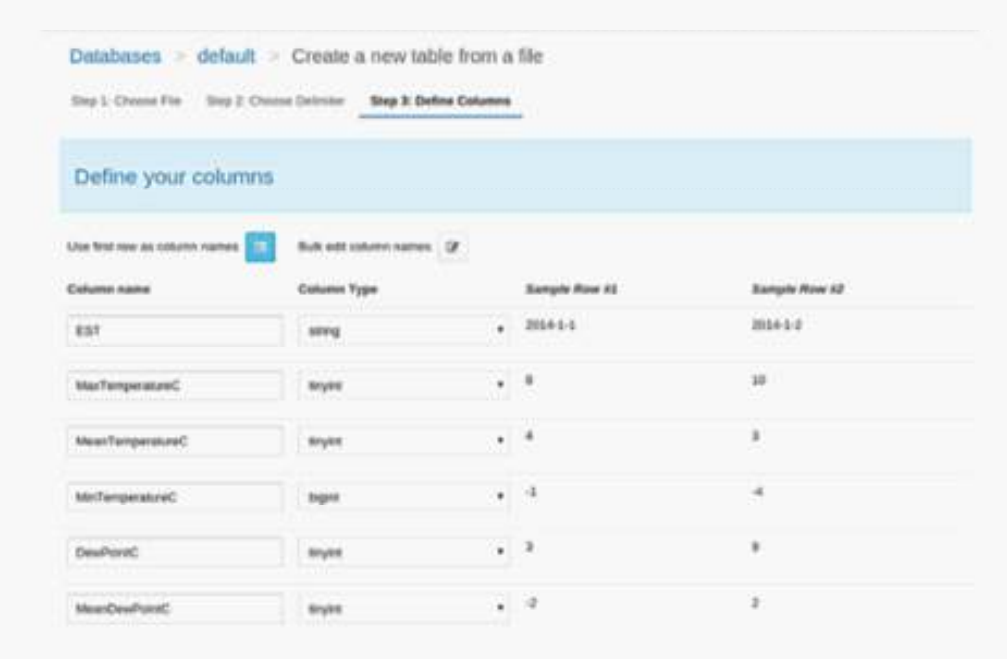



Gambar 9.2: *Membuat tabel Hive dari Hue - step 1*



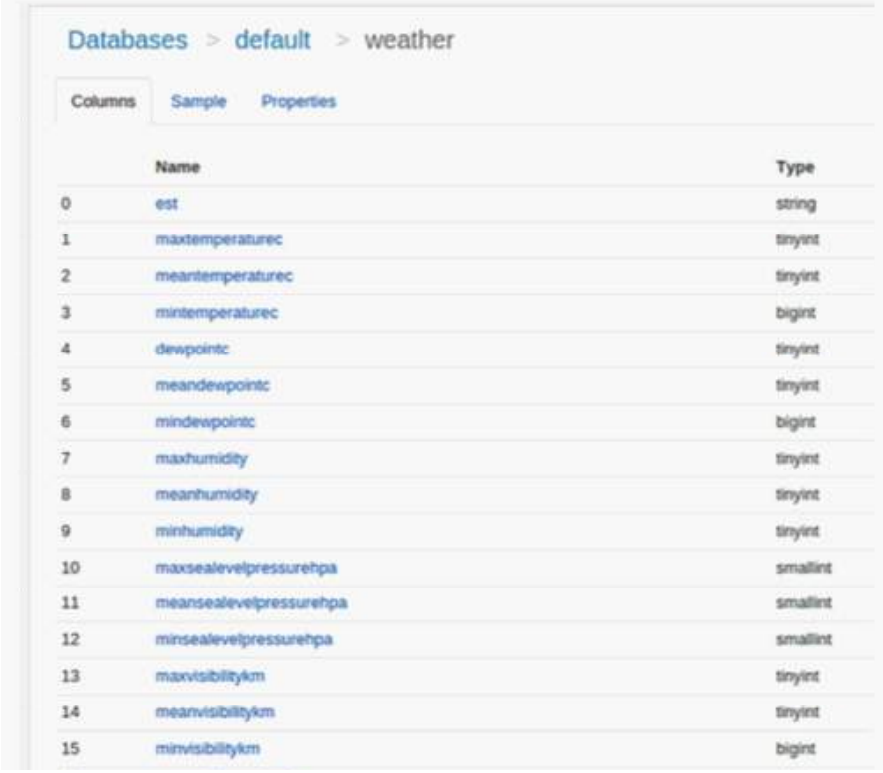

| col_1 | col_2 | col_3 | col_4 | col_5 | col_6 | col_7 | col_8 | col_9 |
|---|---|---|---|---|---|---|---|---|
| EST | Max TemperatureC | Mean TemperatureC | Min TemperatureC | Dew PointC | MeanDew PointC | Min DewpointC | Max Humidity | Mean Humidity |
| 2014-1-1 | 8 | 4 | -1 | 3 | -2 | -3 | 78 | 64 |
| 2014-1-2 | 10 | 3 | -4 | 9 | 2 | -11 | 100 | 82 |
| 2014-1-3 | 1 | -3 | -7 | -10 | -12 | -13 | 62 | 49 |
| 2014-1-4 | 1 | -2 | -4 | -3 | -9 | -13 | 75 | 56 |
| 2014-1-5 | 8 | 4 | 1 | 7 | 2 | -2 | 100 | 88 |
| 2014-1-6 | 8 | -2 | -12 | 7 | -8 | -20 | 85 | 65 |
| 2014-1-7 | -4 | -9 | -15 | -16 | -20 | -22 | 66 | 49 |
| 2014-1-8 | 8 | -1 | -10 | -11 | -16 | -22 | 73 | 42 |
| 2014-1-9 | 8 | 3 | -3 | 2 | -7 | -13 | 76 | 53 |

Gambar 9.3: *Membuat tabel Hive dari Hue - step 2*




Gambar 9.4: *Membuat tabel Hive dari Hue - step 3*

Setelah menyelesaikan wizard pembuatan tabel, Anda dapat melihat kolom-kolom tabel beserta tipe datanya seperti yang ditunjukkan pada Gambar 9.5.

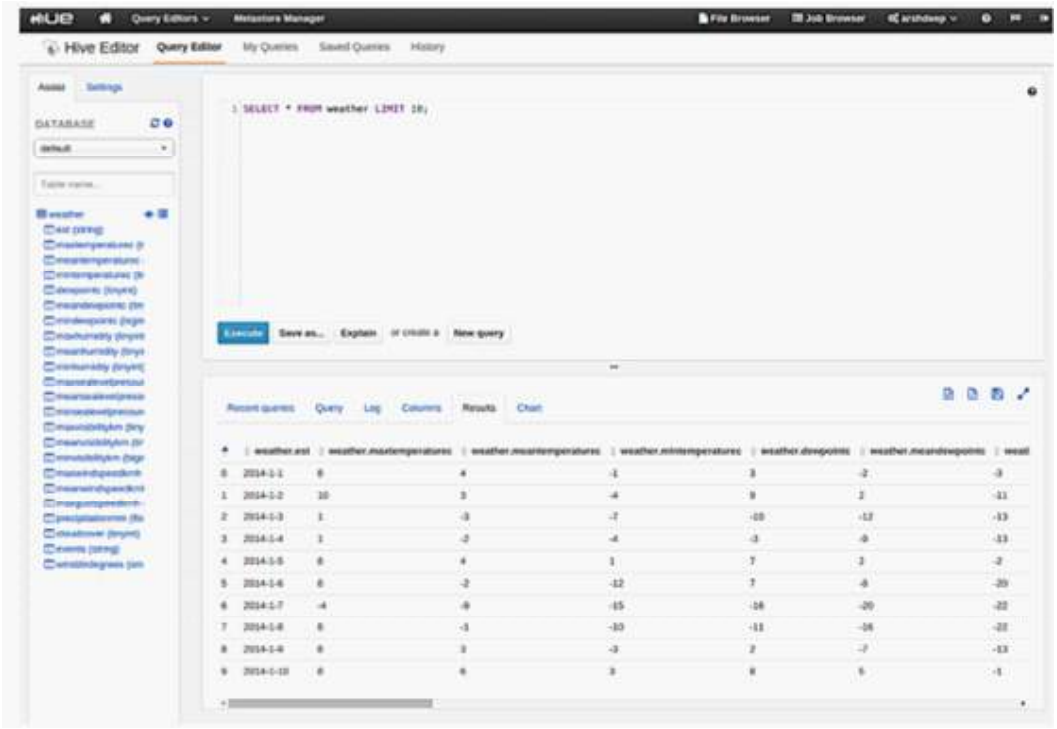



| | Name | Type |
|---|---|---|
| 0 | est | string |
| 1 | maxtemperaturec | tinyint |
| 2 | meantemperaturec | tinyint |
| 3 | mintemperaturec | bigint |
| 4 | dewpointc | tinyint |
| 5 | meandewpointc | tinyint |
| 6 | mindewpointc | bigint |
| 7 | maxhumidity | tinyint |
| 8 | meanhumidity | tinyint |
| 9 | minhumidity | tinyint |
| 10 | maxsealevelpressurehpa | smallint |
| 11 | meansealevelpressurehpa | smallint |
| 12 | minsealevelpressurehpa | smallint |
| 13 | maxvisibilitykm | tinyint |
| 14 | meanvisibilitykm | tinyint |
| 15 | minvisibilitykm | bigint |

| | | |
|---|---|---|
| 16 | maxwindspeedkmh | tinyint |
| 17 | meanwindspeedkmh | tinyint |
| 18 | maxgustspeedkmh | tinyint |
| 19 | precipitationmm | float |
| 20 | cloudcover | tinyint |
| 21 | events | string |
| 22 | winddirdegrees | smallint |

Gambar 9.5: *Tabel Hive yang dibuat dari Hue*

Mari kita lihat contoh beberapa kueri SQL yang dapat dijalankan dari editor kueri Hive di Hue. Gambar 9.6 menunjukkan query SQL untuk mengambil sepuluh record dari tabel. Output query juga dapat dilihat pada gambar.




Gambar 9.6: *Querying data with Hive*

Gambar 9.7 menunjukkan query SQL untuk menemukan suhu maksimum, minimum, dan rata-rata sepanjang tahun.

Gambar 9.7: *Kueri data dengan Hive*

Gambar 9.8 menunjukkan kueri SQL untuk menemukan sepuluh hari paling basah dalam setahun, diurutkan berdasarkan curah hujan

```
SELECT
  est,
  precipitationmm
FROM
  weather
ORDER BY
  precipitationmm DESC
LIMIT 10;
```

Execute | Save as... | Explain | or create a | New query

Recent queries | Query | Log | Columns | Results | Chart

| | est | precipitationmm |
|---|---|---|
| 0 | 2014-4-7 | 75.180000305175781 |
| 1 | 2014-7-19 | 61.470001220703125 |
| 2 | 2014-10-14 | 57.909999847412109 |
| 3 | 2014-1-11 | 53.090000152587891 |
| 4 | 2014-9-3 | 46.229999542236328 |
| 5 | 2014-8-8 | 41.150001525878906 |
| 6 | 2014-9-8 | 40.389999389648438 |
| 7 | 2014-11-23 | 38.099998474121094 |
| 8 | 2014-7-31 | 36.830001831054688 |
| 9 | 2014-11-17 | 34.799999237060547 |

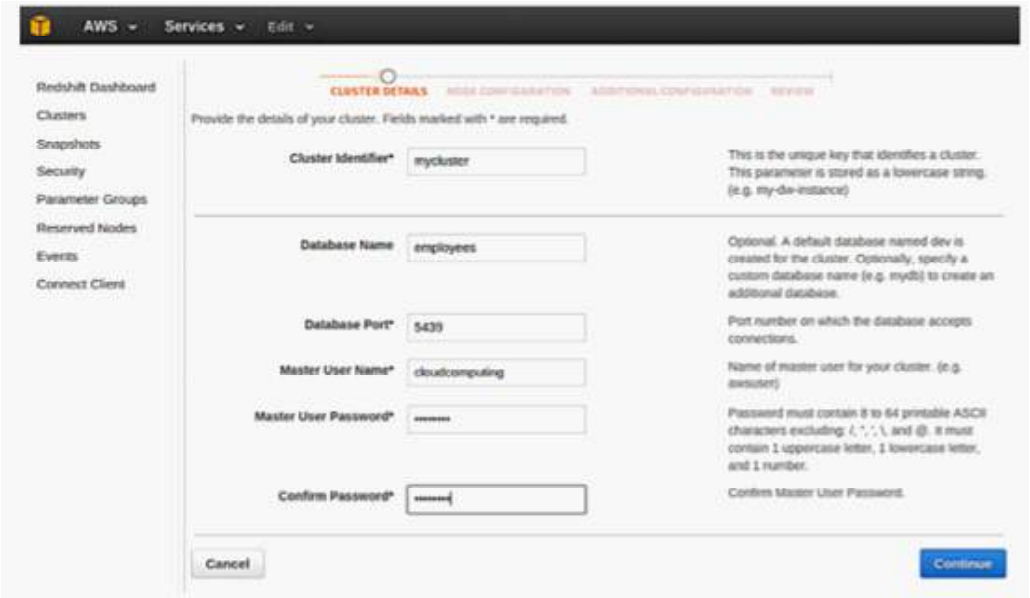
Gambar 9.8: *Kueri data dengan Hive*

## 9.3 Amazon Redshift

Amazon Redshift adalah layanan gudang data terkelola skala besar dan cepat. Redshift berspesialisasi dalam menangani kueri pada kumpulan data dengan ukuran hingga satu mapbyte atau lebih melalui penggunaan arsitektur Massively Parallel Processing (MPP), yang memparalelkan kueri SQL di semua sumber daya di klaster Redshift. Redshift menyediakan penyimpanan berbentuk kolom dan kompresi data tingkat lanjut yang memungkinkan pencarian kunci yang sangat cepat.

Gudang data Redshift terdiri dari kluster yang mencakup kumpulan node. Redshift sangat skalabel dan memungkinkan node tambahan ditambahkan atau dihapus dari cluster sambil tetap beroperasi. Redshift adalah gudang data yang terkelola sepenuhnya dan menyediakan fitur-fitur seperti pencadangan otomatis, toleransi kesalahan, keamanan, dan pemulihan.

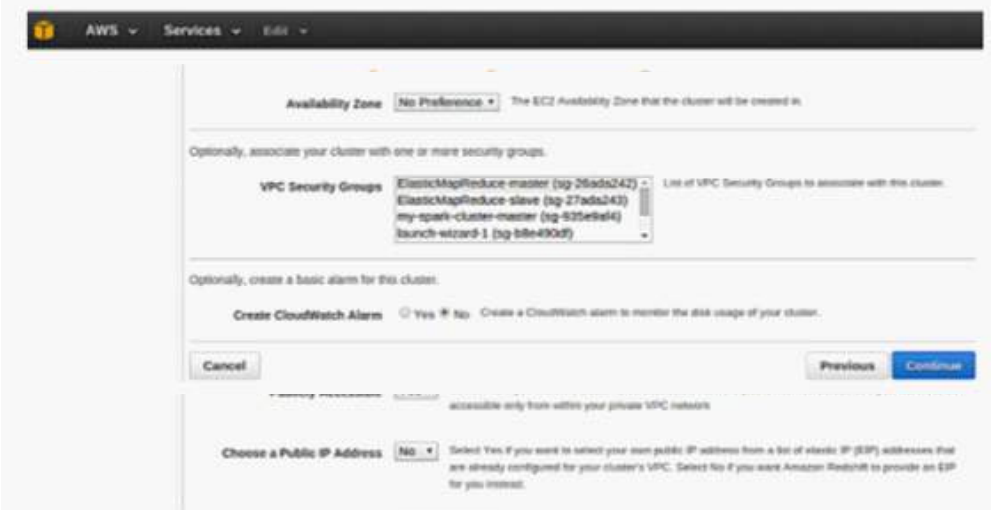
Untuk mulai bekerja dengan Redshift, pertama-tama kluster data harus dibuat menggunakan dasbor Redshift atau API Redshift. Gambar 9.9 menunjukkan screenshot wizard untuk membuat cluster Redshift. Nama unik untuk cluster dan database harus ditentukan.
Pada langkah selanjutnya, tipe node dan tipe cluster perlu ditentukan seperti yang ditunjukkan pada Gambar 9.10.



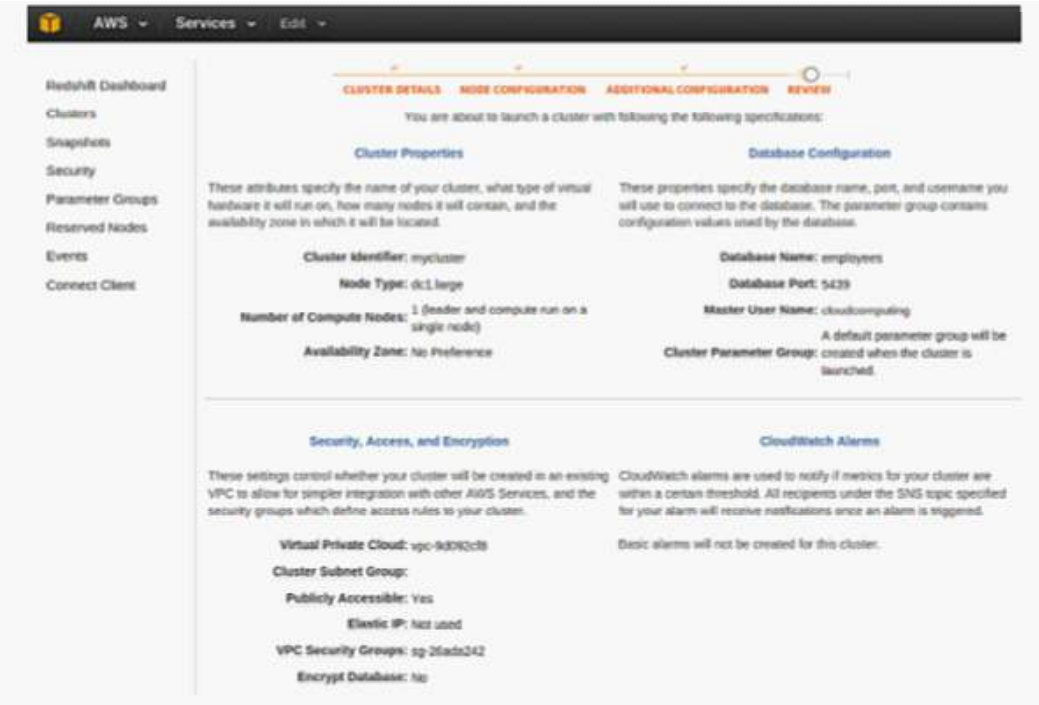

Gambar 9.9: *Screenshot Redshift cluster launch wizard – step 1*


Gambar 9.10: *Screenshot Redshift cluster launch wizard - step 2*

Pada langkah berikutnya, konfigurasi tambahan untuk cluster ditentukan seperti yang ditunjukkan pada Gambar 9.11. Gambar 9.12 menunjukkan halaman review dari wizard peluncuran cluster. Saat cluster diluncurkan, Anda dapat melihat detail cluster dari dasbor Redshift seperti yang ditunjukkan pada Gambar 9.13. Untuk terhubung ke cluster Redshift, Anda perlu mengkonfigurasi grup keamanan untuk mengotorisasi akses. Di grup keamanan untuk cluster yang diluncurkan, tambahkan aturan TCP kustom dan aktifkan port 5439 (yang merupakan port default untuk Redshift).




Gambar 9.11: *Screenshot Redshift cluster launch wizard - step 3*

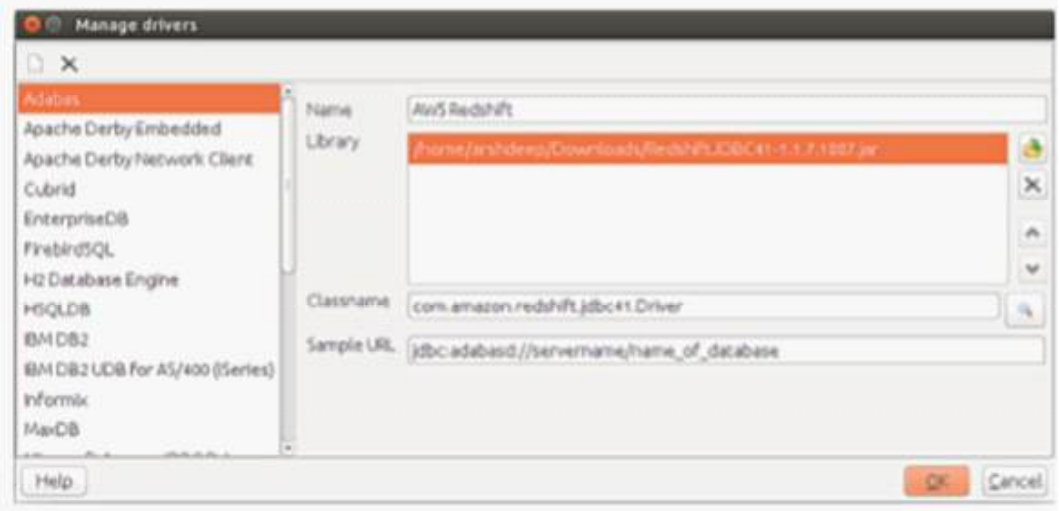



Gambar 9.12: *Screenshot Redshift cluster launch wizard - step 4*

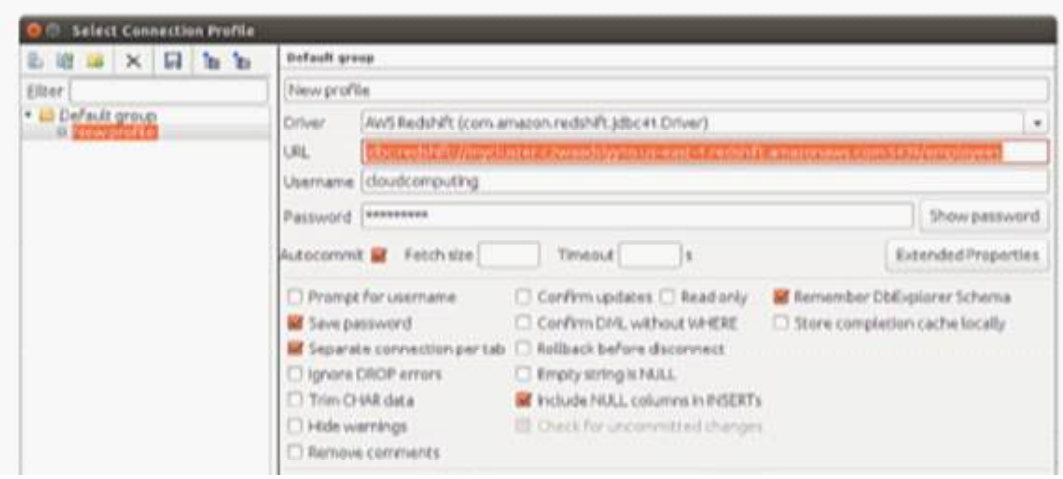
Anda sekarang dapat terhubung ke cluster Redshift dan menjalankan kueri SQL dari klien SQL. Untuk contoh di bab ini, kita akan menggunakan SQL Workbench / J client. Untuk menyiapkan klien SQL Workbench / J untuk menyambung ke Redshift, unduh driver JDBC Redshift dari dasbor Redshift dan tambahkan driver ke SQL Workbench / J dari dialog kelola driver seperti yang ditunjukkan pada Gambar 9.14.

Di SQL Workbench / J, buat profil koneksi baru dan salin URL JDBC dari cluster Redshift dalam dialog profil koneksi seperti yang ditunjukkan pada Gambar 9.15. Anda bisa mendapatkan URL JDBC cluster Redshift dari dasbor Redshift.
Sebagai contoh, kami akan menggunakan kumpulan data karyawan [30]. Box 9.1 menunjukkan pernyataan SQL untuk membuat tabel database.


Gambar 9.13: *Screenshot dasbor Redshift yang menampilkan detail cluster*



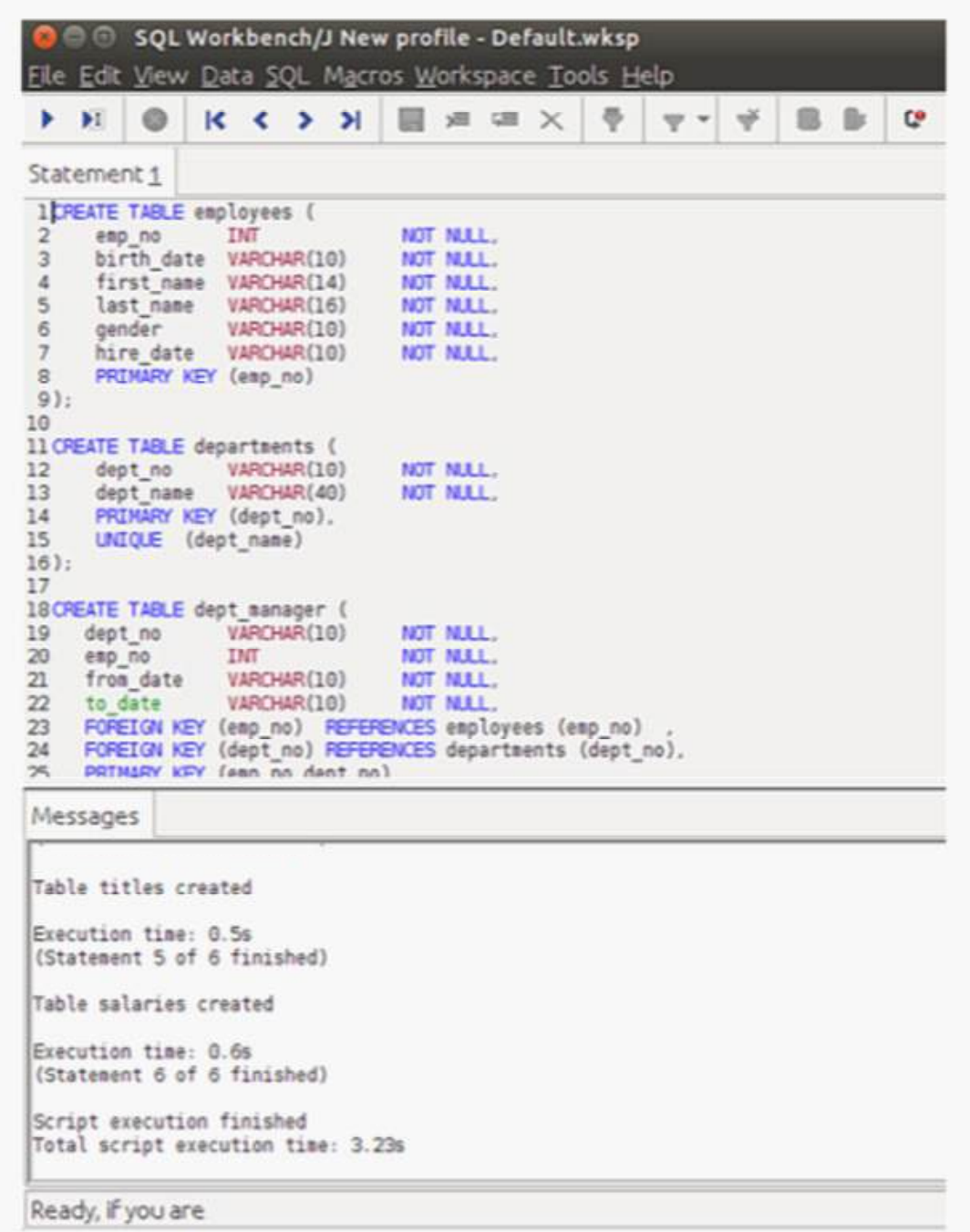

Gambar 9.14: *Screenshot of SQL WorkBench - adding Redshift driver*




Gambar 9.15: *Screenshot dari SQL WorkBench - menyambungkan ke cluster Redshift*

**Box 9.1: SQL statements for creating database tables**

```
CREATE TABLE employees (
   emp_no     INT          NOT NULL,
   birth_date VARCHAR(14)          NOT NULL,
   first_name VARCHAR(20)     NOT NULL,
   last_name VARCHAR(20)     NOT NULL,
   gender     VARCHAR(10)     NOT NULL,
   hire_date VARCHAR(14)          NOT NULL,
   PRIMARY KEY (emp_no)
);
CREATE TABLE departments (
   dept_no     VARCHAR(10)     NOT NULL,
   dept_name VARCHAR(40)     NOT NULL,
   PRIMARY KEY (dept_no),
   UNIQUE (dept_name)
);
CREATE TABLE dept_manager (
   dept_no     VARCHAR(10)     NOT NULL,
   emp_no     INT          NOT NULL,
   from_date   VARCHAR(14)     NOT NULL,
   to_date     VARCHAR(14)     NOT NULL,
   FOREIGN KEY (emp_no) REFERENCES employees (emp_no) ,
   FOREIGN KEY (dept_no) REFERENCES departments (dept_no),
   PRIMARY KEY (emp_no,dept_no)
);
CREATE TABLE titles (
   emp_no     INT          NOT NULL,
   title     VARCHAR(50)     NOT NULL,
   from_date VARCHAR(14)     NOT NULL,
   to_date     VARCHAR(14),
   FOREIGN KEY (emp_no) REFERENCES employees (emp_no),
   PRIMARY KEY (emp_no,title, from_date)
);
CREATE TABLE salaries (
   emp_no     INT          NOT NULL,
   salary     INT          NOT NULL,
   from_date VARCHAR(14)     NOT NULL,
   to_date     VARCHAR(14)     NOT NULL,
   FOREIGN KEY (emp_no) REFERENCES employees (emp_no),
   PRIMARY KEY (emp_no, from_date)
```




Gambar 9.16: *Screenshot SQLWorkBench – Membuat tabel*

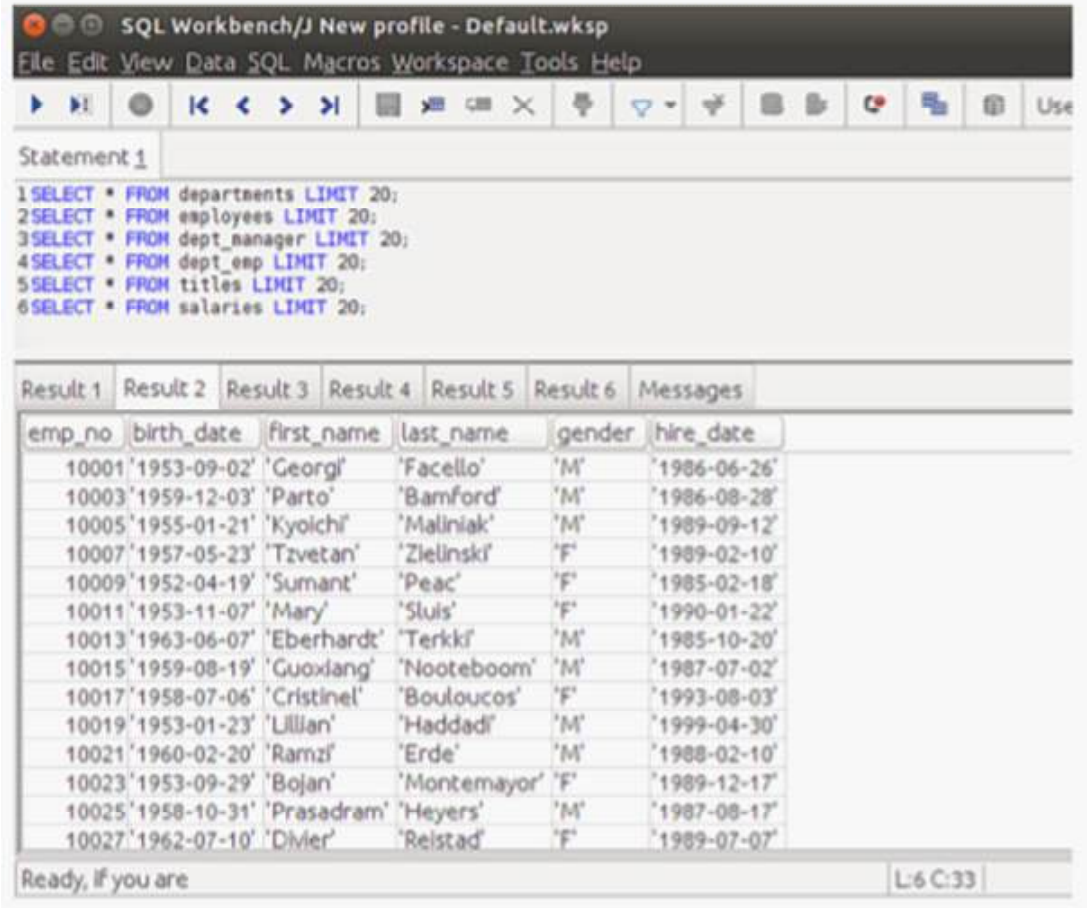
File dataset untuk tabel individual (dalam format CSV) disalin ke bucket Amazon S3. Untuk memuat data ke cluster Redshift, perintah SQL COPY digunakan seperti yang diperlihatkan di Box 9.2.

**Box 9.2: Loading data into tables**

```
copy departments FROM 's3://abahgacloud/departments.csv'
credentials 'aws_access_key_id=<enter-key>;
aws_secret_access_key=<enter-secret>'
delimiter ',';


COPY employees FROM 's3://abahgacloud/employees.csv'
credentials 'aws_access_key_id=<enter-key>;
aws_secret_access_key=<enter-secret>'
delimiter ',';


COPY dept_manager FROM 's3://abahgacloud/dept_manager.csv'
credentials 'aws_access_key_id=<enter-key>;
aws_secret_access_key=<enter-secret>'
delimiter ',';


COPY dept_emp FROM 's3://abahgacloud/dept_emp.csv'
credentials 'aws_access_key_id=<enter-key>;
aws_secret_access_key=<enter-secret>'
delimiter ',';


COPY titles FROM 's3://abahgacloud/titles.csv'
credentials 'aws_access_key_id=<enter-key>;
aws_secret_access_key=<enter-secret>'
delimiter ',';
```

```
COPY salaries FROM 's3://abahgacloud/salaries.csv'
credentials 'aws_access_key_id=<enter-key>;
aws_secret_access_key=<enter-secret>'
delimiter ',';
```

Dengan data yang dimuat ke dalam database, Anda sekarang dapat membuat kueri data menggunakan pernyataan SQL. Gambar 9.17 menunjukkan pernyataan SQL SELECT untuk mendapatkan subset baris dari semua tabel dalam database Karyawan. Gambar 9.18 menunjukkan detail eksekusi kueri di dasbor Redshift.




Gambar 9.17: *Screenshot of SQLWorkBench - querying data*

Gambar 9.18: Screenshot Redshift dashboard menunjukkan detail eksekusi kueri

Sekarang, mari kita lihat beberapa kueri SQL lanjutan. Box 9.3 menunjukkan kueri SQL untuk mengambil gaji (dan tanggal efektif) untuk karyawan dengan ID 10009.

**Box 9.3: Get all the salaries and the effective dates for employee with ID 10009**

```
SELECT first_name, last_name, salary, from_date, to_date
FROM employees, salaries
WHERE employees.emp_no=salaries.emp_no
AND employees.emp_no='10009';

'Sumant'   'Peac'   60929   '1985-02-18'   '1986-02-18'
'Sumant'   'Peac'   64780   '1987-02-18'   '1988-02-18'
'Sumant'   'Peac'   69042   '1989-02-17'   '1990-02-17'
'Sumant'   'Peac'   71434   '1991-02-17'   '1992-02-17'
'Sumant'   'Peac'   76518   '1993-02-16'   '1994-02-16'
'Sumant'   'Peac'   80944   '1995-02-16'   '1996-02-16'
'Sumant'   'Peac'   85875   '1997-02-15'   '1998-02-15'
'Sumant'   'Peac'   90668   '1999-02-15'   '2000-02-15'
```

```
'Sumant'   'Peac'   94443   '2001-02-14'   '2002-02-14'
'Sumant'   'Peac'   64604   '1986-02-18'   '1987-02-18'
'Sumant'   'Peac'   66302   '1988-02-18'   '1989-02-17'
'Sumant'   'Peac'   70889   '1990-02-17'   '1991-02-17'
'Sumant'   'Peac'   74612   '1992-02-17'   '1993-02-16'
'Sumant'   'Peac'   78335   '1994-02-16'   '1995-02-16'
'Sumant'   'Peac'   82507   '1996-02-16'   '1997-02-15'
'Sumant'   'Peac'   89324   '1998-02-15'   '1999-02-15'
'Sumant'   'Peac'   93507   '2000-02-15'   '2001-02-14'
'Sumant'   'Peac'   94409   '2002-02-14'   '9999-01-01'
```

Box 9.4 menunjukkan kueri SQL untuk menemukan gaji terbaru karyawan dengan ID 10009.

**■ Box 9.4: Find most recent salary of the employee with ID 10009**

```
SELECT first_name, last_name, salary
FROM employees, salaries
WHERE employees.emp_no=salaries.emp_no
AND employees.emp_no='10009'
AND to_date=(
   SELECT MAX(to_date)
   FROM salaries
   WHERE emp_no = '10009'
);

'Sumant'   'Peac'   94409
```

Box 9.5 menunjukkan kueri SQL untuk mengambil detail dari manajer departemen dengan ID d009.

**■ Box 9.5: Get details of the manager of the department with ID d009**

```
SELECT * FROM employees
WHERE emp_no = (
   SELECT emp_no
   FROM dept_manager
   WHERE dept_no = 'd009'
   AND to_date = (
      SELECT max(from_date)
      FROM dept_manager
      WHERE dept_no = 'd009'
   )
);

111939   '1960-03-25'   'Yuchang'   'Weedman'   'M'   '1989-07-10'
```

Box 9.6 menunjukkan kueri SQL untuk mengambil semua judul yang dipegang dan tanggal efektif, untuk karyawan dengan ID 10009.

**■ Box 9.6: Get all the titles held and the effective dates for the employee with ID 10009**

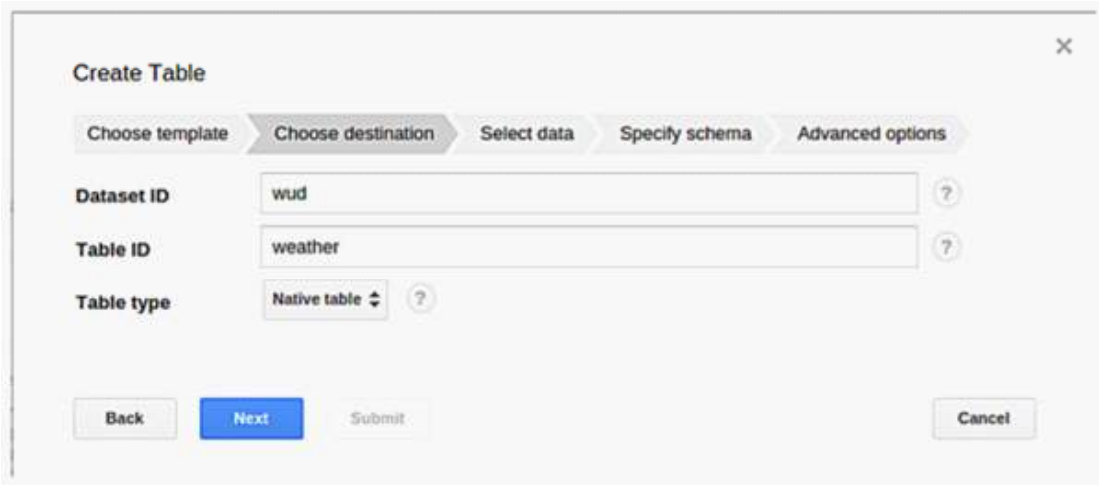
```
SELECT first_name, last_name, title, from_date, to_date
FROM employees, titles
WHERE employees.emp_no=titles.emp_no
AND employees.emp_no='10009';

'Sumant'   'Peac'   'Engineer'   '1990-02-18'   '1995-02-18'
'Sumant'   'Peac'   'Assistant Engineer'   '1985-02-18'   '1990-02-18'
'Sumant'   'Peac'   'Senior Engineer'   '1995-02-18'   '9999-01-01'
```

## 9.4 Google BigQuery

Google BigQuery adalah layanan untuk menanyakan dataset yang sangat besar. BigQuery memungkinkan pembuatan kueri dataset menggunakan kueri seperti SQL. Kueri BigQuery dijalankan pada tabel khusus tambahan yang menggunakan kekuatan pemrosesan infrastruktur Google untuk mempercepat kueri. Untuk membuat kueri data, pertama-tama muat ke BigQuery menggunakan konsol BigQuery atau alat baris perintah BigQuery atau API BigQuery. Data dapat dalam format CSV atau JSON. Data yang diupload dapat dibuat kueri menggunakan dialek SQL BigQuery.

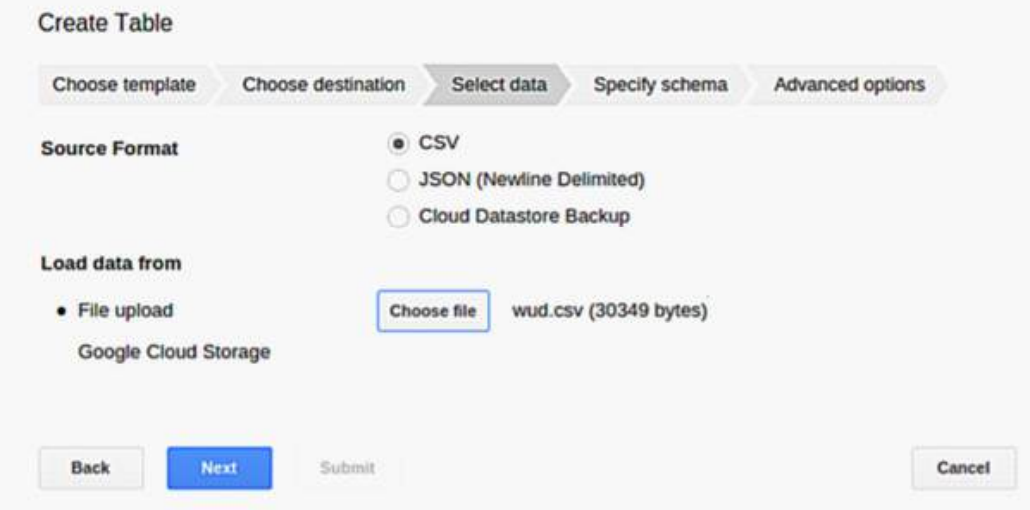
Perbedaan utama antara Amazon Redshift dan Google BigQuery adalah bahwa meski Redshift menawarkan database SQL standar yang memiliki arsitektur Massively Parallel Processing (MPP) dan perlu disediakan sebelum dapat digunakan, BigQuery adalah layanan online yang tidak mengharuskan pengguna untuk menyediakan layanan. BigQuery adalah layanan yang selalu tersedia tempat pengguna dapat memuat data dan kemudian membuat kueri data tersebut. Untuk Redshift, pengguna dikenai biaya berdasarkan jumlah node yang disediakan dan jam kerja node. BigQuery, berbeda dengan Redshift, mengenakan biaya kepada pengguna untuk jumlah data yang disimpan dan jumlah data yang dikonsumsi pada waktu kueri.

Gambar 9.19 menunjukkan screenshot wizard BigQuery untuk memuat data. ID dataset dan ID tabel ditentukan pada langkah pertama.



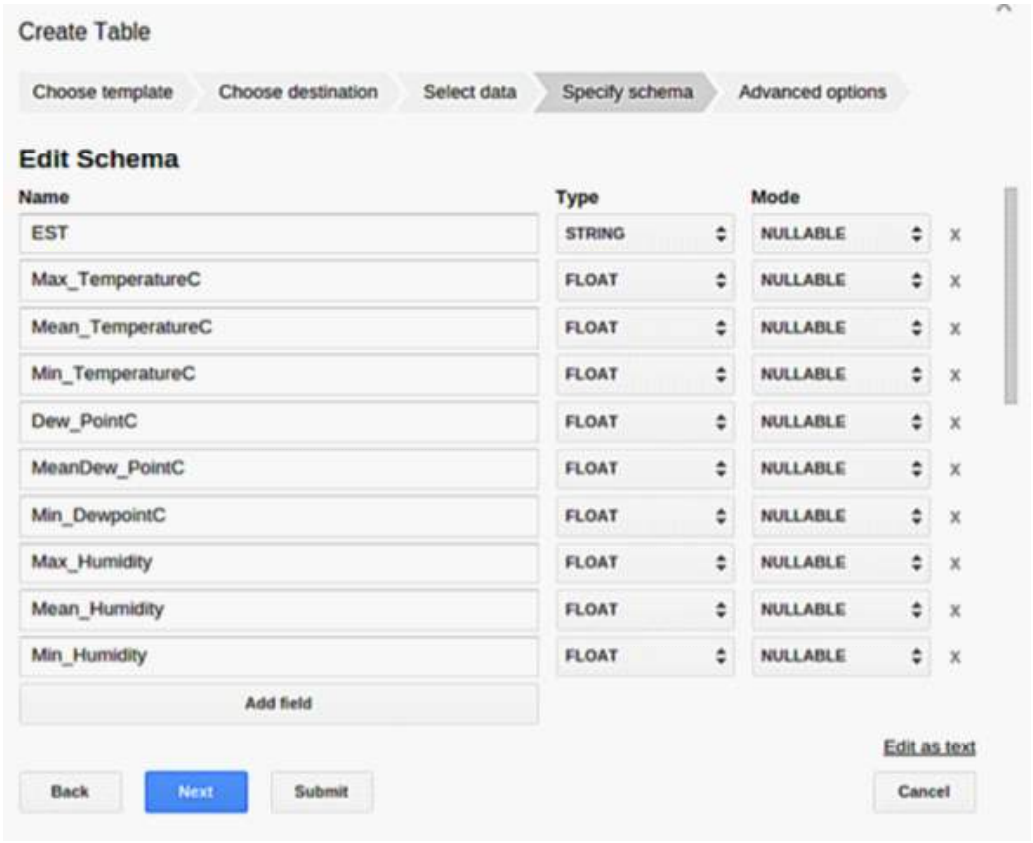

Gambar 9.19: *Screenshot BigQuery dashboard – Pembuatan tabel - step 1*

Gambar 9.20 menunjukkan langkah selanjutnya di mana format file sumber (CSV atau JSON) ditentukan. File data dapat diunggah langsung dari wizard atau jika file sudah ada di Google Cloud Storage, URI file ditentukan. Untuk contoh di bagian ini, kami akan menggunakan kumpulan data cuaca untuk kota Atlanta untuk tahun 2014, diperoleh dari Weather Underground [31].



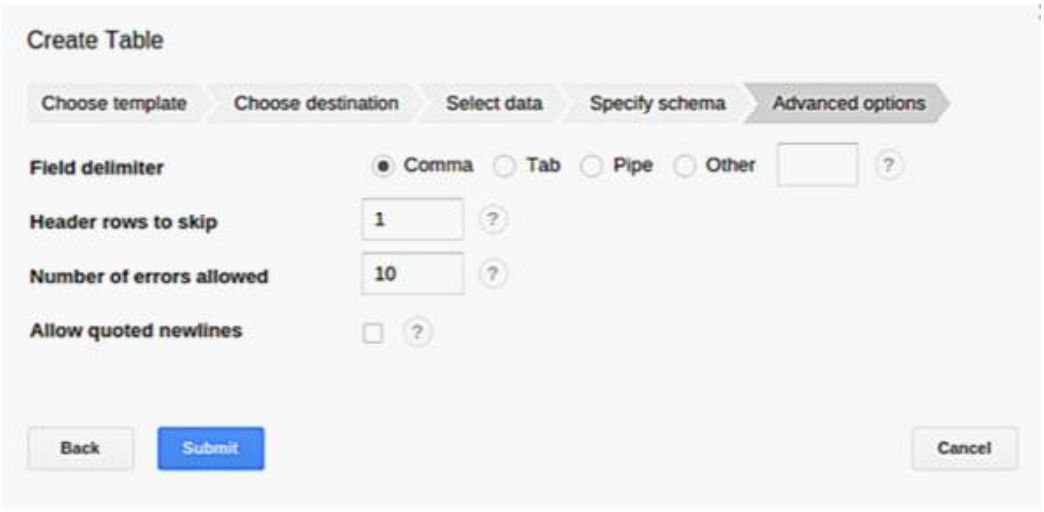

Gambar 9.20: *Screenshot BigQuery dashboard – Pembuatan tabel - step 2*

Pada langkah selanjutnya, skema tabel ditentukan seperti yang ditunjukkan pada Gambar 9.21.




Gambar 9.21: *Screenshot BigQuery dashboard – Pembuatan tabel - step 3*

Pada langkah terakhir, pembatas lapangan yang digunakan dalam file dataset, jumlah baris header untuk dilewati dan opsi lanjutan lainnya ditentukan seperti yang ditunjukkan pada Gambar 9.22.




Gambar 9.22: *Screenshot BigQuery dashboard – Pembuatan tabel - step 4*

Dengan data yang dimuat ke dalam tabel BigQuery, detail tabel tersebut dapat dilihat dari dasbor BigQuery seperti yang ditunjukkan pada Gambar 9.23.

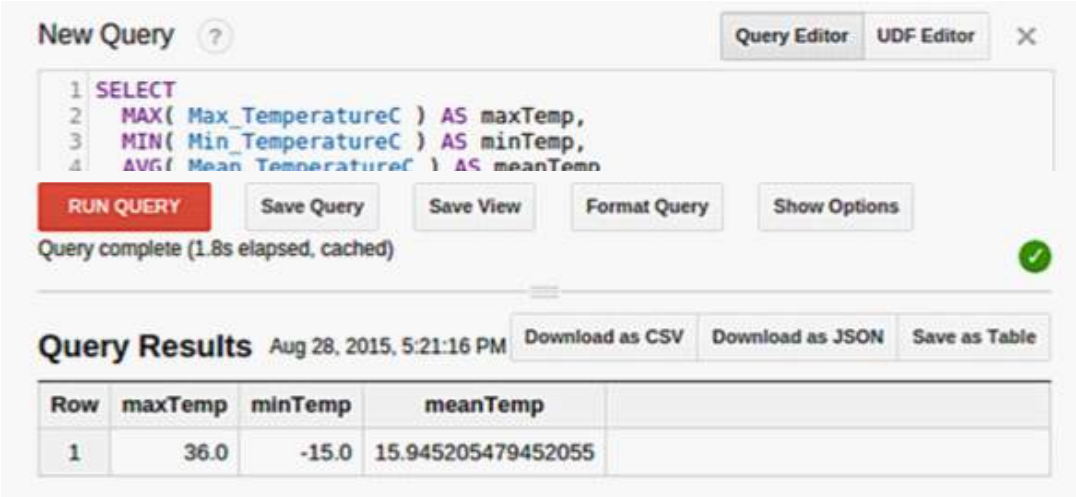

Gambar 9.23: *Screenshot BigQuery dashboard menunjukkan detai tabel*

Sekarang mari kita lihat contoh beberapa kueri SQL yang dapat dieksekusi dari dasbor BigQuery. Gambar 9.24 menunjukkan kueri SQL untuk mengambil sepuluh rekaman dari tabel. Output query juga dapat dilihat pada gambar.

| | | | | | | |
|---|---|---|---|---|---|---|
| 6 | 2014-1-6 | 8.0 | -2.0 | -12.0 | 7.0 | -8.0 |
| 7 | 2014-1-7 | -4.0 | -9.0 | -15.0 | -16.0 | -20.0 |
| 8 | 2014-1-8 | 8.0 | -1.0 | -10.0 | -11.0 | -16.0 |
| 9 | 2014-1-9 | 8.0 | 3.0 | -3.0 | 2.0 | -7.0 |
| 10 | 2014-1-10 | 8.0 | 6.0 | 3.0 | 8.0 | 5.0 |

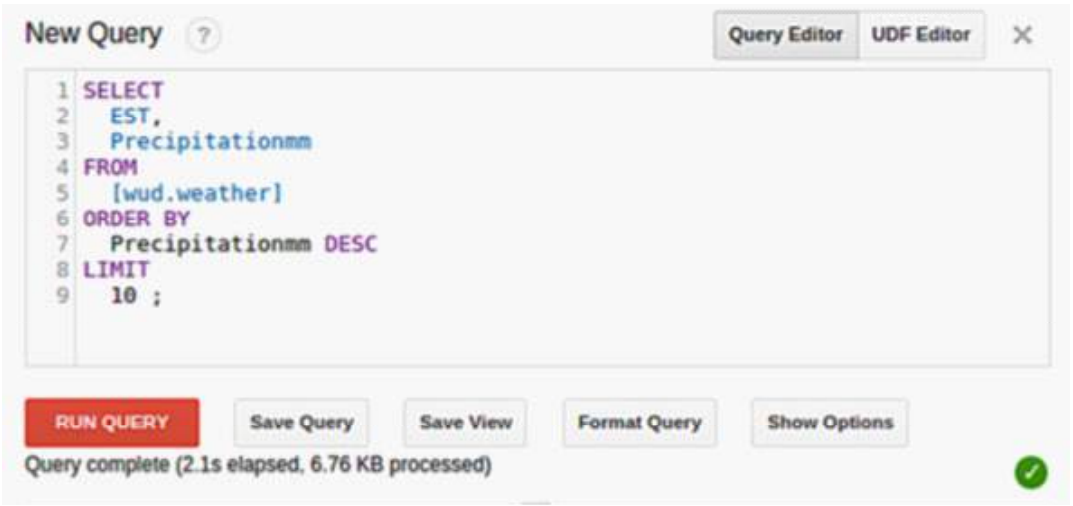
Gambar 9.24: *Screenshot dasbor BigQuery yang menampilkan hasil kueri*

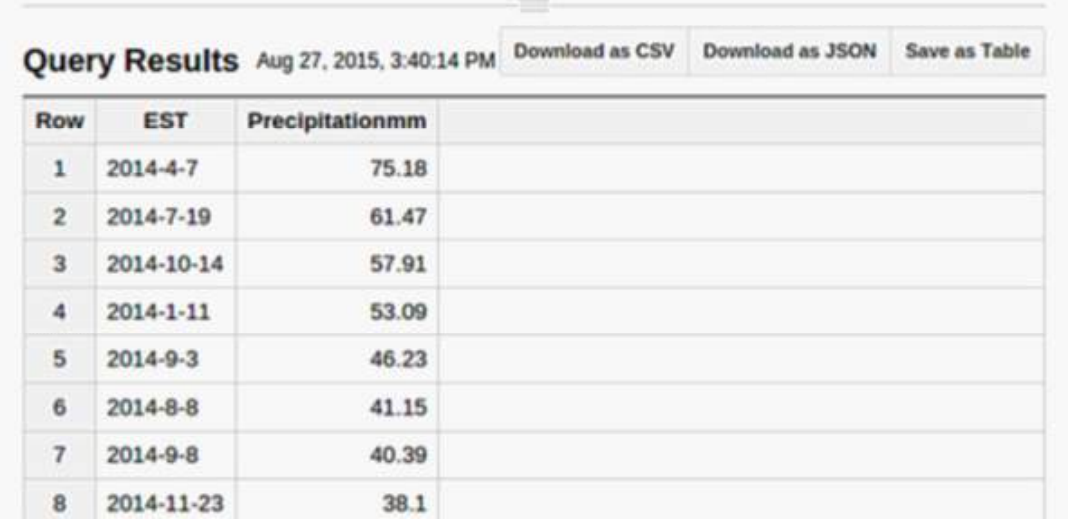
Gambar 9.25 menunjukkan query SQL untuk menemukan suhu maksimum, minimum, dan rata-rata sepanjang tahun.




Gambar 9.25: *Screenshot BigQuery dashboard Menampilkan hasil kueri*

Gambar 9.25 menunjukkan kueri SQL untuk menemukan sepuluh hari paling basah dalam setahun, diurutkan berdasarkan curah hujan.





**Query Results** Aug 27, 2015, 3:40:14 PM | Download as CSV | Download as JSON | Save as Table

| Row | EST | Precipitationmm |
|---|---|---|
| 1 | 2014-4-7 | 75.18 |
| 2 | 2014-7-19 | 61.47 |
| 3 | 2014-10-14 | 57.91 |
| 4 | 2014-1-11 | 53.09 |
| 5 | 2014-9-3 | 46.23 |
| 6 | 2014-8-8 | 41.15 |
| 7 | 2014-9-8 | 40.39 |
| 8 | 2014-11-23 | 38.1 |

Gambar 9.26: *Screenshot BigQuery dashboard showing hasil queri*

Sampai sekarang kita melihat contoh BigQuery yang menggunakan interface web BigQuery. BigQuery juga menyediakan API untuk membuat kumpulan data dan membuat kueri data. Box 9.7 menunjukkan program Python untuk membuat kumpulan data BigQuery. Metode sisipan jobs () di Google BigQuery API digunakan untuk memasukkan kumpulan data baru. Badan permintaan metode ini berisi properti seperti konfigurasi, beban, dan skema. Dalam contoh ini, data dimuat dari file CSV. Properti skema menentukan skema file CSV. Metode jobs (). Insert segera dikembalikan, oleh karena itu, jobs.get dipanggil untuk mendapatkan status pekerjaan.

**Box 9.7: Python program for creating a BigQuery dataset**

```
import json
import uuid
from googleapiclient.discovery import build
from oauth2client.client import GoogleCredentials

credentials = GoogleCredentials.get_application_default()
bigquery_service = build('bigquery', 'v2', credentials=credentials)

PROJECT_ID = 'mycloud'
DATASET_ID = 'wud'
TABLE_ID = 'weather'

job_data = {
  'jobReference': {
    'projectId': PROJECT_ID,
    'job_id': str(uuid.uuid4())
  },
```

```
'configuration': {
  'load': {
    'sourceUris': ['gs://abahqa/wud.csv'],
    'schema': {
      'fields': [
        {
        'name': 'EST',
        'type': 'STRING'
        },
        {
        'name': 'Max_TemperatureC',
        'type': 'FLOAT'
        },
        {
        'name': 'Mean_TemperatureC',
        'type': 'FLOAT'
        },
        {
        'name': 'Min_TemperatureC',
        'type': 'FLOAT'
        },
        {
        'name': 'Dew_PointC',
        'type': 'FLOAT'
        },
        {
        'name': 'MeanDew_PointC',
        'type': 'FLOAT'
        },
        {
        'name': 'Max_Humidity',
        'type': 'FLOAT'
        },
        {
        'name': 'Mean_Humidity',
        'type': 'FLOAT'
        },
        {
        'name': 'Min_Humidity',
        'type': 'FLOAT'
        },
        {
        'name': 'Max_Sea_Level_PressurehPa',
        'type': 'FLOAT'
        },
        {
        'name': 'Mean_Sea_Level_PressurehPa',
        'type': 'FLOAT'
        },
        {
        'name': 'Min_Sea_Level_PressurehPa',
        'type': 'FLOAT'
        },
        {
        'name': 'Max_VisibilityKm',
        'type': 'FLOAT'
        },
        {
```

```
            'name':   'Mean_VisibilityKm',
            'type':   'FLOAT'
            },
            {
            'name':   'Min_VisibilitykM',
            'type':   'FLOAT'
            },
            {
            'name':   'Max_Wind_SpeedKmph',
            'type':   'FLOAT'
            },
            {
            'name':   'Mean_Wind_SpeedKmph',
            'type':   'FLOAT'
            },
            {
            'name':   'Max_Gust_SpeedKmph',
            'type':   'FLOAT'
            },
            {
            'name':   'Precipitationmm',
            'type':   'FLOAT'
            },
            {
            'name':   'CloudCover',
            'type':   'FLOAT'
            },
            {
            'name':   'Events',
            'type':   'FLOAT'
            },
            {
            'name':   'WindDirDegrees',
            'type':   'FLOAT'
            }

          ]
        },
        'destinationTable':  {
          'projectId':  PROJECT_ID,
          'datasetId':  DATASET_ID,
          'tableId':  TABLE_ID
        }
      }
    }
}

insertResponse = bigquery_service.jobs().insert(projectId=PROJECT_ID,
   body=job_data).execute(num_retries=5)

print insertResponse

while True:
  job = bigquery_service.jobs().get(projectId=PROJECT_ID,
  jobId=insertResponse['jobReference']['jobId']).execute()
  print job['status']['state']
  if 'DONE' == job['status']['state']:
```

```
    print 'Done!'
    break

  print 'Loading data...'
  time.sleep(10)
```

Box 9.8 menunjukkan program Python untuk membuat kueri dataset dengan BigQuery. Metode kueri jobs (). Dari Google BigQuery API digunakan untuk membuat kueri dataset. Metode ini menjalankan kueri BigQuery SQL secara sinkron dan menampilkan hasil kueri.

**■ Box 9.8: Python program for querying a dataset with BigQuery**

```
import json
import uuid
from googleapiclient.discovery import build
from oauth2client.client import GoogleCredentials
```

```
credentials = GoogleCredentials.get_application_default()
bigquery_service = build('bigquery', 'v2', credentials=credentials)

PROJECT_ID = 'mycloud'

query_data = {'query':'SELECT EST, Precipitationmm
FROM [wud.weather] ORDER BY Precipitationmm DESC LIMIT 10 ;'}

query_response = bigquery_service.jobs().query(projectId=PROJECT_ID,
body=query_data).execute()

print query_response

print 'Query Results:'
for row in query_response['rows']:
  result_row = []
  for field in row['f']:
    result_row.append(field['v'])
  print ('').join(result_row)
```

## Ringkasan

Dalam bab ini, kami menjelaskan alat dan kerangka kerja untuk kueri interaktif dari big data termasuk Spark SQL, Hive, Google BigQuery, dan Amazon RedShift, bersama dengan contoh pembuatan kueri. Spark SQL adalah komponen Spark yang memungkinkan pembuatan kueri interaktif. Spark SQL berguna untuk membuat kueri data terstruktur dan semi-terstruktur menggunakan kueri seperti SQL. Hive adalah kerangka kerja data warehousing yang dibangun di atas Hadoop. Hive menyediakan bahasa kueri seperti SQL

yang disebut Hive Query Language, untuk membuat kueri data yang berada di HDFS. Amazon Redshift adalah layanan gudang data terkelola skala besar dan cepat. Redshift berspesialisasi dalam menangani kueri pada kumpulan data dengan ukuran hingga satu mapbyte atau lebih melalui penggunaan arsitektur Massively Parallel Processing (MPP), yang memparalelkan kueri SQL di semua sumber daya di kluster Redshift. Google BigQuery memungkinkan kueri dataset menggunakan kueri seperti SQL. Kueri BigQuery dijalankan pada tabel khusus tambahan yang menggunakan kekuatan pemrosesan infrastruktur Google untuk mempercepat kueri.

# Serving Database & Web Framework Bab 10

Bab ini mencakup :

- Serving Databases - SQL and NoSQL
- MySQL
- DynamoDB
- MongoDB
- Cassandra
- Python Web Framework – Django

Di bagian pertama bab ini, kami menjelaskan berbagai opsi untuk melayani database untuk aplikasi big data. Perbandingan database NoSQL relasional dan non-relasional untuk tujuan ini disediakan. Contoh implementasi berbagai database juga disediakan. Di bagian kedua dari bab ini, kami menjelaskan kerangka kerja web Django Python yang dapat digunakan untuk mengembangkan aplikasi web guna menyajikan hasil analisis kepada pengguna.

## 10.1 Relational (SQL) Databases

Database relasional adalah database yang sesuai dengan model relasional yang dipopulerkan oleh IBM Edgar Codd pada tahun 1970 [34]. Database relasional memiliki kumpulan relasi (atau tabel). Relasi adalah sekumpulan tupel (atau baris). Setiap relasi memiliki skema tetap yang mendefinisikan himpunan atribut (atau kolom dalam tabel) dan batasan pada atribut. Setiap tupel dalam suatu relasi memiliki atribut (kolom) yang sama. Tupel dalam suatu relasi dapat memiliki urutan apa pun dan relasinya tidak peka terhadap urutan tupel. Setiap atribut memiliki domain, yang merupakan kumpulan nilai yang memungkinkan untuk atribut tersebut. Hubungan dapat diubah dengan menggunakan operasi penyisipan, pembaruan dan penghapusan. Setiap relasi memiliki kunci primer yang secara unik mengidentifikasi setiap tupel dalam relasi tersebut. Atribut dapat dijadikan kunci utama jika tidak memiliki nilai berulang dalam tupel yang berbeda. Artinya, tidak ada dua tupel yang dapat memiliki nilai yang sama untuk atribut kunci primer.

Database relasional memiliki berbagai kendala yang dijelaskan sebagai berikut:

- Domain Constraint: Domain Constraint membatasi domain dari setiap atribut atau kumpulan nilai yang mungkin untuk atribut tersebut. Batasan domain menentukan bahwa nilai setiap atribut harus berupa nilai dari domain atribut.
- Batasan Integritas Entitas: Batasan integritas entitas menyatakan bahwa tidak ada key-value utama yang boleh nol. Karena kunci primer digunakan untuk mengidentifikasi secara unik setiap tupel dalam suatu relasi, memiliki nilai null untuk key-value primer akan membuat identifikasi tupel tidak mungkin dilakukan dalam relasi.
- Batasan Integritas Referensial: Batasan integritas referensial diperlukan untuk menjaga konsistensi antara tupel dalam dua relasi. Integritas referensial mengharuskan setiap nilai dari satu atribut relasi ada sebagai nilai atribut lain di relasi lain. Dengan kata lain, tupel dalam relasi yang mengacu pada relasi lain harus mengacu pada tupel yang ada pada relasi lain tersebut.
- Foreign Key: Untuk referensi silang antara beberapa relasi, kunci asing digunakan. Kunci asing adalah kunci dalam relasi yang cocok dengan kunci primer relasi lain.
- Database relasional mendukung setidaknya satu sub-bahasa komprehensif, yang paling populer adalah Structured Query Language (SQL). Database relasional memberikan jaminan ACID yang merupakan sekumpulan properti yang menjamin bahwa transaksi database diproses dengan andal, yang dijelaskan sebagai berikut:
- Atomicity: Properti atomicity memastikan bahwa setiap transaksi adalah "semua atau tidak sama sekali". Dengan kata lain, transaksi atom memastikan bahwa semua bagian dari transaksi selesai atau status database tidak berubah. Transaksi yang diselesaikan sebagian jika terjadi pemadaman sistem dapat menyebabkan status tidak valid. Atomicity memastikan bahwa transaksi tidak dapat dibagi dan dilakukan atau dibatalkan.
- Konsistensi: Properti konsistensi memastikan bahwa setiap transaksi membawa database dari satu status valid ke status lain. Dengan kata lain, data dalam database selalu sesuai dengan skema dan batasan yang ditentukan.

| Pro | Kontra |
|---|---|
| Model konsistensi yang terdefinisi dengan baik. Sebuah aplikasi yang berjalan pada database relasional (misalnya Server Microsoft SQL). Model yang mendasari tetap tidak berubah | Kinerja adalah kendala utama untuk database relasional. Performa tergantung pada jumlah relasi dan ukuran relasi. Sulit untuk menskalakan penerapan database relasional. |
| Memberikan jaminan ACID | Dukungan terbatas untuk struktur data yang kompleks. Misalnya. Jika data secara alami diatur dalam cara hierarkis dan disimpan sedemikian rupa, pendekatan hierarki dapat memungkinkan analisis data yang cepat. |

| | |
|---|---|
| Integritas relasional dipelihara melalui batasan integritas entitas dan referensial. | Pengetahuan lengkap tentang struktur database diperlukan untuk membuat kueri ad hoc. |
| Sangat cocok untuk aplikasi Pemrosesan Transaksi Online (OLTP). | Kebanyakan sistem database relasi mahal. |
| Landasan teori yang kuat (berdasarkan model relasional) yang telah dicoba dan diuji selama beberapa tahun. Tersedia database yang stabil dan terstandarisasi. | Beberapa data yang bersifat relasional membatasi ukuran bidang. |
| Desain database dan langkah-langkah normalisasi didefinisikan dengan baik dan struktur yang mendasari dipahami dengan baik | Mengintegrasikan data dari beberapa sistem database relasional bisa jadi merepotkan. |

Tabel 10.1: *Pro dan Kontra dari database relasional*

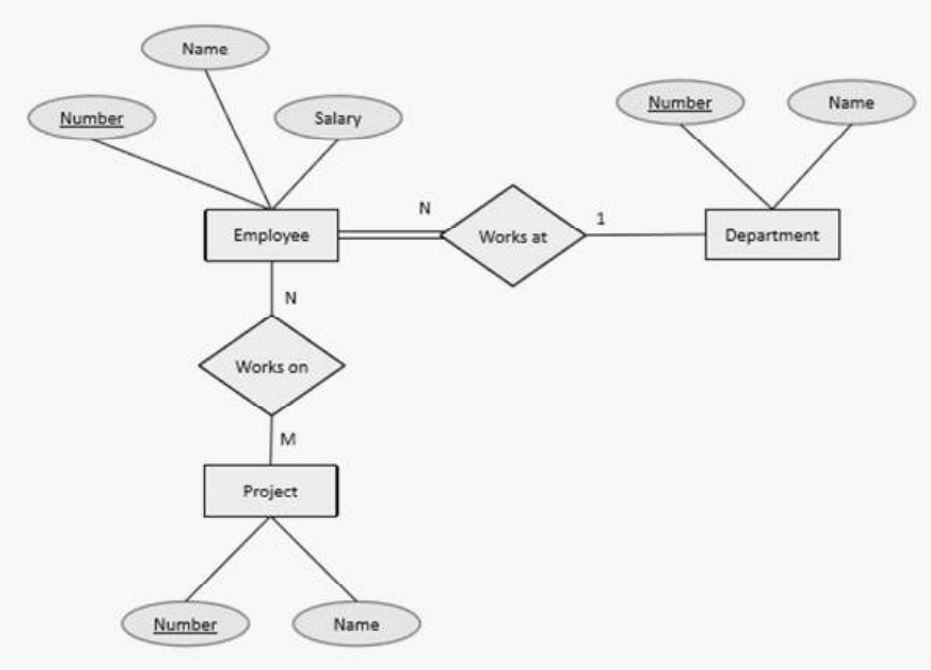
- Isolation: Isolation property memastikan bahwa status database yang diperoleh setelah serangkaian transaksi bersamaan adalah sama seperti jika transaksi dijalankan secara serial. Ini memberikan kontrol konkurensi, yaitu hasil transaksi yang tidak lengkap tidak terlihat oleh transaksi lain. Transaksi diisolasi satu sama lain sampai selesai.
- Durability: Durabilitypropertyensurest thatonceatransactioniscommitted, the data tetap apa adanya, yaitu tidak terpengaruh oleh pemadaman sistem seperti kehilangan daya. Durability menjamin bahwa database dapat melacak perubahan dan dapat pulih dari penghentian yang tidak normal.

Tabel 10.1 mencantumkan beberapa pro dan kontra dari database relasional.

### 10.1.1 My SQL

MySQL adalah Relational Database Management System (RDBMS) open source. MySQL adalah salah satu RDBMS yang paling banyak digunakan dan pilihan yang baik untuk menjadi database penyajian untuk aplikasi analisis data yang datanya terstruktur.

Mari kita lihat contoh penggunaan MySQL untuk aplikasi referensi yang menyimpan catatan karyawan di sebuah perusahaan. Gambar 10.1 menunjukkan diagram Entity-relationship (ER) untuk aplikasi referensi yang secara grafis merepresentasikan entitas dan hubungannya satu sama lain. Diagram ER menunjukkan tiga entitas-Karyawan, Departemen dan Proyek. Perhatikan bahwa hubungan satu-ke-satu ada antara Karyawan dan entitas Departemen, sedangkan hubungan banyak ke banyak ada antara Karyawan dan Proyek.




Gambar 10.1: *Diagram hubungan-entitas (ER) untuk aplikasi referensi*

Untuk memetakan model ER yang direpresentasikan dalam diagram ER ke model relasi, kita mengikuti aturan berikut:

Untuk setiap entitas reguler dalam model ER, buat relasi (tabel).

- Jadikan atribut entitas sebagai atribut tabel (atau kolom dalam tabel). Pilih salah satu atribut kunci entitas sebagai kunci utama untuk relasi tersebut.
- Untuk hubungan 1: 1 antara dua entitas (katakanlah PandQ), sertakan sebagai kunci asing di salah satu relasi, katakan relasi untuk entitas P, kunci utama dari relasi Q lainnya.
- Untuk relasi 1: N antara dua entitas (katakanlah R dan S), sertakan sebagai kunci asing dalam relasi untuk entitas S (di mana S adalah entitas di sisi N relasi), kunci primer relasi untuk entitas R.
- Untuk relasi M: N antara dua entitas (katakanlah U dan V), buat relasi baru dan dalam relasi tersebut masukkan sebagai foreign key, kunci primer dari relasi untuk entitas U dan V. Jika relasi M: N memiliki nilai sederhana atribut, termasuk itu juga di relasi baru.

Mengikuti aturan di atas, kami menghasilkan relasi (tabel) dan atributnya (kolom dalam tabel). Box 10.1 menunjukkan pernyataan SQL 'DAPAT DIBUAT' untuk membuat tabel

**Box 10.1: SQL statements for creating tables**

```
CREATE TABLE department(
number varchar(50) NOT NULL PRIMARY KEY,
name varchar(200) NULL
);

CREATE TABLE employee (
number varchar(100) NOT NULL PRIMARY KEY,
name varchar(100) NOT NULL,
salary varchar(20) NOT NULL,
department_id varchar(20) REFERENCES department (number),
);

CREATE TABLE project (
number varchar(20) NOT NULL PRIMARY KEY,
name varchar(100) NOT NULL
);

CREATE TABLE workson (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
employee_id varchar(20) NOT NULL REFERENCES employee (number),
project_id varchar(20) NOT NULL REFERENCES project (number)
);
```

Setelah membuat tabel, data dapat dimasukkan ke dalam tabel menggunakan pernyataan SQL 'INSERT INTO' seperti yang ditunjukkan di Box 10.2.

**Box 10.2: SQL statements for inserting data into tables**

```
INSERT INTO department VALUES ("1001", "ECE");

INSERT INTO employee (number, name,
salary,department_id) VALUES ("5001",
"Alex", "50000", "1001");

INSERT INTO project VALUES ("201", "Cloud");

INSERT INTO workson(employee_id,project_id)
VALUES ("5001", "201");
```

Akhirnya, data dapat ditanyakan menggunakan pernyataan SELECT seperti yang ditunjukkan pada Box 10.3.

**Box 10.3: SQL statements for querying tables**

```
# Retrieve all employees
SELECT * FROM employee;

# Retrieve top 3 employees with highest salary
SELECT * FROM employee ORDER BY salary DESC LIMIT 3;
```

```
# Retrieve all employees in department 'ECE'
SELECT e.name, e.number, d.name FROM
employee e, department d WHERE d.name='ECE' ;

# Count the number of employees working on 'IoT' project
SELECT COUNT(*) FROM project p, workson w WHERE p.name='IoT' ;
```

## 10.2 Database Non-Relational (NoSQL)

Di Bab-4, kami menjelaskan berbagai jenis database Non-Relasional (NoSQL). Tidak seperti database relasional, database NoSQL tidak memberikan jaminan ACID. Sebagian besar database NoSQL menawarkan konsistensi 'akhirnya', yang berarti bahwa dengan periode waktu yang cukup lama di mana tidak ada pembaruan yang dilakukan, semua pembaruan dapat diharapkan untuk menyebar pada akhirnya melalui sistem dan replika akan konsisten. Beberapa penulis merujuk pada istilah jaminan BASE (Pada dasarnya Tersedia, Soft state, Konsistensi akhir) untuk database NoSQL sebagai lawan jaminan ACID yang disediakan oleh database relasional.

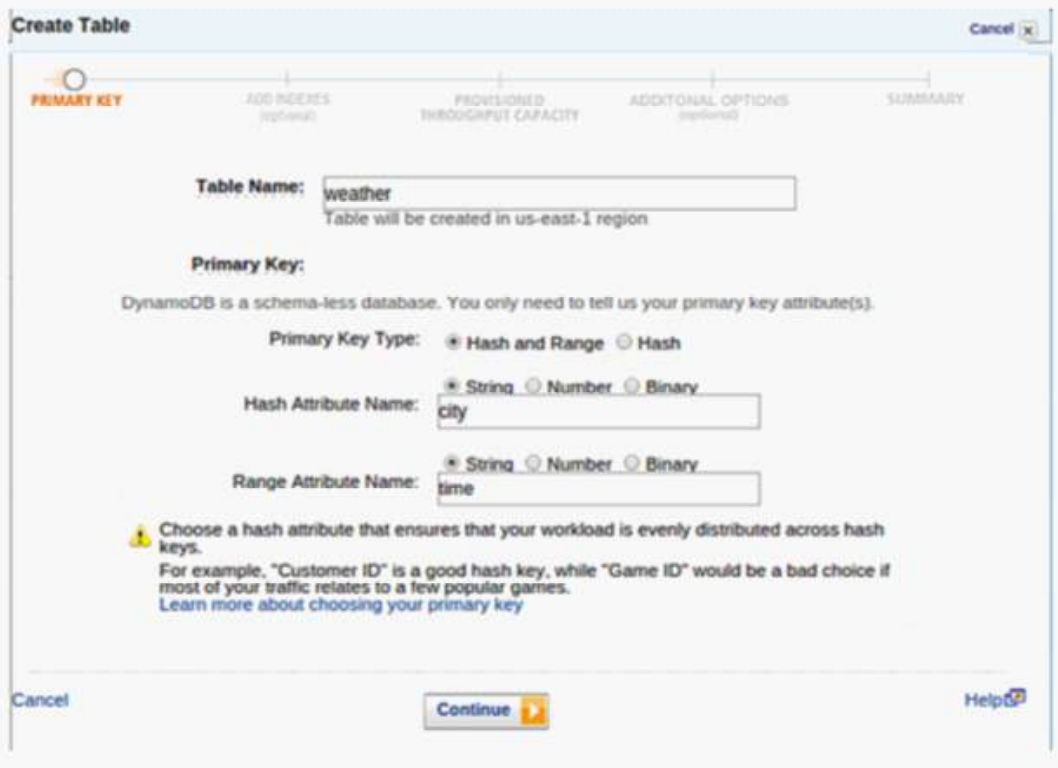
| Pro | Kontra |
|---|---|
| Mudah untuk diperkecil. Kinerja yang lebih tinggi untuk data berskala besar dibandingkan dengan database relasional. Memungkinkan berbagi data di beberapa server. | Tidak memberikan jaminan ACID, oleh karena itu kurang cocok untuk aplikasi seperti pemrosesan transaksi yang membutuhkan konsistensi yang kuat. |
| Sebagian besar solusi bersifat open-source atau lebih murah dibandingkan dengan database relasional. | Tidak ada skema tetap. Tidak ada model penyimpanan data yang umum. Solusi yang berbeda memiliki model penyimpanan data yang berbeda. |
| Ketersediaan tinggi dan toleransi kesalahan disediakan oleh replikasi data. | Dukungan terbatas untuk agregasi (SUM, AVG, COUNT, GROUP BY) dibandingkan dengan database relasional. |
| Mendukung struktur data kompleks dan objek pemrograman asli. | Kinerja untuk gabungan yang kompleks buruk dibandingkan dengan database relasional |
| Tidak ada skema tetap. Mendukung data tidak terstruktur. | Tidak ada pendekatan yang didefinisikan dengan baik untuk desain database, karena solusi yang berbeda memiliki model penyimpanan data yang berbeda. |
| Pengambilan data sangat cepat. Cocok untuk aplikasi real-time. | Kurangnya model yang konsisten dapat menyebabkan solusi yang terkunci, yaitu, migrasi dari satu solusi ke solusi lain mungkin memerlukan pemodelan ulang aplikasi yang signifikan. |
| Sebagian besar solusi memberikan dukungan untuk model pemrograman MapReduce untuk memproses data skala besar. | |

Table 10.2: *Pro dan Kontra pada database non-relational*

Kekuatan pendorong di balik database NoSQL adalah kebutuhan database yang dapat mencapai ukuran terkait kinerja dari skalabilitas tinggi, toleransi kesalahan, dan ketersediaan. Database ini dapat didistribusikan pada sejumlah besar mesin. Toleransi kesalahan disediakan dengan menyimpan beberapa replika data pada mesin yang berbeda. Misalnya, dengan faktor replikasi yang disetel sama dengan N untuk database NoSQL, setiap record memiliki N replika pada mesin yang berbeda. Tabel 10.2 mencantumkan beberapa pro dan kontra dari database non-relasional.

### 10.2.1 Amazon DynamoDB

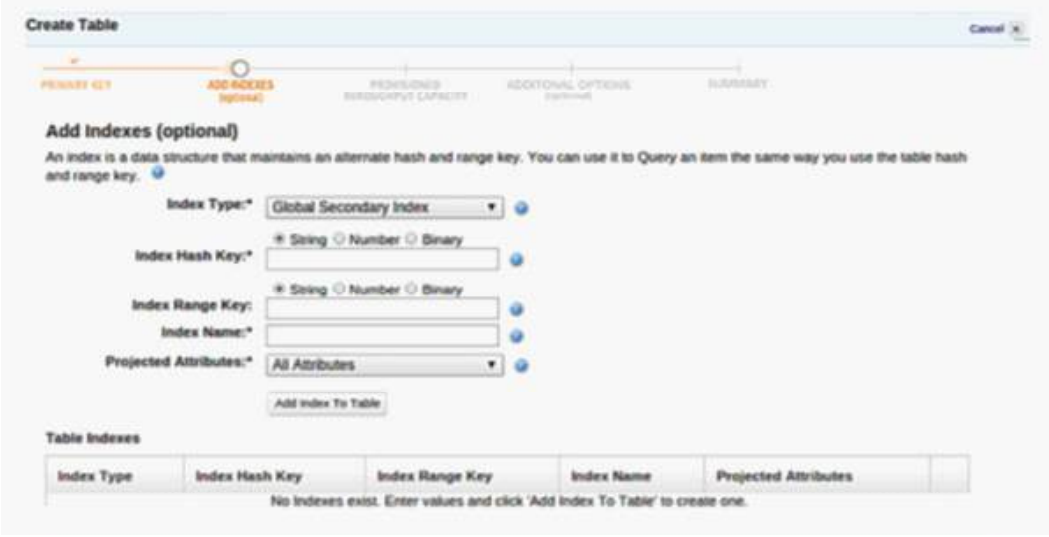
Kami menjelaskan Amazon DynamoDB di Bab-4. Di bagian ini, kami akan menjelaskan contoh penggunaan DynamoDB untuk aplikasi referensi yang mengambil dan menyimpan data cuaca untuk berbagai kota. Langkah pertama adalah membuat tabel DynamoDB dari dashboard DynamoDB seperti yang ditunjukkan Gambar 10.2. Tabel DynamoDB dapat memiliki kunci utama sederhana yang terdiri dari atribut hash (atau kunci partisi) atau kunci utama racomposite yang terdiri dari atribut hash dan atribut jangkauan (atau kunci urutkan). Atribut hash digunakan untuk membuat indeks hash tak berurutan dan atribut range digunakan untuk membuat indeks rentang terurut.




Gambar 10.2: *Membuat tabel DynamoDB - step-1*

Pada langkah berikutnya, Anda dapat menentukan indeks tambahan seperti yang ditunjukkan pada Gambar 10.3. DynamoDB memberikan opsi untuk mendefinisikan satu atau lebih indeks sekunder di atas tabel. Dua jenis indeks sekunder didukung-indeks sekunder global dan indeks sekunder lokal. Indeks sekunder global dapat dibuat dengan atribut kunci hash dan range yang berbeda dari kunci primer. Sedangkan indeks sekunder lokal dapat dibuat dari atribut hash yang sama seperti untuk kunci utama tetapi atribut range yang berbeda.

Pada langkah berikutnya, Anda dapat menentukan kapasitas throughput yang disediakan seperti yang ditunjukkan pada Gambar 10.4. DynamoDB memberikan kinerja yang konsisten dan dapat diprediksi dengan memungkinkan pengguna untuk menentukan kapasitas throughput yang disediakan saat membuat atau memperbarui tabel. Throughput yang disediakan didefinisikan dalam istilah unit kapasitas baca dan tulis. Unit kapasitas baca adalah satu bacaan yang sangat konsisten atau dua bacaan yang pada akhirnya konsisten per detik untuk item berukuran 4 KB. Unit kapasitas tulis mewakili satu tulis per detik untuk item berukuran 1 KB. Misalnya, jika Anda ingin membaca 100 item per detik (setiap item berukuran maksimal 4 KB), kapasitas baca 100 harus disediakan. Untuk item yang lebih besar dari 4 KB, kapasitas baca dihitung dengan membulatkan ke kelipatan 4. Misalnya, untuk membaca 100 item per detik di mana setiap item berukuran 6 KB, kapasitas baca 200 harus disediakan. Demikian pula, untuk penulisan, jika item lebih besar dari 1 KB, kapasitas tulis dihitung dengan membulatkan ke 1 KB berikutnya. Pada langkah berikutnya, Anda dapat menentukan pemberitahuan alarm throughput untuk tabel seperti yang ditunjukkan pada Gambar 10.5.

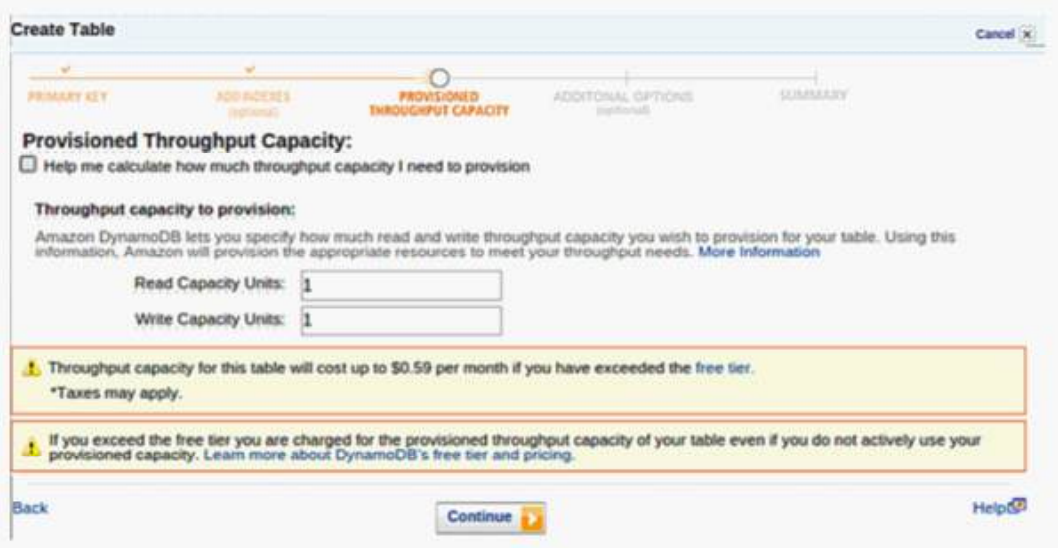



Gambar 10.3: *Membuat tabel DynamoDB - step-2*

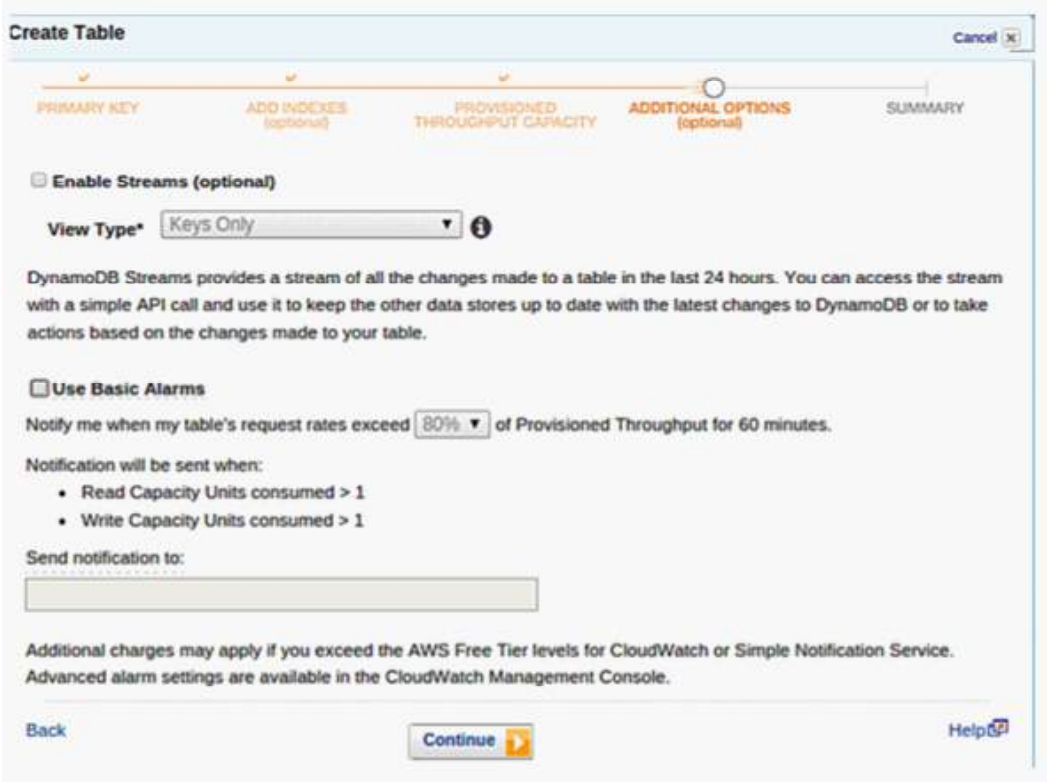
Untuk aplikasi referensi data cuaca kita akan menggunakan pustaka PyOwm Python untuk mengambil data dari OpenWeatherMap (OWM) [35]. Open Weather Map adalah layanan online yang menyediakan API gratis untuk data cuaca. Box 10.4 menunjukkan kode Python untuk mengambil data cuaca terkini dan ramalan cuaca untuk daftar kota dan menulis data ke tabel DynamoDB. Dalam contoh ini, koneksi pertama kali dibuat dengan layanan DynamoDB dan pegangan ke tabel yang ada diambil. Untuk menulis data ke DynamoDB, fungsi thetable.put_item digunakan.

**Box 10.4: Python program for retrieving weather data and writing to DynamoDB**

```
import pyowm
import boto.dynamodb2
from boto.dynamodb2.table import Table
import time
from datetime import date
import datetime
```




Gambar 10.4: *Membuat tabel DynamoDB - step-3*




Gambar 10.5: *Membuat tabel DynamoDB - step-4*

```
import cPickle

PYOWM_KEY='<enter>'
owm = pyowm.OWM(PYOWM_KEY)

REGION="us-east-1"

print "Connecting to DynamoDB"

conn = boto.dynamodb2.connect_to_region(REGION,
aws_access_key_id='<enter>',
aws_secret_access_key='<enter>')


table=Table('weather',connection=conn)


places=['New York,US','Los Angeles,US',
 'Chicago,US','Houston,US', 'Philadelphia,US',
 'Phoenix,US', 'San Antonio,US','San Diego,US',
 'Dallas,US','San Jose,US']

for place in places:
 print place
 observation = owm.weather_at_place(place)
 w= observation.get_weather()
```

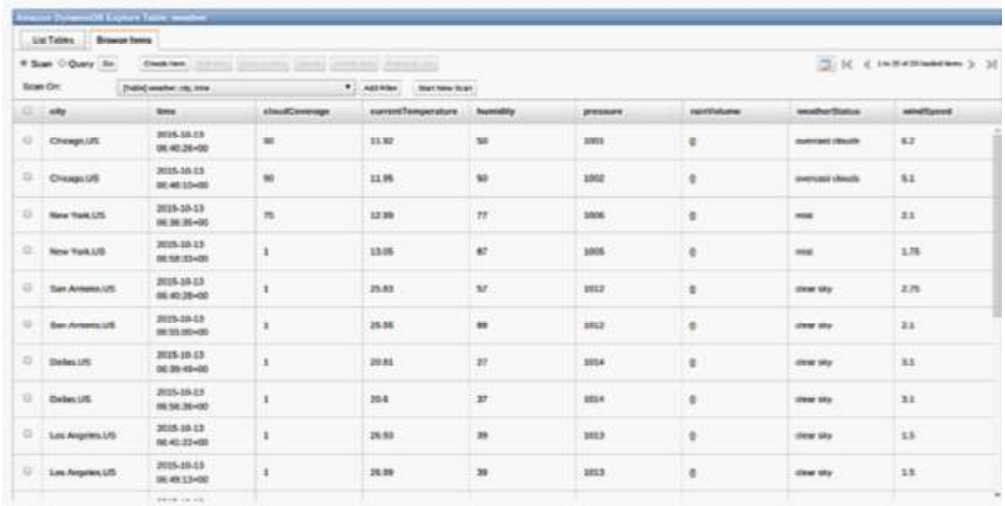
```
item = table.put_item(data={
  'city':place,
  'time': str(w.get_reference_time(timeformat='iso')),
  'currentTemperature': str(w.get_temperature(unit='celsius')['temp']),
  'weatherStatus': str(w.get_detailed_status()),
  'cloudCoverage': str(w.get_clouds()),
  'rainVolume': str(w.get_rain()),
  'windSpeed': str(w.get_wind()['speed']),
  'humidity': str(w.get_humidity()),
  'pressure': str(w.get_pressure()['press'])
})


forecastTable=Table('forecast',connection=conn)

for place in places:
 print place
 fc = owm.daily_forecast(place, limit=7)
 f = fc.get_forecast()
 forecast_list=[]
 for w in f.get_weathers():
  forecast_dict={}
  t=w.get_reference_time()
  forecast_dict['date'] =
  datetime.datetime.fromtimestamp(t).strftime('%Y-%m-%d')
  forecast_dict['tempMin']= w.get_temperature(unit='celsius')['min']
  forecast_dict['tempMax']= w.get_temperature(unit='celsius')['max']

  rain= w.get_rain()
  if rain.has_key('all'):
   forecast_dict['rain'] = rain['all']
  else:
   forecast_dict['rain'] = 0

  forecast_dict['status']= w.get_detailed_status()
  forecast_list.append(forecast_dict)

 item = forecastTable.put_item(data={
  'city':place,
  'forecastList': cPickle.dumps(forecast_list)
 },overwrite=True)
```

Box 10.5 menunjukkan contoh Python untuk membaca data dari DynamoDB. DynamoDB mendukung dua jenis operasi baca-QueryandScan. Operasi kueri memungkinkan Anda meminta tabel dengan memberikan nama dan nilai atribut kunci hash. Selain itu, atribut kunci rentang dan nilai dapat ditentukan bersama dengan operator pembanding untuk memperhalus hasil. Operasi Pindai dapat digunakan untuk membaca semua item dalam tabel.

**Box 10.5: Python program for reading data from DynamoDB**

```
import boto.dynamodb2
from boto.dynamodb2.table import Table

REGION="us-east-1"

conn = boto.dynamodb2.connect_to_region(REGION,
aws_access_key_id='<enter>',
aws_secret_access_key='<enter>')

table=Table('weather',connection=conn)

#Scan table
all_items=table.scan()

for item in all_items:
 print item.items()

#Query for a particular city
results = table.query_2(city__eq='New York,US')

for r in results:
 print r['time']
 print r['currentTemperature']
 print r['weatherStatus']
 print r['pressure']
 print r['humidity']
 print r['windSpeed']
 print r['rainVolume']
 print r['cloudCoverage']
```

Gambar 10.6 dan 10.7 menunjukkan contoh pemindaian dan kueri tabel DynamoDB dari dasbor DynamoDB.



Gambar 10.6: *Scanning table dari Dashboard DynamoDB*

Gambar 10.7: *Querying table dari Dashboard DynamoDB*

### 10.2.2`Cassandra

Cassandra adalah sistem database non-relasional open source yang dapat diskalakan, sangat tersedia, dan toleran terhadap kesalahan. Cassandra memiliki arsitektur terdistribusi, terdesentralisasi, dan peer-to-peer di mana semua node memiliki peran yang sama. Data yang disimpan direplikasi di beberapa node dalam cluster dan kinerja database dapat diskalakan secara linier dengan penambahan node baru. Model data Cassandra mirip dengan model HBase yang dijelaskan pada Bab-4. Unit dasar penyimpanan data adalah Kolom yang memiliki nama, nilai dan cap waktu. Sebuah Baris memiliki banyak kolom dan diidentifikasikan dengan kunci baris yang unik. ColumnFamily memiliki beberapa baris dan dianalogikan dengan tabel dalam database relasional. Baris berbeda dalam ColumnFamily bisa memiliki kumpulan kolom berbeda, dan kolom bisa ditambahkan kapan saja. Keyspace di Cassandra dianalogikan dengan database dalam sistem database relasional. Keyspace berisi elemen operasional seperti strategi replikasi dan faktor replikasi.

Box 10.6 menunjukkan perintah untuk menyiapkan dan menjalankan Cassandra.

**■ Box 10.6: Setting up Cassandra**

```
#Download Cassandra distribution
wget http://www.eu.apache.org/dist/cassandra/2.1.10/
apache-cassandra-2.1.10-bin.tar.gz
tar -xzf apache-cassandra-2.1.10-bin.tar.gz
cd apache-cassandra-2.1.10

#Run Cassandra
bin/cassandra -f

#Run CQL shell
bin/cqlsh
```

Cassandra menyediakan bahasa kueri yang disebut Cassandra Query Language (CQL) yang mirip dengan SQL. Shell CQL dapat digunakan untuk berinteraksi dengan Cassandra.

Pada bagian ini, kami akan menjelaskan contoh penggunaan Cassandra untuk aplikasi data cuaca referensi. Box 10.7 menunjukkan pernyataan CQL untuk membuat tabel untuk menyimpan data cuaca terkini dan prakiraan cuaca.

**Box 10.7: Creating Cassandra tables**

```
#Create keyspace
CREATE KEYSPACE weatherkeyspace
WITH REPLICATION = { 'class' : 'SimpleStrategy', 'replication_factor' :
1 };

#Use keyspace
USE weatherkeyspace;

#Create tables
CREATE TABLE weathernew (
 city text,
 time text,
 currentTemperature float,
 weatherStatus text,
 cloudCoverage text,
 rainVolume text,
 windSpeed float,
 humidity float,
 pressure float,
 PRIMARY KEY (city, time)
);


CREATE TABLE forecast (
 city text,
 forecastList text,
 PRIMARY KEY (city)
);
```

Box 10.8 menunjukkan kode Python untuk mengambil data cuaca terkini dan ramalan cuaca untuk daftar kota dan menulis datanya ke tabel Cassandra.

**Box 10.8: Python program for retrieving weather data and writing to Cassandra**

```
import pyowm
import time
from datetime import date
import datetime
import cPickle
from cassandra.cluster import Cluster

PYOWM_KEY='<enter>'
owm = pyowm.OWM(PYOWM_KEY)

## create Cassandra instance
cluster = Cluster()

## establish Cassandra connection, using local default
session = cluster.connect('weatherkeyspace')
places=['New York,US','Los Angeles,US','Chicago,US',
'Houston,US', 'Philadelphia,US', 'Phoenix,US',
'San Antonio,US','San Diego,US','Dallas,US','San Jose,US']

for place in places:
 print place
 observation = owm.weather_at_place(place)
 w= observation.get_weather()

 rain= w.get_rain()
 if rain.has_key('all'):
  rainVolume = rain['all']
 else:
  rainVolume = 0


 statement="INSERT INTO weathernew (city, time,
  currentTemperature, weatherStatus, cloudCoverage,
  rainVolume, windSpeed, humidity, pressure) VALUES('" +
 str(place)+"','"+ str(w.get_reference_time(timeformat='iso'))+
 "',"+ str(w.get_temperature(unit='celsius')['temp'])+",'"+
 str(w.get_detailed_status())+"','"+ str(w.get_clouds())+"','"+
 str(rainVolume) +"',"+ str(w.get_wind()['speed'])+","+
 str(w.get_humidity())+","+ str(w.get_pressure()['press'])+")"

 session.execute(statement)

for place in places:
 print place
 fc = owm.daily_forecast(place, limit=7)
 f = fc.get_forecast()
 forecast_list=[]
 for w in f.get_weathers():
  forecast_dict={}
  t=w.get_reference_time()
  forecast_dict['date'] =
  datetime.datetime.fromtimestamp(t).strftime('%Y-%m-%d')
```

```
  forecast_dict['tempMin']= w.get_temperature(unit='celsius')['min']
  forecast_dict['tempMax']= w.get_temperature(unit='celsius')['max']

  rain= w.get_rain()
  if rain.has_key('all'):
   forecast_dict['rain'] = rain['all']
  else:
   forecast_dict['rain'] = 0
  forecast_dict['status']= w.get_detailed_status()
  forecast_list.append(forecast_dict)

 statement="INSERT INTO forecast (city, forecastList)
  VALUES('"+str(place)+"',"'"+
  cPickle.dumps(forecast_list).encode("hex") +"')"

 session.execute(statement)
```

Gambar 10.8 menunjukkan screenshot dari kueri yang dieksekusi di shell CQL.

```
cqlsh:weatherkeyspace> SELECT * FROM weather;

 city          | time                     | cloudcoverage | currenttemperature | humidity | pressure | rainvolume | weatherstatus          | windspeed
---------------+--------------------------+---------------+--------------------+----------+----------+------------+------------------------+-----------
     Dallas,US | 2015-10-16 08:23:01+0000 |            20 |              22.23 |       38 |     1020 |          0 |              few clouds |       2.1
     Dallas,US | 2015-10-16 09:59:56+0000 |             1 |              21.15 |       43 |     1022 |          0 |              clear sky |       7.2
    Phoenix,US | 2015-10-16 08:20:56+0000 |             1 |              28.79 |       17 |     1013 |          0 |              clear sky |       2.5
    Phoenix,US | 2015-10-16 09:56:45+0000 |             1 |              25.79 |       34 |     1013 |          0 |              clear sky |       1.5
  New York,US  | 2015-10-16 08:19:31+0000 |            75 |              10.41 |       71 |     1013 |          0 |            broken clouds |       3.6
  New York,US  | 2015-10-16 09:50:11+0000 |            40 |              10.43 |       71 |     1013 |          0 |          scattered clouds |       2.6
  San Diego,US | 2015-10-16 08:20:22+0000 |            90 |              21.84 |       88 |     1013 |          0 |                   mist |       2.6
  San Diego,US | 2015-10-16 09:55:20+0000 |            90 |              21.56 |       88 |     1013 |          0 | light intensity drizzle |       2.6
San Antonio,US | 2015-10-16 08:24:58+0000 |             1 |              17.64 |       77 |     1021 |          0 |              clear sky |       1.5
San Antonio,US | 2015-10-16 09:51:02+0000 |            40 |              17.07 |       82 |     1021 |          0 |          scattered clouds |       2.1
Los Angeles,US | 2015-10-16 08:25:13+0000 |            20 |              19.91 |       83 |     1013 |          0 |              few clouds |       2.6
Los Angeles,US | 2015-10-16 09:59:08+0000 |            75 |              19.92 |       88 |     1013 |          0 |                   mist |         1
    Chicago,US | 2015-10-16 08:21:29+0000 |             1 |               5.96 |       81 |     1021 |          0 |              clear sky |       3.6
    Chicago,US | 2015-10-16 09:54:23+0000 |             1 |               6.04 |       52 |     1022 |          0 |              clear sky |       3.1
     Houston,US | 2015-10-16 08:20:23+0000 |             1 |              17.04 |       88 |     1019 |          0 |                    fog |       1.4
     Houston,US | 2015-10-16 09:57:16+0000 |             1 |              16.33 |       88 |     1019 |          0 |                    fog |       1.4
Philadelphia,US | 2015-10-16 08:18:25+0000 |            20 |               8.84 |       87 |     1014 |          0 |              few clouds |      4.55
Philadelphia,US | 2015-10-16 09:52:39+0000 |             0 |               8.92 |       87 |     1014 |          0 |              clear sky |       1.5
    San Jose,US | 2015-10-16 08:14:35+0000 |             1 |              16.28 |       82 |     1014 |          0 |              clear sky |       2.6
    San Jose,US | 2015-10-16 09:49:21+0000 |             1 |              16.74 |       82 |     1015 |          0 |              clear sky |      0.75

(20 rows)
```

Gambar 10.8: *Tabel Querying Cassandra dari CQL shell*

### 10.2.3 MongoDB

Kami menjelaskan MongoDB di Bab-4. Pada bagian ini, kami akan menjelaskan contoh penggunaan MongoDB untuk aplikasi data cuaca referensi. MongoDB menyediakan shell yang dapat digunakan untuk melakukan berbagai operasi database. Box 10.9 menunjukkan beberapa contoh dari berbagai operasi database yang menggunakan shell MongoDB.

**Box 10.9: Using MongoDB shell commands**

```
#Launch MongoDB shell
mongo localhost:27017

#Switch to new database named weatherdb
> use weatherdb
switched to db weatherdb

#Insert a document
> post = {
 "city" :  "New York,US",
 "windSpeed" :  "5.7",
 "currentTemperature" :  "12.01",
 "time" :  "2015-10-23 09:15:00+00",
 "cloudCoverage" :  "20",
 "rainVolume" :  "{u'1h':  0.25}",
 "humidity" :  "54",
 "pressure" :  "1021",
 "weatherStatus" :  "light rain"
}
> db.collection.insert(post)
WriteResult({ "nInserted" :  1 })

# Retrieve all documents
> db.collection.find()
{ "_id" :  ObjectId("5629fee545c8be72b82fc60e"), "city" :  "New York,US",
"windSpeed" :  "5.7", "currentTemperature" :  "12.01",
"time" :  "2015-10-23 09:15:00+00", "cloudCoverage" :  "20",
"rainVolume" :  "{u'1h':  0.25}", "humidity" :  "54",
"pressure" :  "1021", "weatherStatus" :  "light rain" }

{ "_id" :  ObjectId("5629ff4d45c8be72b82fc60f"), "city" :  "Chicago,US",
"windSpeed" :  "4.6", "currentTemperature" :  "10.98",
"time" :  "2015-10-23 09:01:04+00", "cloudCoverage" :  "90",
"rainVolume" :  "{}", "humidity" :  "66", "pressure" :  "1024",
"weatherStatus" :  "overcast clouds" }

# Retrieve documents matching the query
> db.collection.find({"city" :  "New York,US"})
{ "_id" :  ObjectId("5629fee545c8be72b82fc60e"), "city" :  "New York,US",
"windSpeed" :  "5.7", "currentTemperature" :  "12.01",
"time" :  "2015-10-23 09:15:00+00", "cloudCoverage" :  "20",
"rainVolume" :  "{u'1h':  0.25}", "humidity" :  "54",
"pressure" :  "1021", "weatherStatus" :  "light rain" }

#Show current database name
> db
weatherdb

#Show collections in the database
> show collections
collection
system.indexes
```

Box 10.10 menunjukkan kode Python untuk mengambil data cuaca terkini dan ramalan cuaca untuk daftar kota dan menulis datanya ke MongoDB.

**Box 10.10: Python program for retrieving weather data and writing to MongoDB**

```
import pyowm
import time
from datetime import date
import datetime
import cPickle
from pymongo import MongoClient

PYOWM_KEY='<enter>'
owm = pyowm.OWM(PYOWM_KEY)

client = MongoClient('mongodb://root:password@hostname:53688/weather')
db = client['weather']
weather_collection = db['current']
forecast_collection = db['forecast']


places=['New York,US','Los Angeles,US',
'Chicago,US','Houston,US', 'Philadelphia,US',
'Phoenix,US', 'San Antonio,US','San Diego,US',
'Dallas,US','San Jose,US']

for place in places:
 print place
 observation = owm.weather_at_place(place)
 w= observation.get_weather()

 item = {
  'city':place,
  'time': str(w.get_reference_time(timeformat='iso')),
  'currentTemperature': str(w.get_temperature(unit='celsius')['temp']),
  'weatherStatus': str(w.get_detailed_status()),
  'cloudCoverage': str(w.get_clouds()),
  'rainVolume': str(w.get_rain()),
  'windSpeed': str(w.get_wind()['speed']),
  'humidity': str(w.get_humidity()),
  'pressure': str(w.get_pressure()['press'])
 }

 weather_collection.insert_one(item)


  forecast_dict={}
  t=w.get_reference_time()
  forecast_dict['date'] =
  datetime.datetime.fromtimestamp(t).strftime('%Y-%m-%d')
  forecast_dict['tempMin']= w.get_temperature(unit='celsius')['min']
  forecast_dict['tempMax']= w.get_temperature(unit='celsius')['max']
```

```
forecast_dict={}
t=w.get_reference_time()
forecast_dict['date'] =
datetime.datetime.fromtimestamp(t).strftime('%Y-%m-%d')
forecast_dict['tempMin']= w.get_temperature(unit='celsius')['min']
forecast_dict['tempMax']= w.get_temperature(unit='celsius')['max']

rain= w.get_rain()
if rain.has_key('all'):
 forecast_dict['rain'] = rain['all']
else:
 forecast_dict['rain'] = 0
forecast_dict['status']= w.get_detailed_status()
forecast_list.append(forecast_dict)

item = {
 'city':place,
 'forecastList':  forecast_list
}

forecast_collection.insert_one(item)
```

## 10.3 Python Web Application Framework – Django

Django adalah kerangka aplikasi web sumber terbuka untuk mengembangkan aplikasi web dengan Python [61]. Kerangka aplikasi web, secara umum, adalah kumpulan solusi, paket, dan praktik terbaik yang memungkinkan pengembangan aplikasi web dan situs web dinamis. Django didasarkan pada arsitektur Model-Tempelatee-View dan menyediakan pemisahan model data dari aturan bisnis dan interface pengguna. Django menyediakan API terpadu ke backend database. Jadi, aplikasi web yang dibangun dengan Django dapat bekerja dengan database berbeda tanpa memerlukan perubahan kode apapun. Django terdiri dari pemeta relasional objek, sistem tempelatee web, dan dispatcher URL berbasis ekspresi reguler.

Mengingat pemisahan model, tampilan dan tempelate, fleksibilitas untuk menggunakan database yang berbeda, dikombinasikan dengan kapabilitas yang kuat dari bahasa Python dan ekosistem Python, Django adalah salah satu kerangka aplikasi web yang paling cocok untuk analisis big data dan aplikasi awan.

### 10.3.1 Rancangan Django

Django mengadopsi arsitektur Model-Tempelatee-View (MTV). Peran model, tempelatee dan view adalah sebagai berikut:

***Model***

Model bertindak sebagai definisi dari data yang disimpan dan menangani interaksi dengan database. Model Django adalah kelas Python yang menguraikan variabel dan metode untuk tipe data tertentu

***Tempelate***

Dalam aplikasi web Django tipikal, tempelatenya hanyalah sebuah halaman HTML dengan beberapa placeholder tambahan. Bahasa cetakan Django dapat digunakan untuk membuat berbagai bentuk berkas teks (XML, email, CSS, Javascript, CSV, dll.)

***View***

Tampilan mengikat model ke tempelatee. View adalah tempat Anda menulis kode yang menghasilkan halaman web. View menentukan data apa yang akan ditampilkan, mengambil data dari database dan meneruskan data ke tempelatee. Untuk menjelaskan implementasi model, tempelate dan komponen tampilan dari aplikasi Django kita akan menggunakan aplikasi referensi yang memelihara catatan karyawan di perusahaan. Diagram ER untuk aplikasi untuk aplikasi referensi telah dijelaskan sebelumnya di bab ini.

### 10.3.2 Memulai Pengembangan dengan Django

Django dapat dipasang dengan perintah berikut:

```
# Installing Django
sudo apt-get install python-pip
sudo pip install Django==1.8.5
```

Di bagian ini, Anda akan belajar bagaimana memulai mengembangkan aplikasi web dengan Django.

**Membuat Proyek dan Aplikasi Django**

Box 10.11 menyediakan perintah untuk membuat proyek Django dan aplikasi di dalam proyek.

Ketika Anda membuat proyek Django baru, berkas berikut dibuat:

- __init__.py: File ini memberitahu Python bahwa folder ini adalah paket Python
- manage.py: File ini berisi larik fungsi untuk mengatur proyek
- settings.py: File ini berisi pengaturan proyek
- urls.py: File ini berisi pola URL yang memetakan URL ke halaman

sebuah proyek Django dapat memiliki banyak aplikasi. Aplikasi adalah tempat Anda menulis kode yang membuat aplikasi web Anda berfungsi. Setiap proyek dapat memiliki banyak aplikasi dan setiap aplikasi dapat menjadi bagian dari banyak proyek.
Ketika aplikasi baru dibuat, direktori baru untuk aplikasi tersebut dibuat yang memiliki banyak file termasuk:

- model.py: File ini berisi deskripsi model untuk aplikasi
- views.py: File ini berisi tampilan aplikasi

**Box 10.11: Creating a new Django project and an app in the project**

```
#Create a new project
django-admin.py startproject myproject
```

```
#Create an application within the project
python mangage.py startapp myapp

#Starting development server
python manage.py runserver

#Django uses port 8000 by default
#The project can be viewed at the URL:
#http://localhost:8000
```

Django hadir dengan server Web built-in dan ringan yang dapat digunakan untuk tujuan pengembangan. Ketika server pengembangan Django dimulai proyek standar dapat dilihat di URL: http: // localhost: 8000.

### Mengonfigurasi Database

Sampai sekarang Anda telah mempelajari bagaimana membuat proyek Django baru dan sebuah aplikasi dalam proyek tersebut. Sebagian besar aplikasi web memiliki backend database. Pengembang memiliki banyak pilihan database yang dapat digunakan untuk aplikasi web termasuk database relasional dan non-relasional. Django menyediakan API terpadu untuk backend database sehingga memberikan kebebasan untuk memilih database. Django mendukung berbagai mesin database relasional termasuk MySQL, PostgreSQL, Oracle dan SQLite3. Dukungan untuk database non-relasional seperti MongoDB dapat ditambahkan dengan menginstal mesin tambahan (misalnya mesin Django-MongoDB untuk MongoDB).

Mari kita lihat contoh penyetelan database MySQL dengan proyek Django. Langkah pertama dalam menyiapkan database adalah menginstal dan mengkonfigurasi server database. Setelah menginstal database, langkah selanjutnya adalah menentukan seting database dalam file setting.py dalam proyek Django.

Box 10.12 menunjukkan perintah untuk mengatur MySQL. Box 10.13 memperlihatkan pengaturan database untuk menggunakan MySQL dengan proyek Django

**■ Box 10.12: Setting up MySQL database**

```
#Install MySQL
sudo apt-get install mysql-server mysql-client
sudo mysqladmin -u root -h localhost password 'mypassword'
```

**■ Box 10.13: Configuring MySQL with Django - settings.py**

```
DATABASES = {
   'default':  {
      'ENGINE': 'django.db.backends.mysql',
      'NAME': '<database-name>'
      'USER': 'root'
      'PASSWORD': 'mypassword'
      'HOST': '<hostname>', # set to empty for localhost
      'PORT': '<port>', #set to empty for default port
}
}
```

### Mendefinisikan Model

Model berperan sebagai definisi dari data dalam database. Box 10.14 memperlihatkan kode Python untuk model Django untuk aplikasi referensi. Berbagai tabel database untuk aplikasi ini didefinisikan sebagai Kelas dalam model Django. Setiap kelas yang merepresentasikan tabel database adalah subkelas dari kelas django.db.models.Model yang berisi semua fungsionalitas yang memungkinkan model untuk berinteraksi dengan database. Untuk menyinkronkan model dengan database cukup jalankan perintah berikut:

> python manage.py syncdb

**Box 10.14: Django model for reference application**

```
from django.db import models
from django.contrib.auth.models import User
from django.conf import settings

class Department(models.Model):
 name = models.CharField(verbose_name="Name",
  max_length=1000,blank = True,null=True)

 number = models.CharField(verbose_name="Number",
  max_length=1000,blank = True,primary_key=True)

 def __unicode__(self):
  return str(self.name)+" - "+str(self.number)

class Project(models.Model):
 name = models.CharField(verbose_name="Name",
  max_length=1000,blank = True,null=True)

 number = models.CharField(verbose_name="Number",
  max_length=1000,blank = True,primary_key=True)

 def __unicode__(self):
  return str(self.name)+" - "+str(self.number)
def project_ids(self):
  return "project_"+str(self.id)

class Employee (models.Model):

 name = models.CharField(verbose_name="Name",
  max_length=1000,blank = True,null=True)

 number = models.CharField(verbose_name="Number",
  max_length=1000,blank = True,primary_key=True)

 salary = models.CharField(verbose_name="Salary",
  max_length=1000,blank = True,null=True)
```

```
 department= models.ForeignKey(Department, null=True,blank=True,
  related_name='departmentname', default="null")

 project=models.ManyToManyField(Project,through='WorksOn')

 def __unicode__(self):
  return str(self.name)+" - "+str(self.number)+
   " - "+str(self.department.number)


class WorksOn(models.Model):
employee = models.ForeignKey(Employee)
project = models.ForeignKey(Project)

def __unicode__(self):
  return str(self.employee.name)+" - "+str(self.project.name)
```

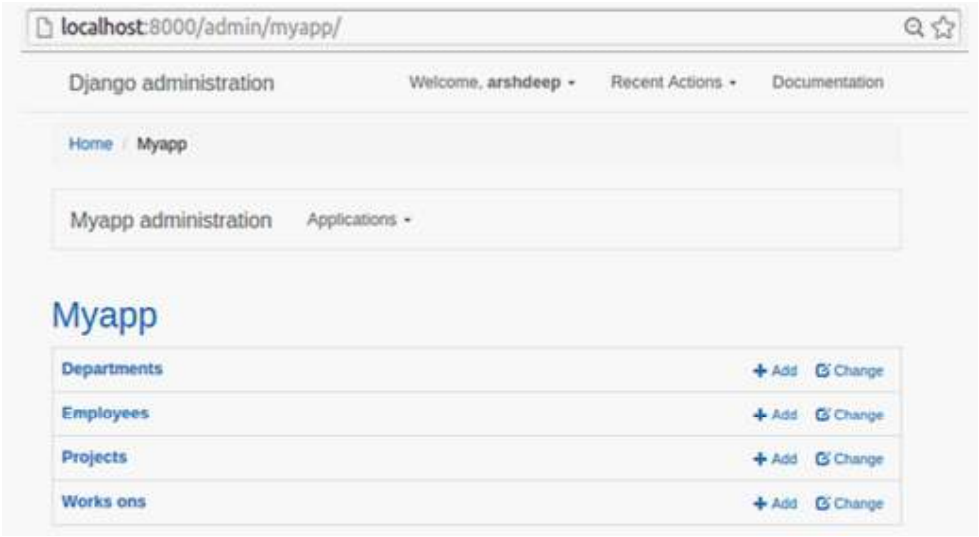
### Situs Admin Django

Django menyediakan sistem administrasi yang memperbolehkan Anda untuk mengatur proyek tanpa menulis kode tambahan. Sistem admin membaca model Django dan menyediakan interface yang dapat digunakan untuk menambahkan konten ke proyek. Situs admin Django diaktifkan dengan menambahkan django.contrib.admin dan django.contrib.admindocs ke bagian INSTALLED_APPS dalam berkas settings.py. Situs admin juga memerlukan definisi pola URL di file urls.py yang dijelaskan nanti di bagian URL. Untuk menentukan model aplikasi yang dapat diedit di interface admin, file baru bernama admin.py dibuat di folder aplikasi seperti yang ditunjukkan pada Box 10.15.

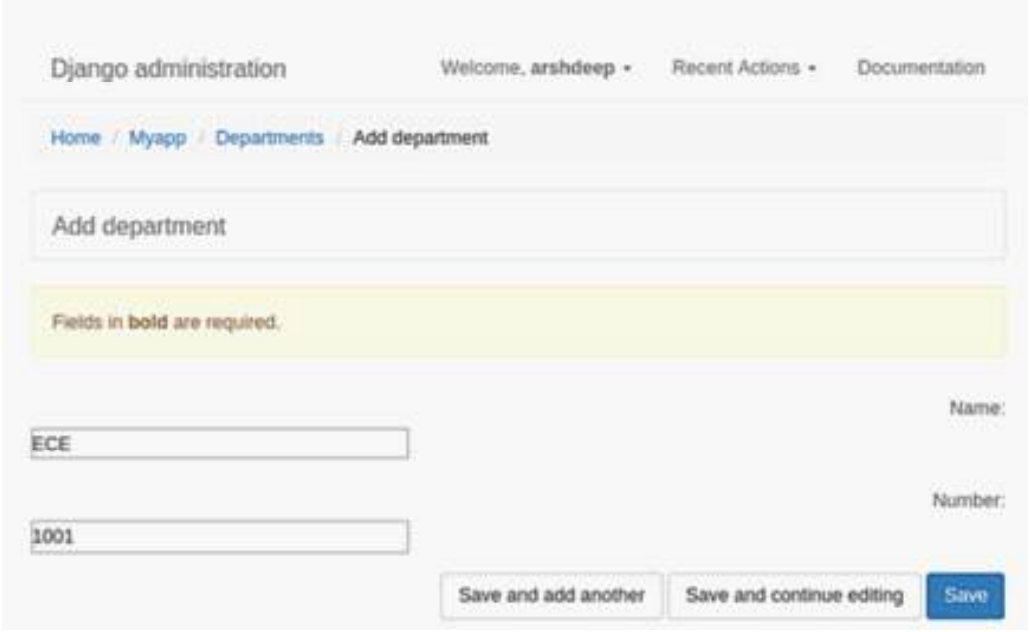
**■ Box 10.15: Enabling admin for Django models**

```
from django.contrib import admin
from django.contrib.auth.models import User

from myapp.models import Department,
 Employee, Project, WorksOn

admin.site.register(Department)
admin.site.register(Employee)
admin.site.register(Project)
admin.site.register(WorksOn)
```

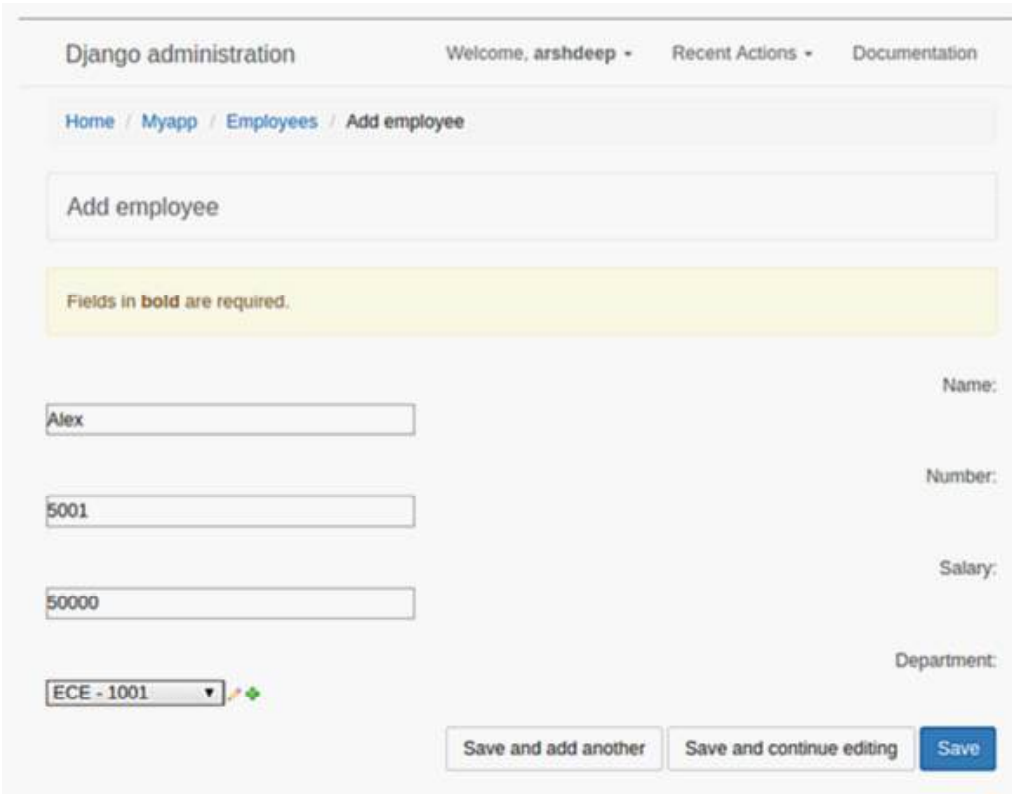
Gambar 10.9 memperlihatkan screenshot dari interface admin Django. Anda dapat melihat semua tabel yang berhubungan dengan model Django di screenshot ini. Gambar 10.10, 10.11, 10.12 dan 10.13 menunjukkan bagaimana menambahkan item baru di tabel Departemen, Karyawan, Proyek dan WorksOn menggunakan situs admin.



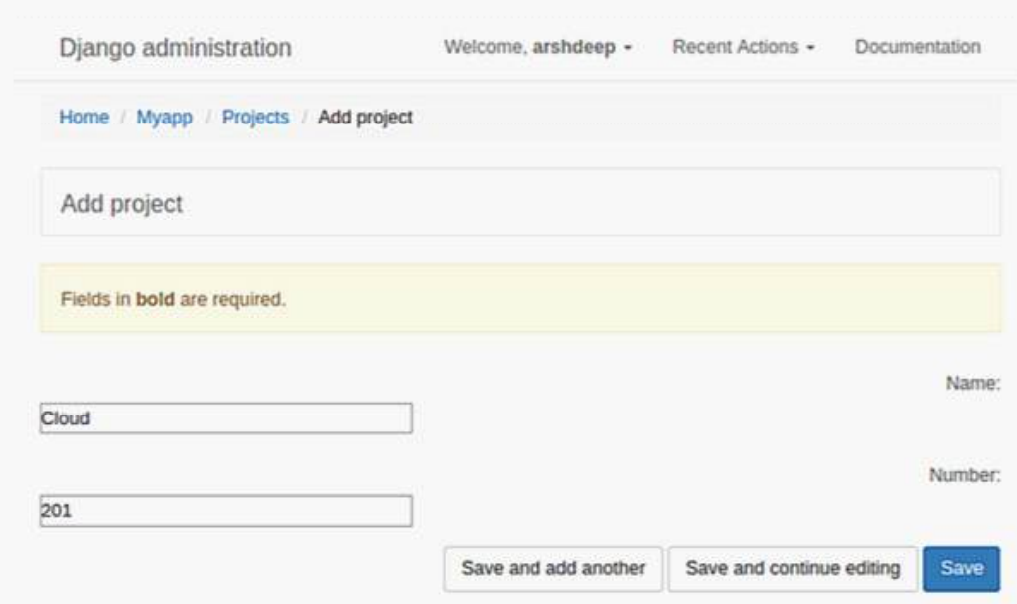

Gambar 10.9: *Screentshot Situs Admin Django*



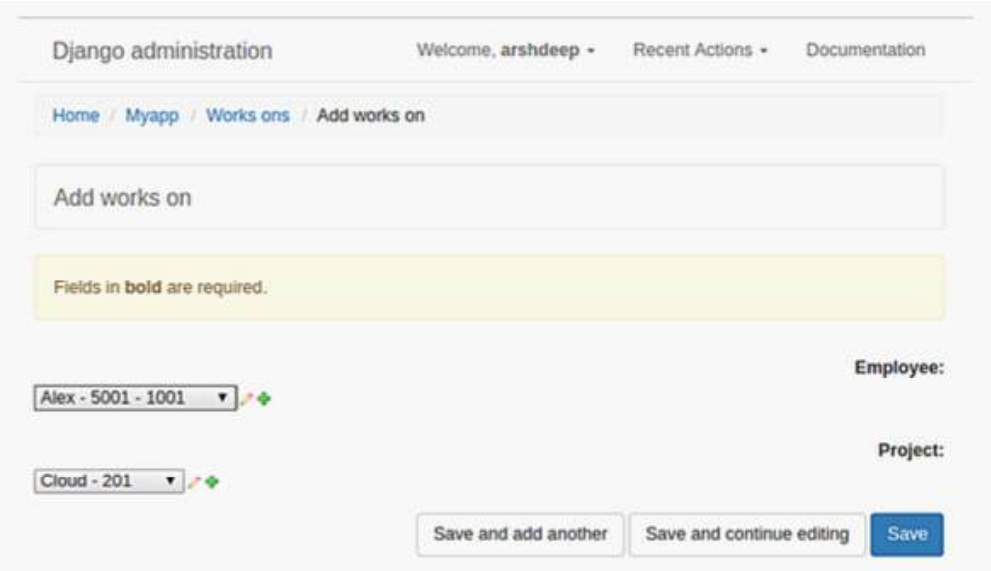

Gambar 10.10: *Situs Admin Django – Penambahan item baru pada tabel departemen*




Gambar 10.11: *Situs Admin Django – Penambahan item baru pada tabel karyawan*




Gambar 10.12: *Situs Admin Django – Penambahan item baru pada tabel project*




Gambar 10.12: *Situs Admin Django – Penambahan item baru pada tabel work-on*

### Mendefinisikan View

View berisi logika yang merekatkan model ke tempelatee. Tampilan menentukan data yang akan ditampilkan dalam tempelatee, mengambil data dari database, dan sandi untuk tempelatee. Sebaliknya, tampilan juga mengekstrak data yang diposting dalam formulir di tempelate dan menyisipkannya ke dalam database. Biasanya, setiap aplikasi halaman web memiliki tampilan terpisah, yang difungsikanisaPython di file views.py. Tampilan juga dapat melakukan tugas tambahan seperti otentikasi, mengirim email, dll.

Box 10.16 memperlihatkan kode sumber untuk tampilan Django untuk aplikasi referensi. Tampilan sesuai dengan halaman web yang menampilkan daftar karyawan, detail karyawan, detail departemen dan detail proyek.

Dalam tampilan, API pemetaan relasional objek bawaan Django digunakan untuk mengambil data dari tabel database. API pemetaan relasional objek memungkinkan pengembang menulis kode generik untuk berinteraksi dengan database tanpa mengkhawatirkan mesin database yang mendasarinya. Jadi kode yang sama untuk interaksi database berfungsi dengan backend database yang berbeda. Anda juga dapat memilih untuk menggunakan pustaka Python yang khusus untuk backend database yang digunakan (misalnya MySQLdb untuk MYSQL, PyMongo untuk MongoDB, dll.) Untuk menulis kode spesifik yang didukung database.

Dalam tampilan yang diperlihatkan di Box 10.16, kueri thetable.objects.all mengembalikan QuerySet dengan semua entri dalam tabel. Untuk mengambil entri tertentu, Anda bisa menggunakan table.objects.filter (□□ kwargs) untuk menyaring kueri yang cocok dengan kondisi yang ditentukan. Misalnya, kueri Employee.objects.filter (number = 0 1230) mengembalikan detail karyawan dengan employee-number123. Untuk tempelatee, fungsi therender_to_response digunakan. Fungsi ini merender tempelatee tertentu dengan kamus konteks tertentu dan mengembalikan objek HttpResponse dengan teks yang direnderr tersebut.

**■ Box 10.16: Django view for reference application**

```
from django.shortcuts import render_to_response
from django.template import RequestContext
from django.shortcuts import render
from django.shortcuts import redirect
from django.contrib.auth import authenticate, login
from django.contrib.auth import logout
from django.contrib.auth.decorators import login_required
from django.contrib.auth.models import User
from myapp.models import Department, Employee, Project, WorksOn

def logout_view(request):
 logout(request)
 return redirect('/')


def home(request):
 if request.method == 'POST':
  username = request.POST['username']
  password = request.POST['password']
```

```
  user = authenticate(username=username, password=password)
  if user is not None:
   if user.is_active:
    login(request, user)
    username = request.user.username
    authorusername = str(request.user.username)


 if request.user.is_authenticated():
  authorusername = str(request.user.username)

  employeecount = Employee.objects.all().count()
  departmentcount = Department.objects.all().count()
  projectcount = Project.objects.all().count()

  return render_to_response('index.html',{
   'authorusername':  authorusername, 'employeecount':employeecount,
    'departmentcount':departmentcount, 'projectcount':projectcount},
    context_instance=RequestContext(request))

 return redirect('/accounts/login')

@login_required
def employees(request):
  if request.user.is_authenticated():
  authorusername = str(request.user.username)
 employees = Employee.objects.all()
 project_list=[]
 len_e=len(employees)
 for it in range(len(employees)):
  templ=employees[it].project.all()
  for items3 in templ:
   project_list.append((items3.name,employees[it].number))
 return render_to_response('employees.html',{
    'authorusername':  authorusername,'employees':employees,
    'len':len_e,'project_list':project_list},
    context_instance=RequestContext(request))

@login_required
def employeedetail(request, query):
 if request.user.is_authenticated():
  authorusername = str(request.user.username)

 employee = Employee.objects.filter(number=query).get()
 project_list=employee.project.all()
 return render_to_response('employeedetail.html',{
  'authorusername':  authorusername,'employee':employee,
  'project_list':project_list},
  context_instance=RequestContext(request))

@login_required
def departmentdetail(request, query):
 if request.user.is_authenticated():
  authorusername = str(request.user.username)
```

```
 department = Department.objects.filter(number=query).get()
 employee_count = Employee.objects.filter(department__pk=query).count()
 return render_to_response('departmentdetail.html',{
  'authorusername': authorusername,'department':department,
  'employee_count':employee_count},
  context_instance=RequestContext(request))

@login_required
def projectdetail(request, query):
 if request.user.is_authenticated():
  authorusername = str(request.user.username)

 project = Project.objects.filter(number=query).get()
 employee_count = Employee.objects.filter(project__pk=query).count()
 return render_to_response('projectdetail.html',{
  'authorusername': authorusername,'project':project,
  'employee_count':employee_count},
  context_instance=RequestContext(request))
```

**Mendefinisikan Tempelate**

Sebuah tempelatee Django biasanya adalah sebuah file HTML (meskipun itu dapat berupa file teks apapun seperti XML, email, CSS, JavaScript, CSV, dll.). Tempelate Django memperbolehkan pemisahan penyajian data dari data aktual dengan menggunakan placeholder dan logika terkait (menggunakan tag tempelate). Tempelate menerima konteks dari tampilan dan menyajikan data dalam variabel konteks di tempat penampung.

Box 10.17, 10.18, 10.19 dan 10.20 menunjukkan tempelate untuk aplikasi referensi. Data diambil dari database dalam tampilan dan diteruskan ke tempelate dalam bentuk kamus konteks. Tag for di loop tempelatee di atas setiap item secara berurutan dan item disisipkan dengan tag placeholder (nama variabel yang dikelilingi oleh tanda kurung kurawal, mis. {{Entry.email}}). Tag tempelatee adalah teks apa pun yang dikelilingi oleh tanda kurung kurawal dan tanda persen (mis. {% Untuk entri dalam student_entries%}). Bahasa cetakan Django menawarkan tag dasar seperti untuk, jika, dll. Dan sejumlah filter built-in untuk memodifikasi output variabel. Filter dilampirkan ke variabel menggunakan karakter pipa (|). Misalnya filter bergabung dalam {{entry.courses.all | join: ","}} menggabungkan daftar dengan string.

**■ Box 10.17: Django template for showing employees list**

```
<html lang="en">
 <head>
 <title> Dashboard</title>
 <!- Bootstrap Core CSS ->
 <link href="/static/css/bootstrap.min.css" rel="stylesheet">
 </head>
```

```
<body>
 <div id="page-wrapper">
 <div class="container-fluid">
  <!- Page Heading ->
  <div class="row">
   <div class="col-lg-12">
    <h1 class="page-header">
     Employees
    </h1>
   </div>
  </div>
  <!- /.row ->
 </div>
 <!- /.row ->
 <div class="row">
 <div class="col-lg-6">
  <div class="panel panel-default">
   <div class="panel-heading">
    <h3 class="panel-title">Employees List</h3>
   </div>
   <div class="panel-body">
    <div class="table-responsive">
     <table class="table table-bordered table-hover table-striped">
      <thead>
       <tr>
        <th>Employee No.</th>
        <th>Name</th>
        <th>Salary</th>
        <th>Department</th>
       </tr>
      </thead>
      <tbody>
       {% for employee in employees%}
       <tr>
        <td>
<a href="/employee/{{employee.number}}/">{{employee.number}}</a></td>
        <td>{{employee.name}}</td>
        <td>{{employee.salary}}</td>
        <td><a href="/department/{{employee.department.number}}/">
{{employee.department.name}}</a></td>
       </tr>
       {% endfor %}
      </tbody>
     </table>
    </div>
    <!- /.row ->
   </div>
  </div>
  <!- /#page-wrapper ->
 </div>
 <!- /#wrapper ->
 <!- jQuery ->
 <script src="/static/js/jquery.js"></script>
 <!- Bootstrap Core JavaScript ->
 <script src="/static/js/bootstrap.min.js"></script>
</body>
</html>
```

**Box 10.18: Django template for showing employee details**

```
<html lang="en">
 <head>
  <title> Dashboard</title>
  <!- Bootstrap Core CSS ->
  <link href="/static/css/bootstrap.min.css" rel="stylesheet">
 </head>
 <body>
  <div id="wrapper">
  <div id="page-wrapper">
  <div class="container-fluid">
   <!- Page Heading ->
   <div class="row">
    <div class="col-lg-12">
     <h1 class="page-header">
      {{employee.name}}
     </h1>
    </div>
   </div>
   <!- /.row ->
  </div>
  <!- /.row ->
  <div class="row">
  <div class="col-lg-6">
   <div class="panel panel-default">
    <div class="panel-heading">
     <h3 class="panel-title">Employee Detail</h3>
    </div>
    <div class="panel-body">
     <div class="table-responsive">
      <table class="table table-bordered table-hover table-striped">
       <tbody>
```

```
        <tr>
         <td>Name</td>
         <td>{{employee.name}}</td>
        </tr>
        <tr>
         <td> Number</td>
         <td>{{employee.number}}</td>
        </tr>
        <tr>
         <td>Salary</td>
         <td>{{employee.salary}}</td>
        </tr>
        <tr>
         <td>Projects</td>
         <td>
          {% for project in project_list%}
          <a href="/project/{{project.number}}/">{{project.name}}</a><br>
          {% endfor %}
         </td>
        </tr>
       </tbody>
      </table>
     </div>
     <!- /.row ->
```

```
  </div>
 </div>
 <!- /#page-wrapper ->
</div>
<!- /#wrapper ->
<!- jQuery ->
<script src="/static/js/jquery.js"></script>
<!- Bootstrap Core JavaScript ->
<script src="/static/js/bootstrap.min.js"></script>
</body>
</html>
```

**■ Box 10.19: Django template for showing project details**

```
<html lang="en">
 <head>
 <title>Dashboard</title>
 <!- Bootstrap Core CSS ->
 <link href="/static/css/bootstrap.min.css" rel="stylesheet">
 </head>
 <body>
 <div id="wrapper">
 <div id="page-wrapper">
 <div class="container-fluid">
  <!- Page Heading ->
  <div class="row">
   <div class="col-lg-12">
    <h1 class="page-header">
     {{project.name}}
    </h1>
   </div>
  </div>
  <!- /.row ->
 </div>
 <!- /.row ->
 <div class="row">
 <div class="col-lg-6">
  <div class="panel panel-default">
   <div class="panel-heading">
    <h3 class="panel-title"> Project Details</h3>
   </div>
   <div class="panel-body">
    <div class="table-responsive">
     <table class="table table-bordered table-hover table-striped">
      <tbody>
       <tr>
        <td>Name</td>
        <td>{{project.name}}</td>
       </tr>
       <tr>
        <td>Number</td>
        <td>{{project.number}}</td>
       </tr>
```

```
    <td>Total Employees Working on the Project</td>
    <td>{{employee_count}}</td>
    </tr>
   </tbody>
  </table>
 </div>
 <!- /.row ->
</div>
</div>
<!- /#page-wrapper ->
</div>
<!- /#wrapper ->
<!- jQuery ->
<script src="/static/js/jquery.js"></script>
<!- Bootstrap Core JavaScript ->
<script src="/static/js/bootstrap.min.js"></script>
</body>
</html>
```

■ **Box 10.20: Django template for showing department details**

```
<html lang="en">
<head>
<title> Dashboard</title>
<!- Bootstrap Core CSS ->
<link href="/static/css/bootstrap.min.css" rel="stylesheet">
</head>
```

```
<body>
<div id="wrapper">
<div id="page-wrapper">
<div class="container-fluid">
 <!- Page Heading ->
 <div class="row">
  <div class="col-lg-12">
   <h1 class="page-header">
    {{department.name}}
   </h1>
  </div>
 </div>
 <!- /.row ->
</div>
<!- /.row ->
<div class="row">
<div class="col-lg-6">
 <div class="panel panel-default">
  <div class="panel-heading">
   <h3 class="panel-title">Department Detail</h3>
  </div>
  <div class="panel-body">
   <div class="table-responsive">
    <table class="table table-bordered table-hover table-striped">
     <tbody>
      <tr>
```

```
      <td>Name</td>
      <td>{{department.name}}</td>
     </tr>
     <tr>
      <td>Number</td>
      <td>{{department.number}}</td>
     </tr>
     <tr>
      <td>Total Employees in Department</td>
      <td>{{employee_count}}</td>
     </tr>
    </tbody>
   </table>
  </div>
  <!- /.row ->
 </div>
 <!- /.container-fluid ->
</div>
<!- /#page-wrapper ->
</div>
<!- /#wrapper ->
<!- jQuery ->
<script src="/static/js/jquery.js"></script>
<!- Bootstrap Core JavaScript ->
<script src="/static/js/bootstrap.min.js"></script>
</body>
</html>
```

**Mendefinisikan Pola URL**

Pola URL adalah cara memetakan URL ke tampilan yang seharusnya menangani permintaan URL. URL yang diminta oleh pengguna dicocokkan dengan pola URL dan tampilan yang sesuai dengan pola yang matriks URL digunakan untuk menangani permintaan. Box 10.21 menunjukkan contoh pola URL untuk aplikasi referensi. Seperti yang terlihat dalam contoh ini, pola URL dibuat menggunakan ekspresi reguler. Persamaan reguler paling sederhana (r'$ ') sesuai dengan root situs web atau beranda. URL yang lebih kompleks memungkinkan menangkap nilai. Misalnya polanya:

url (r'karyawan / (? P <query> \ w +) ',' myapp.views.employeedetail ')

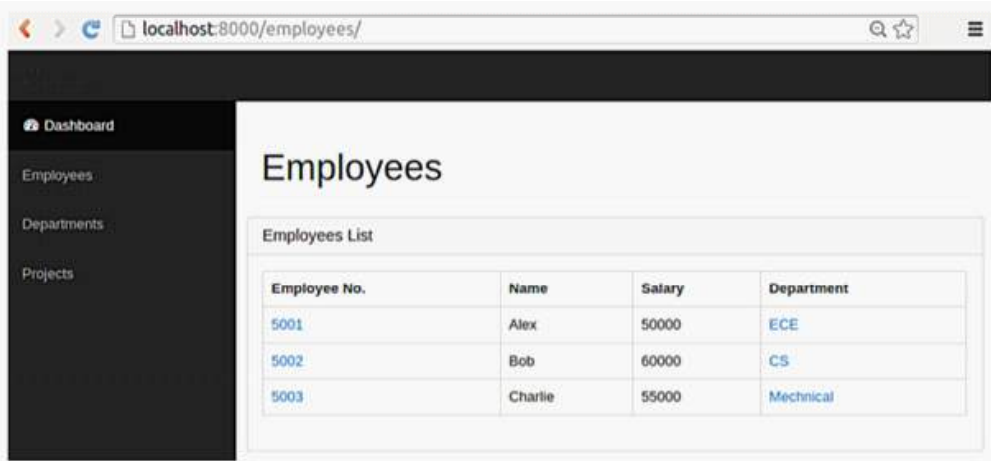
menangkap nomor karyawan dari URL ke kueri variabel dan meneruskannya ke tampilan detail kerja.

**■ Box 10.21: Example of a URL configuration**

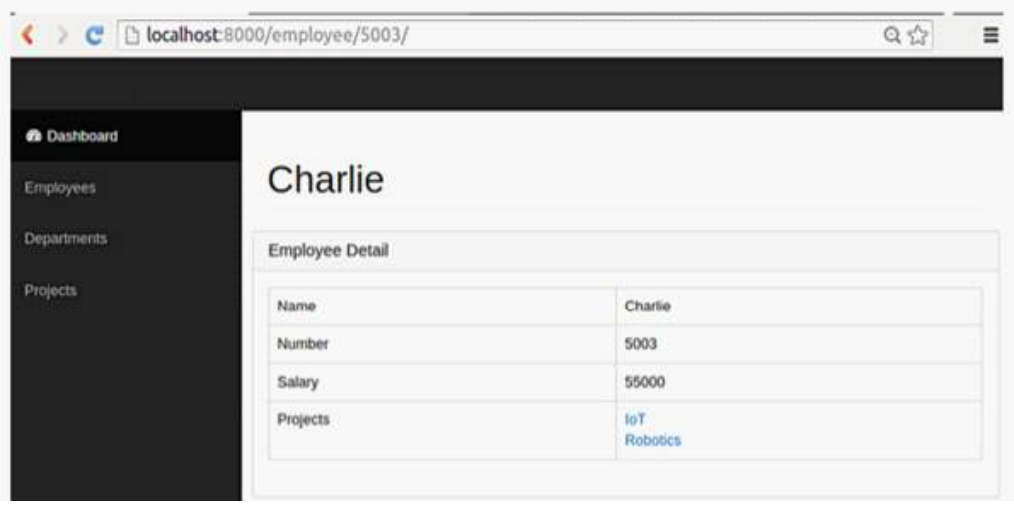
```
from django.conf.urls import patterns, include, url
from myapp import views
from django.contrib import admin

admin.autodiscover()
```

```
urlpatterns = patterns('',
url(r'^$', 'myapp.views.home', name='home'),
url(r'^employees/$', 'myapp.views.employees', name='employees'),
url(r'^employee/(?P<query>\w+)/$', 'myapp.views.employeedetail'),
url(r'^department/(?P<query>\w+)/$', 'myapp.views.departmentdetail'),
url(r'^project/(?P<query>\w+)/$', 'myapp.views.projectdetail'),
url(r'^admin/doc/', include(django.contrib.admindocs.urls)),
url(r'^admin/', include(admin.site.urls)),
)
```

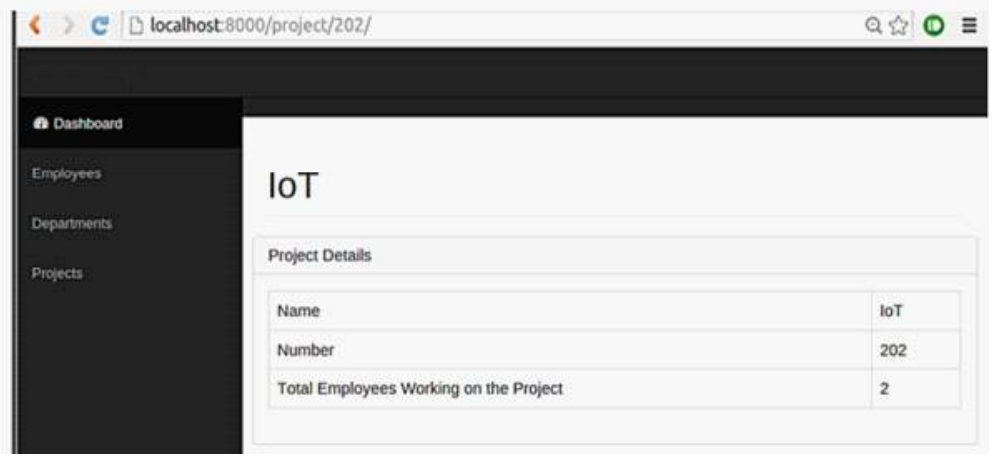
Gambar 10.14,10.15,10.16 dan 10.17menunjukkan berbagai halaman dari aplikasi referensi yang dirender dari tempelatee yang ditunjukkan dalam Box 10.17, 10.18, 10.19 dan 10.20.



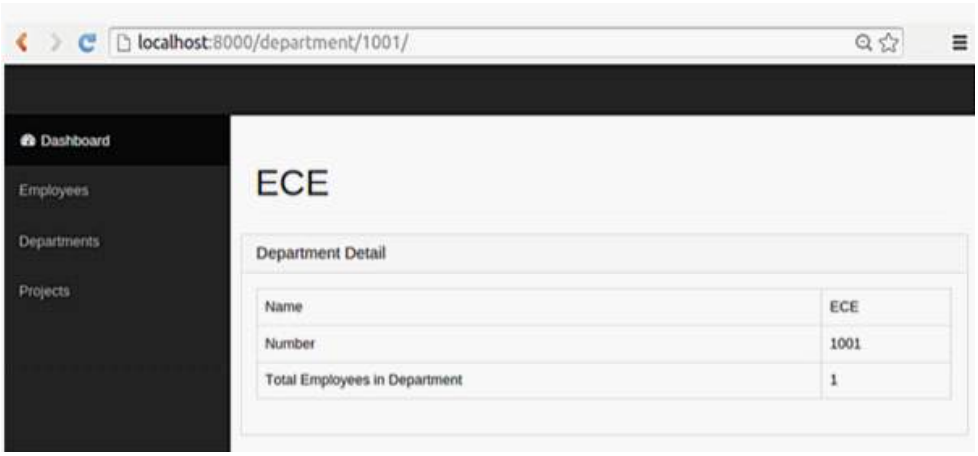

Gambar 10.14: *Screenshot halaman daftar karyawan yang dirender dari tempelatee di Box 10.17*




10.15: Screenshot *halaman detail karyawan yang dirender dari tempelatee di Box 10.18*




Gambar 10.15: *Screenshot halaman detail karyawan yang dirender dari tempelatee di Box 10.18*




Gambar 10.16: *Screenshot halaman detail proyek yang dirender dari tempelatee di Box 10.19*

Studi Kasus : Aplikasi Django untuk melihat data cuaca

Mari kita lihat studi kasus aplikasi Django yang menampilkan data cuaca terkini dan prakiraan cuaca di berbagai kota. Dalam bagianDynamoDB, Cassandra danMongoDB kami menjelaskan contoh-contoh memperoleh data cuaca menggunakan Open Weather Map API dan menulis data ke database masing-masing. Sekarang mari kita membangun aplikasi web Django yang mengambil data cuaca dari database ini dan menyajikan data dalam halaman web. Kami akan menjelaskan tiga implementasi alternatif berdasarkan DynamoDB, Cassandra dan MongoDB. Box 10.22 memperlihatkan kode sumber untuk tampilan Django yang mengambil data cuaca dari tabel DynamoDB dan membuatnya dalam cetakan Django.

**Box 10.22: Django view for retrieving weather data from DynamoDB**

```
from django.shortcuts import render_to_response
from django.template import RequestContext
import boto.dynamodb2
from boto.dynamodb2.table import Table
import cPickle
import pyowm

REGION="us-east-1"

conn = boto.dynamodb2.connect_to_region(REGION,
aws_access_key_id='<enter>',
aws_secret_access_key='<enter>')


table=Table('weather',connection=conn)
forecastTable=Table('forecast',connection=conn)

PYOWM_KEY='<enter>'
owm = pyowm.OWM(PYOWM_KEY)

def home(request):
 if request.method == 'POST':
  city=request.POST.get('city')
 else:
  city = 'New York,US'

 results = table.query_2(city__eq=city, reverse=True, limit=1)
 for r in results:
  time=r['time']
  temp=r['currentTemperature']
  status=r['weatherStatus']
  pressure=r['pressure']
  humidity=r['humidity']
  wind=r['windSpeed']
  rain=r['rainVolume']
  clouds=r['cloudCoverage']

 results = forecastTable.query_2(city__eq=city)
 for r in results:
  data = str(r['forecastList'])

 forecast_list = cPickle.loads(data)

 return render_to_response('index.html',{
  'city':city,'time':time, 'temp':temp, 'status':status,
  'pressure':pressure, 'humidity':humidity,
  'wind':wind, 'rain':rain, 'clouds':clouds,
  'forecast_list':forecast_list},
  context_instance=RequestContext(request))
```

Box 10.23 memperlihatkan kode sumber untuk tampilan Django yang mengambil data cuaca dari tabel Cassandra dan membuatnya dalam cetakan Django.

**■ Box 10.23: Django view for retrieving weather data from Cassandra**

```
from django.shortcuts import render_to_response
from django.template import RequestContext
import cPickle
import pyowm
from cassandra.cluster import Cluster
PYOWM_KEY='<enter>'
owm = pyowm.OWM(PYOWM_KEY)

## create Cassandra instance
cluster = Cluster()

## establish Cassandra connection, using local default
session = cluster.connect('weatherkeyspace')


def home(request):
 if request.method == 'POST':
  city=request.POST.get('city')
 else:
  city = 'New York,US'

 results = session.execute('SELECT * FROM weather
   WHERE city='+city+'ÓRDER BY time DESC LIMIT 1')
 for r in results:
  time=r[1]
  temp=r[2]
  status=r[3]
  pressure=r[4]
  humidity=r[5]
  wind=r[6]
  rain=r[7]
  clouds=r[8]

 results = session.execute('SELECT * FROM forecast
   WHERE city='+city+'')

 for r in results:
  data = r[1]

 forecast_list = cPickle.loads(data.decode("hex"))

 return render_to_response('index.html',{
  'city':city,'time':time, 'temp':temp,
  'status':status, 'pressure':pressure, 'humidity':humidity,
  'wind':wind, 'rain':rain, 'clouds':clouds,
  'forecast_list':forecast_list},
  context_instance=RequestContext(request))
```

Box 10.24 memperlihatkan kode sumber untuk tampilan Django yang mengambil data cuaca dari MongoDB dan membuatnya dalam cetakan Django.

**Box 10.24: Django view for retrieving weather data from MongoDB**

```
from django.shortcuts import render_to_response
from django.template import RequestContext
import pyowm
from pymongo import MongoClient

client = MongoClient('mongodb://root:password@hostname:53688/weather')
db = client['weather']
weather_collection = db['current']
forecast_collection = db['forecast']

PYOWM_KEY='<enter>'
owm = pyowm.OWM(PYOWM_KEY)

def home(request):
 if request.method == 'POST':
  city=request.POST.get('city')
 else:
  city = 'New York,US'

 results = weather_collection.find({"city":
  city}).sort("_id",-1).limit(1)
 for r in results:
  time=r['time']
  temp=r['currentTemperature']
  status=r['weatherStatus']
  pressure=r['pressure']
  humidity=r['humidity']
  wind=r['windSpeed']
  rain=r['rainVolume']
  clouds=r['cloudCoverage']

 results = forecast_collection.find({"city":  city})

 for r in results:
  data = r['forecastList']

 forecast_list = data


 return render_to_response('index.html',{
    'city':city,'time':time, 'temp':temp,
    'status':status, 'pressure':pressure, 'humidity':humidity,
    'wind':wind, 'rain':rain, 'clouds':clouds,
    'forecast_list':forecast_list},
    context_instance=RequestContext(request))
```

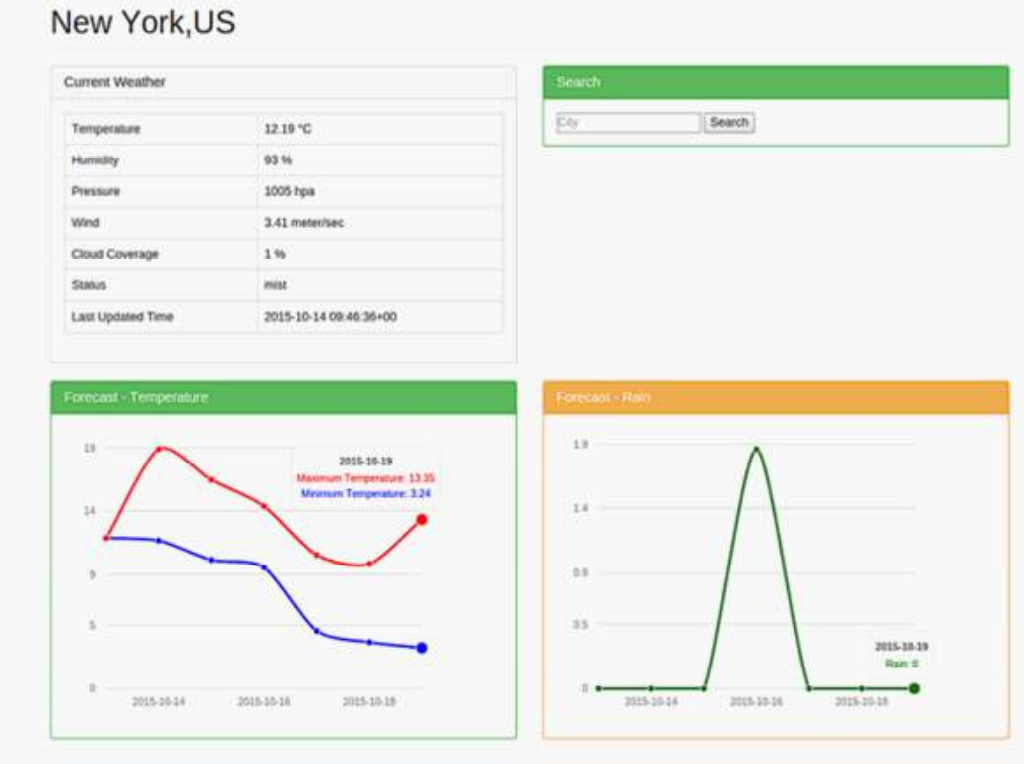
Box 10.25 memperlihatkan kode sumber dari tempelate Django untuk aplikasi cuaca dan Gambar 10.18 memperlihatkan screenshot dari halaman web yang direnderdari tempelate

**Box 10.25: Django template for weather application**

```
<html lang="en">
 <head>
  <title> Dashboard</title>
  <!- Bootstrap Core CSS ->
  <link href="/static/css/bootstrap.min.css" rel="stylesheet">
  <!- Morris Charts CSS ->
  <link href="/static/css/plugins/morris.css" rel="stylesheet">
  <!-[if lt IE 9]>
  <script src="/static/js/html5shiv.js"></script>
  <script src="/static/js/respond.min.js"></script>
  <![endif]->
  <script src="/static/js/raphael-min.js"></script>
  <script src="/static/js/jquery-1.8.2.min.js"></script>
  <script src="/static/js/morris-0.4.1.min.js"></script>
 </head>
 <body>
  <div id="wrapper">
   <div id="page-wrapper">
    <div class="container-fluid">
     <!- Page Heading ->
     <div class="row">
      <div class="col-lg-12">
       <h1 class="page-header">
        {{city}}
       </h1>
      </div>
     </div>
     <!- /.row ->
     <div class="row">
      <div class="col-lg-4">
       <div class="panel panel-default">
        <div class="panel-heading">
         <h3 class="panel-title">Current Weather</h3>
        </div>
        <div class="panel-body">
         <div class="table-responsive">
          <table class="table table-bordered table-hover table-striped">
           <tbody>
            <tr>
             <td>Temperature</td>
             <td>{{temp}} &#176;C</td>
            </tr>

            <tr>
             <td>Humidity</td>
             <td>{{humidity}} %</td>
            </tr>
            <tr>
             <td>Pressure</td>
             <td>{{pressure}} hpa</td>
            </tr>
            <tr>
             <td>Wind</td>
```

```
        <td>{{wind}} meter/sec</td>
       </tr>
       <tr>
        <td>Cloud Coverage</td>
        <td>{{clouds}} %</td>
       </tr>
       <tr>
        <td>Status</td>
        <td>{{status}}</td>
       </tr>
       <tr>
        <td>Last Updated Time</td>
        <td>{{time}}</td>
       </tr>
      </tbody>
     </table>
    </div>
   </div>
  </div>
 </div>
 <div class="col-lg-4">
  <div class="panel panel-green">
   <div class="panel-heading">
    <h3 class="panel-title">Search</h3>
   </div>
   <div class="panel-body">
    <form method="post" action="/">{%csrf_token%}
     <input type="text" name="city" id="city" placeholder="City" />
     <input type="submit" value="Search" />
    </form>
   </div>
  </div>
 </div>
</div>
<!- /.row ->
<div class="row">
 <div class="col-lg-4">
  <div class="panel panel-green">
   <div class="panel-heading">
    <h3 class="panel-title">Forecast - Temperature</h3>
   </div>
   <div class="panel-body">
    <script>
     $(function() {
     Morris.Line({
      element: 'morris-line-chart',
      data: [
       {% for f in forecast_list %}
     { y: '{{f.date}}', 'a': {{f.tempMax}}, 'b': {{f.tempMin}} },
       {% endfor %}
      ],
      xkey: 'y',
      ykeys: ['a', 'b'],
      xLabels:'day',
      lineColors: ['red','blue'],
      labels: ['Maximum Temperature', 'Minimum Temperature']
```

```
      });
      });
    </script>
    <div id="morris-line-chart"></div>
   </div>
  </div>
 </div>
 <div class="col-lg-4">
  <div class="panel panel-yellow">
   <div class="panel-heading">
    <h3 class="panel-title">Forecast - Rain</h3>
   </div>
   <div class="panel-body">
    <script>
     $(function() {
     Morris.Line({
      element:  'morris-line-chart1',
      data:  [
       {% for f in forecast_list %}
    { y:  '{{f.date}}', 'a':  {{f.rain}} },
       {% endfor %}
      ],
      xkey:  'y',
      ykeys:  ['a'],
      xLabels:'day',
      labels:  ['Rain'],
      lineColors:  ['green']
      });
      });
    </script>
    <div id="morris-line-chart1"></div>
   </div>
  </div>
 </div>
 </div>
 <!- /.row ->
</div>
<!- /.container-fluid ->
</div>
<!- /#page-wrapper ->
</div>
<!- /#wrapper ->

<!- jQuery ->
<script src="/static/js/jquery.js"></script>
<!- Bootstrap Core JavaScript ->
<script src="/static/js/bootstrap.min.js"></script>
</body>
</html>
```




## Ringkasan

Pada bab ini kami memberikan perbandingan database NoSQL relasional dan non-relasional dan juga contoh beberapa database populer yang dapat digunakan sebagai database penyajian untuk aplikasi big data. Database relasional memiliki model yang konsisten dan skema tetap untuk relasinya. Database relasional memberikan jaminan Atomicity, Consistency, Isolation and Durability (ACID). Database non-relasional tidak memiliki skema tetap dan tidak memberikan jaminan ACID. Database non-relasional lebih dapat diskalakan dibandingkan dengan database relasional dan memiliki arsitektur yang terdistribusi, sangat tersedia, dan toleran terhadap kesalahan. Kami menjelaskan contoh penggunaan MySQL, DynamoDB, Cassandra, dan MongoDB sebagai database penyajian. Selanjutnya, kami mendeskripsikan kerangka web Django Python yang dapat digunakan untuk mengembangkan aplikasi web untuk menyajikan hasil analisis kepada pengguna. Django mengadopsi arsitektur model-tempelatee-view dan menawarkan API terpadu ke backend database. Kami juga menjelaskan contoh pembuatan model, tampilan dan tempelatee untuk aplikasi referensi.


# Topik Lanjutan

# Analisis Algoritma

Bab 11

Bab ini meliputi :

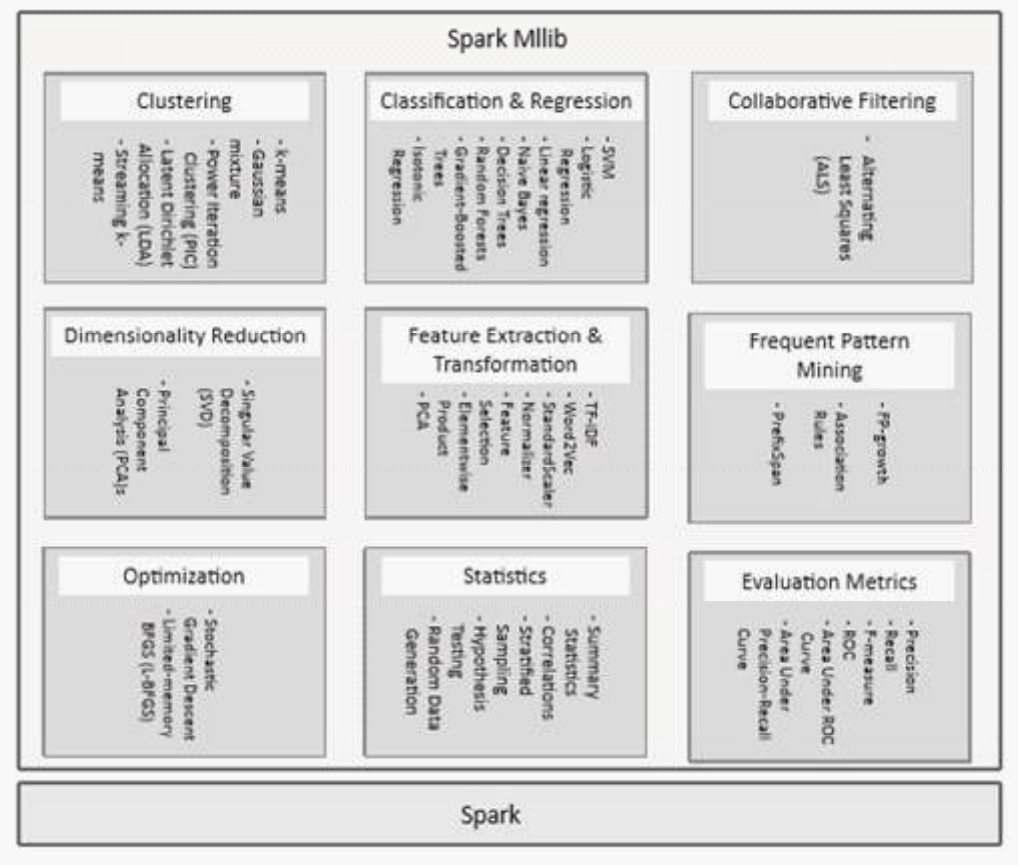
- Machine Learning tools dan frameworks
- Spark MLlib
- H2O
- Clustering Algoritma
- Klasifikasi Algoritma
- Regresi Algoritma
- Sistem Rekomendasi

Dalam bab ini, Anda akan mempelajari algoritme untuk analisis big data termasuk pengelompokan, klasifikasi, regresi, dan rekomendasi. Untuk contoh dalam bab ini, kita akan menggunakan framework machine learning Spark MLlib dan H2O.

## 11.1 Framework

### 11.1.1 Spark Mllib

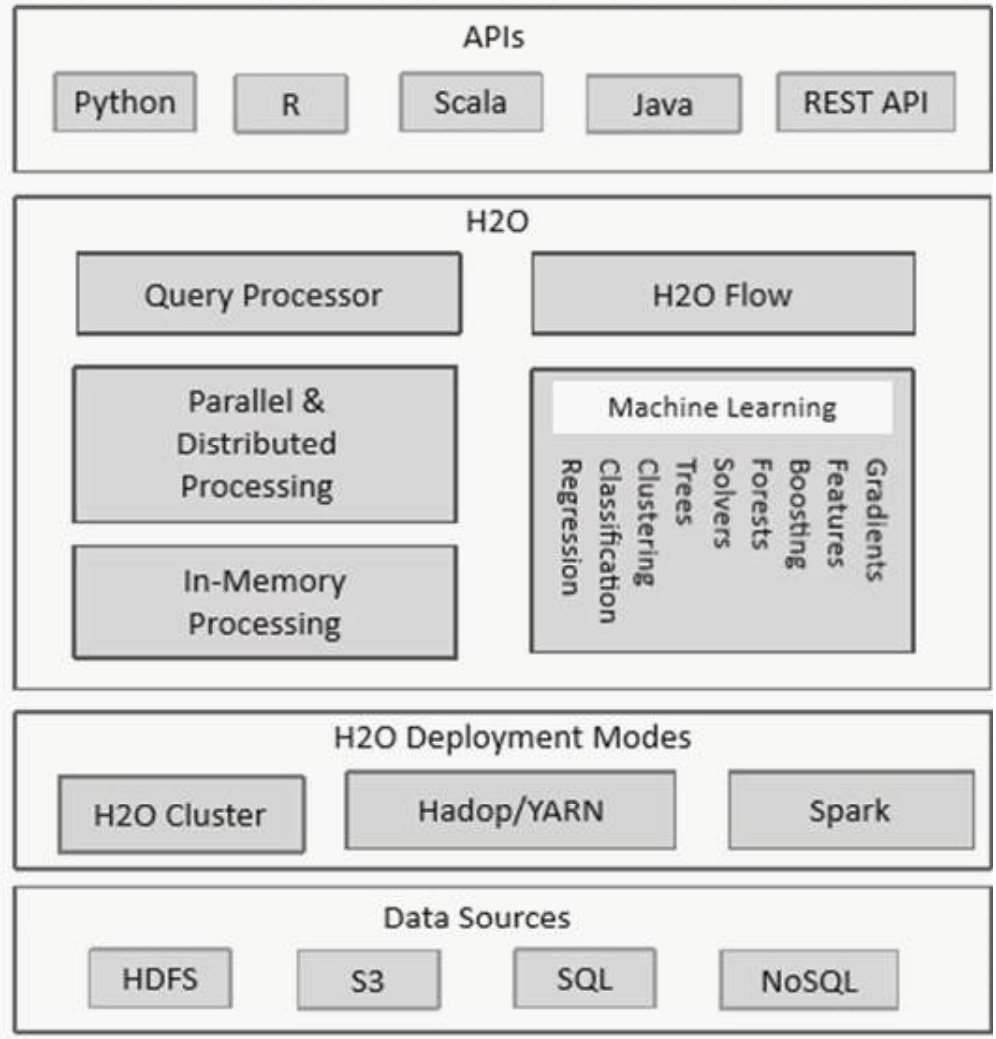
Spark MLlib adalah pustaka pembelajaran mesin Spark yang menyediakan implementasi berbagai algoritma pembelajaran mesin termasuk klasifikasi, regresi, pengelompokan, pemfilteran kolaboratif, dan pengurangan dimensi. MLlib API dibuat di atas dataset terdistribusi (RDD) yang tangguh dari Spark. MLlib juga menyediakan tipe data tingkat tinggi seperti Vektor, LabeledPoint, Peringkat dan Matriks, yang didukung oleh RDD. Manfaat menggunakan MLlib dibandingkan library machine learning adalah menyediakan implementasi paralel dari algoritme machine learning dan dapat memproses dataset terdistribusi besar. Spark MLlib menyediakan API untuk bahasa pemrograman Python, Scala, dan Java. Gambar 11.1 menunjukkan berbagai komponen Spark MLlib.




Gambar 11.1: *Komponen Spark MLlib*

### 11.1.2 H2O

H2O adalah kerangka kerja analisis prediktif sumber terbuka yang menyediakan implementasi berbagai algoritme pembelajaran mesin untuk pengelompokan, klasifikasi, dan pengurangan dimensi. Gambar 11.2menunjukkan berbagai komponen dari kerangka kerja H2O. H2O menyediakanAPI untuk bahasa pemrograman Python, Scala, R dan Java. H2O juga menyediakan interface web bergaya notebook yang disebut H2O yang memungkinkan pengguna mengimpor data dari berbagai sumber, membangun model pembelajaran mesin, dan membuat prediksi menggunakan model tersebut. H2O dapat dijalankan sebagai cluster mandiri atau di atas cluster Hadoop atau Spark yang sudah ada. Pustaka Sparkling Water H2O mengintegrasikan mesin pembelajaran mesin H2O dengan Spark. H2O dapat terhubung ke berbagai sumber data seperti HDFS, S3, SQL, dan NoSQL.




Gambar 11.2: *Komponen H2O*

### *Setting up H2O*

H2O dapat dijalankan sebagai cluster mandiri atau di atas cluster Hadoop. Untuk mengatur cluster mandiri H2O, unduh rilis H2O terbaru dari http://h2o.ai/download/ dan ikuti perintah yang ditunjukkan pada box di bawah ini:

```
■ unzip h2o-3.2.0.5.zip
cd h2o-3.2.0.5
java -jar h2o.jar
```

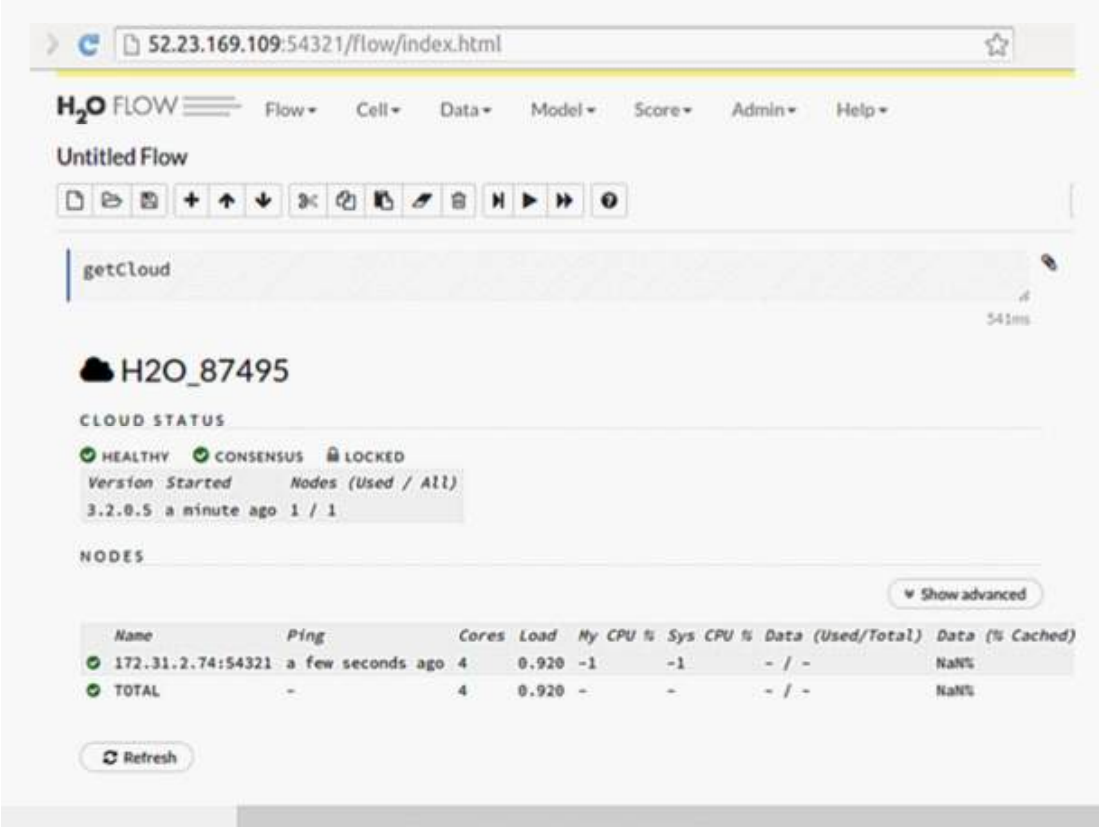
Untuk menjalankan H2O pada kluster Hadoop, unduh rilis H2O khusus untuk versi Hadoop Anda dan ikuti perintah yang ditampilkan di box di bawah ini:

```
■ unzip h2o-3.2.0.5-hdp2.2.zip
cd h2o-3.2.0.5-hdp2.2
hadoop jar h2odriver.jar -nodes 1 -mapperXmx 6g -output hdfsOutputDir
```

Perintah di atas akan meluncurkan cluster H2O dengan satu node dan memori 6GB. URL UI H2O Flow dapat diperoleh dari output seperti yang ditunjukkan di bawah ini:

```
■ Determining driver host interface for mapper->driver callback...
[Possible callback IP address:  172.31.2.74]
[Possible callback IP address:  127.0.0.1]
Using mapper->driver callback IP address and port:  172.31.2.74:41892
(You can override these with -driverif and -driverport.)
Memory Settings:  mapreduce.map.java.opts:  -Xms1g -Xmx1g -XX:PermSize=256m
-verbose:gc -XX:+PrintGCDetails
-XX:+PrintGCTimeStamps -Dlog4j.defaultInitOverride=true
Extra memory percent:  10
mapreduce.map.memory.mb:  1126
15/10/01 05:58:35 INFO impl.TimelineClientImpl:
Timeline service address:  http://ip-172-31-2-74.ec2.internal:8188/ws/v1/timeline/
15/10/01 05:58:35 INFO client.RMProxy:
Connecting to ResourceManager at ip-172-31-2-74.ec2.internal/172.31.2.74:8050
15/10/01 05:58:36 INFO mapreduce.JobSubmitter:  number of splits:1
15/10/01 05:58:36 INFO mapreduce.JobSubmitter:
Submitting tokens for job:  job_1443678868179_0002
15/10/01 05:58:37 INFO impl.YarnClientImpl:
Submitted application application_1443678868179_0002
15/10/01 05:58:37 INFO mapreduce.Job:  The url to track the job:
http://ip-172-31-2-74.ec2.internal:8088/proxy/application_1443678868179_0002/
Job name 'H2O_87495' submitted
JobTracker job ID is 'job_1443678868179_0002'
For YARN users, logs command is
'yarn logs -applicationId application_1443678868179_0002'
Waiting for H2O cluster to come up...
H2O node 172.31.2.74:54321 requested flatfile
Sending flatfiles to nodes...
[Sending flatfile to node 172.31.2.74:54321]
H2O node 172.31.2.74:54321 reports H2O cluster size 1
H2O cluster (1 nodes) is up
(Note:  Use the -disown option to exit the driver after cluster formation)

Open H2O Flow in your web browser:  http://172.31.2.74:54321

(Press Ctrl-C to kill the cluster)
Blocking until the H2O cluster shuts down...
```

Gambar 11.3 menunjukkan sebuah screenshot pada H2O Flow UI.




Gambar 11.3: *H2O Flow UI*

## 11.2 Clustering

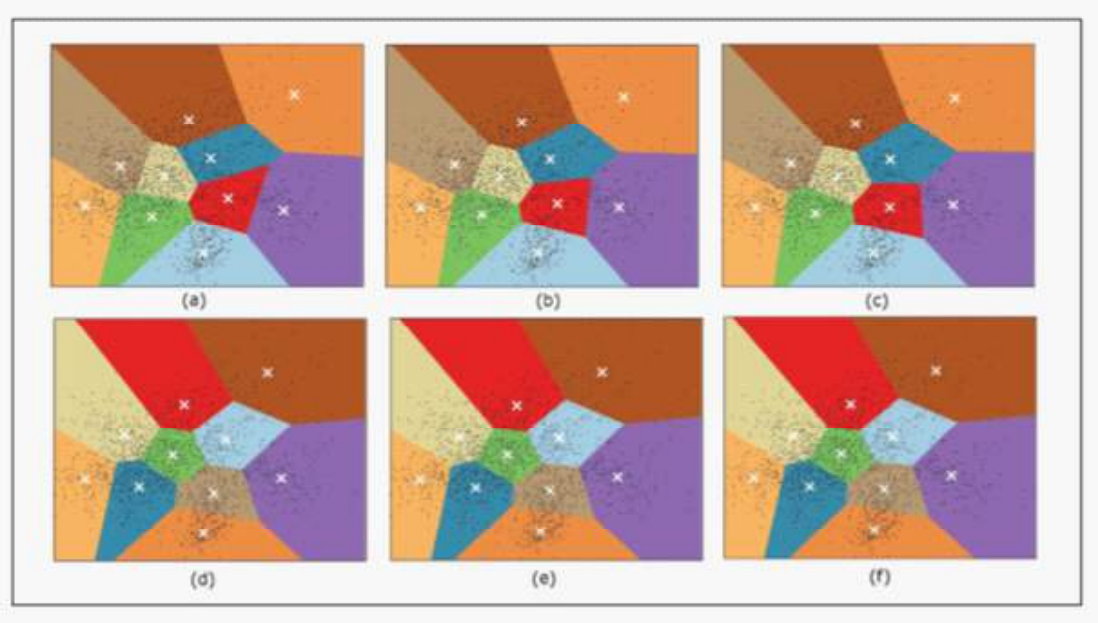
Clustering adalah proses pengelompokan item data yang serupa bersama-sama sehingga item data yang lebih mirip satu sama lain (berkenaan dengan beberapa kriteria kesamaan) daripada item data lainnya diletakkan dalam satu cluster. Pengelompokan big data sangat menarik, dan terjadi dalam aplikasi seperti:

- Mengelompokkan data jejaring sosial untuk menemukan sekelompok pengguna serupa
- Mengelompokkan data catatan kesehatan elektronik (EHR) untuk menemukan pasien serupa.
- Mengelompokkan data sensor untuk mengelompokkan kesalahan serupa atau terkait di mesin
- Mengelompokkan data riset pasar untuk mengelompokkan pelanggan serupa
- Mengelompokkan data aliran klik untuk mengelompokkan pengguna serupa Pengelompokan dicapai oleh algoritme pengelompokan yang termasuk dalam

algoritme kategori luas yang disebut pembelajaran mesin tanpa pengawasan. Algoritme pembelajaran mesin yang tidak diawasi menemukan pola dan struktur tersembunyi dalam data yang tidak memiliki data pelatihan.

### 11.2.1 K-Means

K-means adalah algoritme pengelompokan yang mengelompokkan item data ke dalam kl kluster k, di mana k ditentukan oleh pengguna. Setiap cluster ditentukan oleh titik sentroid. Semua titik dalam cluster lebih dekat (sehubungan dengan beberapa ukuran jarak) ke sentroid mereka dibandingkan dengan sentroid dari cluster tetangga. Pengelompokan K-means dimulai dengan sekumpulan titik sentroid k yang dipilih secara acak dari dataset atau dipilih menggunakan beberapa algoritma inisialisasi seperti pengelompokan kanopi. Algoritme melanjutkan dengan menemukan jarak antara setiap titik data dalam dataset dan titik pusat. Berdasarkan ukuran jarak, setiap titik data ditetapkan ke cluster yang termasuk dalam centroid terdekat. Pada langkah selanjutnya sentroid dihitung ulang dengan mengambil nilai rata-rata dari semua titik data dalam sebuah cluster. Proses ini diulangi sampai sentroid tidak lagi bergerak lebih dari ambang batas yang ditentukan. Algoritme pengelompokan k-means ditunjukkan pada Box 11.1.




Gambar 11.4: *Contoh pengelompokan 300 titik dengan k-means: (a) iterasi 1, (b) iterasi 2, (c) iterasi 3, (d) iterasi 5, (e) iterasi 10, (f) iterasi 100.*

**Box 11.1: k-means clustering algorithm**

```
Mulailah dengan titik sentroid k
    sedangkan sentroid tidak lagi bergerak melampaui ambang batas atau jumlah iterasi
    maksimum tercapai:
        untuk setiap titik dalam kumpulan data:
            untuk setiap sentroid:
                menemukan jarak antara titik dan pusat massa menetapkan titik ke cluster
                milik centroid terdekat
untuk setiap cluster:
    hitung ulang titik pusat dengan mengambil nilai rata-rata dari semua titik dalam kluster
```

Gambar 11.4 menunjukkan contoh pengelompokan 300 titik data. Titik-titik sentroid dihitung ulang setelah setiap literasi, dan dimasukkan ke dalam gambar ini pergerakan sentroid yang hilang setelah sepuluh iterasi.

Ada berbagai ukuran jarak yang dapat digunakan untuk algoritma pengelompokan termasuk:

- Pengukuran jarak euclidean: Ini adalah yang paling sederhana dari semua pengukuran jarak. Jarak Euclidean antara titik p dan q dalam ruang dimensi-N diberikan sebagai:

$$d(p,q) = \sqrt{\sum_{i=1}^{N}(p_i - q_i)^2}$$

- Ukuran jarak kosinus: Ukuran jarak kosinus menemukan garis lintang antara dua vektor (vektor ditarik dari titik awal ke titik).

$$d = cos(\theta) = \frac{A.B}{||A||||B||}$$

- Pengukuran jarak Manhattan: Pengukuran jarak Manhattan adalah jumlah dari perbedaan absolut dari koordinat dua titik yang diberikan sebagai:

$$d(p,q) = \left|\sum_{i=1}^{N}(p_i - q_i)\right|$$

Mari kita lihat contoh pengelompokan k-means menggunakan kerangka kerja H2O. Untuk contoh ini, kami akan menggunakan kumpulan data Wine dari repositori pembelajaran Mesin UCI [39]. Kumpulan data ini memiliki hasil analisis kimiawi anggur yang ditanam di Italia. Analisis kimiawi menentukan jumlah 13 konstituen (seperti alkohol, asam malat, magnesium, dsb.) Yang ditemukan dalam tiga jenis anggur.

Dengan menggunakan dataset wine dan mengabaikan label kelas (tipe wine), mari kita coba mengelompokkan data untuk mengidentifikasi pola dalam data menggunakan H2O. Luncurkan cluster H2O menggunakan perintah berikut:

```
java -jar h2o.jar
```

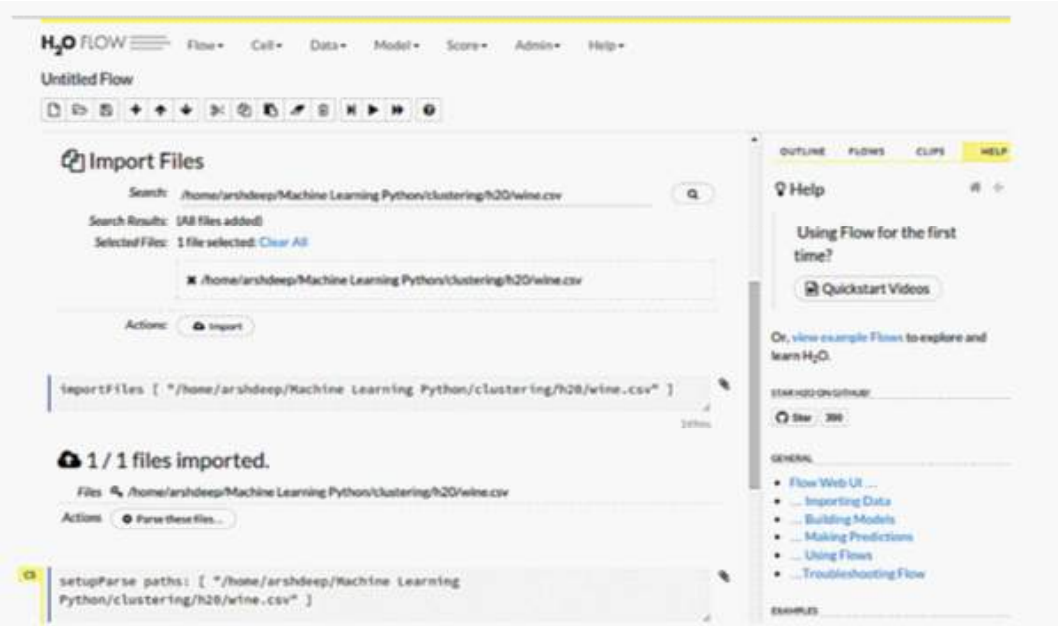
Saat cluster H2O diluncurkan dengan perintah di atas, output akan menyertakan URI dari H2O Flow UI (default untuk mesin lokal adalah http: // localhost: 54321). Dengan cluster H2O diluncurkan, mari kita impor data ke H2O dari H2O Flow UI seperti yang ditunjukkan pada Gambar 11.5. Anda dapat menggunakan perintah H2O importFiles atau memilih opsi impor file dari menu UI H2O Flow. Masukkan jalur ke file data dan tekan tombol impor. Langkah selanjutnya adalah mengurai file yang diimpor. Klik tombol Parse setelah mengimpor file. Gambar 11.6 menunjukkan pengaturan parser. Dalam langkah ini, Anda dapat menentukan berbagai opsi penguraian seperti tipe data untuk setiap kolom, jenis pengurai yang akan digunakan (CSV, XLS, dll.), Pemisah yang digunakan, dll. Untuk sebagian besar penguraian data, H2O secara otomatis mengenali data Tipe. Setelah memilih opsi parse, klik tombol Parse untuk mengurai file. Data dari file yang diurai disimpan dalam bingkai H2O. Gambar 11.7 menunjukkan bingkai H2O yang dibuat dengan mem-parsing file dataset.

Dengan data yang diimpor dan diurai, mari kita sekarang membangun model pengelompokan k-means. Klik tombol Model Build dalam tindakan bingkai yang diurai atau pilih opsi Build Model dari menu. Gambar 11.8 menunjukkan berbagai opsi untuk model. Pilih tipe algoritme menjadi K-means, frame pelatihan sebagai wine.hex, dan masukkan jumlah cluster (k = 3) dan iterasi maksimum (max_iterations = 100). Setelah menentukan opsi model, klik tombol Build Model untuk membangun model.

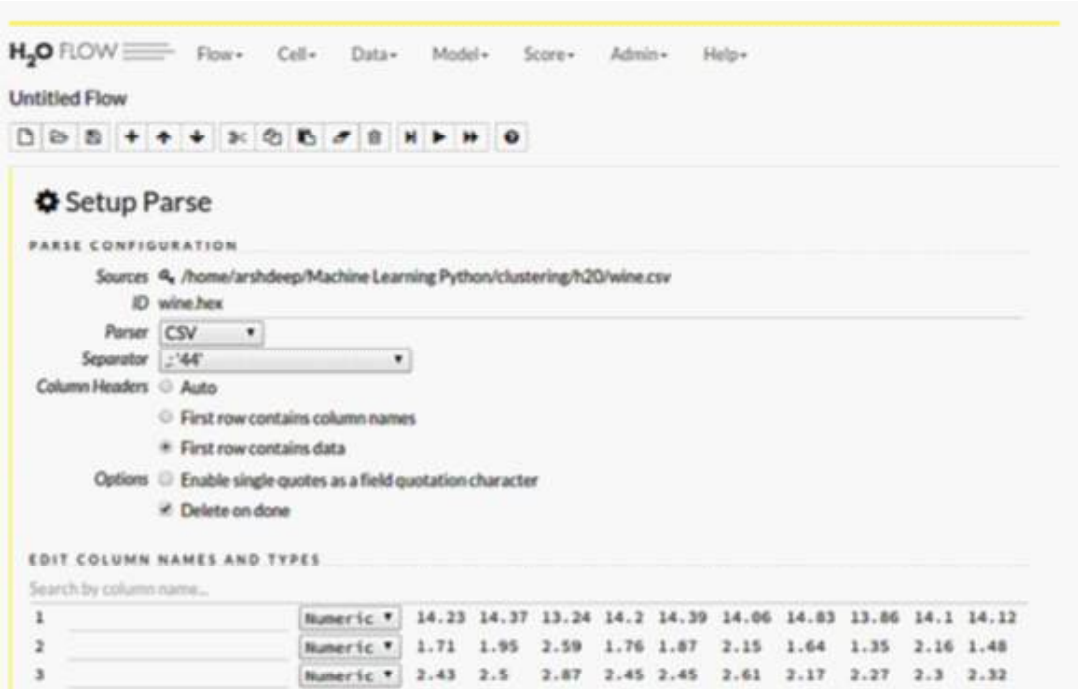
Gambar 11.9 menunjukkan detail model k-means yang dibangun dari dataset Wine. Ringkasan model menunjukkan berbagai statistik model seperti jumlah cluster, jumlah kolom kategorik, jumlah iterasi, dll. Statistik centroid dan mean cluster juga dapat dilihat pada output model.

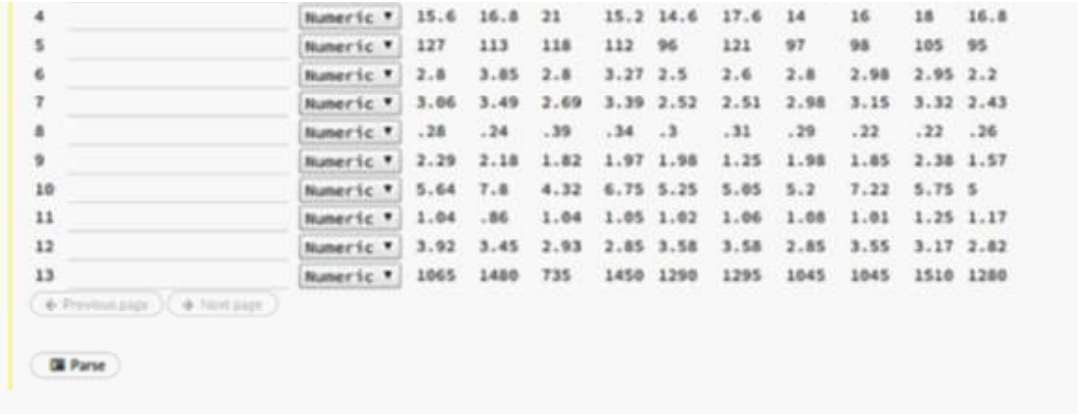
Meskipun nyaman menggunakan H2O Flow UI untuk menganalisis data, untuk kumpulan data yang memerlukan pemrosesan tambahan, Anda dapat mengimplementasikan program Python yang menggunakan H2O Python API. Mari kita lihat implementasi Python untuk mengelompokkan data dengan k-means menggunakan H2O Python API. Box 11.2

menunjukkan program Python untuk pengelompokan k-means menggunakan H2O. Dalam program ini, kami mengimpor pustaka python H2O dan kemudian menginisialisasi H2O. Ini akan meluncurkan cluster H2O baru. Import_frame digunakan untuk mengimpor data ke dalam bingkai H2O. Dengan data yang diimpor, kami menggunakan fungsi kmean untuk membangun model pengelompokan k-means.




Gambar 11.5: *Mengimpor file dataset menggunakan H2O Flow UI*



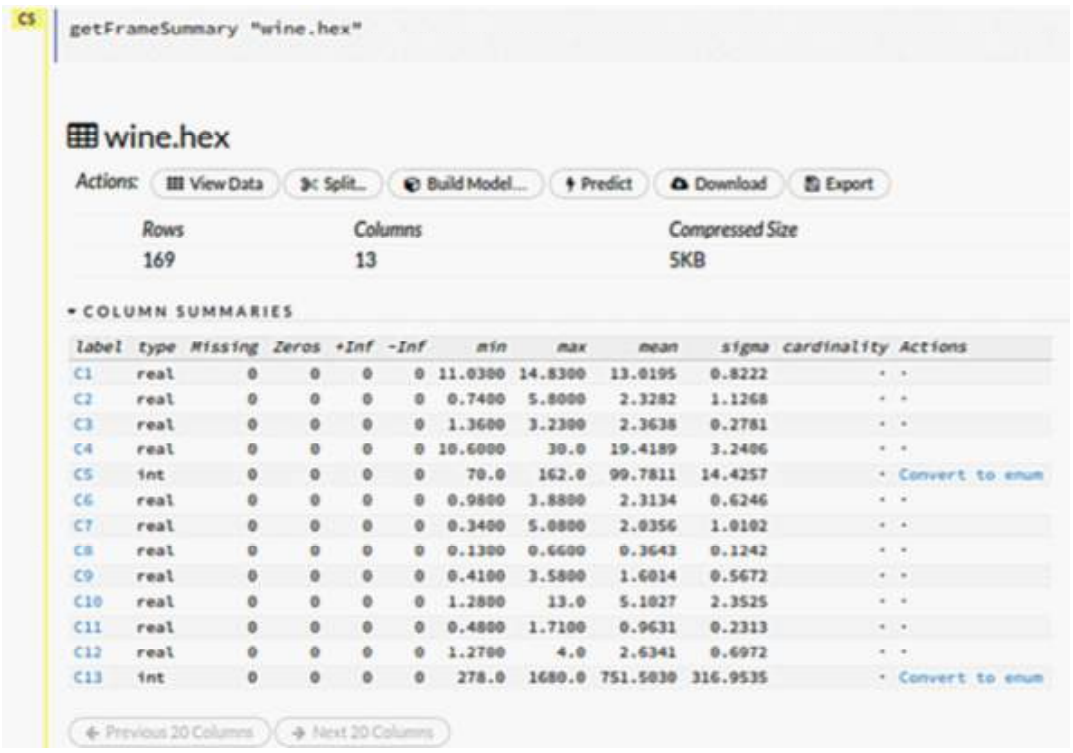


| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | Numeric | 15.6 | 16.8 | 21 | 15.2 | 14.6 | 17.6 | 14 | 16 | 18 | 16.8 |
| 5 | Numeric | 127 | 113 | 118 | 112 | 96 | 121 | 97 | 98 | 105 | 95 |
| 6 | Numeric | 2.8 | 3.85 | 2.8 | 3.27 | 2.5 | 2.6 | 2.8 | 2.98 | 2.95 | 2.2 |
| 7 | Numeric | 3.06 | 3.49 | 2.69 | 3.39 | 2.52 | 2.51 | 2.98 | 3.15 | 3.32 | 2.43 |
| 8 | Numeric | .28 | .24 | .39 | .34 | .3 | .31 | .29 | .22 | .22 | .26 |
| 9 | Numeric | 2.29 | 2.18 | 1.82 | 1.97 | 1.98 | 1.25 | 1.98 | 1.85 | 2.38 | 1.57 |
| 10 | Numeric | 5.64 | 7.8 | 4.32 | 6.75 | 5.25 | 5.05 | 5.2 | 7.22 | 5.75 | 5 |
| 11 | Numeric | 1.04 | .86 | 1.04 | 1.05 | 1.02 | 1.06 | 1.08 | 1.01 | 1.25 | 1.17 |
| 12 | Numeric | 3.92 | 3.45 | 2.93 | 2.85 | 3.58 | 3.58 | 2.85 | 3.55 | 3.17 | 2.82 |
| 13 | Numeric | 1065 | 1480 | 735 | 1450 | 1290 | 1295 | 1045 | 1045 | 1510 | 1280 |

Previous page Next page

Parse

Gambar 11.6: *Parsing file dataset menggunakan H2O Flow UI*



cs `getFrameSummary "wine.hex"`

wine.hex

Actions: View Data | Split... | Build Model... | Predict | Download | Export

| Rows | Columns | Compressed Size |
|---|---|---|
| 169 | 13 | 5KB |

▪ COLUMN SUMMARIES

| label | type | Missing | Zeros | +Inf | -Inf | min | max | mean | sigma | cardinality | Actions |
|---|---|---|---|---|---|---|---|---|---|---|---|
| C1 | real | 0 | 0 | 0 | 0 | 11.0300 | 14.8300 | 13.0195 | 0.8222 | - | - |
| C2 | real | 0 | 0 | 0 | 0 | 0.7400 | 5.8000 | 2.3282 | 1.1268 | - | - |
| C3 | real | 0 | 0 | 0 | 0 | 1.3600 | 3.2300 | 2.3638 | 0.2781 | - | - |
| C4 | real | 0 | 0 | 0 | 0 | 10.6000 | 30.0 | 19.4189 | 3.2406 | - | - |
| C5 | int | 0 | 0 | 0 | 0 | 70.0 | 162.0 | 99.7811 | 14.4257 | - | Convert to enum |
| C6 | real | 0 | 0 | 0 | 0 | 0.9800 | 3.8800 | 2.3134 | 0.6246 | - | - |
| C7 | real | 0 | 0 | 0 | 0 | 0.3400 | 5.0800 | 2.0356 | 1.0102 | - | - |
| C8 | real | 0 | 0 | 0 | 0 | 0.1300 | 0.6600 | 0.3643 | 0.1242 | - | - |
| C9 | real | 0 | 0 | 0 | 0 | 0.4100 | 3.5800 | 1.6014 | 0.5672 | - | - |
| C10 | real | 0 | 0 | 0 | 0 | 1.2800 | 13.0 | 5.1027 | 2.3525 | - | - |
| C11 | real | 0 | 0 | 0 | 0 | 0.4800 | 1.7100 | 0.9631 | 0.2313 | - | - |
| C12 | real | 0 | 0 | 0 | 0 | 1.2700 | 4.0 | 2.6341 | 0.6972 | - | - |
| C13 | int | 0 | 0 | 0 | 0 | 278.0 | 1680.0 | 751.5030 | 316.9535 | - | Convert to enum |

Previous 20 Columns Next 20 Columns

Gambar 11.7: *Melihat bingkai H2O yang dibuat dari file dataset yang diurai*

**▪ Box 11.2: Python program for k-means clustering using H2O**

```
import h2o

h2o.init()

data = h2o.import_frame(path="/home/ubuntu/wine.data.txt")

data.describe()
```


Gambar 11.8: *Membangun model pengelompokan k-means menggunakan H2O Flow* UI

```
model = h2o.kmeans(x=data[1:], k=3,
init="Random", seed=2, standardize=True)

total_within_sumofsquares = model.tot_withinss()
number_of_clusters = len(model.centers())
number_of_dimensions = len(model.centers()[0])
number_of_rows = sum(model.size())
```

Mari kita ulangi contoh clustering menggunakan Spark MLlib. Box 11.3 menunjukkan program Python untuk pengelompokan data menggunakan Spark MLlib. Spark MLlib menyertakan implementasi paralel k-means yang dapat digunakan untuk mengelompokkan big data. Program ini dapat berjalan di shellPySpark. Dalam program ini kami mengimplementasikan fungsi parseVector yang mengambil setiap baris dari file input, membagi baris menjadi kolom individu yang dipisahkan oleh koma, mengubah nilai menjadi mengambang dan mengembalikan array numpy Python. Kelas K - Means dari modul pengelompokan MLlib digunakan untuk membangun model pengelompokan k-means. Setelah model dibangun, metode clusterCenters dari kelas KMeans dapat digunakan untuk melihat pusat cluster.


Gambar 11.9: *H2O Flow UI yang menampilkan detail model pengelompokan k-means*

**Box 11.3: Python program for k-means clustering using Spark MLlib**

```
import numpy as np
from pyspark import SparkContext
from pyspark.mllib.clustering import KMeans


def parseVector(line):
   return np.array([float(x) for x in line.split(',')])


sc = SparkContext(appName="KMeans")

lines = sc.textFile('file:///home/hadoop/wine.data.txt')

data = lines.map(parseVector)

k = 3

model = KMeans.train(data, k)

print("Final centers:  " + str(model.clusterCenters))
```

## Studi Kasus

### Sistem Rekomendasi Lagu

Pada bagian ini, kami akan menjelaskan studi kasus tentang membangun sistem rekomendasi musik menggunakan Apache Spark. Meskipun sebagian besar sistem rekomendasi lagu menggunakan pemfilteran kolaboratif, yang memerlukan akses ke data profil pengguna seperti lagu yang dimainkan oleh pengguna dan peringkat lagu yang diberikan pengguna, dengan tidak adanya informasi tersebut, akan sulit untuk merekomendasikan lagu kepada pengguna. Untuk tujuan studi kasus ini, kami akan menggunakan pendekatan pemfilteran berbasis konten yang tidak memerlukan informasi tentang pengguna, melainkan memanfaatkan properti lagu untuk merekomendasikan lagu baru kepada pengguna.

Untuk studi kasus ini, kami akan menggunakan Kumpulan Data Juta Lagu [44]. Dataset berisi informasi tentang sejuta lagu kontemporer seperti metadata lagu, metadata artis, dan fitur akustik lagu. File dataset tersedia dalam format HDF5. Untuk studi kasus ini, kami akan menggunakan subset dari data. Box 11.4 menunjukkan kode Python untuk membaca file H5 dan mengekstraksi bidang metadata yang menarik ke dalam file CSV.

**Box 11.4: Python program for reading H5 files and exporting meta-data to CSV**

```
import hdf5_getters as GETTERS
import csv

h5 = GETTERS.open_h5_file_read("input.h5")
c = csv.writer(open("data.csv", "w+"))

all_artist_id = []
all_artist_names = []
all_songs_id = []
all_songs_names = []
all_tempo = []
all_loudness = []
all_key_confidence = []
all_mode_confidence = []
all_song_hottness=[]
all_year =[]

artist_id = GETTERS.get_artist_id(h5)
all_artist_id.append(artist_id)

artist_name = GETTERS.get_artist_name(h5)
all_artist_names.append(artist_name)

song_id = GETTERS.get_song_id(h5)
all_songs_id.append(song_id)

song_name = GETTERS.get_title(h5)
all_songs_names.append(song_name)

loudness = GETTERS.get_loudness(h5)
all_loudness.append(loudness)

song_hottness = GETTERS.get_song_hotttnesss(h5)
all_song_hottness.append(song_hottness)

tempo = GETTERS.get_tempo(h5)
all_tempo.append(tempo)

key_confidence = GETTERS.get_key_confidence(h5)
all_key_confidence.append(key_confidence)

mode_confidence = GETTERS.get_mode_confidence(h5)
all_mode_confidence.append(mode_confidence)

year=GETTERS.get_year(h5)
all_year.append(year)

for k in range(len(list(all_artist_names))):
   artist_id=list(all_artist_id)[k]
   artist_name=list(all_artist_names)[k]
   song_id=list(all_songs_id)[k]
```

```
song_name=list(all_songs_names)[k]
loudness=list(all_loudness)[k]
tempo=list(all_tempo)[k]
key_confidence=list(all_key_confidence)[k]
mode_confidence=list(all_mode_confidence)[k]
year=list(all_year)[k]

c.writerow([artist_id, artist_name, song_id, song_name,
 loudness, tempo,key_confidence, mode_confidence, year])
```

Box di bawah ini menunjukkan format dan contoh dari meta-data yang diekstrak ke dalam file CSV dari dataset asli.

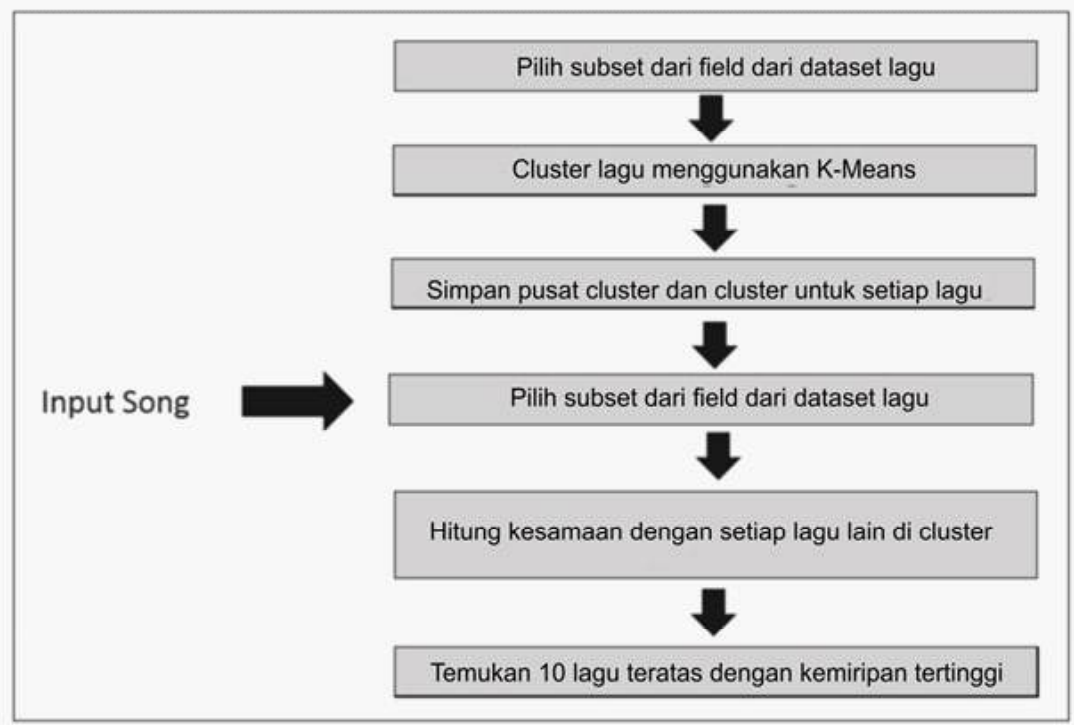
```
■ #Sample of data extracted into CSV
Artist ID, Artist Name, Song ID, Song Name, Loudness, Song Hottness, Tempo, Key Confidence, Mode Confidence,
Year
ARE26EG1187B990AEF, Sunscreem, SOICLQB12A8C13637C, Exodus, -8.955, 0.79, 130.201, 0.625, 0.558, 1995
```

Box 11.5 menunjukkan program Spark untuk menemukan 10 lagu teratas dan 10 artis teratas setiap tahun. Program ini mengambil file CSV yang dihasilkan pada langkah sebelumnya sebagai masukan. Transformasi map digunakan untuk membagi nilai yang dipisahkan koma menjadi kata-kata. Baris kosong kemudian disaring. Langkah selanjutnya adalah memetakan nilai yang dipisahkan koma menjadi (artist_id, artist_name, artist_ hotness, year) dan (song_id, song_name, song_hotness, year). Ada artis dengan banyak lagu, jadi transformasi yang berbeda () digunakan untuk menemukan artis yang unik sehingga tidak diulang saat menghitung artis top. Selanjutnya, loop for digunakan untuk menghitung artis dan lagu teratas setiap tahun. Langkah selanjutnya adalah mengubah kunci dari artist_id dan song_idtoartist_hotnessandsong_hotnesswhich yang dilakukan dengan transformasi map lain. Namun, beberapa lagu memiliki nilai song_hotness sebagai NaN. Jadi nilai-nilai ini harus disaring. Operator sortByKey digunakan untuk menyusun entri dalam urutan menurun nilai hotness dan operator take digunakan untuk mengembalikan daftar 10 lagu dan artis teratas.

**■ Box 11.5: Spark program for finding the top 10 songs and top 10 artists in each year**

```
from pyspark import SparkContext

sc = SparkContext("local", "Simple App")

step1=sc.textFile("file:///output_csv.csv")
step2=step1.map(lambda line:  line.split(",")).filter(lambda line:
len(line)>1)

step3=step2.map(lambda line:  ((str(line[1].encode('utf-8')),
   str(line[0].encode('utf-8')))))

step4=step2.map(lambda line:  ((line[1].encode('utf-8'),
   line[0].encode('utf-8'),float(line[4]),int(line[9])))).distinct()

for i in range (10)
   inp_year = 1990 + i

   step5=step4.filter(lambda line:  line[3]==inp_year).map(lambda
      line:  ((float(line[2]),(line[0],line[1],line[3]))))

   step6=step5.sortByKey(False)

   #Emit top 10 artists
   topArtists=step6.take(10)

   step8=step2.map(lambda line:  ((line[3].encode('utf-8'),
        line[2].encode('utf-8'),float(line[5]),int(line[9])))).distinct()

   step9=step8.filter(lambda line:  line[3]==inp_year).map(lambda line:
        ((float(line[2]),(line[0],line[1],line[3])))).filter(lambda
        line:  not math.isnan(line[0]))

   step10=step9.sortByKey(False)

   #Emit top 10 songs
   topSongs=step10.take(10)

sc.stop
```

Untuk merekomendasikan lagu yang serupa, lagu-lagu tersebut pertama-tama dikelompokkan menjadi 10 kelompok. Lagu dalam cluster tertentu lebih berkorelasi / mirip satu sama lain dibandingkan dengan lagu di cluster lain. MLlib Spark menyediakan implementasi algoritme pengelompokan K-Means. Setelah mengelompokkan lagu, kami menambahkan informasi cluster yang sesuai dari setiap lagu ke dalam file CSV yang juga memiliki metadata lagu. Pusat cluster disimpan dalam file acar. Gambar 11.10 menunjukkan langkah-langkah dalam rekomendasi lagu.




Gambar 11.10: *Langkah-langkah terkait dalam rekomendasi lagu*

Untuk merekomendasikan lagu, pengguna memberikan ID lagu sebagai input. Sistem kemudian mencari file CSV yang berisi informasi cluster untuk menemukan cluster yang sesuai dan memuat file acar untuk menemukan pusat cluster yang sesuai. Langkah selanjutnya adalah menemukan setiap lagu lain dalam cluster yang sesuai selain lagu input pengguna dan menghitung jarak Euclidean antara setiap lagu dan lagu input pengguna. Perhatikan bahwa, setelah pengelompokan, ruang pencarian berkurang secara signifikan dan pencarian ini berpotensi dilakukan secara real time. Lagu-lagu dalam cluster kemudian diurutkan menurut jarak Euclidean mereka dengan lagu input pengguna dan 10 lagu teratas dipilih untuk ditampilkan kepada pengguna.

Box 11.6 menunjukkan program Spark untuk mengelompokkan lagu, dan Box 11.7 menunjukkan program Spark untuk merekomendasikan lagu yang serupa.

**Box 11.6: Spark program for clustering the songs**

```
from pyspark import SparkContext
from pyspark.mllib.clustering import KMeans
from math import sqrt
import sys
import pickle
```

```
sc = SparkContext("local", "App")

def closestPoint(p,centers):
   bestIndex = 0
   closest = float("+inf")
   for i in range(len(centers)):
      tempDist = np.sum((p - centers[i]) ** 2)
      if tempDist < closest:
         closest = tempDist
         bestIndex = i
   return bestIndex

Parseddata=sc.textFile("file:///data.csv").map(lambda line:
line.split(",")).filter(lambda line:
len(line)>1).map(lambda line:
(float(line[4]),float(line[5]),float(line[6]),float(line[7])))

clusters = KMeans.train(Parseddata, 10, maxIterations=100,
   runs=100, initializationMode="random")

print str(clusters.clusterCenters)

cc=clusters.clusterCenters
pickle.dump(cc,open("cluster_centers.p","wb"))

PD1 = sc.textFile("file:///data.csv").map(lambda line:
line.split(",")).filter(lambda line:  len(line)>1).map(lambda p:
(p[0].encode('utf-8'),p[1].encode('utf-8'),p[2].encode('utf-8'),
p[3].encode('utf-8'),float(p[4]),float(p[5]),float(p[6]),
float(p[7]),closestPoint([float(p[4]),float(p[5]),
float(p[6]),float(p[7])],cc),p[8].encode('utf-8')))

PD1.saveAsTextFile("file:///data_clustered.txt")

logData = sc.textFile("file:///data_updated.csv").cache()

sc.stop()
```

**Box 11.7: Spark program for recommending similar songs**

```
from pyspark import SparkContext
from operator import add
from pyspark.mllib.clustering import KMeans
import numpy as np
```

```
from numpy import array
from math import sqrt
import sys
import pickle
import re
import datetime
sc = SparkContext("local", "Simple App")
```

```
def g(x):
    print x

def line_strip(p):
   a = p[0].encode('ascii')
   b = p[1].encode('ascii')
   c = (p[2].encode('ascii')).strip(" ")
   d = p[3].encode('ascii')
   e=float(p[4])
   f=float(p[5])
   g=float(p[6])
   h=float(p[7])
   i=int(p[8].encode('ascii'))
   k = p[9].encode('ascii')
   j=(a,b,c,d,e,f,g,h,i,k)
   return j
```

```
def euclid_dist(p,user_point,center):
   a=user_point-center
   b=p-center

   c=np.array([(user_point[0]/center[0]-1),(user_point[1]/center[1]-1),
      (user_point[2]/center[2]-1),(user_point[3]/center[3]-1)])

   d=np.array([(p[0]/center[0]-1),(p[1]/center[1]-1),
      (p[2]/center[2]-1),(p[3]/center[3]-1)])

   dist1=np.linalg.norm(c)
   dist2= np.linalg.norm(d)
   dot=np.dot(c,d)
   cosine=dot/dist1/dist2
   similarity=np.sqrt(cosine*cosine+(dist1-dist2)*(dist1-dist2))
   return str(similarity)

PD4_read=sc.textFile("file:////data.csv")
PD4_read_count=PD4_read.count()
PD4_iter=PD4_read.map(lambda line:  line.split(",")).filter(lambda
line:  len(line)>1).map(lambda line:line_strip(line)).collect()

for item in range(len(PD4_iter)):
   input_user=PD4_iter[item][2]

   PD4_read2=PD4_read.map(lambda line:  line.split(",")).filter(lambda
      line:  len(line)>1).map(lambda line:line_strip(line))

   PD4_filt=PD4_read2.filter(lambda line:(line[2]==input_user))
```

```
   PD4_filt.foreach(g)
   CF=PD4_filt.collect()

   Cl_found=CF[0][8]
   print 'Cluster Found=',Cl_found
   cc = pickle.load(open("cluster_centers.p", "rb" ))
   print 'Cluster Center=',cc[Cl_found][0],',',
```

```
    cc = pickle.load(open("cluster_centers.p", "rb" ))
    print 'Cluster Center=',cc[Cl_found][0],',',
       cc[Cl_found][1],',',cc[Cl_found][2],',',
       cc[Cl_found][3],'',type(Cl_found)

    PD6=PD4_read2.filter(lambda line:
    (line[8]==Cl_found)).filter(lambda line:(line[2]!=input_user))

    PD7=PD6.map(lambda p:(euclid_dist(np.asarray((p[4],p[5],p[6],p[7])),
       np.asarray((CF[0][4],CF[0][5],CF[0][6],CF[0][7])),cc[Cl_found]),
       (p[0],p[1],p[2],p[3]))).sortByKey().collect()

    post = {"name":  (CF[0][3]).encode('ascii'),
          "artist":(CF[0][1]).encode('ascii'),
          "spark-id":(CF[0][2]).encode('ascii'),
          "year":(CF[0][9]).encode('ascii'),
          "tags":  ["mongodb", "python", "pymongo"],
          "date":  datetime.datetime.utcnow()}
    post["similar"]=[]

    print 'Similar Songs are '

    for item in range(10):
         song = {"name":  (PD7[item][1][3]).encode('ascii'),
            "artist":(PD7[item][1][1]).encode('ascii'),
            "spark-id":(PD7[item][1][2]).encode('ascii')}
         post["similar"].append(song)

    print post

sc.stop()
```

## 11.3 Klasifikasi dan Regresi

Klasifikasi adalah proses mengkategorikan objek ke dalam kategori yang telah ditentukan sebelumnya. Klasifikasi dicapai dengan algoritma klasifikasi yang termasuk dalam kategori algoritma yang luas yang disebut pembelajaran mesin yang diawasi. Pembelajaran yang diawasi melibatkan pengambilan model dari sekumpulan data input dan respons yang diketahui ke data (data pelatihan) dan kemudian menggunakan model yang disimpulkan untuk memprediksi respons ke data baru. Ada berbagai jenis pendekatan klasifikasi untuk analisis big data termasuk:

- Klasifikasi biner: Klasifikasi biner melibatkan pengelompokan data ke dalam dua kategori. Misalnya, mengklasifikasikan sentimen artikel berita menjadi positif atau negatif, mengklasifikasikan keadaan mesin menjadi baik atau rusak, mengklasifikasikan tes kesehatan menjadi positif atau negatif, dll.
- Klasifikasi multi-kelas: Klasifikasi multi-kelas melibatkan lebih dari dua kelas yang didata. Misalnya, masalah klasifikasi ekspresi gen melibatkan banyak kelas.

- Klasifikasi dokumen: Klasifikasi dokumen adalah jenis pendekatan klasifikasi multi kelas dimana data yang akan diklasifikasi berbentuk dokumen teks. Misalnya, mengklasifikasikan artikel berita ke dalam berbagai kategori seperti politik, olahraga, dll.

***Regresi***

Sedangkan dalam klasifikasi, variabel respon adalah kategorikal dan tidak berurutan (misalnya ya / tidak atau positif / negatif dalam klasifikasi biner) di Regresi, variabel respon mengambil nilai kontinu. Regresi melibatkan pemodelan hubungan antara variabel dependen (variabel respon) dan satu atau lebih variabel independen. Misalnya, dalam regresi linier, variabel terikat y dimodelkan sebagai kombinasi linier dari variabel bebas. Dalam regresi, tujuannya adalah untuk mempelajari fungsi h (x) dari set pelatihan, yang dapat memprediksi nilai y. Fungsi h (x) disebut hipotesis. Untuk regresi linier,

$$h(x) = \sum_{i=0}^{n} \theta_i x_i$$

Dimana θ0,θ1, ...,θn adalah parameter.

### 11.3.1 Metrik Evaluasi Kinerja

Masalah klasifikasi Forabinary (dengan dua kelas Positif dan Negatif) kita dapat memiliki empat kemungkinan kasus: (1) Untuk kelas Positif jika prediksinya adalah Positif maka ini adalah TruePositive, (2) Untuk kelas Positif jika prediksinya Negatif maka ini adalah FalseNegative, (3) Untuk kelas Negatif jika prediksinya Negatif maka ini adalah True Negative, (4) Untuk kelas Negatif jika prediksinya adalah Positif maka ini adalah False Positive, Algoritma klasifikasi kinerja dapat dievaluasi menggunakan metrik berikut:

- True Positive Rate (TPR) / Sensitivity / Recall: TruePositiveRate (TPR) juga disebut Sensitivity atau Recall adalah pecahan dari positif yang diklasifikasikan dengan benar.

$$\textbf{TPR} = \frac{\textit{True Positive}}{\textit{(True Positive + False Negative)}}$$

- True Negative Rate (TNR) / Speci fi city: True Negative Rate (TPR) juga disebut Specificity adalah pecahan dari negatif yang diklasifikasikan dengan benar.

$$\textbf{TNR} = \frac{\textit{True Negative}}{\textit{(True Negative + False Positive)}}$$

- FalsePositiveRate (FPR): False Positive Rate (FPR) didefinisikan sebagai,

$$\textbf{FPR} = \frac{\textit{True Positive}}{\textit{(False Positive + True Negative)}}$$

- Presisi: Presisi adalah bagian dari objek yang diklasifikasikan dengan benar. Presisi didefinisikan sebagai,

$$\textbf{Presisi} = \frac{\textit{True Positive}}{\textit{(True Positive + False Positive)}}$$

- Akurasi: Akurasi didefinisikan sebagai,

$$\textbf{Presisi} = \frac{\textit{True Positive + True Negative}}{\textit{TruePositive + TrueNegative + FalsePositive + FalseNegative)}}$$

- F1-score: F1-score adalah ukuran akurasi yang mempertimbangkan presisi dan recall. F1-score adalah alat presisi dan recall yang diberikan sebagai,

$$\textbf{\textit{F1} – Score} = \frac{\textit{2 (Precision)(Recall)}}{\textit{(Precision + Recall)}}$$

- Kurva Receiver Operating Characteristics (ROC): Kurva ROC adalah plot dari True Positive Rate (TPR) versus False Positive Rate (FPR). Untuk nilai yang berbeda dari ambang batas diskriminasi (ambang batas untuk propiglitas di atas yang kami pilih kelas positif), kami mendapatkan sejumlah pasangan nilai (TPR, FPR).
- Area Under Curve (AUC): AUC adalah area di bawah kurva KOP.
- Mean Squared Error (MSE): Mean Squared Error adalah rata-rata dari jumlah kuadrat kesalahan antara nilai perkiraan dan nilai sebenarnya.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(h(x^{(i)}) - y^{(i)})^2 \qquad (11.10)$$

- Koefisien Determinasi (R2): Koefisien Determinasi disebut juga R2 atau R-Squared, adalah ukuran seberapa baik model mampu menjelaskan variasi data. R2 didefinisikan sebagai,

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y^{(i)} - h(x^{(i)}))^2}{\sum_{i=1}^{n}(y^{(i)} - \mu)^2} = 1 - \frac{SSE}{SST} \qquad (11.11)$$

dimana SSE adalah jumlah sisa kuadrat dan SST adalah jumlah total kuadrat. R2 bervariasi antara 0 dan 1. R2 = 1 berarti model menjelaskan semua variabilitas data di sekitar meannya.

### 11.3.2 Naive Bayes

Naive Bayes adalah algoritma klasifikasi propiglistik yang didasarkan pada teorema Bayes dengan asumsi naif tentang independensi atribut fitur. Diberikan variabel kelas C dan variabel fiturF1, ..., Fn, propiglitas bersyarat (posterior) menurutBayestheorem diberikan sebagai,

$$P(C|F1,...,Fn) = \frac{P(F1,...,Fn|C)P(C)}{P(F1,...,Fn)}$$

di mana, P (C | F1, ..., Fn) adalah propiglitas posterior, P (F1, ..., Fn | C) adalah kemungkinan dan P (C) adalah propiglitas sebelumnya dan P (F1, ... , Fn) adalah buktinya. Naive Bayes membuat asumsi naif tentang kemandirian setiap pasang fitur yang diberikan sebagai,

$$P(F_1,...,F_n|C) = \prod_{i=1}^{n} P(F_i|C) \quad (11.13)$$

Dalam prakteknya, karena bukti P (F1, ..., Fn) konstan untuk input yang diberikan dan tidak bergantung pada variabel kelas C, hanya pembilang dari propiglitas posterior yang penting untuk klasifikasi. Oleh karena itu kami mendapatkan,

$$P(C|F_1,...,F_n) \propto P(C)\prod_{i=1}^{n} P(F_i|C) \quad (11.14)$$

Dengan penyederhanaan tersebut maka dapat dilakukan klasifikasi sebagai berikut,

$$C = argmax_C P(C)\prod_{i=1}^{n} P(F_i|C) \quad (11.15)$$

Ada beberapa versi berbeda dari Naive Bayes yang berbeda dalam asumsi naif yang dibuat. Beberapa diantaranya adalah:

- Gaussian Naive Bayes: Gaussian Naive Bayes mengasumsikan kemungkinan P (F1, ..., Fn | C) sebagai,

$$P(F_1,...,F_n|C) = \prod_{i=1}^{n} P(F_i|C) \quad (11.16)$$

Dimana

$$P(F_1,...,F_n|C) = \prod_{i=1}^{n} P(F_i|C) \quad (11.16)$$

- dimana μ adalah mean dan σC adalah deviasi standar untuk nilai-nilai dalam Fi di kelas C. Gaussian Naive Bayes cocok untuk masalah di mana variabel fitur memiliki nilai kontinu yang diasumsikan memiliki distribusi Gaussian.
- MultinomialNaiveBayes: Multinomial Naive Bayes menggunakan distribusi multinomial untuk setiap variabel fitur. Ini cocok untuk masalah yang memiliki fitur diskrit seperti klasifikasi dokumen.

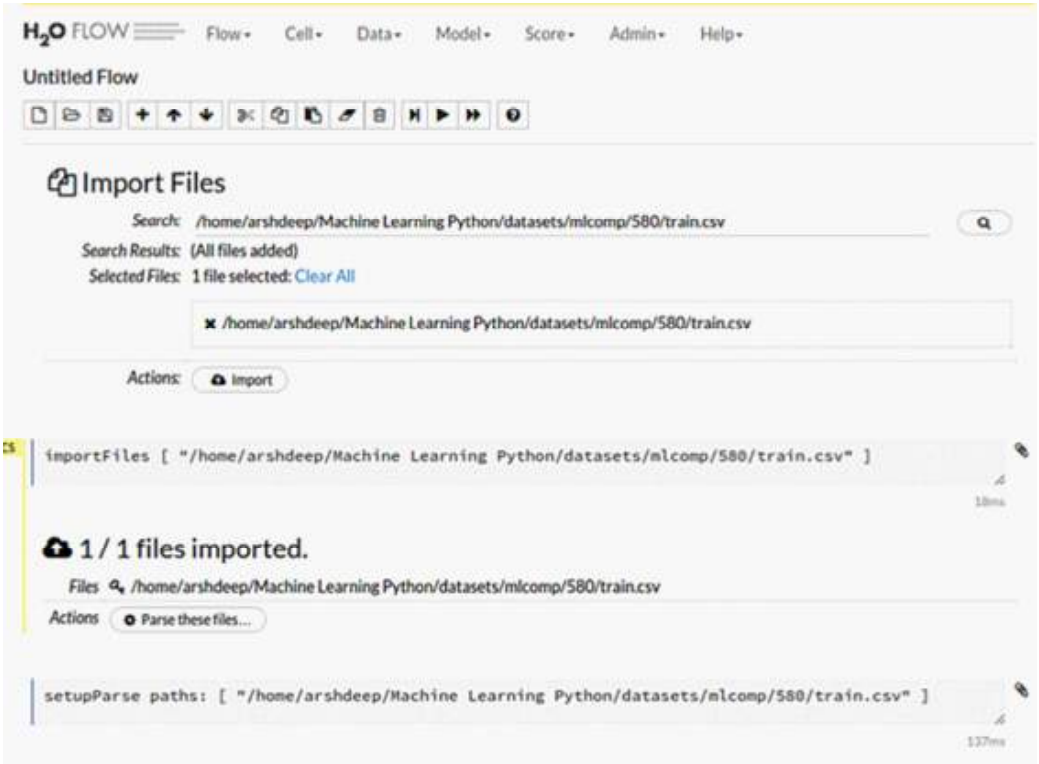
- Bernoulli Naive Bayes: Bernoulli Naive Bayes juga cocok untuk masalah yang memiliki fitur diskrit. Kemungkinan di Bernoulli Naive Bayes adalah sebagai berikut,

$$P(Fi|C) = P(i|Cfi(1-P(i|C))(1-Fi)$$

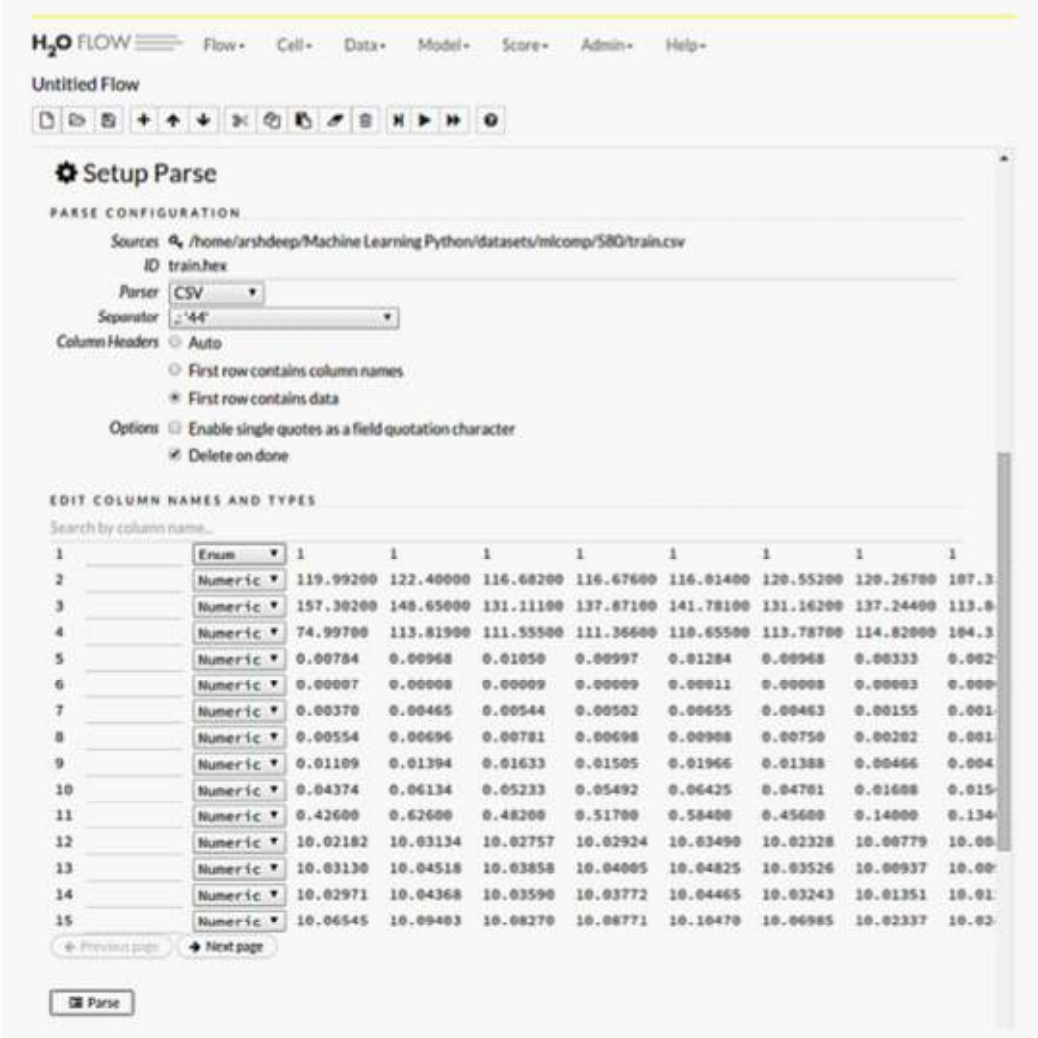
di mana setiap fitur diasumsikan bernilai biner.

Mari kita lihat contoh klasifikasi Naive Bayes menggunakan kerangka kerja H2O. Untuk contoh ini, kita akan menggunakan dataset UCI Parkinsons [43]. Dataset ini terdiri dari berbagai pengukuran suara biomedis. Setiap kolom mewakili ukuran suara tertentu. Kolom status memiliki nilai 1 untuk orang yang menderita penyakit Parkinson dan 0 untuk orang yang tidak menderita penyakit tersebut. Kami akan menggunakan dataset pelatihan untuk melatih model Naive Bayes dan kemudian menggunakan model tersebut untuk membuat prediksi untuk dataset pengujian. (yaitu mengklasifikasikan orang menjadi dua kategori: mereka yang memiliki penyakit Parkinson dan mereka yang tidak memiliki penyakit tersebut).

Langkah pertama adalah mengimpor file dataset ke H2O dari H2O Flow UI seperti yang ditunjukkan pada Gambar 11.11.

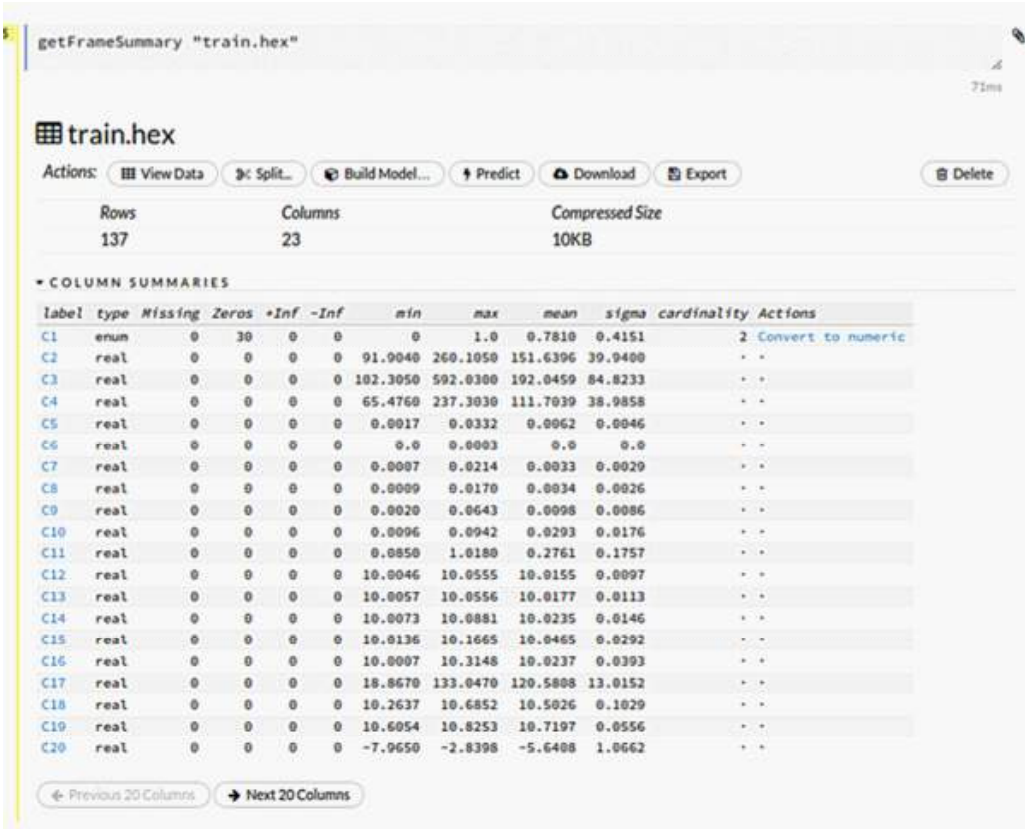



Gambar 11.11: *Mengimpor file dataset menggunakan H2O Flow UI*



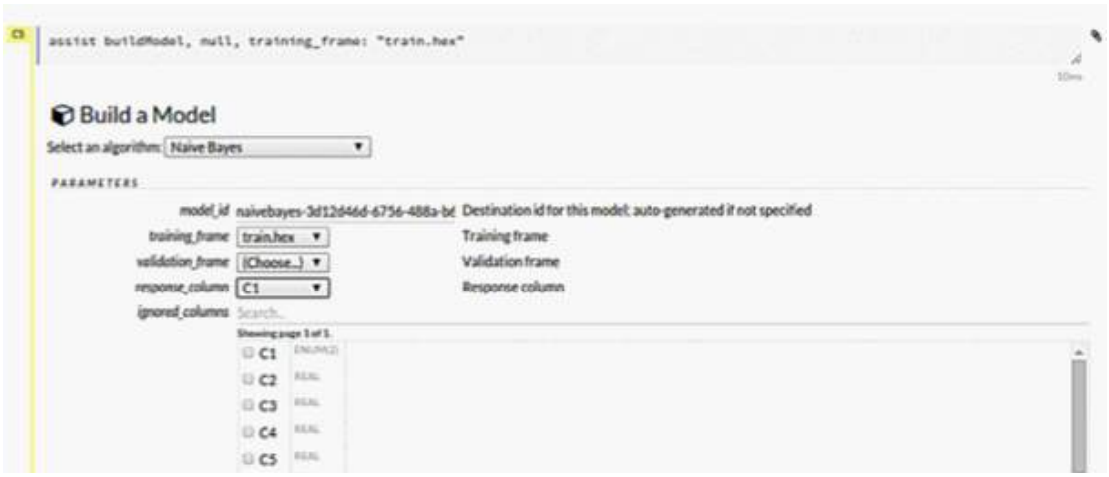

Gambar 11.12: *Parsing file dataset menggunakan H2O Flow UI*

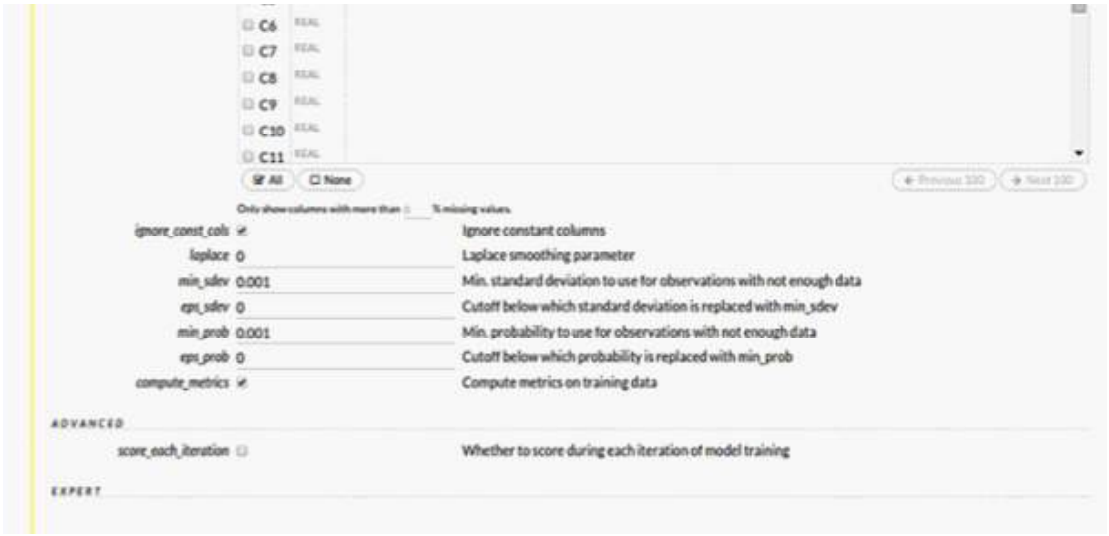
Selanjutnya, kami mengurai file dataset seperti yang ditunjukkan pada Gambar 11.12. Pada langkah ini, Anda dapat menentukan berbagai opsi penguraian. Setelah memilih opsi parse, klik tombol Parse untuk mengurai file. Data dari file yang diurai disimpan dalam bingkai H2O. Gambar 11.13 menunjukkan frame H2O yang dibuat dengan mem-parsing file dataset.




Gambar 11.13: *Menunjukkan bingkai yang dibuat dari file data set yang diurai*

Dengan data yang diimpor dan diurai, sekarang mari kita membangun model klasifikasi Naive Bayes. Klik tombol Build Model dalam tindakan frame yang diurai atau pilih opsi Build Model dari menu. Gambar 11.14 menunjukkan berbagai opsi untuk model. Pilih tipe algoritma menjadi Naive Bayes, frame pelatihan sebagai train.hex dan kolom respon sebagai C1. Setelah menentukan opsi model, klik tombol Build Model untuk membangun model.



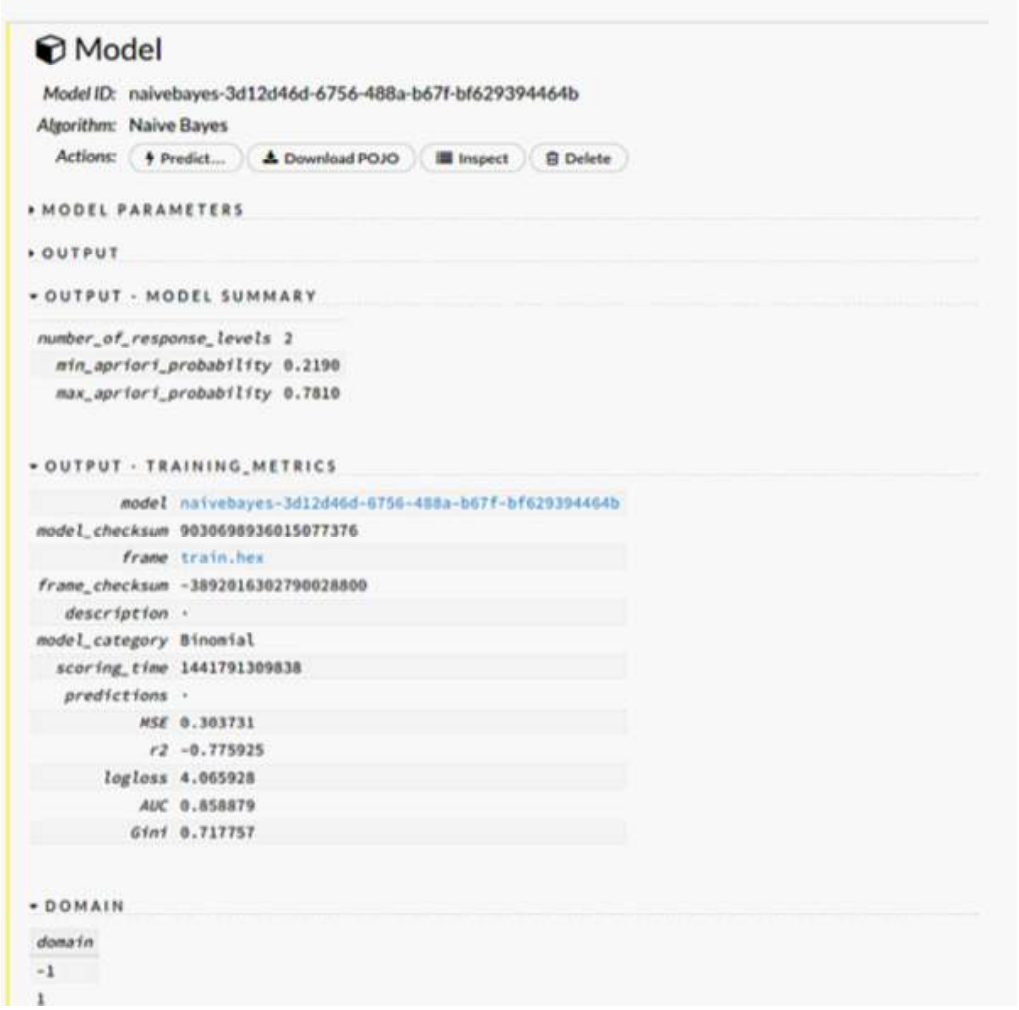



Gambar 11.14: *Membangun model Naive Bayes menggunakan H2O Flow UI*

Gambar 11.15 menunjukkan detail model Naive Bayes. Ringkasan model menunjukkan berbagai model statistik seperti mean squared error (MSE), koefisien determinasi (R2) dan area di bawah kurva (AUC).

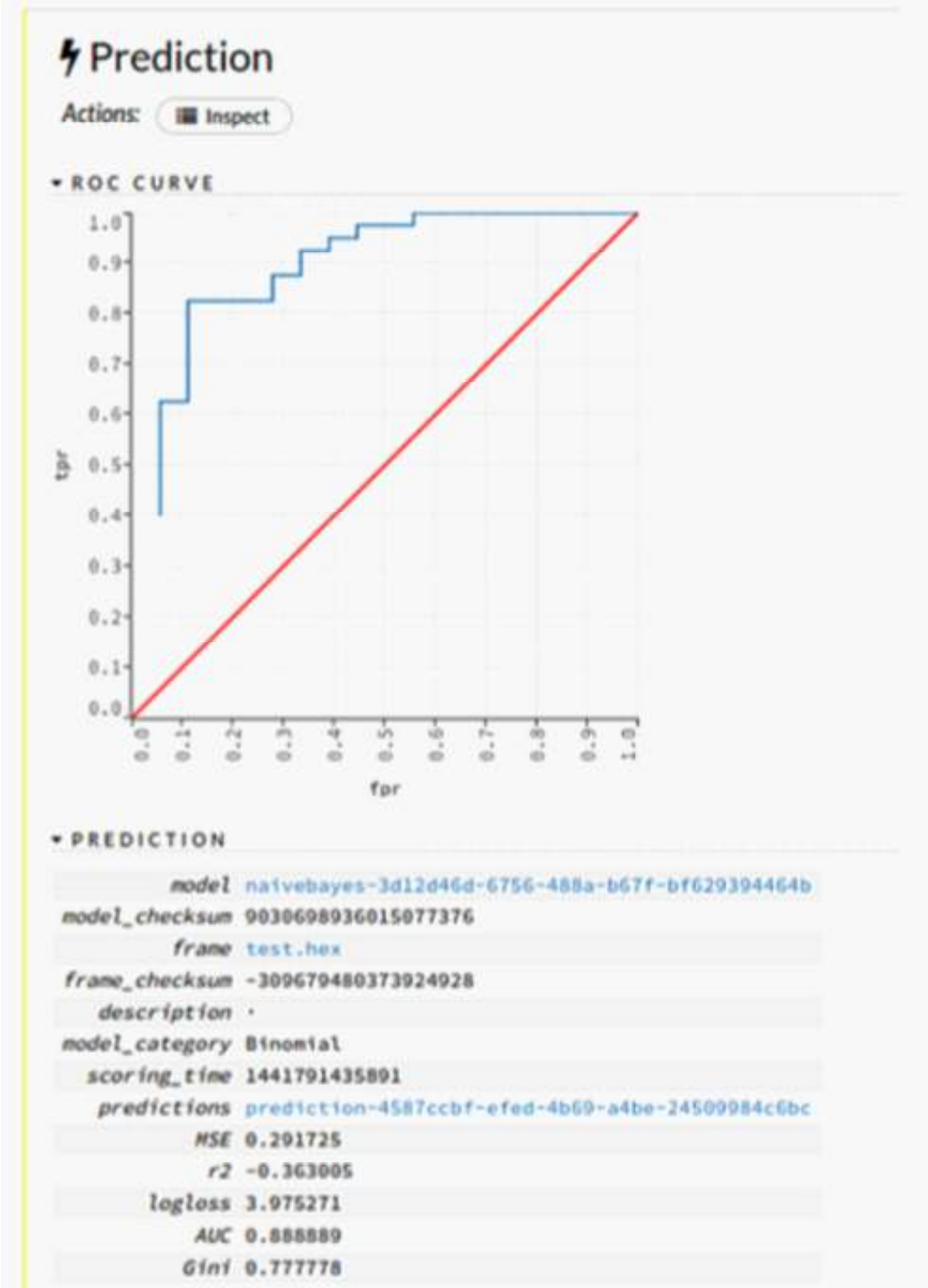
Selanjutnya, impor dan parsing dataset pengujian dengan cara yang sama seperti kita mengimpor dan mengurai dataset pelatihan. Untuk membuat prediksi untuk dataset pengujian, masukkan perintah prediksi atau pilih opsi prediksi dari menu lalu pilih model dan frame pengujian lalu klik tombol Prediksi, seperti yang ditunjukkan pada Gambar 11.16.

Gambar 11.17 menunjukkan ringkasan prediksi yang meliputi plot true positive rate (TPR) dan false positive rate (FPR) yang disebut kurva karakteristik operasi penerima (ROC). Akhirnya, bingkai prediksi dapat diekspor (sebagai file CSV) seperti yang ditunjukkan pada Gambar 11.18.

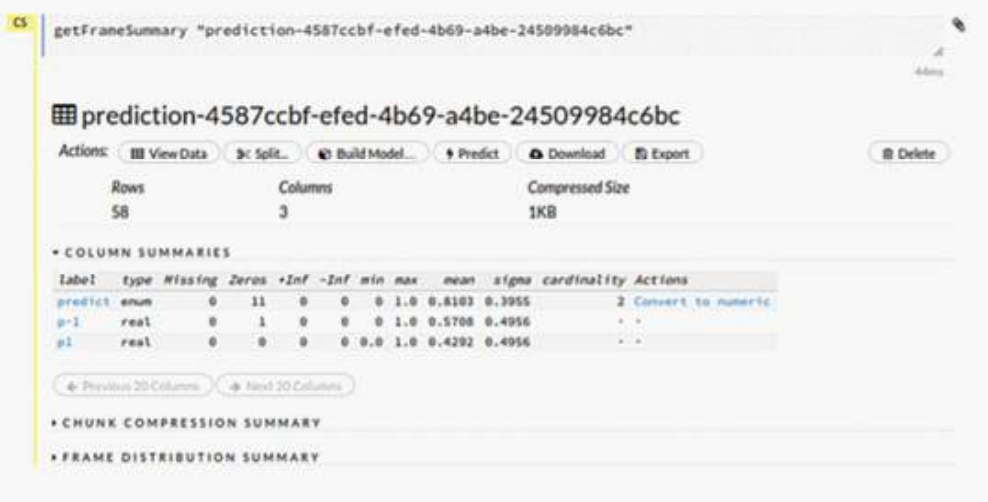



Gambar 11.15: *Melihat detail model Naive Bayes*


Gambar 11.16: *Membuat prediksi untuk data pengujian*




Gambar 11.17: *Melihat hasil prediksi*




Gambar 11.18: *Melihat dan mengekspor bingkai prediksi*

Mari kita lihat implementasi Python dari klasifikasi Naive Bayes menggunakan H2O Python API. Box 11.8 menunjukkan program Python untuk klasifikasi Naive Bayes. Fungsi import_frame digunakan untuk mengimpor data pelatihan dan pengujian. Selanjutnya, kami menentukan kolom respons dalam frame pelatihan dan pengujian. Kami menggunakan fungsi naive_bayes untuk membangun model Naive Bayes. Untuk fungsi ini kita meneruskan data latih tanpa kolom respon (x = latih [1:]), kolom respon di data latih (y = latih [0]), data uji tanpa kolom respon (validation_x = test [1:]) andtheresponsecolumninthetestdata (validation_y = test [0]). Dengan dibangunnya model, kita dapat berfungsi untuk melihat ringkasan model, fungsi prediksi untuk membuat prediksi, dan fungsi model_performance untuk melihat kinerja model dengan data uji. Untuk mengekspor file frame prediksi ke dalamCSV, fungsi download_csv dapat digunakan.

**■ Box 11.8: Python program for Naive Bayes classification using H2O**

```
import h2o

h2o.init()

train = h2o.import_frame(path=h2o.locate("/home/ubuntu/train.csv"))
test = h2o.import_frame(path=h2o.locate("/home/ubuntu/test.csv"))


train[0] = train[0].asfactor()
test[0] = test[0].asfactor()

model = h2o.naive_bayes(x=train[1:], y=train[0], validation_x= test[1:],
validation_y=test[0])

model.show()
```

```
prediction = model.predict(test)

prediction.head()

perf = model.model_performance(test)
perf.show()

print 'Confusion Matrix: '
print (perf.confusion_matrix())

print 'Precision: '
print (perf.precision())

print 'Accuracy: '
print (perf.accuracy())

print 'AUC: '
print (perf.auc())

h2o.download_csv(prediction, 'prediction.csv')
```

**■ Box 11.9: Output of program for Naive Bayes classification using H2O**

```
>>> model.show()

Model Details
=============
H2OBinomialModel :  Naive Bayes
Model Key:  NaiveBayes_model_python_1441787876875_21

Model Summary:

  number_of_response_levels  min_apriori_probability  max_apriori_probability
- -------------------------  -----------------------  -----------------------
  2                          0.218978                 0.781022


ModelMetricsBinomial:  naivebayes
** Reported on train data.  **

MSE: 0.303730509167
R^2:  -0.775924587712
LogLoss:  4.06592781328
AUC: 0.858878504673
Gini:  0.717757009346

Confusion Matrix (Act/Pred) for max f1 @ threshold = 2.32859177356e-09:

   -1  1  Error Rate
--- --- -- ------- ---------
-1  12  18 0.6     (18.0/30.0)
1   1   106 0.0093 (1.0/107.0)
Total 13 124 0.1387 (19.0/137.0)

Maximum Metrics:

metric       threshold value  idx
------------------ ------- ------ ---
```

```
max f1       2.32859e-09 0.917749 87
max f2       1.89463e-10 0.963964 90
max f0point5     2.96126e-07 0.889101 67
max accuracy     1.14672e-08 0.861314 83
max precision    1    1    0
max absolute_MCC    2.32859e-09 0.551268 87
max min_per_class_accuracy 5.64389e-06 0.766355 52

ModelMetricsBinomial:  naivebayes
** Reported on validation data.  **

MSE: 0.291725313697
R²:  -0.36300549344
LogLoss:  3.97527089485
AUC: 0.888888888889
Gini:  0.777777777778

Confusion Matrix (Act/Pred) for max f1 @ threshold = 9.23991456735e-09:

   -1  1  Error Rate
---  ---  --  -----  -------
-1   10  8  0.4444 (8.0/18.0)
1    1  39  0.025  (1.0/40.0)
Total  11  47  0.1552 (9.0/58.0)
```

```
Maximum Metrics:

metric       threshold value  idx
-----------------  -------  -----  ---
max f1       9.23991e-09 0.896552 30
max f2       5.78867e-10 0.952381 33
max f0point5     1.31851e-05 0.916667 18
max accuracy     1.31851e-05 0.844828 18
max precision    0.0441224 0.961538 9
max absolute_MCC    1.31851e-05 0.675148 18
max min_per_class_accuracy 1.31851e-05 0.825  18



>>> prediction
First 10 rows and first 3 columns:
predict    p-1    p1
------  -------  -------
   1  4.04867e-06 0.999996
   1  0.0174331 0.982567
   1  1    9.23991e-09
   1  3.86239e-67 1
   1  1.39996e-27 1
   1  1.10407e-17 1
   1  1    9.9068e-08
   -1  1    1.92925e-09
   -1  1    5.84447e-10
   1  1    1.04785e-08


>>> perf = model.model_performance(test)
>>> perf.show()

ModelMetricsBinomial:  naivebayes
** Reported on test data.  **

MSE: 0.291725313697
R²:  -0.36300549344
```

```
LogLoss:  3.97527089485
AUC: 0.888888888889
Gini:  0.777777777778

Confusion Matrix (Act/Pred) for max f1 @ threshold = 9.23991456735e-09:

   -1  1  Error  Rate
---  ---  --  -----  -------
-1   10  8  0.4444  (8.0/18.0)
1    1   39  0.025  (1.0/40.0)
Total  11  47  0.1552  (9.0/58.0)

Maximum Metrics:

metric          threshold  value    idx
--------------------  -------  -------  ---
max f1          9.23991e-09  0.896552  30
max f2          5.78867e-10  0.952381  33
max f0point5        1.31851e-05  0.916667  18
max accuracy        1.31851e-05  0.844828  18
max precision       0.0441224  0.961538  9
max absolute_MCC      1.31851e-05  0.675148  18
max min_per_class_accuracy  1.31851e-05  0.825   18

>>> print 'Precision:  '
Precision:
>>> print (perf.precision())
[[0.04412240492341057, 0.9615384615384616]]
>>>
>>> print 'Accuracy:  '
Accuracy:
>>> print (perf.accuracy())
[[1.3185115335693565e-05, 0.8448275862068966]]
>>>
>>> print 'AUC: '
AUC:
>>> print (perf.auc())
0.888888888889
```

Sekarang mari kita lihat contoh klasifikasi Naive Bayes menggunakan Spark MLlib. Untuk contoh ini, kami akan menggunakan kumpulan data Wine dari repositori pembelajaran Mesin UCI [39] yang menyertakan hasil analisis kimiawi anggur. Analisis kimiawi menentukan jumlah 13 konstituen (seperti alkohol, asam malat, magnesium, dsb.) Yang ditemukan dalam tiga jenis anggur.

Box 11.10 menunjukkan program Python untuk klasifikasi Naive Bayes menggunakan Spark MLlib. Program ini dapat dijalankan di shell PySpark. Dalam program ini kami mengimplementasikan fungsi parseLine yang mengambil setiap baris dari file input, membagi baris menjadi kolom individual yang dipisahkan oleh koma, mengubah nilai menjadi mengambang dan mengembalikan array numpy Python. Dalam fungsi ini, kami juga mengubah label wine dari 1.0, 2.0 dan 3.0 menjadi 0.0, 1.0 dan 2.0 karena

implementasi Spark dari Naive Bayes mengharapkan label dari 0 ke N-1 di mana N adalah jumlah total kelas dalam data.

Kelas NaiveBayes dari modul klasifikasi MLlib digunakan untuk membangun model Naive Bayes. Setelah model dibangun, metode prediksi dari kelas NaiveBayes dapat digunakan untuk membuat prediksi. Terakhir, kami membandingkan label dalam dataset pengujian dan label yang diprediksi serta menghitung kesalahan pengujian dan keakuratan model.

**■ Box 11.10: Python program for Naive Bayes classification using Spark MLlib**

```
from pyspark.context import SparkContext
from pyspark.mllib.classification import NaiveBayes
from pyspark.mllib.linalg import Vectors
from pyspark.mllib.regression import LabeledPoint

def parseLine(line):
   parts = line.split(',')
   label = float(parts[0])
   if label==1.0:
      label =0.0
   if label==2.0:
      label = 1.0
   if label==3.0:
      label = 2.0
   features = Vectors.dense([float(x) for x in parts[1:]])
   return LabeledPoint(label, features)

sc = SparkContext(appName="NBExample")

trainingData = sc.textFile('file:///home/hadoop/wine.data.txt').map(parseLine)

testData = sc.textFile('file:///home/hadoop/wine.test.txt').map(parseLine)

model = NaiveBayes.train(trainingData, 1.0)

predictions = model.predict(testData.map(lambda x:  x.features))

labelsAndPredictions = testData.map(lambda lp:
lp.label).zip(predictions)

testErr = labelsAndPredictions.filter(lambda v_p:
v_p[0] != v_p[1]).count()/float(testData.count())

print('Test Error = ' + str(testErr))
# Test Error = 0.333333333333

predictionAndLabel = testData.map(lambda p :  (model.predict(p.features),
p.label))

predictionAndLabel.take(10)
#[(0.0, 0.0), (0.0, 0.0), (1.0, 1.0), (1.0, 1.0), (2.0, 1.0), (1.0, 2.0),
(1.0, 2.0), (2.0, 2.0), (2.0, 2.0)]
```

```
accuracy = 1.0 * predictionAndLabel.filter(lambda (x, v):
x == v).count() / testData.count()

print accuracy
#0.666666666667

sc.stop()
```

### 11.3.3 Model Linear Umum

Sedangkan model regresi linier biasa digunakan untuk memodelkan variabel respon yang kontinu, berdistribusi normal dan memiliki varians konstan, Generalized Linear Model (GLM) adalah model regresi linier biasa yang memungkinkan respon variabel yang diskrit, tidak terdistribusi normal dan / atau varians tidak konstan. Model Linier Umum berguna untuk jumlah pemodelan yang bervariasi dalam rentang yang luas (misalnya harga rumah), data kategori dan tidak berurutan (misalnya mengklasifikasikan apakah tumor jinak atau ganas) dan data ordinal (misalnya peringkat film pada skala 0 hingga 10) , di mana nilai numerik yang tepat memiliki signifikansi lain daripada peringkat (dengan kata lain, film dengan peringkat 8 lebih baik daripada film B dengan peringkat 4, tetapi tidak berarti bahwa film A dua kali lebih baik daripada film B).

Model Linear Umum memiliki tiga komponen:

- Random Component: Komponen acak dari GLM adalah distribusi propiglitas variabel respon (y) dari keluarga eksponensial.
- Komponen Sistematis: Komponen acak dari GLM adalah prediktor linier η yang mencakup variabel independen dan parameter model,

$$\eta = \beta 0 + \beta 1x1 + ... + \beta nxn = X\beta\ \beta\ \beta$$

- Fungsi Tautan: Fungsi tautan menentukan hubungan antara nilai yang diharapkan dari variabel respons (E (y)) dan prediktor linier (η),

$$E\ (Y) = \mu = g - 1\ (\eta)$$

Regresi linier adalah kasus khusus Model Linier Umum di mana komponen acak (distribusi propiglitas variabel respons) adalah distribusi normal; komponen sistematik, prediksi nilai kontinu, dan fungsi tautanfungsi identitas.

$$\eta = X\beta\ \beta\ \beta = g\ (\mu) = \mu$$

Regresi logistik adalah kasus khusus Model Linier Umum dimana komponen acak adalah distribusi Bernoulli, komponen sistematis memprediksi nilai kategorikal dan fungsi tautan adalah fungsi Logit.

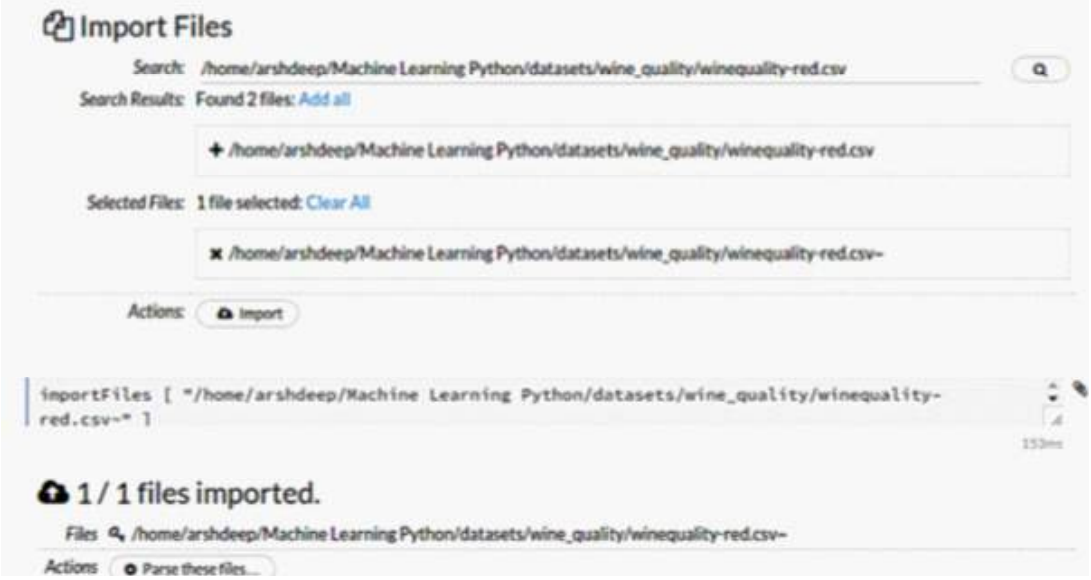
$$\eta = X\beta\ \beta\ \beta = g\ (\mu) = \ln\ \mu\ 1 - \mu$$

Dalam regresi logistik, nilai prediksi adalah propiglitas yang dibatasi hingga (0,1) melalui fungsi Logit.

***Regresi***

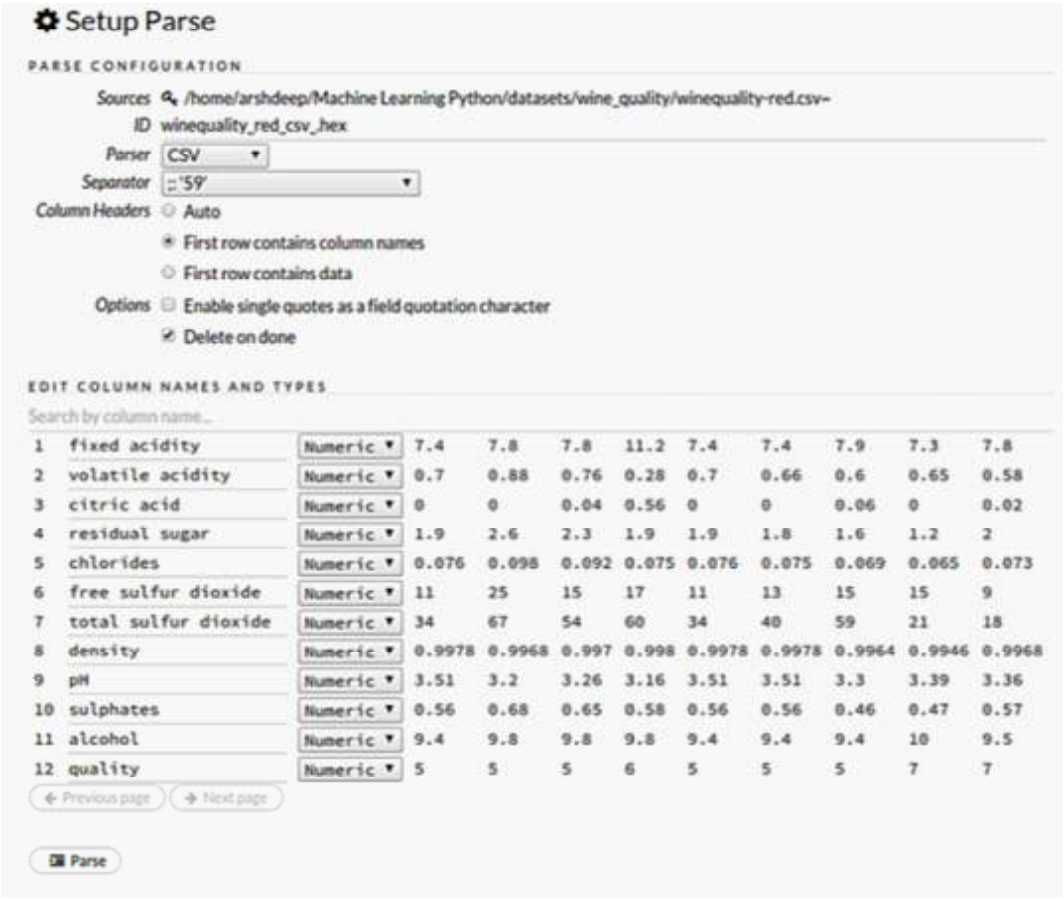
Mari kita lihat contoh regresi linier menggunakan kerangka kerja H2O. Untuk contoh ini, kami akan menggunakan kumpulan data Kualitas Anggur UCI [40]. Kumpulan data ini mencakup data uji fisikokimia untuk varian merah dan putih anggur Portugis. Terdapat 11 variabel input (seperti keasaman tetap, keasaman volatil, asam sitrat, dll.), Dan variabel output berupa skor kualitas antara 0 dan 10.

Langkah pertama adalah mengimpor file dataset ke H2O dari H2O Flow UI seperti yang ditunjukkan pada Gambar 11.19.




Gambar 11.19: Mengimpor file dataset dari H2O Flow UI

Langkah selanjutnya adalah mengurai file dataset. Klik tombol Parse setelah mengimpor file. Gambar 11.20 menunjukkan bagaimana mengatur parser. Pada langkah ini, Anda dapat menentukan berbagai opsi penguraian. Setelah memilih opsi parse, klik tombol Parse untuk mengurai file.




Gambar 11.20: *Parsing file dataset yang diimpor menggunakan H2O Flow UI*

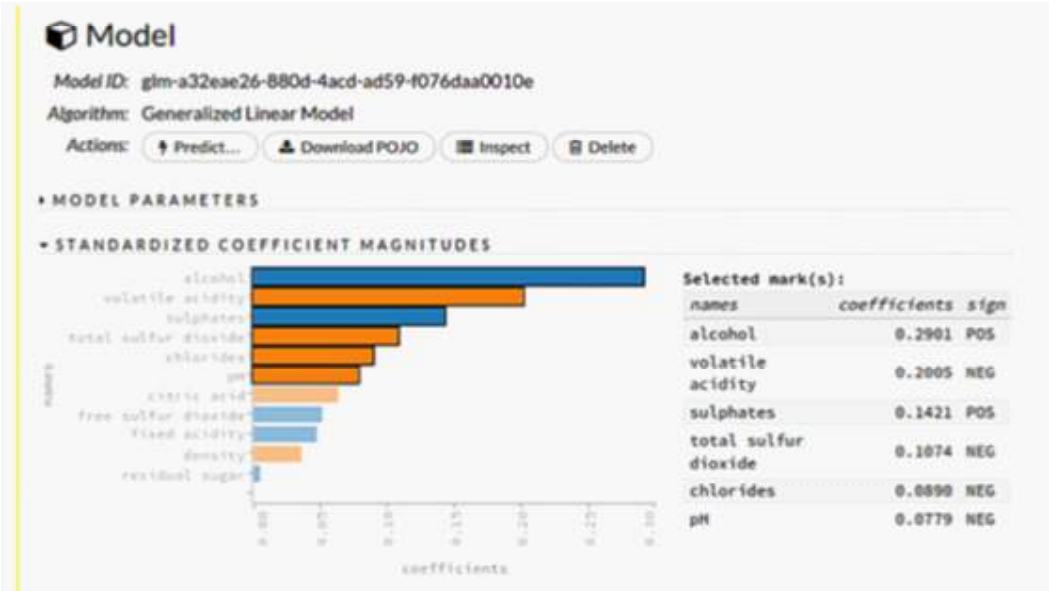
Data dari file yang diurai disimpan dalam bingkai H2O. Gambar 11.21 menunjukkan frame H2O yang dibuat dengan mem-parsing file dataset.

Data tersebut dibagi menjadi training dan test frame seperti yang ditunjukkan pada Gambar 11.22. Pada langkah selanjutnya, kami membangun model GLM. Gambar 11.23 menunjukkan berbagai opsi untuk model. Pilih tipe algoritma yang akan digeneralisasikan Linear Model, frame pelatihan (wine_frame_0.750) dan family (komponen acak untuk GLM) menjadi Gaussian. Pilih opsi default keluarga untuk fungsi tautan, yang secara otomatis akan memilih fungsi tautan default untuk keluarga yang dipilih (keluarga Gaussian). Setelah menentukan opsi model, klik tombol Build Model untuk membangun model.


Gambar 11.21: *Melihat bingkai H2O yang dibuat dari file dataset yang diurai*

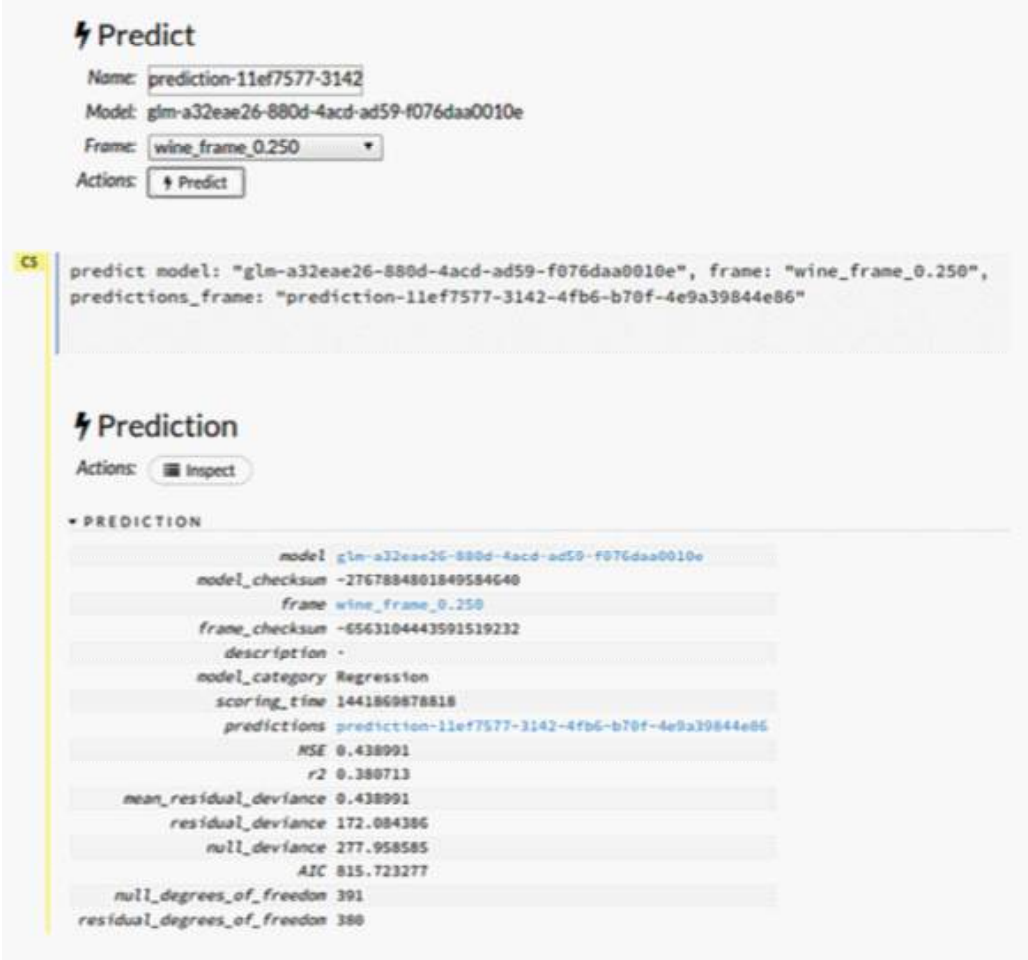

Gambar 11.22: *Memisahkan kerangka H2O menjadi kerangka pelatihan dan pengujian*


Gambar 11.23: *Membangun model GLM dari H2O Flow UI*




Gambar 11.24: *Melihat detail model GLM*

Gambar 11.24menunjukkan ringkasan model termasuk besaran koefisien standar untuk model regresi linier. Untuk membuat prediksi untuk dataset pengujian, masukkan perintah prediksi atau pilih opsi prediksi dari menu, lalu pilih model dan bingkai pengujian, lalu klik tombol Prediksi. Gambar 11.25 menunjukkan ringkasan prediksi. Akhirnya, bingkai prediksi dapat diekspor (sebagai file CSV) seperti yang ditunjukkan pada Gambar 11.26.




Gambar 11.25: *Membuat prediksi dengan model GLM*


Gambar 11.26: *Mengekspor bingkai prediksi dari H2O Flow UI*

Box 11.11 menunjukkan implementasi Python dari contoh yang sama menggunakan pustaka Python H2O dan Box 11.12 menunjukkan output program.

**Box 11.11: Python program for GLM regression using H2O**

```
import h2o

h2o.init()

data = h2o.import_frame(path=h2o.locate("/home/ubuntu/winequality.csv"))

data_split = data.split_frame(ratios = [0.75,0.25])
train = data_split[0]
test = data_split[1]

train[11] = train[11].asfactor()
test[11] = test[11].asfactor()

model = h2o.glm(x=train[0:11], y=train[11],
   validation_x= test[0:11], validation_y=test[11],
   family='gaussian', link='identity')

model.show()

prediction = model.predict(test)

prediction.head()

perf = model.model_performance(test)
perf.show()

h2o.download_csv(prediction, '/home/ubuntu/prediction.csv')
```

**Box 11.12: Output of program for GLM regression using H2O**

```
>>> model.show()
Model Details
=============
H2ORegressionModel :  Generalized Linear Model
Model Key:  GLM_model_python_1441864435922_77

ModelMetricsRegressionGLM: glm
** Reported on train data.  **

MSE: 0.407159182645
R^2:  0.378282135573
Mean Residual Deviance:  0.407159182645
Null degrees of freedom:  1198
Residual degrees of freedom:  1187
Null deviance:  785.217681401
Residual deviance:  488.183859991
AIC: 2351.25188426

ModelMetricsRegressionGLM: glm
** Reported on validation data.  **

MSE: 0.459194439276
R^2:  0.274000886521
```

```
Mean Residual Deviance:  0.459194439276
Null degrees of freedom:  399
Residual degrees of freedom:  388
Null deviance:  258.264326472
Residual deviance:  183.67777571
AIC: 849.838209107


>>> prediction = model.predict(test)

>>>prediction.head()
First 10 rows and first 1 columns:
predict
-------
5.16886
5.56495
6.16603
6.28652
5.15812
6.12444
6.12444
6.12444
5.42637
6.12444

>>> perf = model.model_performance(test)
>>> perf.show()

ModelMetricsRegressionGLM: glm
** Reported on test data.  **

MSE: 0.459194439276
R^2:  0.274000886521
Mean Residual Deviance:  0.459194439276
Null degrees of freedom:  399
Residual degrees of freedom:  388
Null deviance:  258.264326472
Residual deviance:  183.67777571
AIC: 849.838209107
```

**Klasifikasi**

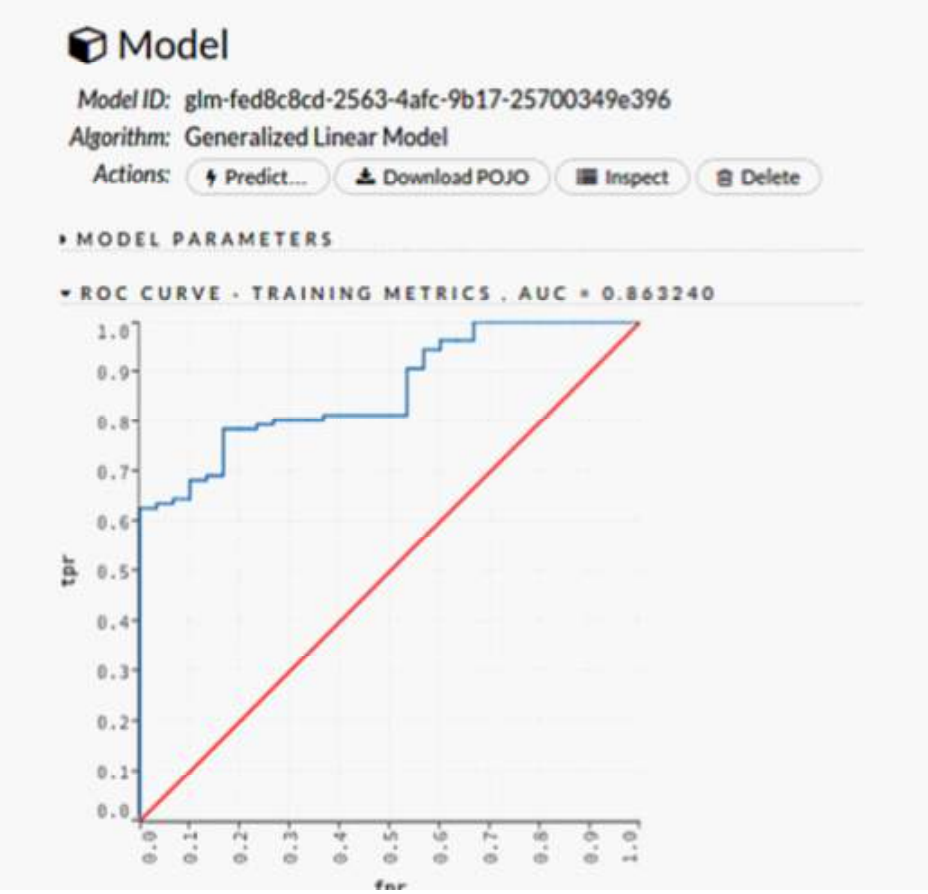
Mari kita lihat contoh klasifikasi GLM (model Regresi Logistik) menggunakan kerangka kerja H2O. Untuk contoh ini, kami akan menggunakan dataset UCI Parkinsons. Impor dan parsing dataset dengan cara yang sama seperti yang ditunjukkan pada contoh klasifikasi Naive Bayes (Gambar 11.11 dan 11.12).

Dengan data yang diimpor dan diurai, sekarang mari kita membuat model klasifikasi GLM. Klik tombol Model Build dalam tindakan bingkai yang diurai atau pilih opsi Build Model dari menu. Gambar 11.27 menunjukkan berbagai opsi untuk model. Pilih jenis algoritme yang akan digeneralisasikan Model Linear, kerangka pelatihan, dan keluarga (komponen acak untuk GLM) yang akan dijadikan Binomial. Pilih opsi default keluarga untuk fungsi tautan, yang secara otomatis akan memilih fungsi tautan default untuk keluarga yang dipilih (keluarga Binomial). Setelah menentukan opsi model, klik tombol Build Model untuk

membangun model. Gambar 11.28 dan 11.29 menunjukkan detail model GLM termasuk kurva ROC dan besaran koefisien standar.

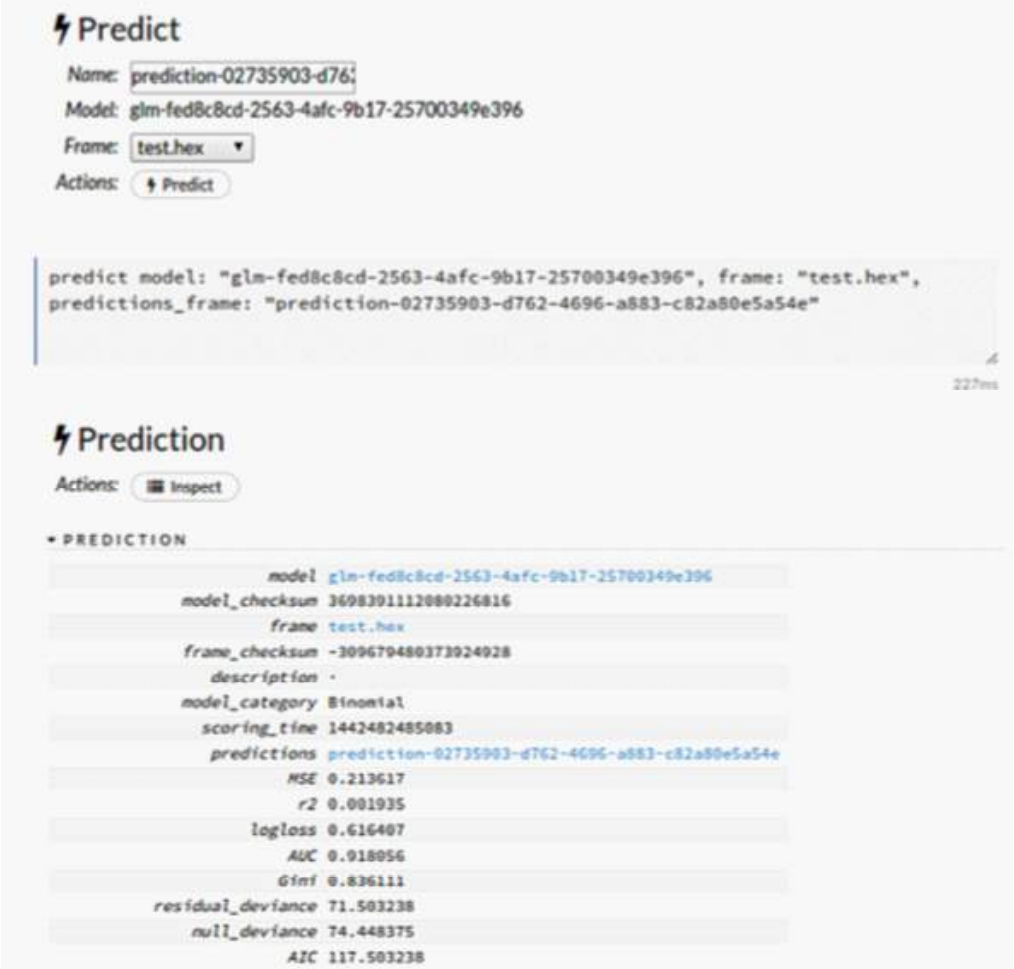
Selanjutnya, impor dan parse dataset pengujian dengan cara yang sama seperti kita mengimpor dan mengurai dataset pelatihan. Untuk membuat prediksi untuk dataset pengujian, masukkan perintah prediksi atau pilih opsi prediksi dari menu, lalu pilih model dan bingkai pengujian, lalu klik tombol Prediksi. Gambar 11.30 menunjukkan ringkasan prediksi. Akhirnya, bingkai prediksi dapat diekspor (sebagai file CSV) seperti yang ditunjukkan pada Gambar 11.31.


Gambar 11.27: *Membangun model GLM dari H2O Flow UI*

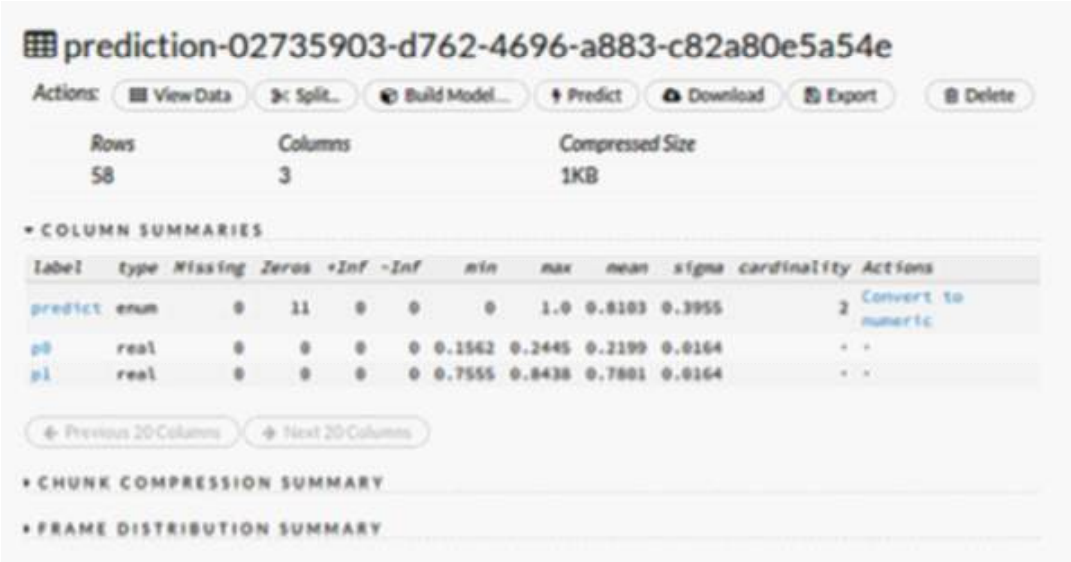



Gambar 11.28: *Melihat detail model GLM*


Gambar 11.29: *Melihat detail model GLM*




Gambar 11.30: *Membuat prediksi menggunakan model GLM*




Gambar 11.31: *Mengekspor bingkai prediksi Bahga & Madisetti*

Box 11.13 menunjukkan implementasi Python dari contoh yang sama menggunakan pustaka Python H2O dan Box 11.14 menunjukkan output program.

**■ Box 11.13: Python program for GLM classification using H2O**

```
import h2o

h2o.init()

train = h2o.import_frame(path=h2o.locate("/home/ubuntu/train.csv"))
test = h2o.import_frame(path=h2o.locate("/home/ubuntu/train.csv"))

train[0] = train[0].asfactor()
test[0] = test[0].asfactor()

model = h2o.glm(x=train[1:], y=train[0],
   validation_x= test[1:], validation_y=test[0],
   family='binomial', link='logit')

model.show()

prediction = model.predict(test)

prediction.head()

perf = model.model_performance(test)
perf.show()

print 'Confusion Matrix: '
print (perf.confusion_matrix())

print 'Precision: '
print (perf.precision())

print 'Accuracy: '
print (perf.accuracy())

print 'AUC: '
print (perf.auc())

h2o.download_csv(prediction, '/home/ubuntu/prediction.csv')
```

**■ Box 11.14: Output of program for GLM classification using H2O**

```
>>> model.show()
Model Details
=============
H2OBinomialModel :  Generalized Linear Model
Model Key:  GLM_model_python_1442483162375_2

GLM Model:

family:  binomial
```

```
link:  logit
regularization:  Elastic Net (alpha = 0.5, lambda = 0.04384 )
number_of_predictors_total:  22
number_of_active_predictors:  10
number_of_iterations:  6
training_frame:  py0cdd7344-bfca-42a5-a122-7d4db9da9615

ModelMetricsBinomialGLM: glm
** Reported on train data.  **

MSE: 0.10081486084
R²:  0.410531425821
LogLoss:  0.3256156667
Null degrees of freedom:  136
Residual degrees of freedom:  127
Null deviance:  144.017560202
Residual deviance:  89.2186926758
AIC: 109.218692676
AUC: 0.893769470405
Gini:  0.78753894081

Confusion Matrix (Act/Pred) for max f1 @ threshold = 0.439216360065:

   -1  1  Error  Rate
---  ---  --  -----  --------
-1   15  15  0.5    (15.0/30.0)
1    0   107 0      (0.0/107.0)
Total  15  122  0.1095  (15.0/137.0)

Maximum Metrics:

metric                  threshold  value     idx
----------------------  ---------  --------  ---
max f1                  0.439216   0.934498  121
max f2                  0.439216   0.972727  121
max f0point5            0.734741   0.903491  94
max accuracy            0.480394   0.890511  119
max precision           0.998257   1         0
max absolute_MCC        0.439216   0.662212  121
max min_per_class_accuracy  0.748936  0.766667  89

ModelMetricsBinomialGLM: glm
** Reported on validation data.  **

MSE: 0.104925736208
R²:  0.509763643607
LogLoss:  0.337658509376
Null degrees of freedom:  57
Residual degrees of freedom:  48
Null deviance:  74.4483748993
Residual deviance:  39.1683870876
AIC: 59.1683870876
AUC: 0.927777777778
Gini:  0.855555555556

Confusion Matrix (Act/Pred) for max f1 @ threshold = 0.568524772441:

   -1  1  Error  Rate
---  ---  --  -----  -------
-1   12  6  0.3333  (6.0/18.0)
1    0   40 0      (0.0/40.0)
Total  12  46  0.1034  (6.0/58.0)
```

```
Maximum Metrics:

metric         threshold  value   idx
-------------- ---------- ------- ---
max f1         0.568525   0.930233 45
max f2         0.568525   0.970874 45
max f0point5       0.767793   0.914634 30
max accuracy       0.568525   0.896552 45
max precision      0.999028   1    0
max absolute_MCC     0.568525   0.761387 45
max min_per_class_accuracy 0.749704   0.8   34

>>> prediction = model.predict(test)
>>> prediction.head()
First 10 rows and first 3 columns:
predict    p0   p1
------- -------- -----
   1 0.0329049 0.967095
   1 0.0378409 0.962159
   1 0.412367 0.587633
   1 0.00664681 0.993353
   1 0.0211856 0.978814
   1 0.08681   0.91319
   1 0.431475 0.568525
  -1 0.715825 0.284175
  -1 0.743189 0.256811
  -1 0.83547   0.16453

>>> perf = model.model_performance(test)
>>> perf.show()

ModelMetricsBinomialGLM: glm
** Reported on test data.  **

MSE: 0.104925736208
R^2:  0.509763643607
LogLoss:  0.337658509376
Null degrees of freedom:  57
Residual degrees of freedom:  48
Null deviance:  74.4483748993
Residual deviance:  39.1683870876
AIC: 59.1683870876
AUC: 0.927777777778
Gini:  0.855555555556

Confusion Matrix (Act/Pred) for max f1 @ threshold = 0.568524772441:

   -1  1  Error Rate
--- --- -- ------ --------
-1  12  6  0.3333 (6.0/18.0)
1   0   40 0    (0.0/40.0)
Total 12  46 0.1034 (6.0/58.0)

Maximum Metrics:

metric         threshold  value   idx
-------------- ---------- ------- ---
max f1         0.568525   0.930233 45
max f2         0.568525   0.970874 45
max f0point5       0.767793   0.914634 30
max accuracy       0.568525   0.896552 45
max precision      0.999028   1    0
max absolute_MCC     0.568525   0.761387 45
```

```
max min_per_class_accuracy 0.749704  0.8  34
>>>
>>> print 'Confusion Matrix:  '
Confusion Matrix:
>>> print (perf.confusion_matrix())

Confusion Matrix (Act/Pred) for max f1 @ threshold = 0.568524772441:

   -1  1 Error Rate
--- --- -- ----- -------
-1  12  6 0.3333 (6.0/18.0)
1   0   40 0   (0.0/40.0)
Total 12  46 0.1034 (6.0/58.0)

>>>
>>> print 'Precision:  '
Precision:
>>> print (perf.precision())
[[0.9990275569821115, 1.0]]
>>>
>>> print 'Accuracy:  '
Accuracy:
>>> print (perf.accuracy())
[[0.568524772440684, 0.896551724137931]]
>>>
>>> print 'AUC: '
AUC:
>>> print (perf.auc())
0.927777777778
```

Sekarang mari kita ulangi contoh yang sama menggunakan Spark MLlib. Box 11.15 menunjukkan program Python untuk klasifikasi GLM (model regresi logistik) menggunakan SparkMLlib. Program ini dapat dijalankan di shell PySpark. Dalam program ini kami mengimplementasikan fungsi parseLine yang mengambil setiap baris dari file input, membagi baris menjadi kolom individual yang dipisahkan oleh koma, mengubah nilai menjadi mengambang dan mengembalikan LabeledPoints. Kelas LogisticRegressionWithLBFGS dari modul klasifikasi MLlib digunakan untuk membangun model Regresi Logistik. Setelah model dibuat, metode prediksi kelas LogisticRegressionWithLBFGS dapat digunakan untuk membuat prediksi. Terakhir, kami membandingkan label dalam dataset pengujian dan label yang diprediksi serta menghitung kesalahan pengujian. Prediksi dapat disimpan ke file teks menggunakan fungsi saveAsTextFile.

**■ Box 11.15: Python program for GLM classification (logistic regression) using Spark MLlib**

```
from pyspark.context import SparkContext
from pyspark.mllib.classification import LogisticRegressionWithLBFGS
from pyspark.mllib.linalg import Vectors
from pyspark.mllib.regression import LabeledPoint
```

```
def parseLine(line):
   parts = line.split(',')
   label = float(parts[0])
   if label==-1.0:
      label =0.0
   features = Vectors.dense([float(x) for x in parts[1:]])
   return LabeledPoint(label, features)

sc = SparkContext(appName="GLMExample")

trainingData = sc.textFile('file:///home/hadoop/train.csv').map(parseLine)

testData = sc.textFile('file:///home/hadoop/test.csv').map(parseLine)

model = LogisticRegressionWithLBFGS.train(trainingData)

predictions = model.predict(testData.map(lambda x:  x.features))

labelsAndPredictions = testData.map(lambda lp:
            lp.label).zip(predictions)

testErr = labelsAndPredictions.filter(lambda v_p:
        v_p[0] != v_p[1]).count()/ float(testData.count())

print('Test Error = ' + str(testErr))

labelsAndPredictions.saveAsTextFile('file:///home/hadoop/prediction.txt')
sc.stop()
```

### 11.3.4 Pohon Keputusan

Decision Trees adalah metode pembelajaran terbimbing yang menggunakan pohon yang dibuat dari aturan keputusan sederhana yang dipelajari dari data pelatihan sebagai model prediksi. Model prediktif berupa pohon yang dapat digunakan untuk memprediksi nilai suatu variabel target berdasarkan beberapa variabel atribut. Setiap node di pohon sesuai dengan satu atribut dalam dataset di mana "pemisahan" dilakukan. Setiap daun di pohon keputusan mewakili nilai variabel target. Proses pembelajaran melibatkan pemisahan secara rekursif pada atribut sampai semua sampel di simpul anak memiliki nilai yang sama dari variabel target atau membagi hasil lebih lanjut tanpa perolehan informasi lebih lanjut. Untuk memilih atribut terbaik untuk pemisahan di setiap tahap, metrik yang berbeda dapat digunakan. Dua metrik paling populer yang digunakan untuk menentukan atribut terbaik untuk pemisahan adalah:

- Penguatan Informasi: Isi informasi dari variabel acak diskrit X dengan fungsi massa propiglitas (PMF), P (X), didefinisikan sebagai,

$$I(X) = -log_2 2P(X)$$

Keuntungan informasi didefinisikan berdasarkan entropi variabel acak yang didefinisikan sebagai,

$$H(X) = E[I(X)] = E[-log2P(Xi)= \Sigma ilog2P(xi)$$

Entropi adalah ukuran ketidakpastian dalam variabel acak dan memilih atribut dengan informasi tertinggi lagi menghasilkan pembagian yang mengurangi ketidakpastian paling besar pada tahap itu.

- GiniCoefisiensi: Koefisien Gini mengukur ketidaksetaraan, yaitu seberapa sering sampel yang dipilih secara acak yang diberi label berdasarkan distribusi label, akan diberi label secara tidak benar. Koefisien Gini didefinisikan sebagai,

$$G(X)= 1-\Sigma iP(xi)2$$

Ada berbagai algoritma untuk membangun pohon keputusan, yang populer adalah ID3 dan C4.5. Mari kita lihat langkah-langkah yang terlibat dalam algoritma ID3:

- Atributnya terpisah. Jika tidak, diskritkan atribut kontinu.
- Hitung entropi setiap atribut menggunakan kumpulan data.
- Pilih atribut dengan perolehan informasi tertinggi.
- Buat cabang untuk setiap nilai atribut yang dipilih.
- Ulangi dengan atribut yang tersisa.

Algoritme ID3 dapat mengakibatkan overfitting pada data pelatihan dan dapat menjadi mahal untuk dilatih terutama untuk atribut kontinu. Algoritma C4.5 merupakan perpanjangan dari algoritma ID3. C4.5 mendukung atribut diskrit dan kontinu. Untuk mendukung atribut kontinu, C4.5 menemukan ambang batas untuk atribut kontinu dan kemudian membagi berdasarkan nilai ambang batas. C4.5 mencegah overfitting dengan pemangkasan pohon setelah dibuat. Pemangkasan melibatkan pemindahan atau penggabungan cabang-cabang yang memberikan sedikit kekuatan diskriminatif.

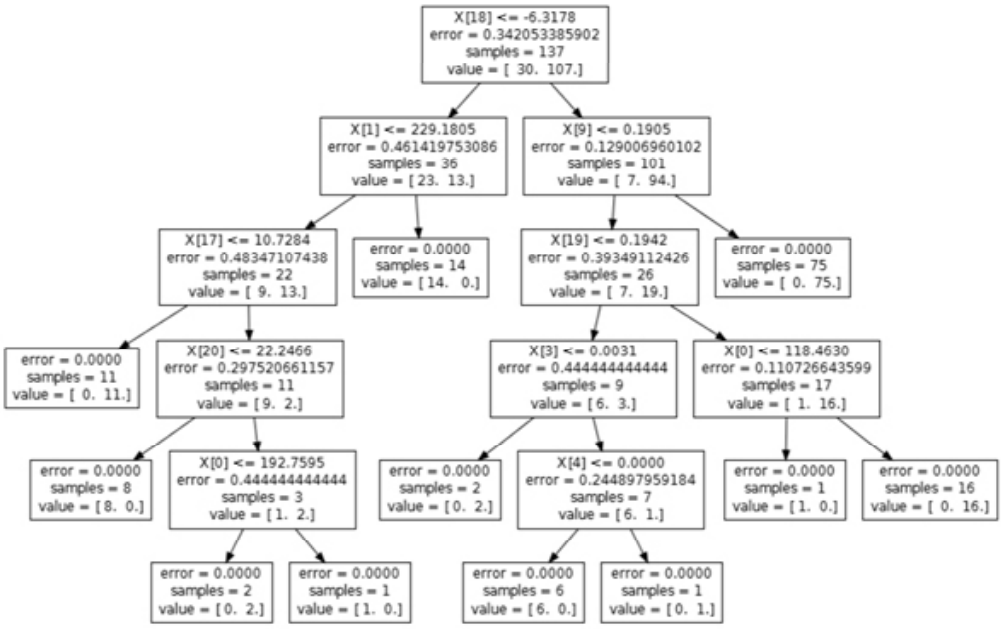
Sekarang mari kita lihat contoh klasifikasi Decision Tree menggunakan Spark MLlib. Untuk contoh ini, kami akan menggunakan dataset UCI Parkinsons. Box 11.16 menunjukkan program Python untuk klasifikasi Decision Tree menggunakan Spark MLlib. Program ini dapat dijalankan di shell PySpark. Dalam program ini kami mengimplementasikan fungsi parseLine yang mengambil setiap baris dari file input, membagi baris menjadi kolom individual yang dipisahkan oleh koma, mengubah nilai menjadi mengambang dan mengembalikan array numpy Python. Dalam fungsi ini, kami juga mengubah label wine dari -1,0, 1,0 menjadi 0,0 dan 1,0 karena Spark mengharapkan label dari 0 ke N-1 di mana N adalah jumlah total kelas dalam data.

Kelas DecisionTree dari modul klasifikasi MLlib digunakan untuk membangun model Pohon Keputusan. Setelah model dibuat, metode prediksi dari kelas DecisionTree dapat digunakan untuk membuat prediksi. Terakhir, kami membandingkan label dalam dataset pengujian dan label yang diprediksi dan menghitung kesalahan pengujian model.

**■ Box 11.16: Python program for Decision Tree classification using Spark MLlib**

```
import sys

from pyspark.context import SparkContext
from pyspark.mllib.tree import DecisionTree, DecisionTreeModel
from pyspark.mllib.linalg import Vectors
from pyspark.mllib.regression import LabeledPoint

def parseLine(line):
   parts = line.split(',')
   label = float(parts[0])
   if label==1.0:
      label =1
   if label==-1.0:
      label = 0
   features = Vectors.dense([float(x) for x in parts[1:]])
   return LabeledPoint(label, features)

sc = SparkContext(appName="DTExample")
```

```
trainingData = sc.textFile('file:///home/hadoop/train.csv').map(parseLine

testData = sc.textFile('file:///home/hadoop/test.csv').map(parseLine)


model = DecisionTree.trainClassifier(trainingData, numClasses=3,
categoricalFeaturesInfo=, impurity='gini', maxDepth=5, maxBins=32)

predictions = model.predict(testData.map(lambda x:  x.features))

labelsAndPredictions = testData.map(lambda lp:
lp.label).zip(predictions)

labelsAndPredictions.take(10)
#[(1.0, 1.0), (1.0, 1.0), (1.0, 1.0), (1.0, 1.0), (1.0, 1.0),
(1.0, 1.0), (1.0, 0.0), (0.0, 0.0), (0.0, 0.0), (0.0, 0.0)]


testErr = labelsAndPredictions.filter(lambda v_p:
v_p[0] != v_p[1]).count()/ float(testData.count())

print('Test Error = ' + str(testErr))
#Test Error = 0.155172413793

labelsAndPredictions.saveAsTextFile('file:///home/hadoop/prediction.txt')

sc.stop()
```




Gambar 11.32: *Contoh pohon keputusan yang dihasilkan*

Gambar 11.32 menunjukkan contoh pohon keputusan yang dibuat untuk dataset Parkinson. Memperlihatkan atribut-atribut yang memisahkan dilakukan setiap langkah dan nilai-nilai pemisahan. Juga menunjukkan kesalahan, jumlah sampel di setiap node dan jumlah sampel di setiap kelas (dalam array nilai). Misalnya split pertama dilakukan pada kolom ke-19 (atribut X [18]) dan jumlah sampel pada training set adalah 137. Pada split pertama terdapat 30 sampel di kelas pertama dan 107 sampel di kelas kedua kelas.

### 11.3.5 Random Forest

Random Forest adalah metode pembelajaran ensemble yang didasarkan pada pohon keputusan acak [38]. Random Forest melatih sejumlah pohon keputusan dan kemudian mengambil suara mayoritas dengan menggunakan mode kelas yang diprediksi oleh pohon individu. Algoritme Random Forest (algoritme Breiman) ditampilkan di Box 11.17.

Mari kita lihat contoh klasifikasi Random Forest menggunakan kerangka kerja H2O. Untuk contoh ini, kami akan menggunakan kumpulan data Wine. Impor dan parsing file dataset dengan cara yang sama seperti yang dilakukan pada contoh untuk k-means clustering (seperti yang ditunjukkan pada Gambar 11.5 dan Gambar 11.6). Dengan data yang diimpor dan diurai, mari kita sekarang membangun model Random Forest. Klik tombol Build Model dalam tindakan frame yang diurai atau pilih opsi Build Model dari menu. Gambar 11.33

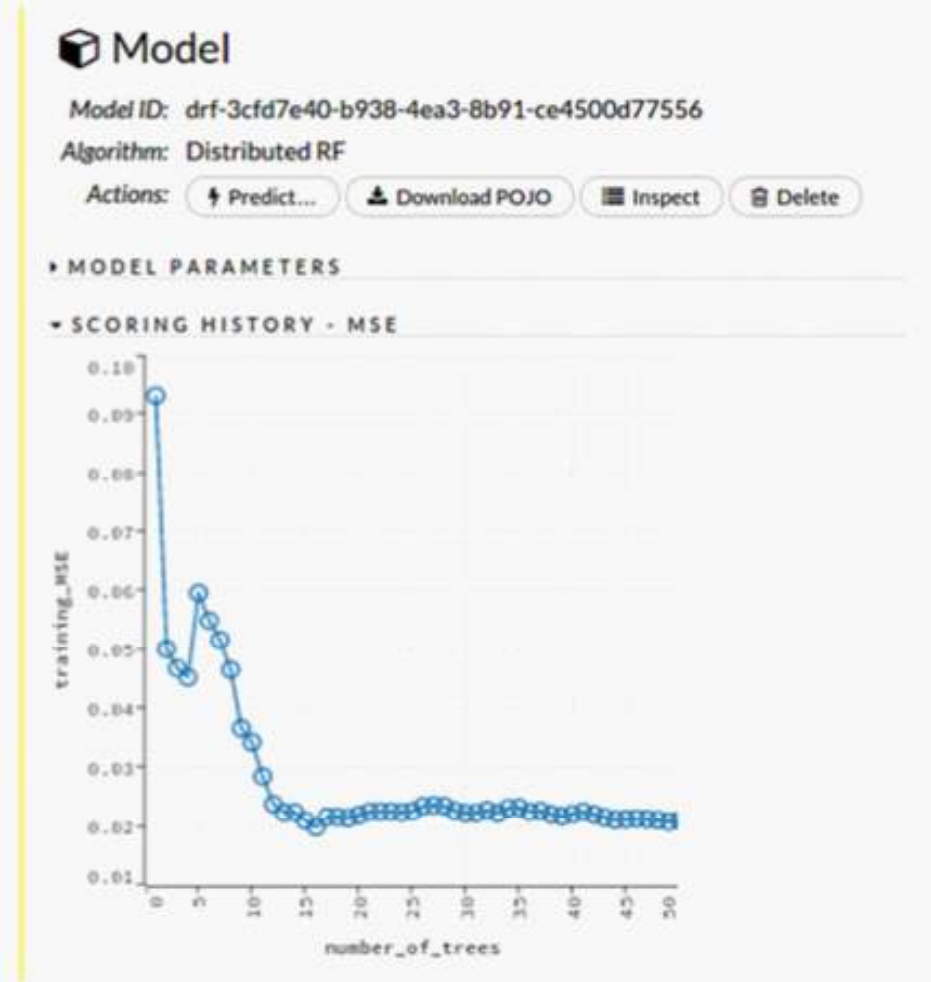
menunjukkan berbagai opsi untuk model. Pilih jenis algoritma yang akan didistribusikan RF, kerangka pelatihan, dan kolom respons sebagai C1. Setelah menentukan opsi model, klik tombol Build Model untuk membangun model.

**■ Box 11.17: Random Forest algorithm**

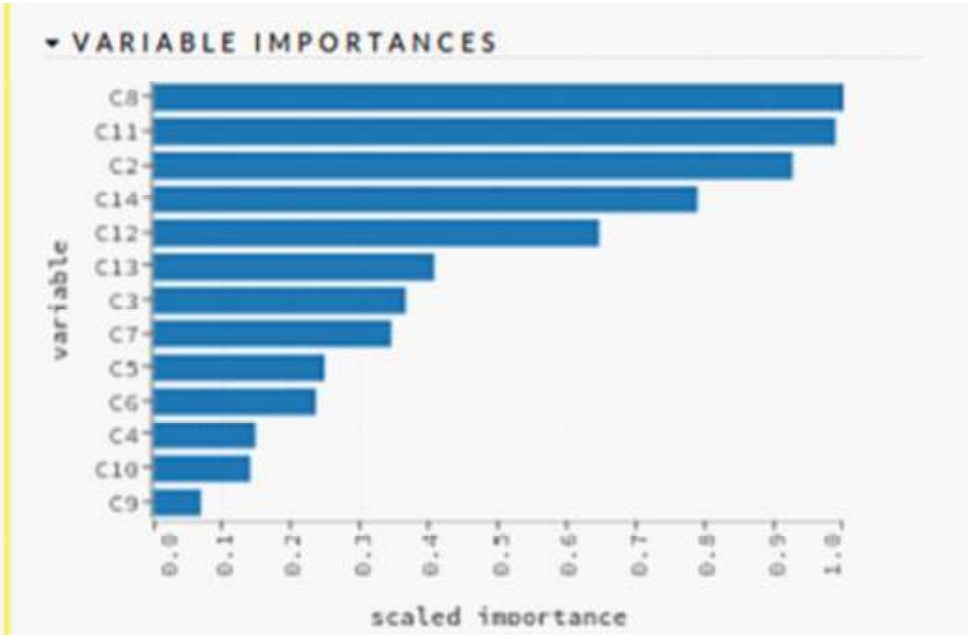
1. Gambarkan sampel bootstrap (n kali dengan penggantian dari sampel N di set pelatihan) dari set data
2. Latih pohon keputusan
   -Sampai pohon tumbuh dewasa (ukuran maksimal)
   - Pilih simpul daun berikutnya
   - Pilih m atribut (m jauh lebih sedikit dari jumlah total atribut M) secara acak.
   - Pilih atribut terbaik dan pisahkan seperti biasa
3. Ukur kesalahan di luar tas
   - Gunakan sampel lainnya (tidak dipilih dalam bootstrap) untuk memperkirakan kesalahan pohon, dengan memprediksi kelasnya.
4. Ulangi langkah 1-3 k kali untuk menghasilkan k pohon.
5. Buat prediksi dengan suara terbanyak di antara k pohon

Gambar 11.34, 11.35 dan 11.36 menunjukkan detail model Random Forest yang dibangun dari dataset Wine.


Gambar 11.33: *Membangun model Random Forest menggunakan H2O Flow UI*




Gambar 11.34: *Melihat detail model Random Forest*



▾ TRAINING METRICS - CONFUSION MATRIX

| | 1 | 2 | 3 | Error | Rate |
|---|---|---|---|---|---|
| 1 | 59 | 0 | 0 | 0 | 0 / 59 |
| 2 | 0 | 70 | 1 | 0.0141 | 1 / 71 |
| 3 | 0 | 0 | 48 | 0 | 0 / 48 |
| Total | 59 | 70 | 49 | 0.0056 | 1 / 178 |

▾ OUTPUT

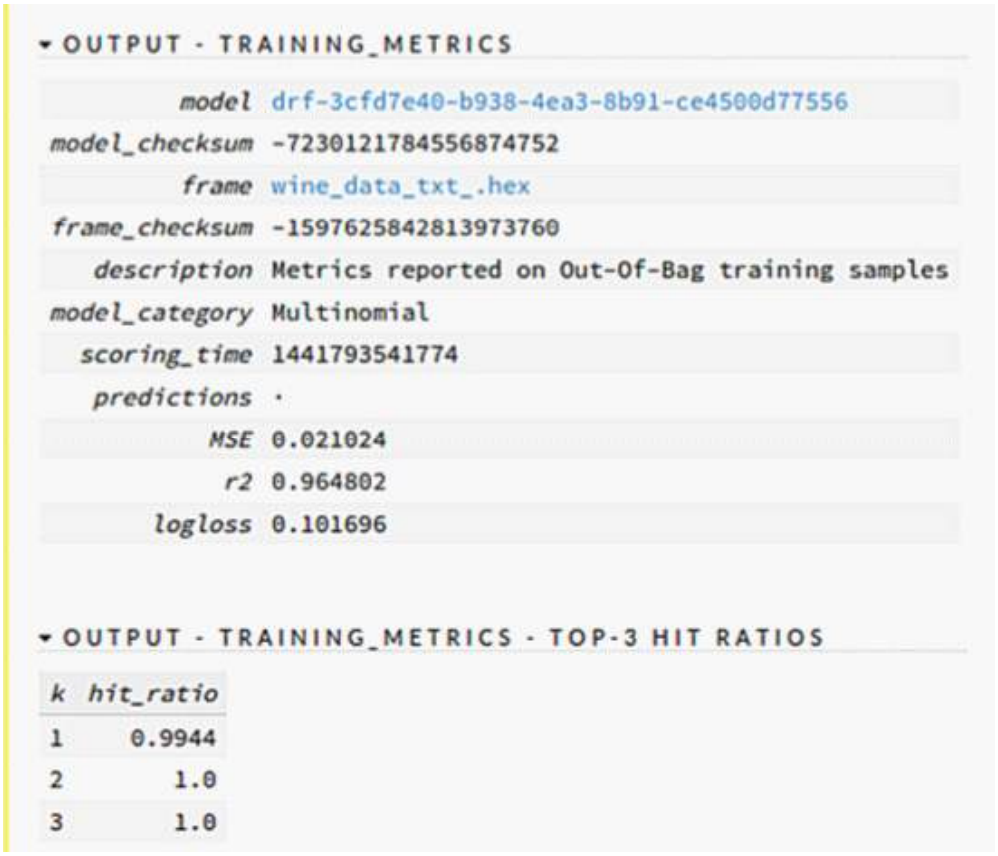
| | |
|---|---|
| cross_validation_models | · |
| cross_validation_predictions | · |
| model_category | Multinomial |
| validation_metrics | · |
| cross_validation_metrics | · |
| status | DONE |
| start_time | 1441793541073 |
| end_time | 1441793541789 |
| run_time | 716 |
| init_f | 0 |

Gambar 11.35: *Melihat detail model Random Forest*

▾ OUTPUT - MODEL SUMMARY

| | |
|---|---|
| number_of_trees | 150 |
| model_size_in_bytes | 24301 |
| min_depth | 3 |
| max_depth | 9 |
| mean_depth | 4.6467 |
| min_leaves | 4 |
| max_leaves | 18 |
| mean_leaves | 8.4867 |

▸ OUTPUT - SCORING HISTORY



▾ OUTPUT - TRAINING_METRICS

| | |
|---|---|
| model | drf-3cfd7e40-b938-4ea3-8b91-ce4500d77556 |
| model_checksum | -7230121784556874752 |
| frame | wine_data_txt_.hex |
| frame_checksum | -1597625842813973760 |
| description | Metrics reported on Out-Of-Bag training samples |
| model_category | Multinomial |
| scoring_time | 1441793541774 |
| predictions | · |
| MSE | 0.021024 |
| r2 | 0.964802 |
| logloss | 0.101696 |

▾ OUTPUT - TRAINING_METRICS - TOP-3 HIT RATIOS

| k | hit_ratio |
|---|---|
| 1 | 0.9944 |
| 2 | 1.0 |
| 3 | 1.0 |

Tabel 11.36: *Melihat ringkasan model dan metrik pelatihan*

Selanjutnya, impor dan parsing dataset pengujian dengan cara yang sama seperti kita mengimpor dan mengurai dataset pelatihan. Untuk membuat prediksi untuk dataset pengujian, masukkan perintah prediksi atau pilih opsi prediksi dari menu, lalu pilih model dan bingkai pengujian, lalu klik tombol Prediksi. Gambar 11.37 menunjukkan ringkasan prediksi.

```
predict model: "drf-3cfd7e40-b938-4ea3-8b91-ce4500d77556", frame: "wine_test.hex",
predictions_frame: "prediction-4f4b3701-339c-4569-8c82-5d5ec191cd6a"
```

Prediction

Actions: Inspect

▾ PREDICTION

| | |
|---|---|
| model | drf-3cfd7e40-b938-4ea3-8b91-ce4500d77556 |
| model_checksum | -7230121784556874752 |
| frame | wine_test.hex |


Gambar 11.37: *Melihat hasil prediksi*

Box 11.18 menunjukkan implementasi Python dari contoh yang sama menggunakan pustaka Python H2O.

**Box 11.18: Python program for Random Forest classification using H2O**

```
import h2o

h2o.init()

train = h2o.import_frame(path=h2o.locate("/home/ubuntu/wine.data.txt"))
test = h2o.import_frame(path=h2o.locate("/home/ubuntu/wine.test.txt"))

train[0] = train[0].asfactor()
test[0] = test[0].asfactor()

model = h2o.random_forest(x=train[1:], y=train[0],
validation_x= test[1:],    validation_y=test[0], seed=12, ntrees=10,
   max_depth=20, balance_classes=True)

model.show()

prediction = model.predict(test)

prediction.head()

perf = model.model_performance(test)
perf.show()
```

```
print 'Confusion Matrix: '
print (perf.confusion_matrix())

print 'Precision: '
print (perf.precision())

print 'Accuracy: '
print (perf.accuracy())

print 'AUC: '
print (perf.auc())

h2o.download_csv(prediction, 'prediction.csv')
```

Box 11.19 menunjukkan output program untuk klasifikasi Random Forest menggunakan H2O.

**■ Box 11.19: Output of program for Random Forest classification using H2O**

```
model.show()
Model Details
=============
H2OMultinomialModel : Distributed RF
Model Key: DRF_model_python_1441787876875_28

Model Summary:

  number_of_trees  model_size_in_bytes  min_depth  max_depth  mean_depth  min_leaves  max_leaves  mean_leaves
- ---------------  -------------------  ---------  ---------  ----------  ----------  ----------  -----------
  30               4805                 3          9          4.86667     5           15          8.4


ModelMetricsMultinomial: drf
** Reported on train data. **

MSE: 0.0242244865001
R^2: 0.963923431738
LogLoss: 0.383920432907
```

```
Confusion Matrix:

1 2 3 Error  Rate
-- -- -- ------ -----
70 0 0 0    0 / 70
0 66 2 0.0294118 2 / 68
0 1 68 0.0144928 1 / 69
70 67 70 0.0144928 3 / 207

Top-3 Hit Ratios:

k hit_ratio
-- -------
1 0.985507
2 0.995169
3 1

ModelMetricsMultinomial: drf
** Reported on validation data. **
```

```
MSE: 0.0110899144456
R^2:  0.982034338598
LogLoss:  0.0536092821889

Confusion Matrix:

1 2 3 Error Rate
-- -- -- ----- ----
2 0 0 0   0 / 2
0 3 0 0   0 / 3
0 0 4 0   0 / 4
2 3 4 0   0 / 9

Top-3 Hit Ratios:

k hit_ratio
-- -------
1 1
2 1
3 1

Variable Importances:

variable relative_importance scaled_importance percentage
-------- -------------------- ------------------ ----------
C13      310.435             1                  0.266665
C2       197.323             0.635635           0.169502
C14      165.101             0.531839           0.141823
C11      99.1768             0.319477           0.0851934
C8       72.4592             0.233412           0.0622428
C3       59.1769             0.190626           0.0508333
C12      53.6374             0.172782           0.0460749
C10      44.0346             0.141848           0.0378259
C6       43.2904             0.139451           0.0371867
C5       41.5202             0.133749           0.0356661
C7       40.4724             0.130373           0.034766
C9       21.3045             0.0686277          0.0183006
C4       16.2047             0.0521999          0.0139199

perf = model.model_performance(test)
>>> perf.show()
```

```
ModelMetricsMultinomial:  drf
** Reported on test data.  **

MSE: 0.0110899144456
R^2:  0.982034338598
LogLoss:  0.0536092821889

Confusion Matrix:

1 2 3 Error Rate
-- -- -- ----- ----
2 0 0 0   0 / 2
0 3 0 0   0 / 3
0 0 4 0   0 / 4
2 3 4 0   0 / 9

Top-3 Hit Ratios:

k hit_ratio
-- -------
1 1
2 1
3 1
```

```
>>>
>>> print 'Confusion Matrix:  '
Confusion Matrix:
>>> print (perf.confusion_matrix())

Confusion Matrix:

1  2  3  Error Rate
-- -- -- ----- ----
2  0  0  0   0 / 2
0  3  0  0   0 / 3
0  0  4  0   0 / 4
2  3  4  0   0 / 9
```

Mari kita ulangi contoh yang samamenggunakanSparkMLlibdengan kumpulan dataWined. Box11.20 showsthepythonprogramforRandomForestclassi fi cationusingSparkMLlib. Program ini dapat dijalankan di shell PySpark. Dalam program ini kami mengimplementasikan fungsi parseLine yang mengambil setiap baris dari file input, membagi baris menjadi kolom individual yang dipisahkan oleh koma, mengubah nilai menjadi mengambang dan mengembalikan LabeledPoints. Dalam fungsi ini, kami juga mengubah label anggur dari 1.0, 2.0 dan 3.0 menjadi 0.0, 1.0 dan 2.0 karena Spark mengharapkan label dari 0 ke N-1 di mana N adalah jumlah total kelas dalam data. Kelas RandomForest dari modul klasifikasi MLlib digunakan untuk membangun model Random Forest. Setelah model dibuat, metode prediksi digunakan untuk membuat prediksi. Terakhir, kami membandingkan label dalam dataset pengujian dan label yang diprediksi dan menghitung kesalahan pengujian. Prediksi dapat disimpan ke file teks menggunakan fungsi saveAsTextFile.

**■ Box 11.20: Python program for Random Forest classification using Spark MLlib**

```
import sys

from pyspark.context import SparkContext
from pyspark.mllib.tree import RandomForest, RandomForestModel
from pyspark.mllib.linalg import Vectors
from pyspark.mllib.regression import LabeledPoint

def parseLine(line):
   parts = line.split(',')
   label = float(parts[0])
   if label==1.0:
      label =0.0
   if label==2.0:
      label = 1.0
   if label==3.0:
      label = 2.0
   features = Vectors.dense([float(x) for x in parts[1:]])
   return LabeledPoint(label, features)
```

```
sc = SparkContext(appName="RandomForestExample")

trainingData = sc.textFile('file:///home/hadoop/wine.data.txt').map(parseLine)

testData = sc.textFile('file:///home/hadoop/wine.test.txt').map(parseLine)

model = RandomForest.trainClassifier(trainingData, numClasses=3,
      categoricalFeaturesInfo=,
      numTrees=3, featureSubsetStrategy="auto",
      impurity='gini', maxDepth=4, maxBins=32)

predictions = model.predict(testData.map(lambda x:  x.features))

labelsAndPredictions = testData.map(lambda lp:
lp.label).zip(predictions)

labelsAndPredictions.take(10)
#[(0.0, 0.0), (0.0, 0.0), (1.0, 1.0), (1.0, 1.0), (1.0, 1.0), (2.0, 2.0),
(2.0, 2.0), (2.0, 2.0), (2.0, 2.0)]


testErr = labelsAndPredictions.filter(lambda v_p:
v_p[0] != v_p[1]).count()/float(testData.count())

print('Test Error = ' + str(testErr))
#0

labelsAndPredictions.saveAsTextFile('file:///home/hadoop/prediction.txt')
```

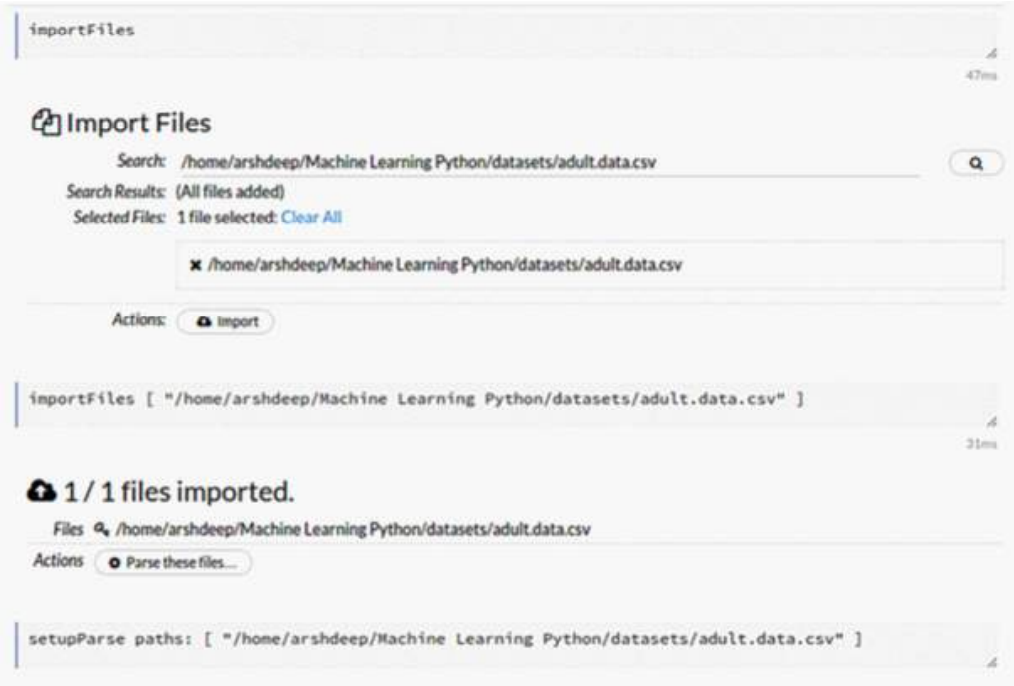
### 11.3.6 Gradient Boosting Machine

Gradient Boosting Machine adalah algoritma pembelajaran ensemble seperti Random Forest. Ketika Decision Trees digunakan sebagai model prediksi dalam Gradient Boosting, model tersebut disebut Gradient-Boosted Trees (GBT). Di Random Forest, setiap pohon dibangun secara independen dari sampel acak (bootstrap), sedangkan di GBT pohon keputusan dilatih pada setiap langkah yang mengoreksi dan melengkapi pohon yang dibangun sebelumnya. Sedangkan di Random Forest, ensembel dibangun dari sampel data acak, di GBT ensembel dibangun di atas residual (kesalahan pohon sebelumnya). Peningkatan Gradien adalah proses yang menggabungkan pelajar yang lemah menjadi pelajar yang kuat dengan secara berurutan meningkatkan pelajar di setiap langkah dengan menambahkan penaksir baru yang dilatihkan pada sisa di setiap langkah [45]. Misalkan fm (x) menjadi model pada langkah m. Pada langkah m + 1 model ditingkatkan sebagai berikut:

$$Fm+1(x) = fm(x)+h(x)$$

di mana h (x) adalah penduga baru yang dilatih pada residual, yaitu, h(x) = y−fm(x).

Mari kita lihat contoh lain dari klasifikasi GBM menggunakan kerangka kerja H2O. Untuk contoh yang berbeda kita akan menggunakan dataset UCI Adult [41] yang juga disebut dataset Census Income. Dataset ini mencakup data pendapatan sensus dengan atribut seperti usia, pendidikan, pekerjaan, dll. Tugas prediksi untuk dataset ini adalah untuk menentukan apakah seseorang berpenghasilan lebih dari 50 ribu setahun. Mari kita impor dulu file dataset menggunakan H2O Flow UI seperti yang ditunjukkan pada Gambar 11.38.




Gambar 11.38: *Mengimpor file dataset dari H2O Flow UI*

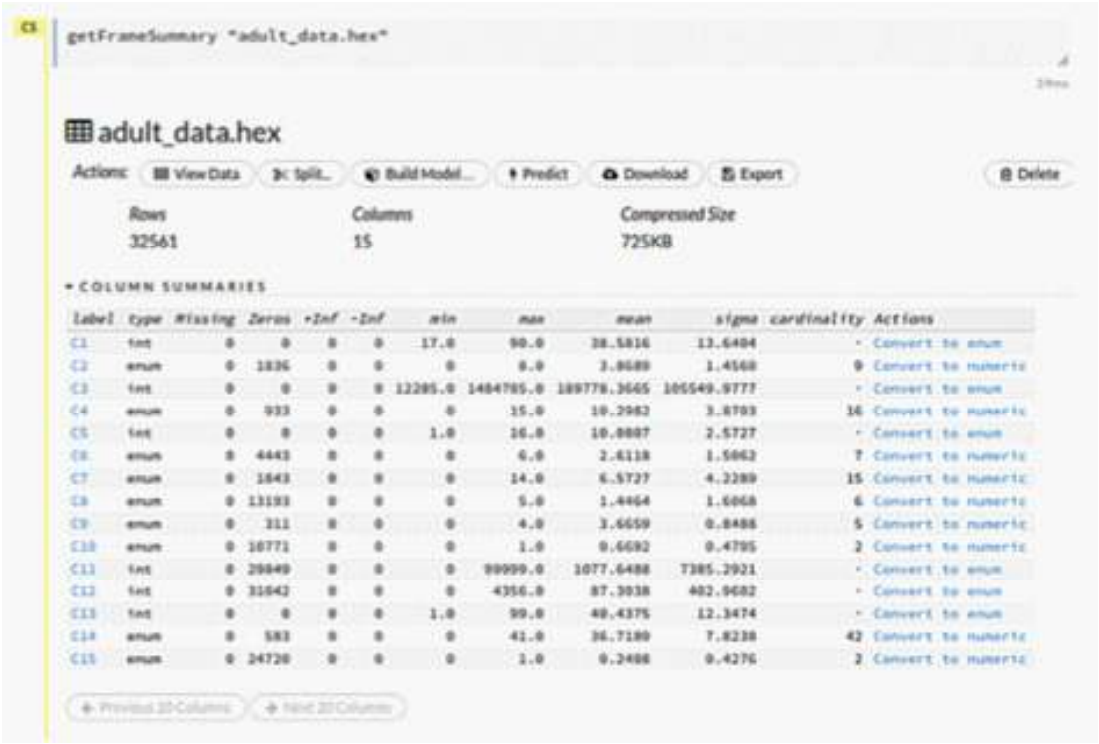
Langkah selanjutnya adalah mengurai file dataset. Gambar 11.39 menunjukkan bagaimana menyiapkan parser. Setelah memilih opsi parse, klik tombol Parse untuk mengurai file. Data dari file yang diurai disimpan dalam bingkai H2O. Gambar 11.40 menunjukkan frame H2O yang dibuat dengan mem-parsing file dataset.

Dengan data yang diimpor dan diurai, mari kita sekarang membangun model GBM. Klik tombol Build Model dalam tindakan frame yang diurai atau pilih opsi Build Model dari menu. Gambar 11.41 menunjukkan berbagai opsi untuk model. Pilih tipe algoritma untuk menjadi Gradient Boosting Machine, frame pelatihan sebagai adult_data.hex, kolom respon sebagai C15 dan jumlah pohon menjadi 50. Setelah menentukan opsi model, klik tombol Build Model untuk membangun model.

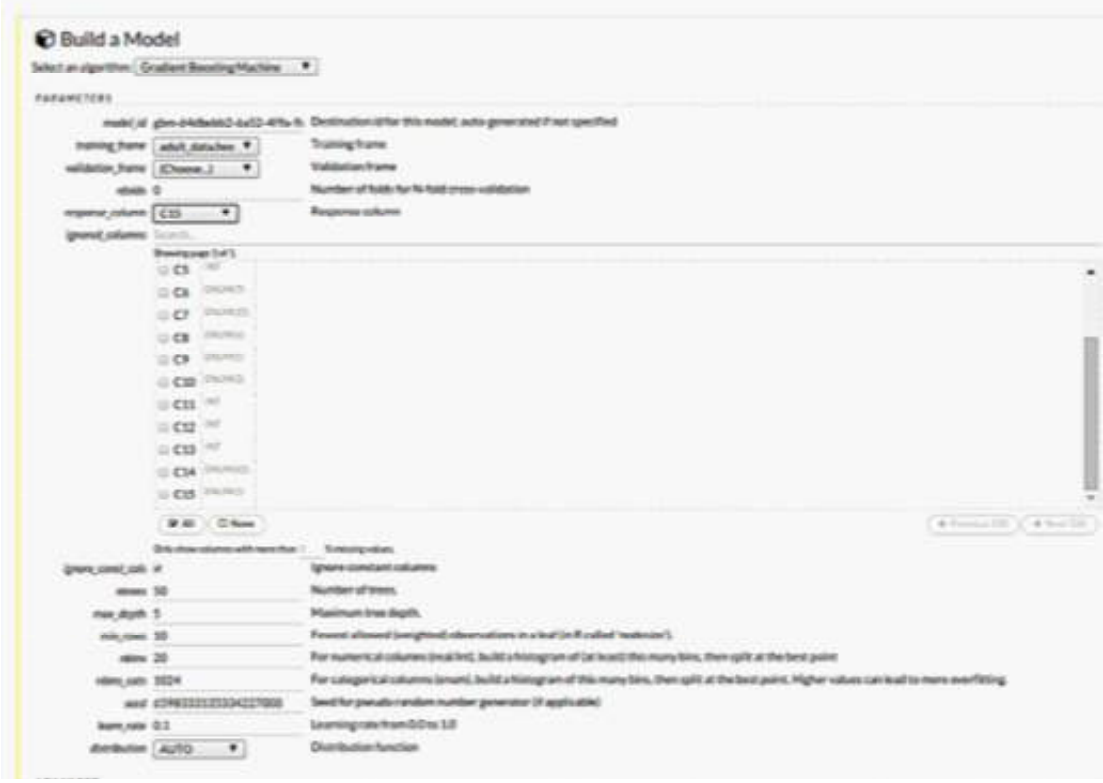

Gambar 11.39: *Parsing file dataset yang diimpor menggunakan H2O Flow UI*

Gambar 11.42menunjukkan detail model GB yang dibangun dari dataset Internet. Ringkasan model menunjukkan berbagai statistik model seperti mean squared error (MSE), koefisien determinasi (R2) dan area di bawah kurva (AUC).

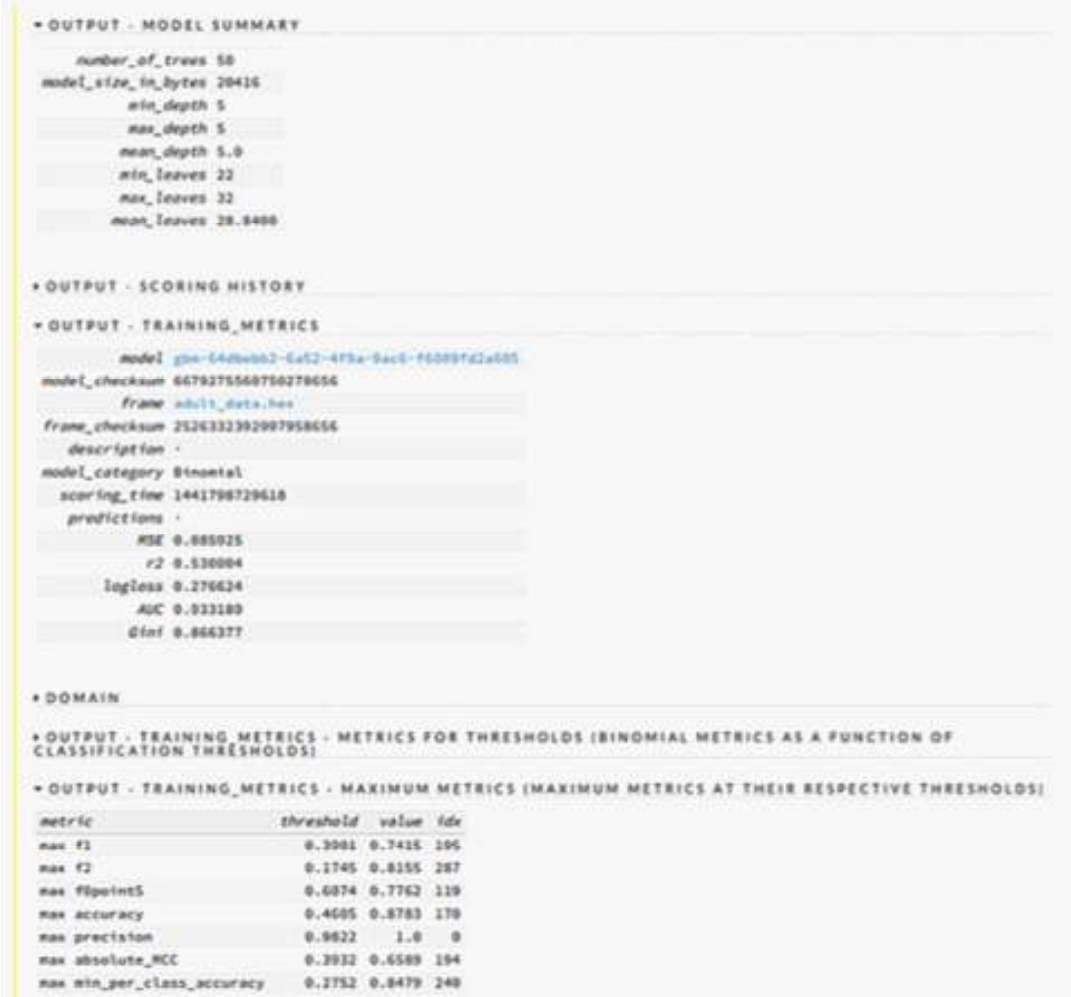
Selanjutnya, impor dan parsing dataset pengujian dengan cara yang sama seperti kita mengimpor dan mengurai dataset pelatihan. Untuk membuat prediksi untuk dataset pengujian, masukkan perintah prediksi atau pilih opsi prediksi dari menu, lalu pilih model dan bingkai pengujian, lalu klik tombol Prediksi. Gambar 11.43 menunjukkan ringkasan prediksi termasuk kurva KOP.




Gambar 11.40: *Melihat bingkai H2O yang dibuat dari file dataset yang diurai*



Gambar 11.41: *Membangun model GBM dari H2O Flow UI*



▾ OUTPUT - MODEL SUMMARY

| | |
|---|---|
| number_of_trees | 50 |
| model_size_in_bytes | 20416 |
| min_depth | 5 |
| max_depth | 5 |
| mean_depth | 5.0 |
| min_leaves | 22 |
| max_leaves | 32 |
| mean_leaves | 28.8400 |

▸ OUTPUT - SCORING HISTORY

▾ OUTPUT - TRAINING_METRICS

| | |
|---|---|
| model | gbm-64dbe0b2-6a52-4f8a-9ac6-f6089fd2a605 |
| model_checksum | 6679275560750278656 |
| frame | adult_data.hex |
| frame_checksum | 2526332392907958656 |
| description | - |
| model_category | Binomial |
| scoring_time | 1441798729618 |
| predictions | - |
| MSE | 0.085025 |
| r2 | 0.530004 |
| logloss | 0.276624 |
| AUC | 0.933189 |
| Gini | 0.866377 |

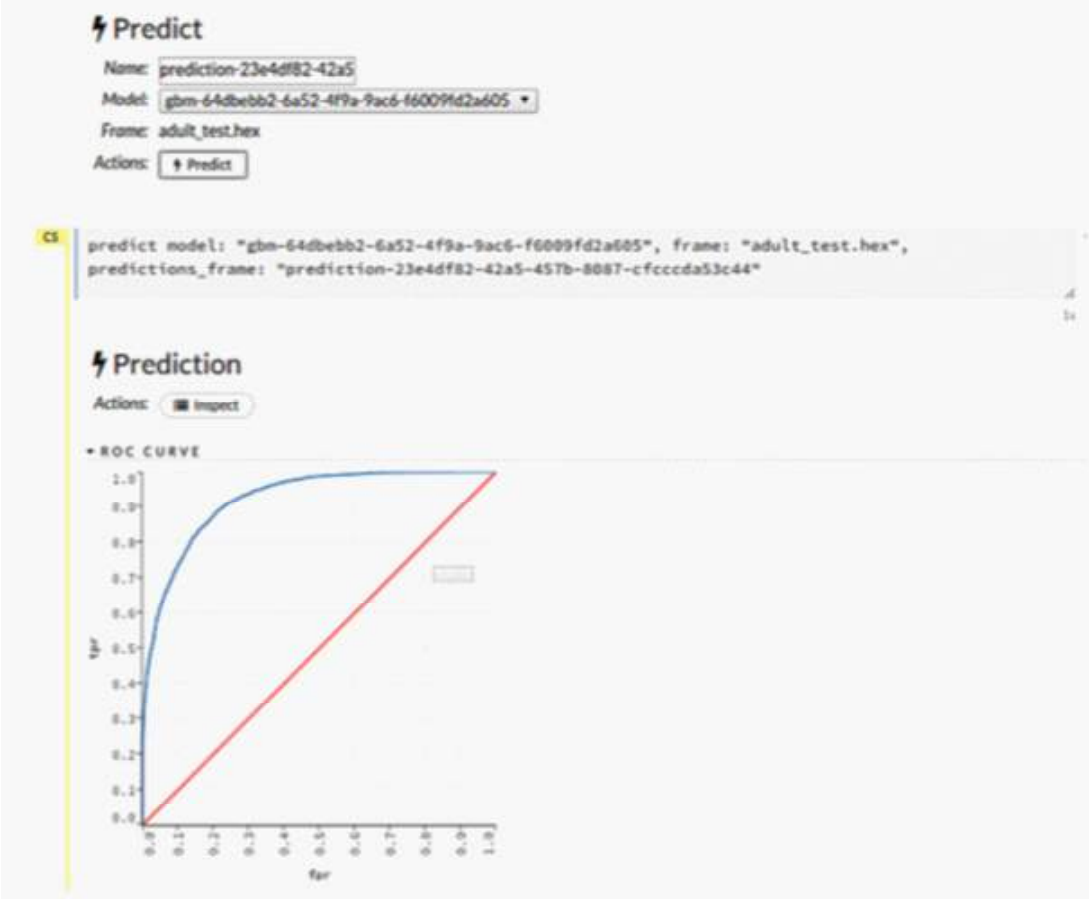
▸ DOMAIN

▸ OUTPUT - TRAINING_METRICS - METRICS FOR THRESHOLDS (BINOMIAL METRICS AS A FUNCTION OF CLASSIFICATION THRESHOLDS)

▾ OUTPUT - TRAINING_METRICS - MAXIMUM METRICS (MAXIMUM METRICS AT THEIR RESPECTIVE THRESHOLDS)

| metric | threshold | value | idx |
|---|---|---|---|
| max f1 | 0.3981 | 0.7415 | 195 |
| max f2 | 0.1745 | 0.8155 | 287 |
| max f0point5 | 0.6874 | 0.7762 | 119 |
| max accuracy | 0.4605 | 0.8783 | 170 |
| max precision | 0.9622 | 1.0 | 0 |
| max absolute_MCC | 0.3932 | 0.6589 | 194 |
| max min_per_class_accuracy | 0.2752 | 0.8479 | 240 |

Gambar 11.42: *Melihat detail model GBM*




Gambar 11.43: *Melihat hasil prediksi*

Box 11.21 menunjukkan implementasi Python dari contoh yang sama menggunakan pustaka Python H2O dan Box 11.22 menunjukkan output program.

**Box 11.21: Python program for GBM classification using H2O**

```
import h2o

h2o.init()

train = h2o.import_frame(path=h2o.locate("/home/ubuntu/adult.data.csv"))
test = h2o.import_frame(path=h2o.locate("/home/ubuntu/adult.test.csv"))

train[14] = train[14].asfactor()
test[14] = test[14].asfactor()

model = h2o.gbm(x=train[0:14], y=train[14], validation_x= test[0:14],
        validation_y=test[14], distribution = "bernoulli",
        ntrees=50, learn_rate=0.1)
```

```
model.show()

prediction = model.predict(test)

prediction.head()

perf = model.model_performance(test)
perf.show()

print 'Confusion Matrix: '
print (perf.confusion_matrix())

print 'Precision: '
print (perf.precision())

print 'Accuracy: '
print (perf.accuracy())

print 'AUC: '
print (perf.auc())

h2o.download_csv(prediction, 'prediction.csv')
```

**■ Box 11.22: Output of program for GBM classification using H2O**

```
>>> model.show()
Model Details
=============
H2OBinomialModel :  Gradient Boosting Machine
Model Key:  GBM_model_python_1441787876875_57

Model Summary:

 number_of_trees model_size_in_bytes min_depth max_depth mean_depth min_leaves max_leaves mean_leaves
- --------------- ------------------- --------- --------- ---------- ---------- ---------- -----------
 50    20417    5  5  5   22  32  28.84


ModelMetricsBinomial:  gbm
** Reported on train data.  **

MSE: 0.0859247340554
R^2:  0.530004451115
LogLoss:  0.276624227866
AUC: 0.933188510192
Gini:  0.866377020384

Confusion Matrix (Act/Pred) for max f1 @ threshold = 0.390088332148:

   <=50K >50K Error Rate
--- ----- ---- ----- ----------
<=50K 22634 2086 0.0844 (2086.0/24720.0)
>50K 1992  5849 0.254 (1992.0/7841.0)
Total 24626 7935 0.1252 (4078.0/32561.0)

Maximum Metrics:

metric       threshold value  idx
```

```
------------------ ------- ----- ---
max f1             0.390088  0.741506 195
max f2             0.174504  0.815484 287
max f0point5       0.607379  0.776164 119
max accuracy       0.460468  0.87829 170
max precision      0.982202  1    0
max absolute_MCC   0.393242  0.658941 194
max min_per_class_accuracy 0.275248  0.847851 240

ModelMetricsBinomial:  gbm
** Reported on validation data.  **

MSE: 0.0905793133771
R^2:  0.497962558845
LogLoss:  0.28949793511
AUC: 0.92284044478
Gini:  0.84568088956

Confusion Matrix (Act/Pred) for max f1 @ threshold = 0.36326916669:

   <=50K >50K Error Rate
--- ----- ---- ----- -----------
<=50K 11150 1285 0.1033 (1285.0/12435.0)
>50K 992  2854 0.2579 (992.0/3846.0)
Total 12142 4139 0.1399 (2277.0/16281.0)

Maximum Metrics:

metric             threshold value  idx
------------------ ------- ----- ---
max f1             0.363269  0.71484 211
max f2             0.172859  0.799287 291
max f0point5       0.553204  0.757278 141
max accuracy       0.514136  0.871568 155
max precision      0.9865  1    0
max absolute_MCC   0.468827  0.625792 169
max min_per_class_accuracy 0.26159  0.836973 250

Variable Importances:

variable relative_importance scaled_importance percentage
-------- -------------------- ------------------ ----------
C8    5012.7       1       0.302438
C11    3214.44      0.641258     0.193941
C4    2816.11      0.561795     0.169908
C6    1444.81      0.28823     0.0871717
C7    1423.55      0.283988     0.0858887
C12    921.112      0.183756     0.0555747
C1    780.083      0.155621     0.0470658
C13    464.666      0.0926977    0.0280353
C14    195.269      0.0389548    0.0117814
C2    184.469      0.0368003    0.0111298
C3    53.7663      0.010726     0.00324396
C10    41.7206      0.00832298    0.00251718
C5    17.0094      0.00339327    0.00102625
C9    4.61617      0.000920896    0.000278514
>>> prediction = model.predict(test)
>>>
>>>prediction.head()
First 10 rows and first 3 columns:
predict   <=50K   >50K
------- ------ ------
<=50K   0.984939 0.015061
```

```
[[0.5141359851152871, 0.8715680854984338]]
>>>
>>> print 'AUC: '
AUC:
>>> print (perf.auc())
0.92284044478
```

Mari kita ulangi contoh yang sama dari klasifikasi GBM menggunakan Spark MLlib dengan dataset Dewasa. Box 11.23 menunjukkan program python untuk klasifikasi GBM menggunakan Spark MLlib. Karena beberapa atribut dalam kumpulan data ini adalah non-angka (atribut string), kami menggunakan kelas Tokenizer dan HashingTF dari Spark untuk ekstraksi dan transformasi fitur. Tokenizer mengubah string input menjadi huruf kecil dan kemudian membaginya dengan spasi putih. HashingTF adalah transformator yang mengambil kumpulan istilah (kumpulan kata) dan mengubah kumpulan tersebut menjadi vektor fitur panjang tetap (frekuensi istilah). Perhatikan juga bahwa dalam contoh ini kami mengonversi label dari <= 50and> 50 menjadi0.0.and1.0 karena Spark mengharapkan label dari 0 ke N-1 di mana Nisthetotal jumlah kelas dalam data. Kelas GradientBoostedTrees dari modul klasifikasi MLlib digunakan untuk membangun model GBM. Setelah model dibuat, metode prediksi dari kelas GradientBoostedTrees digunakan untuk membuat prediksi.

**■ Box 11.23: Python program for GBM classification using Spark MLlib**

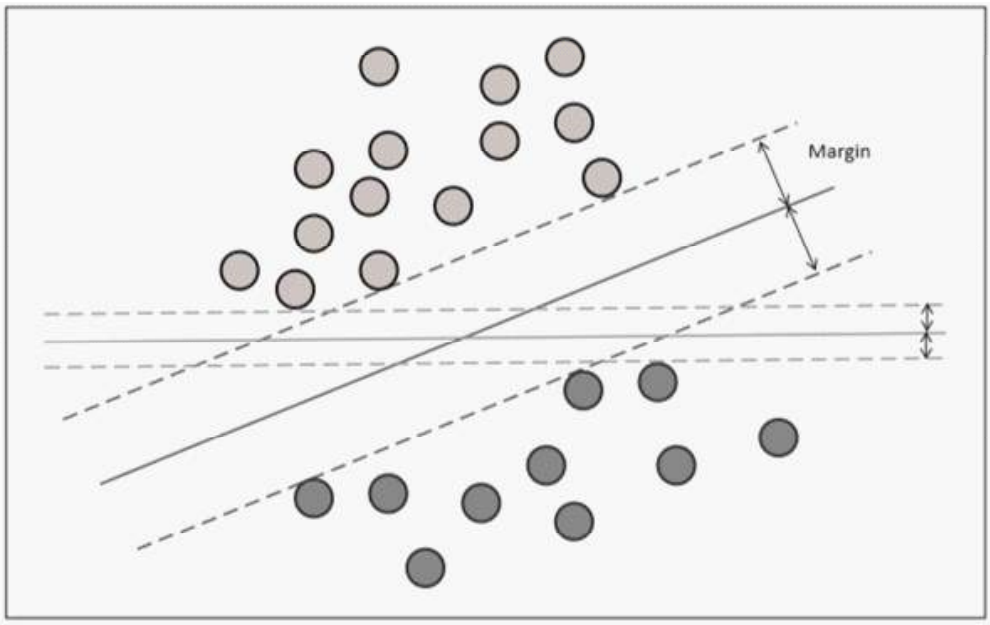
```
from pyspark.context import SparkContext
from pyspark.mllib.tree import GradientBoostedTrees
from pyspark.mllib.regression import LabeledPoint
from pyspark.mllib.feature import HashingTF
from pyspark.ml.feature import HashingTF, Tokenizer
from pyspark.sql import Row

LabeledDocument = Row("text", "label")

def parseLine(line):
parts = line.split(',')
label = parts[-1].strip()
if label=='<=50K':
label =0.0
if label=='>50K':
label =1.0
text=line
return LabeledDocument(text, label)

lines = sc.textFile('file:///home/hadoop/data.csv').map(parseLine)

training = lines.toDF()

training.show()
```

```
tokenizer = Tokenizer(inputCol="text", outputCol="words")
hashingTF = HashingTF(inputCol=tokenizer.getOutputCol(),
outputCol="features")

tokenized = tokenizer.transform(training)
hashed = hashingTF.transform(tokenized)
```

```
hashedrdd=hashed.select('label','features').rdd

data=hashedrdd.map(lambda a:  LabeledPoint(float(a.label), a.features))

(trainingData, testData) = data.randomSplit([0.7, 0.3])

model = GradientBoostedTrees.trainClassifier(p, categoricalFeaturesInfo=,
numIterations=3)

predictions = model.predict(p.map(lambda x:  x.features))

predictions.take(10)
```

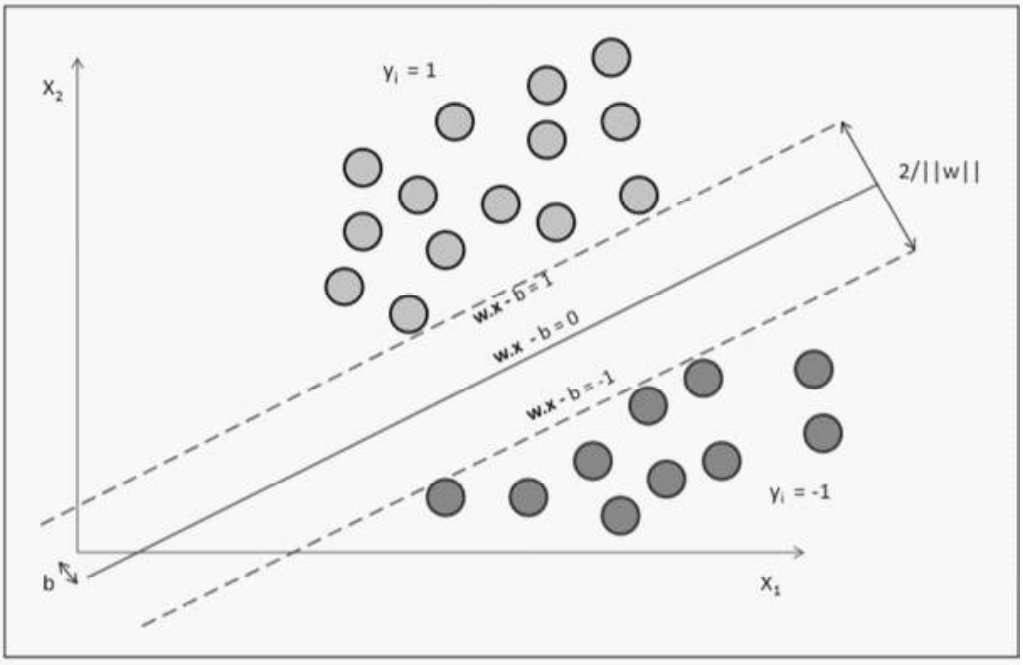
### 11.3.7 Support Vector Machine

Support Vector Machine (SVM) adalah pendekatan pembelajaran mesin yang diawasi yang digunakan untuk klasifikasi dan regresi. Bentuk dasar dari SVM adalah kelas biner yang mengklasifikasikan titik data ke dalam salah satu kelas [37]. Pelatihan SVM melibatkan penentuan hyperplane margin maksimum yang memisahkan kedua kelas. Hyperplane margin maksimum adalah yang memiliki pemisahan terbesar dari titik data pelatihan terdekat. Gambar 11.44 menunjukkan margin untuk SVM. Diketahui satu dataset pelatihan (xi, yi) di mana xi adalah sebuah vektor berdimensi n dan yi = 1ifxi adalah kelas1andyi = −1ifxi adalah kelas2, astandardSVMfndsahyperplane wx − b = 0, yang mengoreksipemisahan pelatihandatapointsandhasama−ximummargin yang merupakan jarak antara dua hyperplanes = 1.x −bargin x − b = −1, seperti yang ditunjukkan pada Gambar 11.45.




Gambar 11.44: *Margins untuk sebuah SVM*

Hyperplane optimal dengan margin maksimum dapat diperoleh dengan menyelesaikan masalah pemrograman kuadratik berikut,




Gambar 11.45: *Margin Maximum hyperplane*

$$f(x) = sign\left(C\sum_{i=1}^{l}\alpha_i y_i K(x_i, x) - b\right) \quad (11.23)$$

tunduk pada yi (w.xi − b) ≥1 − ξi, ξi> 0, 1 <i <I dimana C adalah parameter soft margin dan ξ adalah variabel slack untuk kasus yang tidak dapat dipisahkan. Hyperplane optimal diberikan sebagai,

$$min_{\{w,b\}}\frac{1}{2}||w||^2 + C\sum_{i=1}^{l}\xi_i \quad (11.22)$$

dimanaαi adalah pengali Lagrange dan K (xi, x) adalah fungsi kernel. SVM standar adalah kelas dua kelas di mana hasilnya adalah 1 atau − 1. Jika tidak dapat dipisahkan, titik data dalam ruang berdimensi terbatas asli dimapkan ke ruang berdimensi lebih tinggi di mana mereka dapat dipisahkan dengan mudah. Kinerja klasifikasi SVM bergantung pada pemilihan kernel, parameter kernel, dan parameter margin halusC. Kernel yang umum digunakan meliputi:

- Linear: k(xi,xj) =< xi,xj >
- Polynomial: k(xi,xj) = (γ < xi,xj > +r)d
- Radial Basis Function (RBF): k(xi,xj) = exp(−γ||xi−xj||2)
- Sigmoid: k(xi,xj) = (tanh < xi,xj > +r)

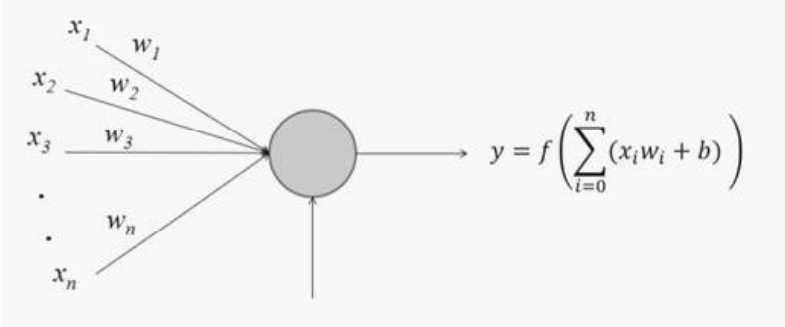
Mari kita lihatexampleofSVMclassi fi kationusingSparkMLlib. Untuk contoh keempat, kami akanmenggunakandatasetWined. Box 11.24 menunjukkan program Python untuk klasifikasi SVM. Program ini dapat dijalankan di PySparkshell. Dalam program ini kami mengimplementasikan fungsi parseLine yang mengambil setiap baris dari file input, membagi baris menjadi kolom individual yang dipisahkan oleh koma, mengubah nilai menjadi mengambang dan mengembalikan LabeledPoints. Dalam fungsi ini, kami juga mengubah label anggur dari 1.0, 2.0 dan 3.0 menjadi 0.0, 1.0 dan 2.0 karena Spark mengharapkan label dari 0 ke N-1 di mana N adalah jumlah total kelas dalam data. Kelas SVMWithSGD dari modul klasifikasi MLlib digunakan untuk membangun model SVM. Setelah model dibangun, metode prediksi digunakan untuk membuat prediksi. Terakhir, kami membandingkan label dalam dataset pengujian dan label yang diprediksi serta menghitung kesalahan pengujian. Prediksi dapat disimpan ke file teks menggunakan fungsi saveAsTextFile.

**Box 11.24: Python program for SVM classification using Spark MLlib**

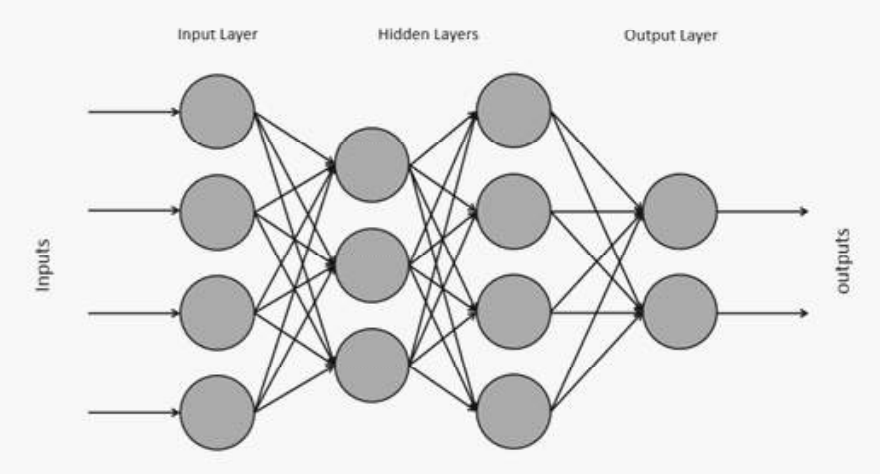
```
from pyspark.context import SparkContext
from pyspark.mllib.tree import SVMWithSGD
from pyspark.mllib.linalg import Vectors
from pyspark.mllib.regression import LabeledPoint

def parseLine(line):
   parts = line.split(',')
   label = float(parts[0])
   if label==1.0:
      label =0.0
   if label==2.0:
      label = 1.0
   if label==3.0:
      label = 2.0
   features = Vectors.dense([float(x) for x in parts[1:]])
   return LabeledPoint(label, features)

sc = SparkContext(appName="SVMExample")

trainingData = sc.textFile('file:///home/hadoop/wine.data.txt').map(parseLine)

testData = sc.textFile('file:///home/hadoop/wine.test.txt').map(parseLine)

model = SVMWithSGD.train(trainingData, iterations=100)

predictions = model.predict(testData.map(lambda x:  x.features))

labelsAndPredictions = testData.map(lambda lp:
lp.label).zip(predictions)

testErr = labelsAndPredictions.filter(lambda v_p:
v_p[0] != v_p[1]).count()/float(testData.count())

print('Test Error = ' + str(testErr))

labelsAndPredictions.saveAsTextFile('file:///home/hadoop/prediction.txt')

sc.stop()
```

### 11.3.8 Deep Learning

Algoritma Pembelajaran Mendalam didasarkan pada jaringan saraf tiruan. Jaringan saraf artifisial terinspirasi dari jaringan saraf biologis dan mencakup sistem neuron yang saling berhubungan. Gambar 11.46 menunjukkan struktur neuron dalam jaringan saraf tiruan. Neuron memiliki banyak masukan (xi) dan setiap masukan memiliki bobot (wi). Kombinasi masukan yang berbobot digabungkan, dan fungsi aktivasi (f) diterapkan ke masukan gabungan. Sebuah bias b juga ditambahkan yang menjelaskan ambang aktivasi neuron.




Gambar 11.46: *Neuron*

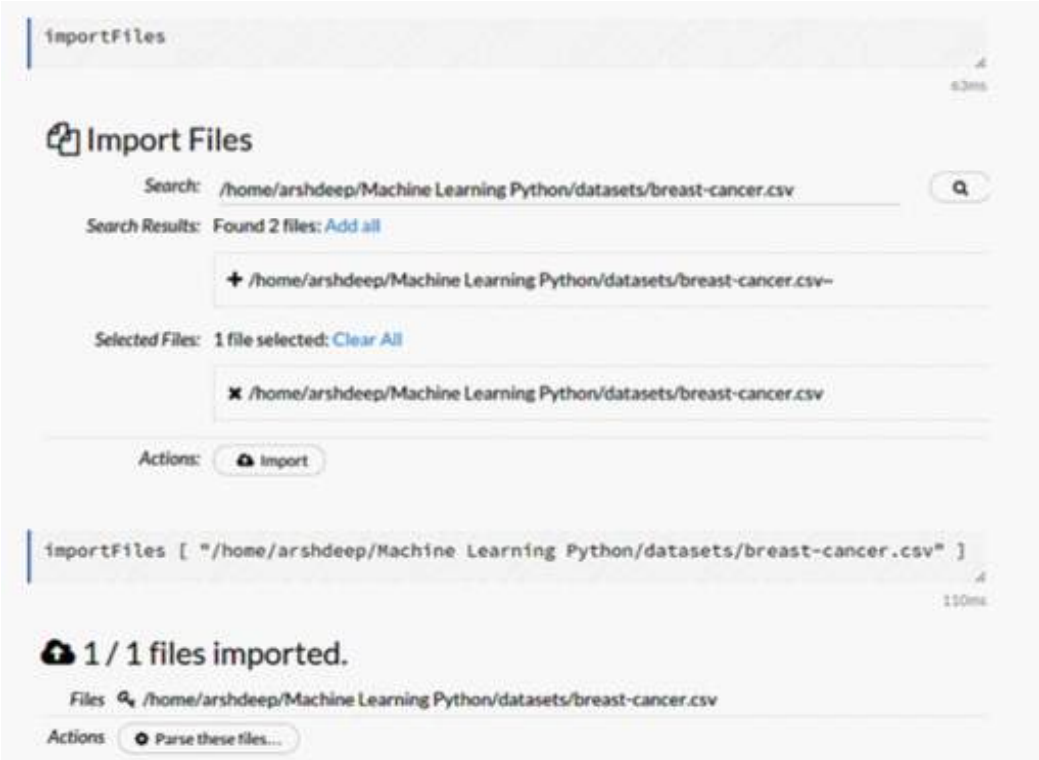



Gambar 11.47: *Multi-layer feed-forward artificial neural network*

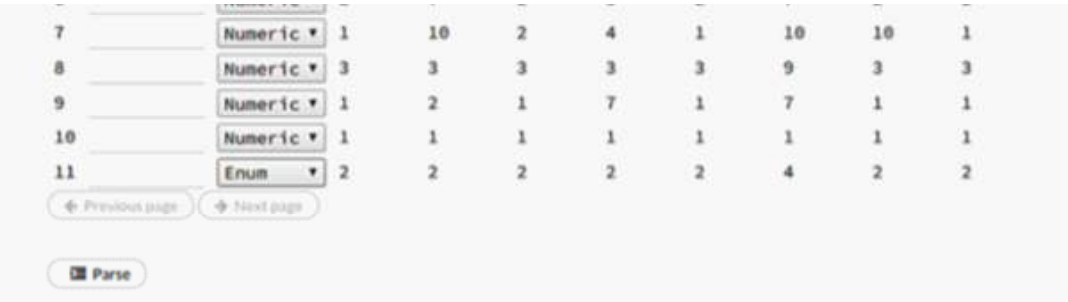
Di bagian ini, kita akan melihat contoh Deep Learning menggunakan kerangka kerja H2O. Implementasi pembelajaran dalam H2O didasarkan pada jaringan saraf tiruan buatan multi-layer feed-forward yang mencakup beberapa layer neuron yang saling berhubungan seperti yang ditunjukkan pada Gambar 11.47. Neuron di setiap layer terhubung langsung ke neuron di layer berikutnya dan neuron terdekat dalam jaringan. Kemudian jumlah neuron di layer masukan sama dengan jumlah fitur di masukan dan jumlah neuron di layer output sama dengan jumlah output. Dalam proses pembelajaran, dengan masukan dan output yang diketahui, sistem menyesuaikan bobot untuk meminimalkan kesalahan prediksi.

Mari kita lihat contoh Deep Learning menggunakan kerangka kerja H2O. Dalam contoh ini, kami akan menggunakan dataset kanker payudara UCI [42]. Dataset terdiri dari pengukuran sepuluh atribut yang masing-masing menggambarkan fitur yang dihitung dari gambar digital dari fineneedleaspirate (FNA) massa payudara. Variabel kelas memiliki dua nilai (2 untuk jinak, 4 untuk ganas).

Langkah pertama adalah mengimpor file dataset ke H2O dari H2O Flow UI seperti yang ditunjukkan pada gamba 11.48

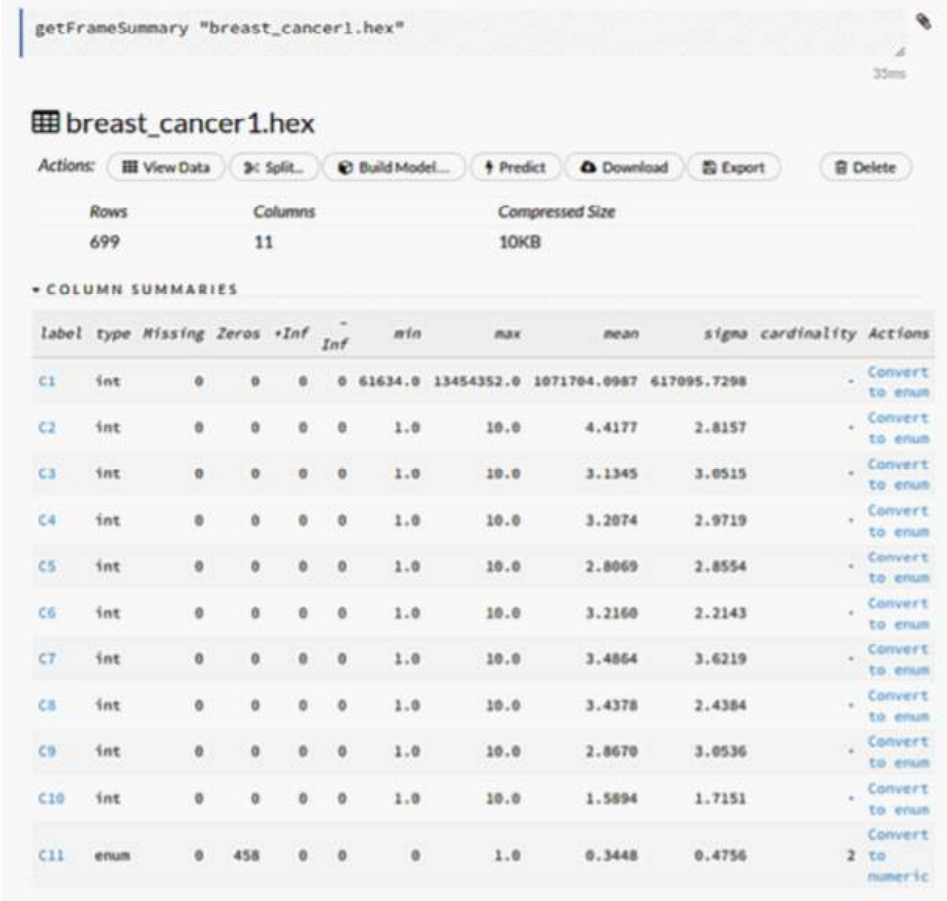



Gambar 11.48: *Mengimpor file dataset dari H2O Flow UI*




Gambar 11.49: *Parsing file dataset yang diimpor menggunakan H2O Flow UI*

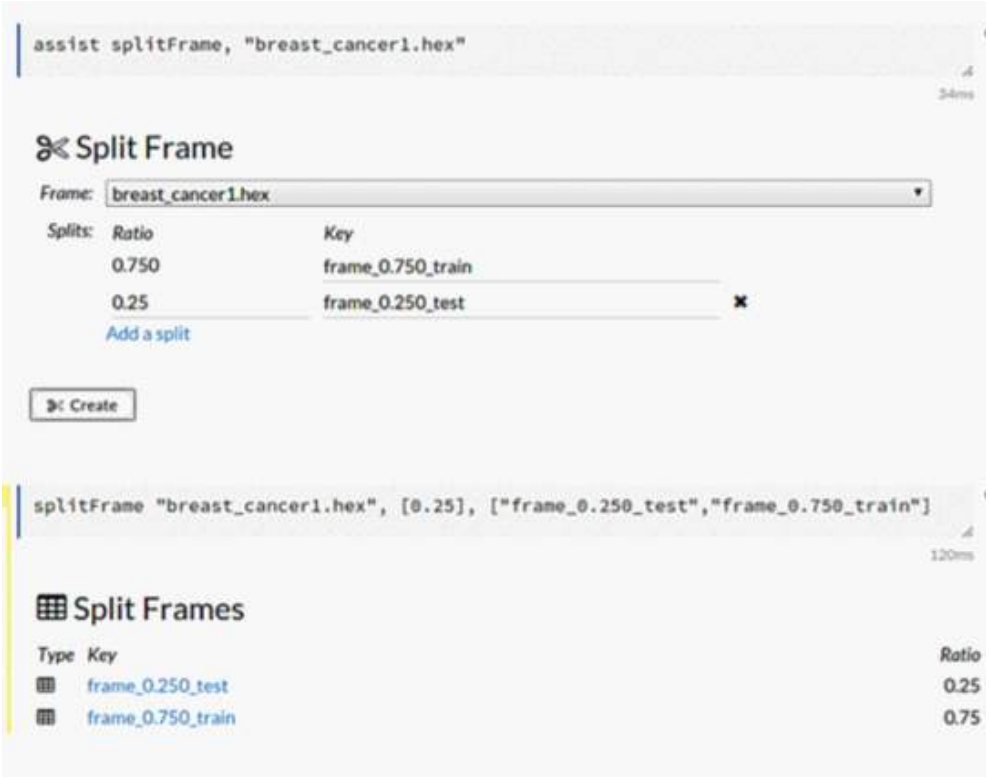
Langkah selanjutnya adalah mengurai file dataset. Klik tombol Parse setelah mengimpor file. Gambar 11.49 menunjukkan bagaimana menyiapkan parser. Pada langkah ini, Anda dapat menentukan berbagai opsi penguraian. Setelah memilih opsi parse, klik tombol Parse untuk mengurai file. Data dari file yang diurai disimpan dalam bingkai H2O. Gambar 11.50 menunjukkan frame H2O yang dibuat dengan mengurai file dataset. Data tersebut dibagi menjadi kerangka pelatihan dan pengujian seperti yang ditunjukkan pada Gambar 11.51.




Gambar 11.50: *Melihat bingkai H2O yang dibuat dari file dataset yang diurai*

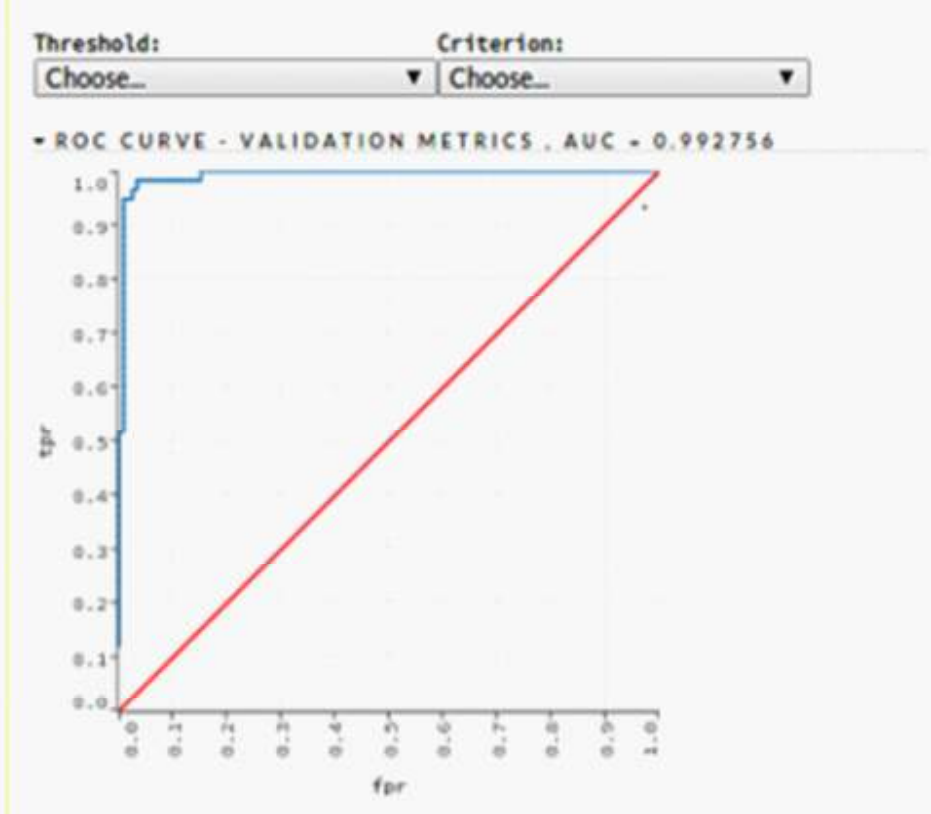
Dengan data yang diimpor dan diurai, sekarang mari kita membuat model Pembelajaran Mendalam. Klik tombol Bangun Model dalam tindakanframe yang dipisahkan atau pilih opsiBuildModeldari menu. Gambar 11.52 menunjukkan berbagai opsi untuk model. Pilih tipe algoritma menjadi Pembelajaran Mendalam, frame pelatihan (frame_0.750_train), frame validasi (frame_0.250_test) dan kolom respon sebagai C11. Setelah menentukan opsi model, klik tombol Build Model untuk membangun model.

Gambar 11.53 dan 11.54 menunjukkan rincian model Deep Learning seperti kurva ROC training dan validasi, serta berbagai statistik model.




Gambar 11.51: *Memisahkan kerangka H2O menjadi kerangka pelatihan dan pengujian*


Gambar 11.52: *Membangun model pembelajaran mendalam dari H2O Flow UI*




Gambar 11.53: *Melihat detail model deep learning*

OUTPUT - VALIDATION_METRICS

| | |
|---|---|
| model | deeplearning-0d64488c-656d-4137-ba5a-eda74d2c7822 |
| model_checksum | 4447524057521643008 |
| frame | |
| frame_checksum | 0 |
| description | Metrics reported on temporary (load-balanced) validation frame |
| model_category | Binomial |
| scoring_time | 1441795605381 |
| predictions | |
| MSE | 0.020251 |
| r2 | 0.867228 |
| logloss | 0.155852 |
| AUC | 0.992756 |
| Gini | 0.985511 |

OUTPUT - VALIDATION_METRICS - METRICS FOR THRESHOLDS (BINOMIAL METRICS AS A FUNCTION OF CLASSIFICATION THRESHOLDS)

OUTPUT - VALIDATION_METRICS - MAXIMUM METRICS (MAXIMUM METRICS AT THEIR RESPECTIVE THRESHOLDS)

| metric | threshold | value | idx |
|---|---|---|---|
| max f1 | 0.9948 | 0.9640 | 46 |
| max f2 | 0.2392 | 0.9727 | 51 |
| max f0point5 | 0.9948 | 0.9752 | 46 |
| max accuracy | 0.9948 | 0.9774 | 46 |
| max precision | 1.0 | 1.0 | 0 |
| max absolute_MCC | 0.9948 | 0.9486 | 46 |
| max min_per_class_accuracy | 0.2392 | 0.9664 | 51 |

Gambar 11.54: *Melihat hasil validasi model deep learning*

Box 11.25 menunjukkan implementasi Python dari contoh yang sama menggunakan pustaka Python H2O dan Box 11.26 menunjukkan output program.

**Box 11.25: Python program for Deep Learning classification using H2O**

```
import h2o

h2o.init()

data = h2o.import_frame(path=h2o.locate("/home/ubuntu/breast-cancer.csv"))

data_split = data.split_frame(ratios = [0.8,0.2])
train = data_split[0]
test = data_split[1]

train[10] = train[10].asfactor()
test[10] = test[10].asfactor()

model = h2o.deeplearning(x=train[0:10], y=train[10],
   validation_x= test[0:10], validation_y=test[10],
   variable_importances=True, loss="Automatic")

model.show()

prediction = model.predict(test)

prediction.head()
```

```
perf = model.model_performance(test)
perf.show()

print 'Confusion Matrix:  '
print (perf.confusion_matrix())

print 'Precision:  '
print (perf.precision())

print 'Accuracy:  '
print (perf.accuracy())

print 'AUC: '
print (perf.auc())

h2o.download_csv(prediction, '/home/ubuntu/prediction.csv')
```

**Box 11.26: Output of the program for Deep Learning classification using H2O**

```
>>> model.show()
Model Details
=============
H2OBinomialModel :  Deep Learning
Model Key:  DeepLearning_model_python_1441787876875_43

Status of Neuron Layers:
```

```
 layer units type  dropout  l1 l2 mean_rate  rate_RMS  momentum mean_weight weight_RMS mean_bias  bias_RMS
- ----- ----- ------ ------- --- --- --------- --------- ------- ---------- --------- ---------- ---------
 1  10  Input  0.0
 2  200  Rectifier 0.0  0.0 0.0 0.0051586363 0.004284294 0.0   -0.0038918303 0.09932074 0.47159594  0.01983153
 3  200  Rectifier 0.0  0.0 0.0 0.042859647 0.12210885 0.0   -0.0014745063 0.06997566 0.99559456  0.007935624
 4  2  Softmax    0.0 0.0 0.0019927532 0.0024083003 0.0   -0.03239843 0.40508103 7.2054856e-05 0.0016424061


ModelMetricsBinomial:  deeplearning
** Reported on train data.  **

MSE: 0.0431564371515
R^2:  0.814549813835
LogLoss:  0.287324374181
AUC: 0.99416925658
Gini:  0.98833851316

Confusion Matrix (Act/Pred) for max f1 @ threshold = 0.954812765121:

   2  4 Error Rate
--- --- --- ------ ---------
2   339 14 0.0397 (14.0/353.0)
4   2 204 0.0097 (2.0/206.0)
Total 341 218 0.0286 (16.0/559.0)

Maximum Metrics:

metric         threshold  value   idx
---------------- ------- ------- ---
max f1          0.954813   0.962264 63

max f2          0.913903   0.980861 66
max f0point5       0.999977   0.961945 30
max accuracy       0.992785   0.971377 59
max precision       0.999999   0.993711 9
max absolute_MCC      0.954813   0.940216 63
max min_per_class_accuracy 0.993953   0.968839 56

ModelMetricsBinomial:  deeplearning
** Reported on validation data.  **

MSE: 0.0254503614803
R^2:  0.864264738772
LogLoss:  0.145122767088
AUC: 1.0
Gini:  1.0

Confusion Matrix (Act/Pred) for max f1 @ threshold = 0.999943256378:

   2  4 Error Rate
--- --- --- ------ --------
2   105 0 0   (0.0/105.0)
4   0 35 0   (0.0/35.0)
Total 105 35 0   (0.0/140.0)

Maximum Metrics:

metric         threshold  value  idx
---------------- ------- ------- ---
max f1          0.999943   1    14
max f2          0.999943   1    14
max f0point5       0.999943   1    14
max accuracy       0.999943   1    14
max precision       1     1    0
max absolute_MCC      0.999943   1    14
max min_per_class_accuracy 0.999943   1    14
```

```
Variable Importances:

variable  relative_importance  scaled_importance  percentage
--------  --------------------  -----------------  ----------
C9        1                     1                  0.108707
C8        0.981525              0.981525           0.106698
C3        0.954005              0.954005           0.103707
C7        0.94194               0.94194            0.102395
C2        0.941813              0.941813           0.102381
C4        0.92944               0.92944            0.101036
C1        0.89463               0.89463            0.0972522
C10       0.889871              0.889871           0.0967349
C5        0.864056              0.864056           0.0939286
C6        0.801792              0.801792           0.0871601


>>> prediction = model.predict(test)
>>>
>>> prediction.head()
First 10 rows and first 3 columns:
predict    p2    p4
---------  ------  -------
   2 0.985993  0.0140068
   2 0.957854  0.042146
   2 0.957854  0.042146
   2 0.999733  0.000266882
   2 0.998995  0.00100492
   2 0.979549  0.0204512
   4 5.65749e-10 1
   2 0.992313  0.0076873
   2 0.974153  0.025847
   4 2.68044e-06 0.999997


>>> perf = model.model_performance(test)
>>> perf.show()

ModelMetricsBinomial:  deeplearning
** Reported on test data.  **

MSE: 0.0254503614803
R^2:  0.864264738772
LogLoss:  0.145122767088
AUC: 1.0
Gini:  1.0

Confusion Matrix (Act/Pred) for max f1 @ threshold = 0.999943256378:

  2  4  Error  Rate
--- --- --- ------- --------
2   105 0 0   (0.0/105.0)
4   0 35 0   (0.0/35.0)
Total 105 35 0   (0.0/140.0)

Maximum Metrics:

metric         threshold  value  idx
-------------  ---------  -----  ---
max f1         0.999943   1    14
max f2         0.999943   1    14
max f0point5       0.999943   1    14
max accuracy       0.999943   1    14
max precision      1      1    0
max absolute_MCC     0.999943   1    14
max min_per_class_accuracy 0.999943   1    14
```

```
>>> print 'Confusion Matrix:  '
Confusion Matrix:
>>> print (perf.confusion_matrix())

Confusion Matrix (Act/Pred) for max f1 @ threshold = 0.999943256378:

  2 4 Error Rate
--- -- -- ----- -------
2  105 0 0   (0.0/105.0)
4  0 35 0   (0.0/35.0)
Total 105 35 0   (0.0/140.0)

>>>
>>> print 'Precision:  '
Precision:
>>> print (perf.precision())
[[1.0, 1.0]]
>>>
>>> print 'Accuracy:  '
Accuracy:
>>> print (perf.accuracy())
[[0.9999432563781738, 1.0]]
>>>
```

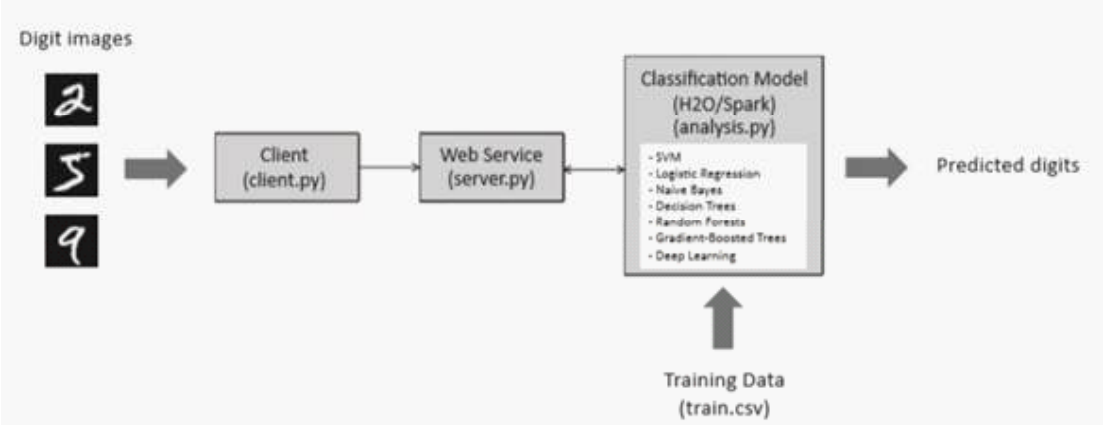
```
>>> print 'AUC: '
AUC:
>>> print (perf.auc())
1.0
```

## 11.4 Studi Kasus : Mengklasifikasikan Digit Tulisan Tangan

Dalam bagian ini, kami akan menjelaskan studi tentang sistem bangunan untuk mengklasifikasikan angka tulisan tangan. Untuk studi kasus ini, kami akan menggunakan dataset MNIST (Modifikasi National Institute of Standards and Technology) [46]. Database angka tulisan tangan MNIST adalah kumpulan 60.000 gambar angka tulisan tangan yang diambil sampelnya dari dokumen yang ditulis oleh karyawan Biro Sensus AS dan siswa sekolah menengah Amerika. Digit telah dinormalisasi ukurannya dan dipusatkan pada gambar skala abu-abu berukuran tetap dengan dimensi 28 x 28 piksel. Setiap gambar memiliki total 784 piksel (28 × 28). Setiap piksel memiliki nilai antara0 dan 255 (dengan angka yang lebih tinggi berarti piksel yang lebih gelap). File dataset yang digunakan dalam contoh di bagian ini dapat diunduh dari [47]. File CSV ini berisi 60.000 baris (satu kolom untuk setiap gambar) dan 785 kolom (kolom 1-784 menunjukkan nilai piksel dan kolom 785 memiliki label gambar - 0 hingga 9).

Gambar 11.55 menunjukkan komponen dari sistem klasifikasi digit. Kami akan menjelaskan dua alternatif implementasi dari sistem berdasarkan H2O dan Spark. Model klasifikasi dibangun menggunakan H2O atau Spark dengan data pelatihan. Komponen layanan

web membuat sistem klasifikasi tersedia sebagai layanan web. Klien Python digunakan untuk menguji sistem.




Gambar 11.55: *Pengenalan digit*

### 11.4.1 Klasifikasi Digit denga H20

Box 11.27 menunjukkan implementasi Python dari komponen analisis untuk klasifikasi digit menggunakan H2O. Dalam program ini, kami menerapkan kelas Mesin Analisis dengan metode untuk melatih model klasifikasi dan membuat prediksi. Jenis model klasifikasi (Naive Bayes, Random Forest, Deep Learning, GBM) ditentukan dalam konstruktor kelas.

**Box 11.27: Analysis component for digit classification using H2O - analysis.py**

```
import h2o

class AnalysisEngine:
 def make_prediction(self, data):
  testframe=h2o.H2OFrame(data)
  prediction = self.model.predict(testframe)
  result = str(prediction[0].as_data_frame())
  return result

 def train_model(self, trainingFile, responseColumn, modelType):
  train = h2o.import_frame(path=h2o.locate(trainingFile))
  train[responseColumn] = train[responseColumn].asfactor()

  if modelType=='deeplearning':
   self.model = h2o.deeplearning(x=train[0:responseColumn],
       y=train[responseColumn])
```

```
  elif modelType=='gbm':
   self.model = h2o.gbm(x=train[0:responseColumn],
        y=train[responseColumn])

  elif modelType=='naivebayes':
   self.model = h2o.naive_bayes(x=train[0:responseColumn],
        y=train[responseColumn])

  elif modelType=='randomforest':
   self.model = h2o.random_forest(x=train[0:responseColumn],
        y=train[responseColumn])

 def __init__(self):
  h2o.init()
  trainingFile = "/home/ubuntu/h2o/data/mnistdata/test.csv"
  responseColumn = 784
  modelType='deeplearning'

  self.train_model(trainingFile, responseColumn, modelType)
```

Box 11.28 menunjukkan implementasi Python dari komponen server untuk klasifikasi digit menggunakan H2O. Untuk komponen server, kami menggunakan kerangka web Flask Python. Ketika komponen server dijalankan, itu membuat sebuah instance dari kelas AnalysisEngine dari komponen analisis. Komponen server memperlihatkan titik akhir (/ prediksi). Ketika klien mengirim permintaan HTTP POST ke titik akhir ini dengan data gambar, fungsi make_prediction dari kelas AnalysisEngine dipanggil untuk mengklasifikasikan gambar.

**■ Box 11.28: Server component for digit classification using H2O - server.py**

```
import json
from flask import Flask, request

from analysish2o import AnalysisEngine

app = Flask(__name__)

@app.route('/predict', methods=['POST'])
def predict():
 data = request.json['data']
 result = analysis_engine.make_prediction(data)
 return json.dumps(result)


if __name__ == '__main__':
 global analysis_engine

 analysis_engine = AnalysisEngine()

 app.run(host='0.0.0.0', port=5000, debug=False)
```

Box 11.29 menunjukkan klien Python untuk menguji sistem klasifikasi digit. Klien ini membaca file gambar, mengubahnya menjadi daftar nilai piksel dan membuat permintaan HTTP POST ke server dengan data gambar.

**Box 11.29: Python client for digit classification - client.py**

```
import requests
import numpy as np
import Image
import json

imgFilename = '/home/ubuntu/5.png'
img = Image.open(imgFilename).convert('L')
imga = np.asarray(img.getdata())
imgl = imga.tolist()

payload={'data':imgl}

headers = {'content-type':  'application/json'}
r = requests.post("http://localhost:5000/predict",
  data=json.dumps(payload), headers=headers)

print r.text
```

Untuk menguji sistem, jalankan file server.py terlebih dahulu dan kemudian jalankan file client.py.

### 11.5.2 Klasifikasi Digit dengan Spark

Box 11.30 menunjukkan implementasi Python dari komponen analisis untuk klasifikasi digit menggunakan Spark. Dalam program ini, kami menerapkan kelas Mesin Analisis dengan metode untuk melatih model klasifikasi dan membuat prediksi. Jenis model klasifikasi (Naive Bayes, Decision Tree, Random Forest, GBM) ditentukan dalam konstruktor kelas.

**Box 11.30: Spark analysis component for digit classification - analysis.py**

```
from pyspark.mllib.linalg import Vectors
from pyspark.mllib.regression import LabeledPoint
from pyspark.mllib.classification import NaiveBayes
from pyspark.mllib.tree import RandomForest
from pyspark.mllib.tree import DecisionTree
from pyspark.mllib.tree import GradientBoostedTrees

class AnalysisEngine:
 def parseLine(self,line):
  parts = line.split(',')
  label = float(parts[784])
```

```
  features = Vectors.dense([float(x) for x in parts[0:784]])
  return LabeledPoint(label, features)

 def make_prediction(self, data):
  testData = Vectors.dense([float(x) for x in data[0:784]])
  result = self.model.predict(testData)
  return result

 def train_model(self,sc,modelType):
  data = sc.textFile('file:////home/hadoop/train.csv')
  header = data.first() #extract header
  data = data.filter(lambda x:x !=header) #filter out header
  trainingData = data.map(self.parseLine)

  if modelType=='naivebayes':
   self.model = NaiveBayes.train(trainingData, 1.0)

  elif modelType=='randomforest':
   self.model = RandomForest.trainClassifier(trainingData,
       numClasses=10, categoricalFeaturesInfo={},
       numTrees=10, featureSubsetStrategy="auto",
       impurity='gini', maxDepth=10, maxBins=32)

  elif modelType=='decisiontree':
   self.model = DecisionTree.trainClassifier(trainingData,
       numClasses=10, categoricalFeaturesInfo={},
       impurity='gini', maxDepth=10, maxBins=32)

  elif modelType=='gbm':
   self.model = GradientBoostedTrees.trainClassifier(trainingData,
    categoricalFeaturesInfo=, numIterations=3)

 def __init__(self, sc):
  modelType='naivebayes'
  self.train_model(sc, modelType)
```

Box 11.31 menunjukkan implementasi Python dari komponen server untuk klasifikasi digit menggunakan Spark. Untuk komponen server, kami menggunakan kerangka web Flask Python. Saat komponen server dijalankan, ia membuat Spark Context dan meneruskannya ke konstruktor kelas AnalysisEngine untuk membuat instance kelas. Komponen server memperlihatkan titik akhir (/ prediksi). Ketika klien mengirim permintaan HTTP POST ke titik akhir ini dengan data gambar, fungsi make_prediction dari kelas AnalysisEngine dipanggil untuk mengklasifikasikan gambar.

**Box 11.31: Server component for digit classification using Spark - server.py**

```
import analysis
import json
from flask import Flask, request
from pyspark import SparkContext, SparkConf

app = Flask(__name__)

@app.route('/mnist/predict', methods=['POST'])
def predict():
 data = request.json['data']
 result = analysis_engine.make_prediction(data)
 return json.dumps(result)


if __name__ == '__main__':
 global analysis_engine
 conf = SparkConf().setAppName("MySparkApp")
 sc = SparkContext(conf=conf, pyFiles=['analysis.py'])
 analysis_engine = analysis.AnalysisEngine(sc)

 app.run(host='0.0.0.0', port=5000, debug=False)
```

Untuk menguji sistem, jalankan file server.py terlebih dahulu (bin / spark-submit server.py) dan kemudian jalankan file client.py (python client.py).

## 11.5 Studi Kasus : Data Analisis Genomee (Implementasi)

Dalam Bab-1, kami menjelaskan studi kasus tentang analisis data genome. Dua jenis analisis berikut dijelaskan: (1) memprediksi respon obat berdasarkan ekspresi gen, (2) menemukan korelasi antara nilai ekspresi dari semua pasang gen untuk menemukan gen yang memiliki pola ekspresi serupa dan gen yang memiliki pola ekspresi berlawanan.
Pada bagian ini, kami akan menjelaskan implementasi dari kedua jenis analisis tersebut. Untuk analisis pertama, kami akan menggunakan Spark untuk membangun model regresi untuk memprediksi respons obat. Variabel target untuk model regresi adalah respon obat pasien, dan variabel independen adalah nilai ekspresi gen. Namun, sebelum kita dapat membangun model regresi, kita harus melakukan beberapa transformasi dan penggabungan agar data sesuai untuk membangun model.
Box 11.32 menunjukkan implementasi program untuk membangun model regresi untuk memprediksi respons obat. Dalam program ini, pertama, kita membaca empat file dataset (ditunjukkan pada Gambar 1.6) dan mengubahnya menjadi Spark DataFrames, sehingga kita dapat menerapkan operasi SparkSQL untuk memfilter, mentransformasikan dan menggabungkan dataset. Dengan kerangka data yang dibuat, kami memilih gen dengan

serangkaian fungsi tertentu dan menggabungkan meta-data gen dengan meta-data pasien dan data larik mikro. Selanjutnya, kami melakukan pivot hasil untuk mendapatkan nilai ekspresi setiap jenis gen untuk setiap pasien (ini adalah tabel 'g2' di kode). Kemudian kami memilih ID pasien, penyakit dan respon obat dari meta-data pasien (ini adalah tabel 'g3' di kode). Selanjutnya, kita menggabungkan tabel 'g2' dan 'g3' untuk mendapatkan tabel 'g4' yang memiliki semua data dalam format yang tepat untuk membuat model regresi. Kami menggunakan modul LinearRegressionWithSGD Spark MLib untuk membuat model regresi linier. Kami menjelaskan langkah-langkah ini dengan sampel kecil data di Bab-1 (seperti yang ditunjukkan pada Gambar 1.8).

**■ Box 11.32: Spark implementation for predicting drug response using regression model**

```
from pyspark import SparkContext
from pyspark.sql import SQLContext, Row
from pyspark.mllib.regression import LabeledPoint
from pyspark.mllib.regression import LinearRegressionWithSGD
from pyspark.mllib.regression import LinearRegressionModel

sc = SparkContext(appName="App")

sqlContext = SQLContext(sc)

genes = sc.textFile('/home/ubuntu/GeneMetaData-10-10.txt')
header = genes.first() #extract header
genes = genes.filter(lambda x:x !=header)

gparts = genes.map(lambda l:  l.split(", "))
geneframe = gparts.map(lambda p:  Row(geneid=int(p[0]),
        target=int(p[1]), position = long(p[2]),
        length=int(p[3]), function=int(p[4])))

schemaGene = sqlContext.createDataFrame(geneframe)
schemaGene.registerTempTable("genes")


patients = sc.textFile('/home/ubuntu/PatientMetaData-10-10.txt')
header = patients.first() #extract header
patients = patients.filter(lambda x:x !=header)

pparts = patients.map(lambda l:  l.split(", "))
patientsframe = pparts.map(lambda p:  Row(patientid=int(p[0]),
          age=int(p[1]), gender=int(p[2]),
          zipcode=int(p[3]), disease=int(p[4]),
          drugResponse = float(p[5])))

schemaPatients = sqlContext.createDataFrame(patientsframe)
schemaPatients.registerTempTable("patients")

geo = sc.textFile('/home/ubuntu/GEO-10-10.txt')
header = geo.first() #extract header
geo = geo.filter(lambda x:x !=header)
```

```
geoparts = geo.map(lambda l:  l.split(", "))
geoframe = geoparts.map(lambda p:  Row(geneid=int(p[0]),
     patientid=int(p[1]), exValue = float(p[2])))

schemaGEO = sqlContext.createDataFrame(geoframe)
schemaGEO.registerTempTable("geo")

g = sqlContext.sql("SELECT p.patientid, p.disease,
       e.geneid, e.exValue, p.drugResponse FROM
       genes AS g, patients AS p, geo AS e
       WHERE g.function < 300 AND
       g.geneid = e.geneid
       AND p.patientid = e.patientid")

g.registerTempTable("responses")

g2=g.groupBy('patientid').pivot('geneid').sum('exValue')

g2.registerTempTable("gen")

g3 = sqlContext.sql("SELECT patientid, disease,
drugResponse FROM patients")

g3.registerTempTable("gen3")

g4 = sqlContext.sql("SELECT * FROM gen3, gen WHERE
gen3.patientid=gen.patientid")


def parsePoint(x):
  return LabeledPoint(x[2], x[4:])

parsedData = g4.map(parsePoint)

# Build the model
model = LinearRegressionWithSGD.train(parsedData)

# Evaluate the model on training data
valuesAndPreds = parsedData.map(lambda p:
       (p.label, model.predict(p.features)))

MSE = valuesAndPreds.map(lambda
    (v, p):  (v - p)**2).reduce(lambda x, y:  x + y) / valuesAndPreds.count()

print("Mean Squared Error = " + str(MSE))
```

Untuk jenis analisis kedua, kita akan menggunakan Spark untuk menghitung korelasi antara nilai ekspresi dari semua pasangan gen. Box 11.33 menunjukkan implementasi program untuk contoh ini. Setelah memuat file dataset dan mengubah dataset menjadi Spark DataFrames, kami memilih pasien dengan penyakit tertentu dan menggabungkan hasilnya dengan tabel microarray. Selanjutnya, kita pivot tabel pada langkah sebelumnya untuk mendapatkan nilai ekspresi semua gen untuk setiap pasien. Kami menggunakan

tabel ini untuk membuat matriks korelasi yang memiliki korelasi antara nilai ekspresi dari semua pasangan gen. Kami menjelaskan langkah-langkah ini dengan sampel kecil data di Bab-1 (seperti yang ditunjukkan pada Gambar 1.9).

**■ Box 11.33: Spark implementation for computing correlation between the expression levels of all pairs of genes**

```
from pyspark import SparkContext
from pyspark.sql import SQLContext, Row
from pyspark.mllib.stat import Statistics
from pyspark.mllib.linalg import Vectors
#sc = SparkContext(appName="App")

sqlContext = SQLContext(sc)


genes = sc.textFile('/home/ubuntu/GeneMetaData-10-10.txt')
header = genes.first() #extract header
genes = genes.filter(lambda x:x !=header)

gparts = genes.map(lambda l:  l.split(", "))
geneframe = gparts.map(lambda p:  Row(geneid=int(p[0]),
         target=int(p[1]), position = long(p[2]),
         length=int(p[3]), function=int(p[4])))

schemaGene = sqlContext.createDataFrame(geneframe)
schemaGene.registerTempTable("genes")


patients = sc.textFile('/home/ubuntu/PatientMetaData-10-10.txt')
header = patients.first() #extract header
patients = patients.filter(lambda x:x !=header)

pparts = patients.map(lambda l:  l.split(", "))
patientsframe = pparts.map(lambda p:  Row(patientid=int(p[0]),
      age=int(p[1]), gender=int(p[2]), zipcode=int(p[3]),
      disease=int(p[4]), drugResponse = float(p[5])))

schemaPatients = sqlContext.createDataFrame(patientsframe)
schemaPatients.registerTempTable("patients")

geo = sc.textFile('/home/ubuntu/GEO-10-10.txt')
header = geo.first() #extract header
geo = geo.filter(lambda x:x !=header)

geoparts = geo.map(lambda l:  l.split(", "))
geoframe = geoparts.map(lambda p:  Row(geneid=int(p[0]),
     patientid=int(p[1]), exValue = float(p[2])))

schemaGEO = sqlContext.createDataFrame(geoframe)
schemaGEO.registerTempTable("geo")
```

```
g = sqlContext.sql("SELECT p.patientid, p.disease,
      e.geneid, e.exValue FROM patients AS p,
      geo AS e WHERE p.disease =18
      AND p.patientid = e.patientid")

gl=g.groupBy('patientid').pivot('geneid').sum('exValue')
```

```
def parseFunc(x):
  return Vectors.dense(x[1:])

parsedData = gl.map(parseFunc)

pearsonCorr = Statistics.corr(parsedData)

print(str(pearsonCorr).replace('nan', 'NaN'))
```

## 11.6 Sistem Rekomendasi

- Filter berbasis konten: Dalam pendekatan filter berbasis konten, rekomendasi diberikan kepada pengguna (untuk item seperti buku, film, lagu, atau restoran) berdasarkan fitur atau karakteristik item. Ide dasar di balik pendekatan ini adalah jika pengguna menyukai suatu barang, dia mungkin juga menyukai barang serupa lainnya. Dengan kata lain, pendekatan ini menemukan semua item yang mirip dengan item yang disukai pengguna dan merekomendasikan item tersebut kepada pengguna. Pendekatan ini tidak memerlukan nilai pengguna atau preferensi pengguna implisit. Meskipun pendekatan ini berfungsi untuk merekomendasikan item yang mirip dengan item yang disukai pengguna, pendekatan ini tidak merekomendasikan sesuatu yang baru yang mungkin disukai pengguna. Untuk menemukan item yang serupa, digunakan ukuran kesamaan (seperti cosine similarity) atau metode lingkungan (seperti metode clustering). Pendekatan ini mensyaratkan item memiliki fitur bermakna tertentu yang dapat digunakan untuk menghitung kesamaan. Namun, jika tidak memungkinkan untuk mengekstrak fitur yang berarti dari item, pendekatan filter kolaboratif digunakan.
- Pemfilteran kolaboratif: Pemfilteran kolaboratif memungkinkan untuk merekomendasikan item (atau memfilter item dari kumpulan item) berdasarkan preferensi pengguna dan preferensi kolektif pengguna lain (yaitu memanfaatkan informasi kolaboratif yang tersedia pada peringkat item pengguna). Pemfilteran kolaboratif memanfaatkan peringkat yang diberikan oleh pengguna ke berbagai item untuk merekomendasikan item kepada pengguna yang belum mereka peringkat. Masukan ke sistem rekomendasi apa pun yang menggunakan pemfilteran kolaboratif adalah data tentang peringkat

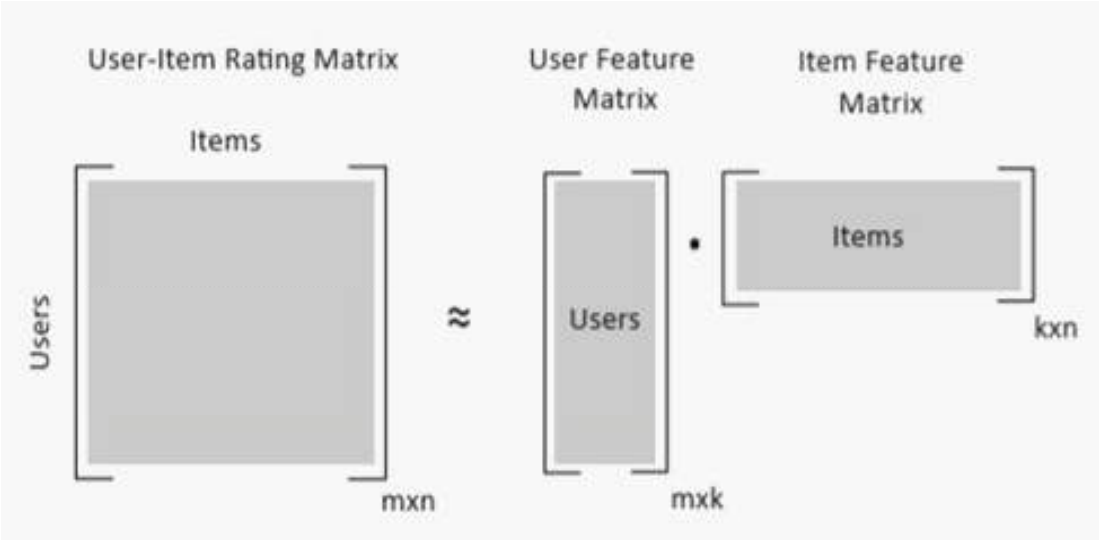
pengguna untuk item yang berbeda. Pendekatan pemfilteran kolaboratif terdiri dari dua jenis:

- Pendekatan berbasis memori: Ada dua jenis pendekatan berbasis memori: pemfilteran kolaboratif berbasis pengguna dan pemfilteran kolaboratif berbasis item. Pemfilteran kolaboratif berbasis pengguna menemukan pengguna yang mirip dengan pengguna tertentu dan merekomendasikan item yang mereka sukai. Pemfilteran kolaboratif berbasis item menemukan item yang serupa dengan item yang sebelumnya disukai pengguna. Kesamaan antara pengguna (dalam pemfilteran kolaboratif berbasis pengguna) atau item (dalam pemfilteran kolaboratif berbasis item) dihitung menggunakan peringkat pengguna dari item tersebut.
- Pendekatan berbasis model: Dalam pendekatan pemfilteran kolaboratif berbasis model, model peringkat pengguna dibuat terlebih dahulu dan kemudian model tersebut digunakan untuk membuat prediksi. Metode ini mengadopsi pendekatan propiglistik dan memprediksi peringkat pengguna untuk item yang belum diberi peringkat oleh pengguna.

Manfaat menggunakan pemfilteran kolaboratif atas pemfilteran berbasis konten adalah dapat menemukan pola tersembunyi dan merekomendasikan sesuatu yang baru.

### 11.6.1 Alternating Least Squares (ALS)

Pada bagian ini, kami akan menjelaskan pendekatan pemfilteran kolaboratif berbasis model berdasarkan algoritma Alternating Least Squares (ALS).




Gambar 11.56: *Memfaktorkan matriks peringkat item-pengguna ke dalam vektor fitur pengguna dan fitur item*

Mari kita rumuskan masalah pemfilteran kolaboratif. Membiarkan
m = jumlah pengguna
n = jumlah item
k = jumlah faktor laten (atau jumlah fitur pengguna / item)
r (u, i) = rating yang diberikan oleh pengguna u untuk item i
w (i, j) = 1 jika pengguna i telah menilai item j dan 0 sebaliknya
x (u) = vektor fitur untuk pengguna u
y (i) = vektor ciri untuk item i

Gambar 11.56 menunjukkan matriks nilai pengguna di mana setiap kolom termasuk kolom penyebab dan kolom adalah peringkat yang diberikan untuk item. Dengan adanya matriks penilaian item pengguna, tujuan pembelajarannya adalah untuk mempelajari fitur laten pengguna dan item (yang mewakili preferensi pengguna dan fitur item). Dengan kata lain, diberikan matriks item-pengguna berdimensi m × n, kita ingin memfaktorkan matriks tersebut menjadi matriks m × k (vektor fitur pengguna) dan matriks k × n (vektor fitur item).

Mempelajari fitur pengguna (x (1), x (2), ..., x (m)) untuk semua pengguna dan fitur item (y (1), y (2), ..., y (n)) untuk semua item , kita dapat mendefinisikan fungsi biaya untuk x (u) dan y (i) sebagai berikut:

$$J(x^{(1)},x^{(2)},\ldots,x^{(m)}) = min_{x^{(u)}} \frac{1}{2}\sum_{u=1}^{m}\sum_{i:w(i,j)=1}(x^{(u)T}y^{(i)} - r^{(u,i)})^2 + \frac{\lambda}{2}\sum_{u=1}^{m}\sum_{l=1}^{k}(x_l^{(u)})^2$$

$$J(y^{(1)},y^{(2)},\ldots,y^{(n)}) = min_{y^{(i)}} \frac{1}{2}\sum_{i=1}^{n}\sum_{u:w(i,j)=1}(x^{(u)T}y^{(i)} - r^{(u,i)})^2 + \frac{\lambda}{2}\sum_{i=1}^{n}\sum_{l=1}^{k}(y_l^{(i)})^2$$

dimana λ adalah parameter regularisasi yang ditambahkan untuk mencegah overfitting data. Fungsi biaya untuk x (u) dan y (i) dapat digabungkan sebagai berikut:

$$J(x^{(1)},\ldots,x^{(m)},y^{(1)},\ldots,y^{(n)}) = min_{(x^{(u)},y^{(i)})} \frac{1}{2}\sum_{(u,i):w(i,j)=1}(x^{(u)T}y^{(i)} - r^{(u,i)})^2 + \frac{\lambda}{2}\left(\sum_{u=1}^{m}\sum_{l=1}^{k}(x_l^{(u)})^2 + \sum_{i=1}^{n}\sum_{l=1}^{k}(y_l^{(i)})^2\right)$$

1. Untuk mengatasi masalah optimasi ini, algoritma Alternating Least Squares (ALS) dapat digunakan.
2. Algoritma ALS diringkas sebagai berikut:
3. Inisialisasi x (u) dan y (i) (vektor fitur pengguna dan item) ke nilai acak.
4. Perbaiki vektor item (y (i)) dan selesaikan untuk vektor pengguna yang optimal (x (u)) dengan meminimalkan fungsi biaya J (x (u), y (i)).
5. Perbaiki vektor pengguna (x (u)) dan selesaikan untuk vektor item optimal (y (i)) dengan meminimalkan fungsi biaya J (x (u), y (i)).
6. Ulangi sampai konvergensi.

Mari kita lihat contoh sistem untuk membuat rekomendasi film menggunakan pendekatan filter kolaboratif. Untuk contoh ini, kami akan menggunakan kumpulan data MovieLens [48] yang menyertakan peringkat yang diberikan oleh pengguna untuk film. Untuk tujuan pengembangan, versi yang lebih kecil dari kumpulan data (MovieLens 100K) yang mencakup 100.000 peringkat dari 943 pengguna pada 1682 film, digunakan. Untuk menguji kode yang berfungsi dengan kumpulan big data, Anda dapat menggunakan kumpulan data MovieLens 20M yang mencakup 20 juta peringkat yang diterapkan ke 27.000 film oleh 138.000 pengguna.

Box 11.34 menunjukkan implementasi Python dari sistem rekomendasi yang menggunakan implementasi dari algoritma Alternating Least Squares (ALS) Spark MLlib. Dalam contoh ini, pertama-tama kami memuat kumpulan data MovieLens dan membaginya menjadi kumpulan data pelatihan dan pengujian. File dataset dipisahkan tab dengan kolom berikut: id pengguna | id item | peringkat | cap waktu

Peringkat diuraikan menjadi objek Peringkat Spark yang mewakili tupel (pengguna, produk, peringkat). Fungsi kereta Sasis ALS digunakan untuk model ALS. Fungsi latihan mengambil parameter input seperti data pelatihan, peringkat (jumlah faktor laten dalam model), jumlah iterasi dan lambda (parameter regularisasi). Model ALS kemudian digunakan untuk memprediksi peringkat untuk pengguna dan produk tertentu (menggunakan fungsi prediksi (pengguna, produk)). Kelas ALS Spark juga menyediakan fungsi lain seperti predictAll (user_product) yang mengembalikan daftar peringkat yang diprediksi untuk input pengguna dan pasangan produk, fungsi recommendProducts (pengguna, num) untuk menampilkan produk jumlah teratas untuk pengguna tertentu, recommendUsers (product, num ) untuk menampilkan jumlah pengguna teratas untuk produk tertentu. Untuk melihat fitur pengguna dan produk, fungsi userFeatures () dan productFeatures () dapat digunakan.

**■ Box 11.34: Python program for building a recommendation system based on ALS**

```
from pyspark.mllib.recommendation import ALS, Rating

# Load and parse the data
data = sc.textFile("file:///home/hadoop/ml-100k/u.data")

(trainingRatings, testRatings) = data.randomSplit([0.7, 0.3])

trainingRatings.first()
#Output:  u'196,242,3,881250949'

testRatings.first()
#Output:  u'244,51,2,880606923'

trainingData = trainingRatings.map(lambda l:
   l.split(',')).map(lambda l:
   Rating(int(l[0]), int(l[1]), float(l[2])))

trainingData.first()
#Output:  Rating(user=196, product=242, rating=3.0)

testData = testRatings.map(lambda l:
   l.split(',')).map(lambda l:
   (int(l[0]), int(l[1])))

testData.first()
#Output:  (244, 51)

# Build the recommendation model using Alternating Least Squares
rank = 10
numIterations = 50
model = ALS.train(trainingData, rank, numIterations)

#Predict rating for the given user and product.
model.predict(253, 465)
#Output:  4.5738394508197189

#Return a list of predicted ratings for input user and product pairs
predictions = model.predictAll(testData)
predictions.first()
#Rating(user=58, product=1084, rating=1.0564932954594659)

predictions = predictions.map(lambda l:  ((l[0], l[1]), l[2]))
predictions.take(5)
#Output:  [((58, 1084), 1.0564932954594659),
#((316, 1084), 5.7316694387562022),
#((330, 1084), 3.5890840644277131),
#((195, 1084), 4.4394359038624369),
#((541, 1084), 5.9725270274011484)]

testRatings = testRatings.map(lambda l:
   l.split(',')).map(lambda l:  ((int(l[0]), int(l[1])), float(l[2])))
```

```
testRatings.take(5)
#[((244, 51), 2.0), ((115, 265), 2.0),
#((6, 86), 3.0),
#((200, 222), 5.0),
#((234, 1184), 2.0)]

ratingsAndPredictions = testRatings.join(predictions)
ratingsAndPredictions.take(5)
#[((105, 333), (3.0, 2.4070588034521849)),
#((109, 365), (4.0, 2.974229204549999)),
#((360, 14), (5.0, 4.5625799637027171)),
#((720, 286), (5.0, 3.9581844170381464)),
#((501, 829), (3.0, 2.4709893659891664))]

MSE = ratingsAndPredictions.map(lambda r:  (r[1][0] - r[1][1])**2).mean()
print "Mean Squared Error = " + str(MSE)
#Output:  Mean Squared Error = 1.29853954685

#Recommend the top N products for a given user
model.recommendProducts(253, 5)
#Output:  [Rating(user=253, product=1063, rating=6.7525296445231611),
#Rating(user=253, product=459, rating=6.7239131863565023),
#Rating(user=253, product=844, rating=6.643750789521941),
#Rating(user=253, product=960, rating=6.175726236721804),
#Rating(user=253, product=394, rating=6.1395628318225901)]

#Recommend the top N users for a given product
model.recommendUsers(465, 5)
#Output:  [Rating(user=519, product=465, rating=7.5049478754749002),
#Rating(user=180, product=465, rating=7.3478113160070091),
#Rating(user=217, product=465, rating=7.2194201952177766),
#Rating(user=808, product=465, rating=6.5398839496324266),
#Rating(user=93, product=465, rating=6.4988971770196038)]


#View features corresponding to a user
model.userFeatures().take(1)[0]
#Output:  (2, array('d', [-0.041698437184095383, -0.29158979654312134,
#0.607492327690124 51, 0.6784324049949646, -0.12671113014221191,
#0.76399964094161987, -0.52530914545059204, 0.25506862998008728,
#0.54997712373733521, -1.36256337165832 52]))


#View features corresponding to a product
model.productFeatures().take(1)[0]
#Output:  (2, array('d', [-1.2220950126647949, 0.26224410533905029,
#0.31355467438697815, 0.7695726752281189,
#0.072056755423545837, 1.233315110206604,
#0.5064246654510498, -0.024322325363755226,
#-0.10120454430580139, -0.98879802227020264]))
```

### 11.6.2 ingular Value Decomposition (SVD)

Pada bagian ini, kami akan menjelaskan pendekatan pemfilteran kolaboratif berdasarkan algoritma Singular Value Decomposition (SVD).
SVD adalah metode pemfaktoran matriks yang dapat digunakan untuk memfaktorkan matriks X dari dimensi (n × d) menjadi matriks U dimensi (n × n), S dimensi (d × d) dan V dimensi (n × d) sebagai berikut:

$$Xn\times d = Un\times nSd\times dVT\ n\times d$$

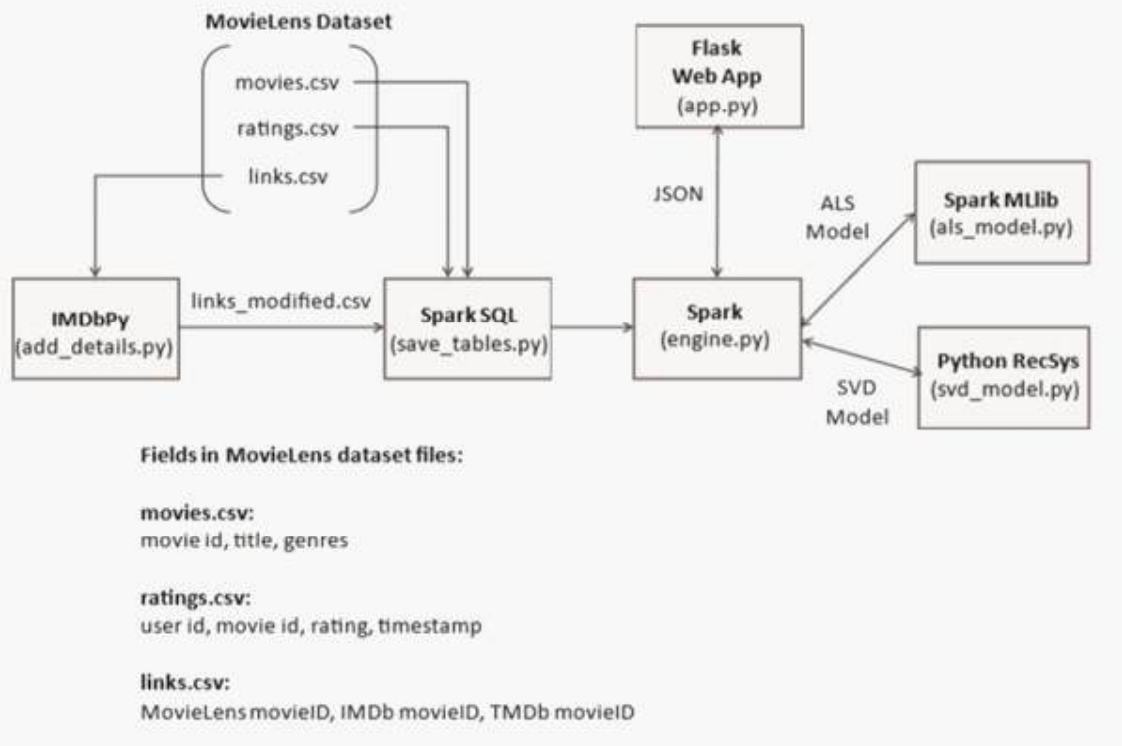
Untuk sistem rekomendasi yang mempertahankan peringkat item pengguna, matriks item-pengguna biasanya sangat jarang, karena matriks mungkin memiliki jumlah pengguna dan item yang sangat besar tetapi pengguna tertentu mungkin hanya memberi peringkat sejumlah kecil item. Matriks dalam kasus ini akan terdiri dari sebagian besar bidang yang disetel ke 0 dengan hanya beberapa bidang yang berisi nilai aktual, yang merupakan peringkat. Bekerja dengan matriks renggang seperti itu melibatkan kompleksitas ruang yang tinggi untuk menyimpan seluruh matriks dan meningkatkan kompleksitas waktu untuk mengurai matriks dan menghitung hubungan untuk item dan pengguna. SVD dapat digunakan untuk reduksi dimensi matriks untuk mengurangi matriks renggang untuk mengekstrak hubungan laten antara pengguna dan item. Ini membantu dalam mendapatkan hubungan antara pengguna dan item dengan cara yang lebih efisien dan membantu meningkatkan keakuratan sistem rekomendasi.

### 11.6.3 Studi Kasus: Sistem Rekomendasi Film

Pada bagian ini kami menjelaskan studi kasus sistem rekomendasi film yang menggunakan algoritma rekomendasi berbasis ALS dan SVD. Untuk mengimplementasikan ALS kami telah menggunakan Spark MLLib dan untuk SVD kami telah menggunakan pustaka Python-RecSys [28].

Gambar 11.57 menunjukkan arsitektur sistem rekomendasi. Dataset yang digunakan untuk sistem rekomendasi adalah dataset MovieLens [48]. Selain dataset MovieLens, informasi tambahan ditambahkan ke dataset dengan IMDbPY, API untuk mengakses database IMDb [49]. Dari kumpulan data MovieLens kami menggunakan file berikut: film.csv, rating.csv, dan links.csv. The movies.csv memiliki format: id film, judul, genre. Rating.csv memiliki format: id pengguna, id film, peringkat, stempel waktu. Link.csv formatnya: MovieLens movieID, IMDb movieID, TMDb movieID.

Karena ID film MovieLens memiliki hubungan satu-ke-satu dengan ID film IMDb, IMDbPy digunakan untuk mengambil objek film yang terkait dengan ID film MovieLens. Objek film berisi daftar sutradara dan daftar pemeran. Untuk sistem rekomendasi film, sutradara pertama dalam daftar dan sembilan anggota pemeran teratas ditambahkan ke file links. csv. Diberikan ID film, file link yang dimodifikasi (links_modi fi ed.csv) dapat digunakan untuk mendapatkan lebih banyak informasi tentang film untuk presentasi front end. Box 11.35 menunjukkan kode Python untuk menambahkan detail ke file tautan menggunakan IMDbPy. File dataset MovieLens dan file link yang dimodifikasi diubah menjadi SparkSQL DataFrames dan disimpan sebagai tabel yang digunakan dalam mesin rekomendasi pada saat dijalankan. Box 11.36 menunjukkan program Python untuk menyimpan file dataset sebagai tabel SparkSQL.




Gambar 11.57: *Arsitektur untuk sistem rekomendasi film*

**■ Box 11.35: Python program for adding details to links file using IMDbPy - add_details.py**

```
import imdb
import csv
import codecs

#Add director and cast info to the links.csv file
```

```
#Fetch from IMDB server
ia = imdb.IMDb(accessSystem='http')

file_name = 'datasets/links.csv'

old = open(file_name, 'rb')
new = codecs.open('links_modified.csv', 'wb', 'utf-8')
reader = csv.reader(old, delimiter=',')
next(reader)

new.write('movieId,imdbId,tmdId,director,cast\n')

for row in reader:
 id = row[1]
 m = ia.get_movie(id)

 director=''
 cast_list=[]
 cast = []

 if m.get('director'):
  director = m.get('director')[0].get('name')

 if m.get('cast'):
  cast_list = m.get('cast')
  l = len(cast_list)
  if l >= 10:
   cast_list = cast_list[0:9]
  else:
   cast_list = cast_list[0:l]

 cast = [c['name'] for c in cast_list]
 cast_elements = '|'.join(cast)
 line = [row[0], id, row[2], director, cast_elements]
 new.write(','.join(line))
 new.write('\n')
 print id

old.close()
new.close()
```

**■ Box 11.36: Python program for saving the dataset files as SparkSQL tables - save_tables.py**

```
import os
import re
from pyspark import SparkContext
from pyspark.sql import SQLContext, Row

# Regex used to seperate movie movieId, name, year, and genres
RE = re.compile(r'(?P<movieId>\d+),
"?(?P<name>.+)\((?P<year>\d+))
?"?,(?P<genres>.+)')
```

```
# Initialize the Spark context
sc = SparkContext("local", "DataImporter")
# Initialize the SparkSQL context
sqlContext = SQLContext(sc)

#----Import Movies File----------

# Read in the text file as an RDD
data = sc.textFile('movies.csv')

header = data.first() # Get the csv header
# Filter out the csv header
#data = data.filter(lambda line:  line != header)
# Split the CSV file into rows
```

```
# Formatter that takes the CSV line and
# outputs it as a list of datapoints
# Uses a regex with named groups
def formatter(line):
 m = RE.match(line) # Seperates datapoints
 if (m != None):
  m = m.groupdict()
  movieId = int(m['movieId'])
  name = m['name']
  year = int(m['year'])
  genres = m['genres'].split('|')
  return [movieId, name, year, genres]

data = data.map(formatter)
# Filter out rows that dont match
data = data.filter(lambda line:  line != None)
# Map the data into a Row data object to prepare it for insertion
rows = data.map(lambda r:  Row(movieId=r[0], name=r[1],
  year=r[2], genres=r[3]))

# Create the schema for movies and register a table for it
schemaMovies = sqlContext.createDataFrame(rows)
schemaMovies.registerTempTable("movies")
schemaMovies.save('tables/movies')

#----Import Ratings File----------

# Regex used to seperate movie movieId, name, year, and genres
RE = re.compile(r'(?P<userId>\d+),(?P<movieId>\d+),
(?P<rating>\d\d),(?P<timestamp>\d+)')

# Read in the text file as an RDD
data = sc.textFile('ratings.csv')

header = data.first() # Get the csv header
# Filter out the csv header
#data = data.filter(lambda line:  line != header)
# Split the CSV file into rows
def formatter(line):
 m = RE.match(line) # Seperates datapoints
 if (m != None):
  m = m.groupdict()
```

```
    userId = int(m['userId'])
    movieId = int(m['movieId'])
    rating = float(m['rating'])
    timestamp = m['timestamp']
    return [userId, movieId, rating, timestamp]

data = data.map(formatter)
# Filter out rows that dont match
data = data.filter(lambda line:  line != None)
# Map the data into a Row data object to prepare it for insertion
rows = data.map(lambda r:  Row(userId=r[0], movieId=r[1],
  rating=r[2], timestamp=r[3]))
```

```
# Create the schema for movies and register a table for it
schemaRatings = sqlContext.createDataFrame(rows)
schemaRatings.registerTempTable("ratings")
schemaRatings.save('tables/ratings')

#----Import Details File----------

# Regex used to seperate movie movieId, imdbId, and tmdbId
RE = re.compile(r'(?P<movieId>\d+),(?P<imdbId>\d+),
(?P<tmdbId>\d+),(?P<director>.+),(?P<cast>.+)')

# Read in the text file as an RDD
data = sc.textFile('links_modified.csv')

header = data.first() # Get the csv header
# Filter out the csv header
#data = data.filter(lambda line:  line != header)
# Split the CSV file into rows
def formatter(line):
 m = RE.match(line) # Seperates datapoints
 if (m != None):
  m = m.groupdict()
  movieId = int(m['movieId'])
  imdbId = int(m['imdbId'])
  if m['tmdbId'] != None:
   tmdbId = int(m['tmdbId'])
  else:
   tmdbId = -1
  director = m['director']
  cast = m['cast'].split('|')
  print [movieId, imdbId, tmdbId, director, cast]
  return [movieId, imdbId, tmdbId, director, cast]

data = data.map(formatter)
# Filter out rows that dont match
data = data.filter(lambda line:  line != None)
# Map the data into a Row data object to prepare it for insertion
rows = data.map(lambda r:  Row(movieId=r[0], imdbId=r[1],
 tmdbId=r[2], director=r[3], cast=r[4]))

# Create the schema for movies and register a table for it
schemaLinks = sqlContext.createDataFrame(rows)
schemaLinks.registerTempTable("detail")
schemaLinks.save('tables/detail')
```

### *Building ALS Model*

Kami menggunakan Spark MLlib untuk mengimplementasikan algoritma ALS dan untuk membangun model ALS. Box 11.37 menunjukkan kode Python untuk melatih dan menyimpan model ALS. Model disimpan sebagai file dan kemudian digunakan dalam mesin rekomendasi. Pendekatan ini efisien karena model hanya perlu dihitung sekali. Dengan demikian, waktu latency dari algoritma rekomendasi berkurang karena pendekatan rekomendasi berbasis model ini.

**■ Box 11.37: Python program for training and saving an ALS model - als_model.py**

```
from pyspark import SparkContext
from pyspark.mllib.recommendation import ALS
from pyspark.mllib.recommendation import Rating

sc = SparkContext("local", "collaborative_filtering") #initializing sc

#Loading the data using SparkContext
ratings = "./ratings.csv"
data = sc.textFile(ratings)
ratings_data = data.map(lambda l:  l.split(','))
ratings = ratings_data.map(lambda l:  Rating(int(l[0]),
 int(l[1]), float(l[2])))

#Building the recommendation model using Alternating Least Squares
rank = 10
numIterations = 5
model = ALS.train(ratings, rank, numIterations)

#Lets save the model for future use
model_path = "./ALS_Model"
model.save(sc,model_path)
```

### *Building SVD Model*

Untuk membangun model SVD, kami menggunakan pustaka Python-RecSys yang menyediakan implementasi algoritma SVD. Box 11.38 menunjukkan kode Python untuk melatih dan menyimpan model SVD. Model disimpan sebagai file dan kemudian digunakan dalam mesin rekomendasi.

**■ Box 11.38: Python program for training and saving an SVD model - svd_model.py**

```
import recsys.algorithm
from recsys.algorithm.factorize import SVD

#SVD Model Computation
```

```
#To obtain make the script verbose.
recsys.algorithm.VERBOSE = True

#Load the ratings file
svd = SVD()
svd.load_data(filename='ratings.csv',
    sep=',' , format={'col':0, 'row':1, 'value':2, 'ids':int})

#Now, lets compute the SVD. k = 100
svd.compute(k=k, min_values=10, pre_normalize=None,
  mean_center=True, post_normalize=True,
  savefile='movielens_model')

print("Model Computed and Created")
```

***Rekomendasi Mesin***

Mesin rekomendasi menerima masukan dari aplikasi web (ID pengguna dan nama film), dan memberikan rekomendasinya menggunakan model ALS dan SVD yang telah dilatih sebelumnya. Box 11.39 menunjukkan kode Python untuk mesin rekomendasi.

Model ALS memberikan rekomendasi dengan mencari pengguna yang mirip dengan pengguna tertentu dan kemudian memprediksi peringkat untuk film berperingkat teratas dari pengguna serupa. Film-film yang direkomendasikan pada dasarnya adalah daftar film-film yang memiliki prediksi rating tinggi di antara film-film pilihan dari para pembuat film tersebut. Untuk studi kasus, daftar film yang direkomendasikan dibatasi sebanyak lima.

Model SVD memberikan dua jenis rekomendasi:

- Merekomendasikan film untuk pengguna tertentu dalam database: Ini menghasilkan daftar film yang berperingkat teratas oleh pengguna lain yang serupa dengan pengguna target. Ini adalah rekomendasi film yang dipersonalisasi dan mengembalikan daftar film yang lebih sesuai dengan preferensi pengguna.
- Merekomendasikan film berdasarkan film tertentu: Rekomendasi film yang tidak dipersonalisasi ini menggunakan film sebagai input dan memprediksi peringkat film lain dan mengembalikan daftar film tersebut.

**■ Box 11.39: Recommendation engine program - engine.py**

```
import recsys.algorithm
from recsys.algorithm.factorize import SVD
from operator import add
from pyspark import SparkContext
from pyspark.sql import SQLContext, Row
from pyspark import SparkConf
from pyspark.mllib.recommendation import ALS
from pyspark.mllib.recommendation import MatrixFactorizationModel
from pyspark.mllib.recommendation import Rating
```

```
recsys.algorithm.VERBOSE = True

'''
This is the main file which initializes the backend and gets the machine
learning algorithms running on spark.  Use this file for running.
'''

def get_counts_and_averages(ID_and_ratings_tuple):
 nratings = len(ID_and_ratings_tuple[1])
 return ID_and_ratings_tuple[0],
 (nratings, float(sum(x for x in ID_and_ratings_tuple[1])) / nratings)

class RecommendationSystem():
 def __init__(self, sc, datapath='frontend/', model='movielens_model'):
  self.sc = sc
  self.start = True
  self.sqlContext = SQLContext(self.sc)

  self.svd = SVD(filename=datapath+model)

  self.als_model_path = datapath + 'ALS_Model'
  self.als_model = MatrixFactorizationModel.load(sc, self.als_model_path)
  self.movie_df = self.sqlContext.read.load(datapath+'tables/movies')
  self.detail_df = self.sqlContext.read.load(datapath+'tables/detail')
  self.rating_df = self.sqlContext.read.load(datapath+'tables/ratings')


 # call this function to get all recommendations
 def get_all_recomm(self, userid, moviename):
  movieid = self.get_movie_id(moviename)

  # all recommendation algorithms return a list of movie ids
  recom1 = self.svd_recomm(userid)
  recom2 = self.svd_similar(movieid)
  recom3 = self.als_new(userid)

  #get info about the movie based on movie ids
  brief_info1 = self.get_brief_list(recom1)
  brief_info2 = self.get_brief_list(recom2)
  brief_info3 = self.get_brief_list(recom3)

  return [brief_info1, brief_info2, brief_info3]

 # get movie id based on movie name input
 def get_movie_id(self, moviename):
  r = self.movie_df.where(self.movie_df['name'].startswith(moviename)).first()

  # return movie id 1 if not found
  if r is None:
   return 1

  return r['movieId']

 # svd recommendation algorithm based on the user's rating history
 def svd_recomm(self, userid, only_unknown):
  # output format:  (movieid, similarity value)
  similar_list = self.svd.recommend(userid, n=10,
```

```
      only_unknowns=True, is_row=True)

  movieid_list = self.get_id_list(similar_list)
  return movieid_list

 # svd recommendation algorithm based on similar movie
 def svd_similar(self, movieid):
  similar_list = self.svd.similar(movieid)
  movieid_list = self.get_id_list(similar_list)
  return movieid_list


 # an ALS recommendation algorithm based on user rating history
 def als_new(self, userid):
  recommended_movies = self.als_model.recommendProducts(userid, 10)
  recommended_movie_list = []
  for movie in recommended_movies:
   recommended_movie_list.append(movie[1])

  return recommended_movie_list

 # return a list of movie id
 def get_id_list(self, l):
  movieid_list = []
  for s in l:
   movieid_list.append(s[0])
  return movieid_list

 # get a list of movie info given a list of movie ids
 def get_brief_list(self, movieList):
  info_list = []
  for m in movieList:
   info = self.get_brief(m)
   if info['title'] != 'unknown':
    info_list.append(info)
   if len(info_list) == 5:
    break

  return info_list

 # get movie info (title, direction, genres, rating, cast)
 def get_brief(self, movieid):
  info = {}
  info['movieid'] = movieid
  info['title'] = 'unknown'
  info['genres'] = 'unknown'
  info['rating'] = 0
  info['director'] = 'unknown'
  info['cast'] = 'unknown'

  m = self.movie_df.where(self.movie_df['movieId'] == movieid).first()
  if m is not None:
   info['title'] = m['name']
   info['genres'] = m['genres']
   if len(info['genres']) > 3:
    info['genres'] = info['genres'][0:3]
```

```
d = self.detail_df.where(self.detail_df['movieId'] == movieid).first()
if d is not None:
 info['director'] = d['director']
 info['cast'] = d['cast']

r = self.rating_df.where(self.rating_df['movieId'] == movieid)

# default rating to be 4.6
if r.count()==0:
 info['rating'] = 4.6
else:
 avg = r.map(lambda row:row['rating']).reduce(lambda x, y:  x+y)/r.count()
 info['rating'] = avg

return info
```

***Aplikasi Web***

Box 11.40 menunjukkan kode Python untuk aplikasi web Flask. Karena keterbatasan ruang, kami belum menyertakan file HTML, JavaScript dan CSS untuk aplikasi web. Gambar 11.58 menunjukkan screenshot dari aplikasi web. Di interface web, cari nama film di bilah pencarian. Rekomendasi berdasarkan tiga algoritme berbeda tercantum dalam tiga kolom terpisah. Rekomendasi film di kolom kiri merupakan output dari algoritma SVD berdasarkan user-ID. Hasil rekomendasi SVD berdasarkan nama film yang diinput tertera di kolom tengah. Kolom kanan adalah hasil dari filter kolaboratif ALS. Informasi singkat dari setiap film yang direkomendasikan, seperti judul, peringkat, dua genre teratas, sutradara, dan sembilan pemeran film teratas, juga ditampilkan.

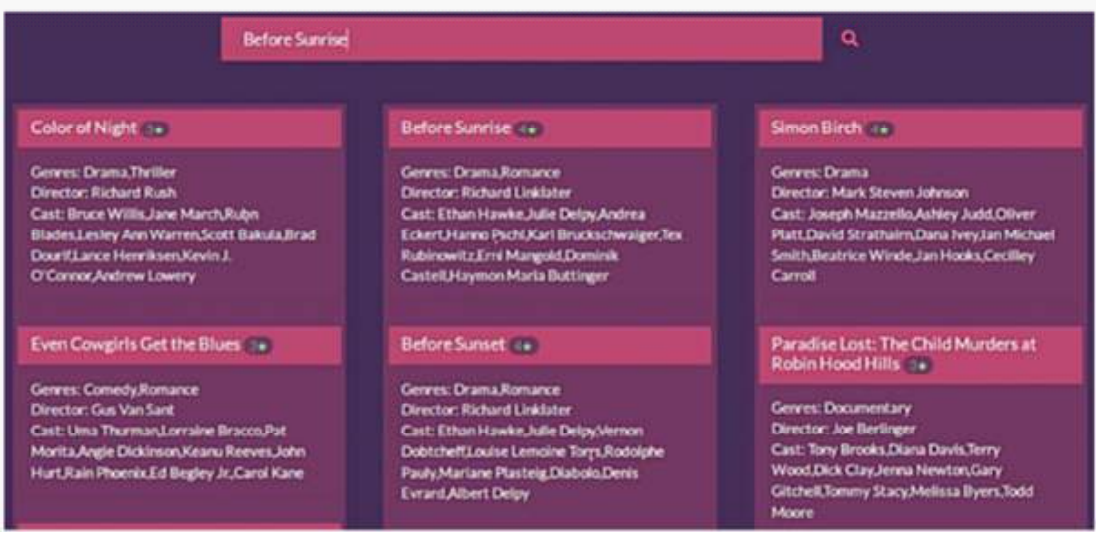
Aplikasi web Flask digunakan untuk menyajikan file HTML, CSS dan JavaScript statis dan bertindak sebagai server, serta menghubungkan mesin rekomendasi backend ke front end. Kode JavaScript menangani interaksi pengguna dan memulai pemrosesan backend dengan mengirimkan permintaan HTTP POST ke aplikasi Flask yang selanjutnya memanggil fungsi untuk mendapatkan rekomendasi menggunakan model ALS dan SVD. JavaScript menunggu respons JSON untuk permintaan POST-nya, lalu mengurai respons dan memasukkan informasi ke dalam tempelatee HTML.

**■ Box 11.40: Python Flask web application - app.py**

```
from flask import Flask
from flask import request, render_template, jsonify, url_for
import json
from engine import RecommendationSystem
from pyspark import SparkContext, SparkConf
```

```
import imdb
import csv
import codecs


#Add director and cast info to the links.csv file

#Fetch from IMDB server
ia = imdb.IMDb(accessSystem='http')

file_name = 'datasets/links.csv'

old = open(file_name, 'rb')
new = codecs.open('links_modified.csv', 'wb', 'utf-8')
reader = csv.reader(old, delimiter=',')
next(reader)

new.write('movieId,imdbId,tmdId,director,cast\n')

for row in reader:
 id = row[1]
 m = ia.get_movie(id)

 director=''
 cast_list=[]
 cast = []

 if m.get('director'):
  director = m.get('director')[0].get('name')

 if m.get('cast'):
  cast_list = m.get('cast')
  l = len(cast_list)
  if l >= 10:
   cast_list = cast_list[0:9]
  else:
   cast_list = cast_list[0:l]

 cast = [c['name'] for c in cast_list]
 cast_elements = '|'.join(cast)
 line = [row[0], id, row[2], director, cast_elements]
 new.write(','.join(line))
 new.write('\n')
 print id

old.close()
new.close()
app = Flask(__name__)

conf = SparkConf().setAppName("movie_recommendation_server")
sc = SparkContext(conf=conf, pyFiles=['frontend/engine.py'])

global data
global userid
```

```
@app.route("/")
def index():
 global data
 global userid
 data = {"data":  "Empty"}
 userid = 1
 return render_template('index.html')

# change user id through url
@app.route("/<int:user_id>")
def index_id(user_id):
 global data
 global userid
 data = {"data":  "Empty"}
 userid = user_id
 return render_template('index.html')

# post movie recommendation results
@app.route("/data", methods=['POST'])
def post_data():
 global data
 global userid
 d = request.get_data()
 data = json.loads(d)

 # calling backend to get all movie recommendations
 info = recomsys.get_all_recomm(userid, data['data'])
 return jsonify({'data':  info})

if __name__ == "__main__":
 global data
 global recomsys

 # initialize backend engine
 recomsys = RecommendationSystem(sc)

 data = {"data":  "Empty"}
 app.run()
```




Gambar 11.58: *Screenshot dari aplikasi web rekomendasi film*

## Ringkasan

Dalam bab ini kami memberikan gambaran umum tentang algoritma analisis big data untuk pembelajaran mesin yang mencakup pengelompokan, klasifikasi, regresi dan rekomendasi. Implementasi dan contoh penerapan algoritme ini menggunakan kerangka kerja pembelajaran mesin Spark MLlib dan H2O. Clustering adalah proses pengelompokan item data yang serupa menjadi satu sehingga item data yang lebih mirip satu sama lain daripada item data lainnya ditempatkan dalam satu cluster. Thek-meansclusteringalgor ithmgroupsdataitemsintokclusters, like thatallpoints in a cluster is near to their centroid dibandingkan dengan centroids of tetangga cluster. Kami menjelaskan berbagai pengukuran jarak yang dapat digunakan untuk algoritma pengelompokan termasuk pengukuran jarak Euclidean, Cosine, dan Manhattan. Selanjutnya, kami menjelaskan klasifikasi dan algoritma regresi. Klasifikasi adalah proses mengkategorikan objek ke dalam kategori yang telah ditentukan sebelumnya. Sedangkan dalam klasifikasi variabel respon bersifat kategorikal dan tidak berurutan, sedangkan pada Regresi variabel respon mengambil nilai kontinyu. Naive Bayes adalah algoritma klasifikasi propiglistik yang didasarkan pada teorema Bayes dengan asumsi naif tentang independensi atribut fitur. Generalized Linear Model (GLM) adalah generalisasi dari model regresi linier biasa yang memungkinkan variabel respon yang diskrit, tidak terdistribusi normal dan / atau varians tidak konstan. Pada Decision Trees, model prediktif berupa pohon yang dapat digunakan untuk memprediksi nilai suatu variabel target berdasarkan beberapa variabel atribut. Random Forest melatih sejumlah pohon keputusan dan kemudian mengambil suara mayoritas dengan menggunakan mode

kelas yang diprediksi oleh pohon individu. Di Random Forest, setiap pohon dibangun secara independen dari sampel acak (bootstrap), sedangkan di Pohon yang Didorong Gradien, pohon keputusan dilatih pada setiap langkah yang mengoreksi dan melengkapi pohon yang dibangun sebelumnya. Dalam SVM, hyperplane margin maksimum ditentukan, yang memisahkan kedua kelas. Selanjutnya, kami mendeskripsikan implementasi khusus dari pembelajaran mendalam, yang didasarkan pada jaringan saraf tiruan buatan multi-layer feed-forward yang mencakup banyak layer neuron yang saling berhubungan. Selanjutnya, kami memberikan perbandingan algoritme rekomendasi. Sementara dalam pendekatan pemfilteran berbasis konten, rekomendasi diberikan kepada pengguna berdasarkan fitur atau karakteristik item, pemfilteran kolaboratif menggunakan peringkat yang diberikan oleh pengguna ke berbagai item untuk merekomendasikan item kepada pengguna yang belum mereka nilai.

# Visualisasi Data

Bab 12

Bab ini mencakup :

- Lightning
- Pygal
- Seaborn

Dalam bab ini, kami akan menjelaskan kerangka kerja dan pustaka Python untuk visualisasi data. Visualisasi dapat membantu dalam memahami data dan hasil analisis dengan cepat dan mudah. Ketika jumlah data sangat besar, visualisasi menjadi penting karena membantu kita memahami pola dalam hasil atau data yang mungkin tidak terlihat.

## 12.1 Frameworks & Libraries

### 12.1.1 Lightning

Lightning adalah kerangka kerja untuk membuat visualisasi interaktif berbasis web [67]. Lightning menyediakan REST API dan pustaka klien untuk bahasa pemrograman Python, Scala, R dan JavaScript. Lightning dapat diterapkan baik dalam mode server atau dalam mode tanpa server lokal. Server Lightning dapat diinstal dan dijalankan menggunakan perintah berikut:

```
sudo apt-get install nodejs npm
sudo npm install -g lightning-server
lightning-server
```

Lightning juga dapat dijalankan tanpa server menggunakan klien Python. Klien Lightning Python dapat diinstal dengan perintah berikut:

```
sudo pip install lightning-python
```

Untuk membuat visualisasi, Anda dapat menggunakan Lightning REST API atau salah satu pustaka klien. Contoh dalam bab ini menggunakan pustaka klien Lightning Python untuk membuat visualisasi.

### 12.1.2 Pygal

Karena kami telah menggunakan Python sebagai bahasa pemrograman utama untuk contoh dalam buku ini, pustaka diagram Python mungkin berguna dalam memvisualisasikan data dan hasil analisis. Pustaka Python Pygal adalah pustaka diagram yang mudah digunakan yang mendukung diagram dari berbagai jenis. Diagram yang dibangun dengan Pygal dapat dirender dalam format output seperti SVG, PNG atau di browser. Library Pygal dapat diinstal dengan perintah berikut:

```
sudo apt install python-scipy python-pandas
sudo pip install seaborn
```

### 12.1.3 Seaborn

Seaborn adalah pustaka visualisasi Python untuk merencanakan plot statistik yang menarik [36]. Seaborn dibangun di atas matplotlib dan menggunakan struktur data dari pustaka numpy dan pandas Python serta rutinitas statistik dari scipy dan statsmodels. Seaborn dapat diinstal dengan perintah berikut:

## 12.2. Contoh Visualisasi

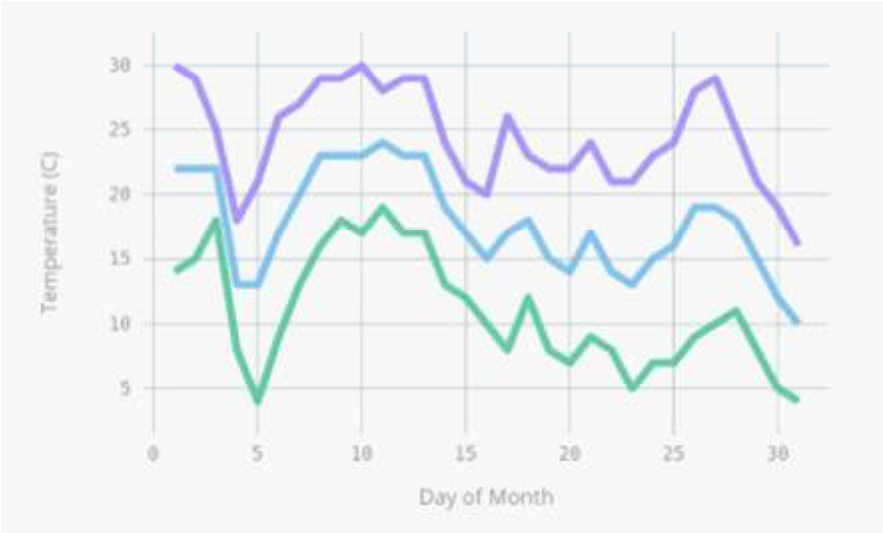
### 12.2.1 Diagram Garis

Diagram Garis adalah salah satu diagram paling sederhana yang dapat digunakan untuk menampilkan informasi sebagai rangkaian titik data yang dihubungkan oleh sebuah garis. Mari kita lihat contoh grafik garis plot untuk suhu maksimum, minimum dan rata-rata yang tercatat pada bulan Oktober 2014 di Atlanta. Data tersebut diperoleh dari Weather Underground [31]. Box 12.1 menunjukkan kode Python yang membuat diagram garis menggunakan Lightning dan hasilnya ditunjukkan pada Gambar 12.1.

**Box 12.1: Python program for plotting line chart using Lightning**

```
from lightning import Lightning

from numpy import random

lgn = Lightning(ipython=True,local=True)

x = range(1,32)
```

```
y1 = [30,29,25,18,21,26,27,29,29,30,28,29,
29,24,21,20,26,23,22,22,24,21,21,23,24,
28,29,25,21,19,16]

y2 = [22,22,22,13,13,17,20,23,23,23,24,23,
23,19,17,15,17,18,15,14,17,14,13,15,16,
19,19,18,15,12,10]

y3 = [14,15,18,8,4,9,13,16,18,17,19,17,
17,13,12,10,8,12,8,7,9,8,5,7,7,9,10,
11,8,5,4]

lgn.line([y1,y2,y3],thickness=6,index=x,
   xaxis='Day of Month',yaxis='Temperature (C)')
```



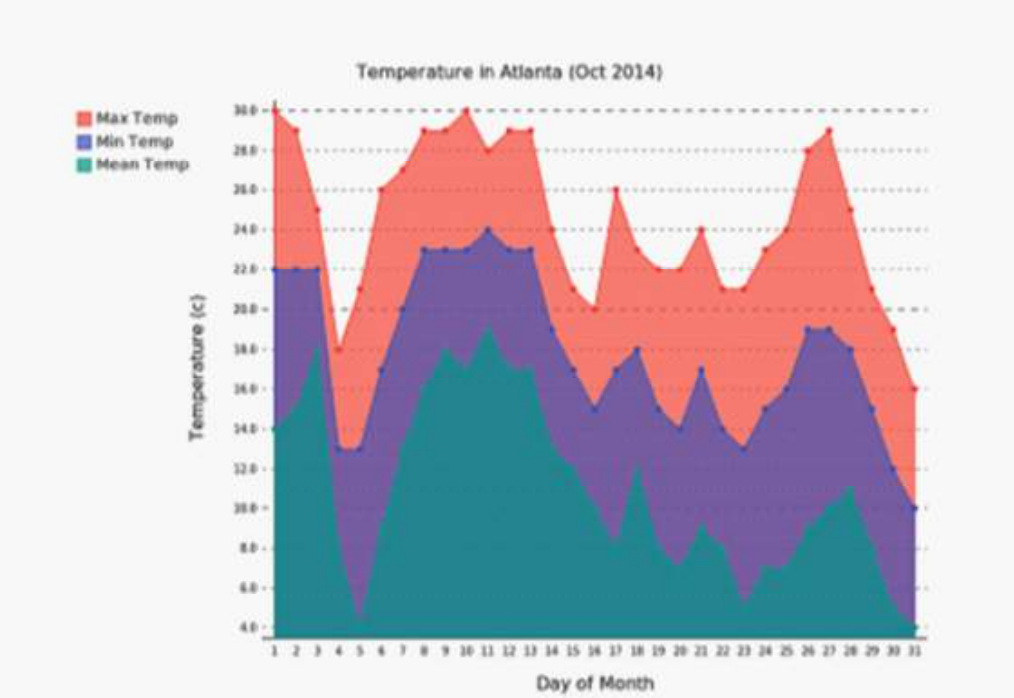

Gambar 12.1: *Diagram garis diplot dengan Lightning*

Mari kita ulangi contoh menggunakan pustaka charting Pygal. Box 12.2 menunjukkan kode Python yang membuat diagram garis menggunakan Pygal dan hasilnya ditunjukkan pada Gambar 12.2.

**■ Box 12.2: Python program for plotting line chart using Pygal**

```
import pygal

line_chart = pygal.Line(fill=True)
line_chart.x_title = 'Day of Month'
line_chart.y_title = 'Temperature (c)'
line_chart.title = 'Temperature in Atlanta (Oct 2014)'

line_chart.x_labels = ['1','2','3','4','5','6','7', '8','9','10','11',
'12','13', '14','15', '16','17','18','19','20','21','22',
'23', '24','25','26','27', '28','29','30','31']
```

```
line_chart.add('Max Temp', [30, 29, 25, 18, 21, 26, 27, 29, 29, 30, 28,
29, 29, 24, 21, 20, 26, 23, 22, 22, 24, 21, 21,
23, 24, 28, 29, 25, 21, 19, 16])

line_chart.add('Min Temp', [22, 22, 22, 13, 13, 17, 20, 23, 23, 23, 24,
23, 23, 19, 17, 15, 17, 18, 15, 14, 17, 14, 13,
15, 16, 19, 19, 18, 15, 12, 10, ])

line_chart.add('Mean Temp',    [14, 15, 18, 8, 4, 9, 13, 16,
18, 17, 19, 17, 17, 13, 12, 10, 8, 12, 8, 7, 9, 8, 5, 7, 7, 9, 10,
11, 8, 5, 4, ])

line_chart.render_to_png('line.png')
```




Gambar 12.2: *Diagram garis diplot dengan Pygal*

### 12.2.2 Scatter Plot (Plot sebar)

Plot sebar dapat digunakan untuk memvisualisasikan dua variabel di sepanjang sumbu X dan Y. Plot sebar dan berguna untuk mengidentifikasi hubungan antara dua dataset, misalnya memasang garis regresi untuk data bivariat.

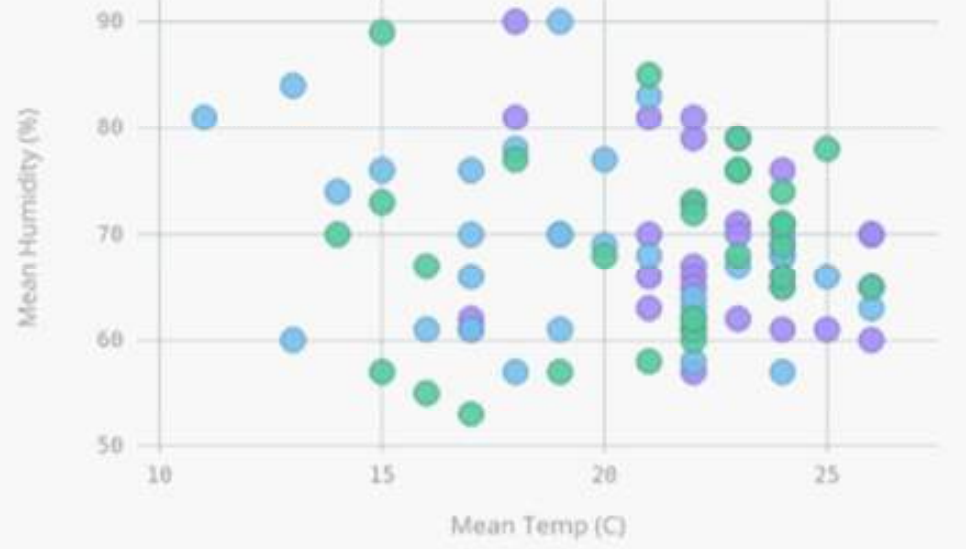
Mari kita lihat contoh plot sebar untuk memvisualisasikan suhu rata-rata dan kelembaban rata-rata yang tercatat di Atlanta pada Mei 2012, Mei 2013, dan Mei 2014. Box 12.3

menunjukkan kode Python yang membuat plot sebar menggunakan Lightning dan hasilnya ditunjukkan pada Gambar 12.3. Grup (ditampilkan dalam warna berbeda) menunjukkan data dari tahun yang berbeda untuk bulan Mei.

**Box 12.3: Python program for plotting scatter plot using Lightning**

```
from lightning import Lightning

from numpy import random

lgn = Lightning(ipython=True,local=True)

#Mean Temp
x=[25,23,23,24,24,24,22,23,18,17,17,
21,18,22,21,22,21,22,21,22,23,22,22,22,24,
26,26,26,26,26,24,19,18,17,13,14,11,15,17,
20,19,20,16,13,17,22,22,21,21,19,23,24,24,
25,18,17,19,22,24,23,24,26,16,17,16,19,22,
22,21,22,22,21,23,24,24,22,15,15,14,15,18,
20,22,23,26,24,24,23,23,24,24,25,24 ]

#Mean Humidity
y=[61,62,71,76,69,66,79,79,81,62,61,66,90,
81,81,67,70,66,63,57,70,73,65,61,65,65,
60,65,70,70,61,70,78,76,84,74,81,76,70,
69,70,77,61,60,66,58,63,68,83,90,76,70,
68,66,57,61,61,64,57,67,68,63,67,53,55,
57,60,61,58,62,73,85,76,71,65,72,73,57,
70,89,77,68,62,68,65,66,69,76,79,74,71,78,71]

g=[1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,2,2,2,2,2,2,2,2,2,
2,2,2,2,2,2,2,2,2,2,2,2,2,
2,2,2,2,2,2,2,3,3,3,3,3,3,3,
3,3,3,3,3,3,3,3,3,3,3,3,3,3,
3,3,3,3,3,3,3,3,3,3]

lgn.scatter(x,y,group=g,
   xaxis='Mean Temp (C)',yaxis='Mean Humidity (%)')
```

Petir mendukung jenis plot pencar khusus di mana ukuran titik dapat dibuat proporsional dengan variabel ketiga. Box 12.4 menunjukkan kode Python membuat plot pencar di mana ukuran titik sebanding dengan kecepatan angin rata-rata. Outputnya ditunjukkan pada Gambar 12.4. Titik warna yang berbeda menunjukkan data dari tahun yang berbeda.

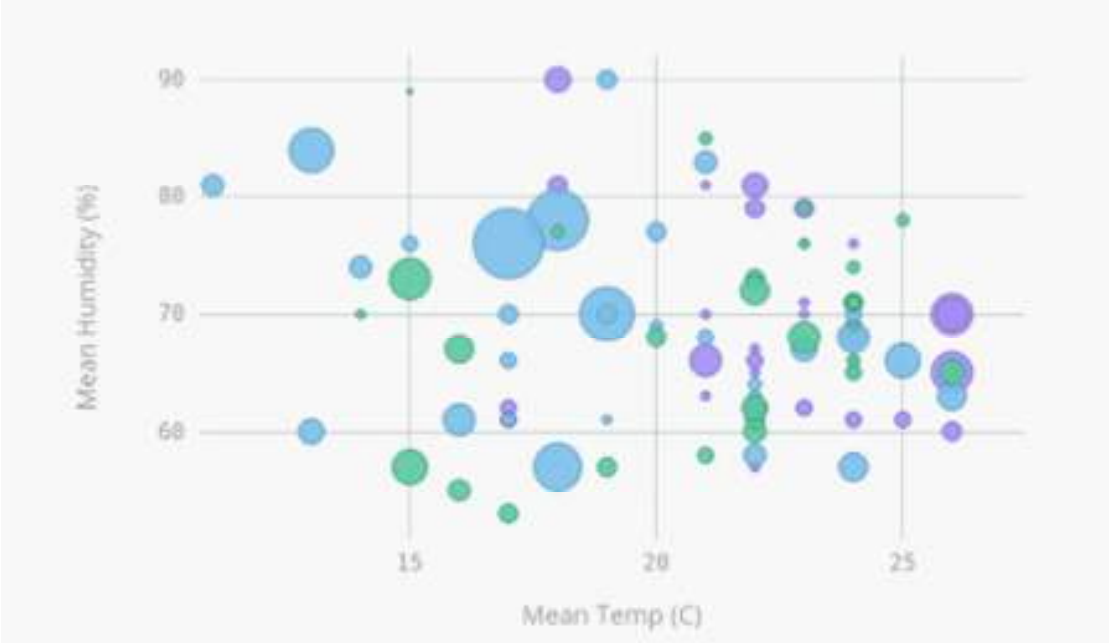



Gambar 12.3: *Plot sebar diplot dengan Lightning*

**Box 12.4: Python program for plotting scatter plot using Lightning**

```
from lightning import Lightning

from numpy import random

lgn = Lightning(ipython=True,local=True)

#Mean Temp
x=[25,23,23,24,24,24,22,23,18,17,17,
21,18,22,21,22,21,22,21,22,23,22,22,22,24,
26,26,26,26,26,24,19,18,17,13,14,11,15,17,
20,19,20,16,13,17,22,22,21,21,19,23,24,24,
25,18,17,19,22,24,23,24,26,16,17,16,19,22,
22,21,22,22,21,23,24,24,22,15,15,14,15,18,
20,22,23,26,24,24,23,23,24,24,25,24 ]

#Mean Humidity
y=[61,62,71,76,69,66,79,79,81,62,61,66,90,
81,81,67,70,66,63,57,70,73,65,61,65,65,
60,65,70,70,61,70,78,76,84,74,81,76,70,
69,70,77,61,60,66,58,63,68,83,90,76,70,
68,66,57,61,61,64,57,67,68,63,67,53,55,
57,60,61,58,62,73,85,76,71,65,72,73,57,
70,89,77,68,62,68,65,66,69,76,79,74,71,78,71]

g=[1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,2,2,2,2,2,2,2,2,2,
2,2,2,2,2,2,2,2,2,2,2,2,2,
2,2,2,2,2,2,2,3,3,3,3,3,3,
3,3,3,3,3,3,3,3,3,3,3,3,3,
3,3,3,3,3,3,3,3,3]
```

```
#Mean Wind Speed
s=[5,5,3,3,5,3,6,6,6,5,5,10,8,8,
3,3,3,5,3,3,3,5,3,2,2,2,6,13,13,
10,5,17,19,22,14,7,7,5,6,4,6,6,10,
8,5,7,4,5,7,6,3,5,7,11,15,4,3,4,
9,8,10,9,9,6,7,6,7,6,5,7,6,4,3,
6,5,9,13,11,3,2,4,6,8,10,8,4,3,
3,4,4,5,4,4]

lgn.scatter(x,y,group=g,size=s,
   xaxis='Mean Temp (C)',yaxis='Mean Humidity (%)')
```

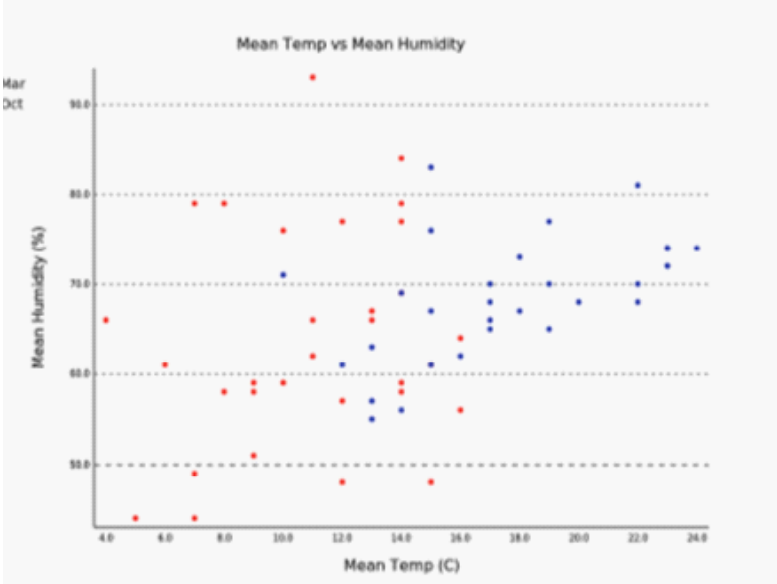



Gambar 12.4: *Plot sebar diplot dengan Lightning*

Mari kita lihat contoh lain dari plot pencar yang dibangun menggunakan pustaka Pygal. Box 12.5 menunjukkan kode Python yang membuat plot sebar suhu rata-rata dan kelembaban rata-rata untuk bulan Maret dan Oktober tahun 2014 di Atlanta. Outputnya ditunjukkan pada Gambar 12.5.

**Box 12.5: Python program for plotting scatter plot using Pygal**

```
import pygal

xy_chart = pygal.XY(stroke=False)
xy_chart.title = 'Mean Temp vs Mean Humidity'
xy_chart.x_title = 'Mean Temp (C)'
xy_chart.y_title = 'Mean Humidity (%)'

xy_chart.add('Mar', [(11, 62), (13, 66), (8, 79), (4, 66),
(9, 58), (6, 61), (10, 76), (11, 66), (14, 59), (14, 58), (16, 64),
(14, 69), (7, 44), (9, 59), (16, 56), (14, 84), (11, 93), (7, 79),
(12, 77), (12, 57), (10, 59), (15, 61), (13, 67), (8, 58), (7, 49),
(5, 44), (9, 51), (14, 77), (14, 79), (12, 48), (15, 48) ])

xy_chart.add('Oct', [(22, 68), (22, 70), (22, 81), (13, 55), (13, 57),
(17, 66), (20, 68), (23, 74), (23, 74), (23, 72), (24, 74), (23, 74),
(23, 72), (19, 77), (17, 68), (15, 76), (17, 65), (18, 67), (15, 67),
(14, 69), (17, 70), (14, 56), (13, 63), (15, 61), (16, 62), (19, 65),
(19, 70), (18, 73), (15, 83), (12, 61), (10, 71)])

xy_chart.render_to_png('scatter.png')
```



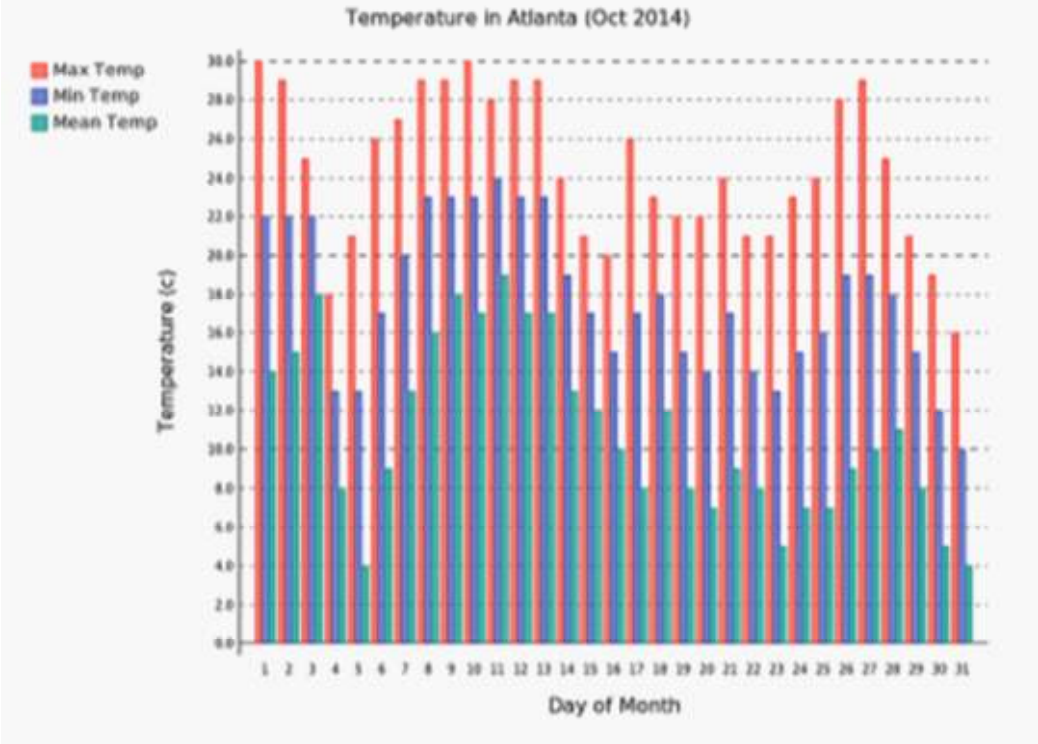

Gambar 12.5: *Plot sebar diplot dengan Pygal*

### 12.2.3 Diagram Batang

Diagram batang dapat digunakan untuk menampilkan data yang dikelompokkan sebagai batang dengan panjang yang sebanding dengan nilai yang diwakili. Box 12.6 menunjukkan kode Python untuk membuat diagram batang dari suhu maksimum, minimum dan rata-rata yang tercatat pada bulan Oktober 2014 di Atlanta. Outputnya ditunjukkan pada Gambar 12.6.

**Box 12.6: Python program for plotting bar chart using Pygal**

```
import pygal

bar_chart = pygal.Bar()
bar_chart.x_title = 'Day of Month'
bar_chart.y_title = 'Temperature (C)'
bar_chart.title = 'Temperature in Atlanta (Oct 2014)'

bar_chart.x_labels = ['1','2','3','4','5','6','7', '8','9','10','11',
'12','13', '14','15', '16','17','18','19','20','21','22',
'23', '24','25','26','27', '28','29','30','31']

bar_chart.add('Max Temp', [30, 29, 25, 18, 21, 26, 27, 29, 29, 30, 28,
29, 29, 24, 21, 20, 26, 23, 22, 22, 24, 21, 21,
23, 24, 28, 29, 25, 21, 19, 16])

bar_chart.add('Min Temp', [22, 22, 22, 13, 13, 17, 20, 23, 23, 23, 24,
23, 23, 19, 17, 15, 17, 18, 15, 14, 17, 14, 13,
15, 16, 19, 19, 18, 15, 12, 10, ])

bar_chart.add('Mean Temp',    [14, 15, 18, 8, 4, 9, 13, 16,
18, 17, 19, 17, 17, 13, 12, 10, 8, 12, 8, 7, 9, 8, 5, 7, 7, 9, 10,
11, 8, 5, 4, ])

bar_chart.render_to_png('bar.png')
```



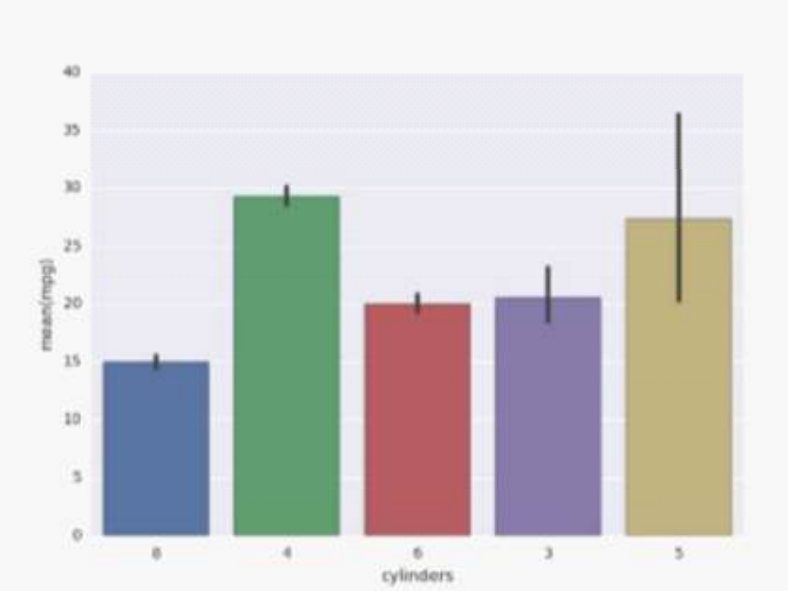

Gambar 12.6: *Diagran Bar yang diplot dengan Pygal*

Mari kita lihat contoh lain dari plot batang yang diplot menggunakan Seaborn. Plot batang Seaborn menunjukkan perkiraan tendensi sentral (mean atau median) untuk variabel numerik (sebagai tinggi batang persegi panjang) dan ketidakpastian di sekitar estimasi tersebut (menggunakan batang error). Untuk membuat plot box kita akan menggunakan Kumpulan Data MPG Otomatis dari repositori pembelajaran Mesin UCI [32]. Dataset ini dapat digunakan untuk masalah regresi untuk memprediksi konsumsi bahan bakar siklus kota (inmpg), antara atribut diskrit multi nilai (silinder, tahun model, asal) dan empat atribut kontinu (perpindahan, tenaga kuda, berat, percepatan). Box 12.7 menunjukkan program Python untuk membuat plot batang menggunakan Seaborn dan Gambar 12.7 menunjukkan plot batang. Plot ini menunjukkan nilai rata-rata konsumsi bahan bakar (dalam mpg) untuk jumlah silinder yang berbeda di mobil.

**Box 12.7: Python program for plotting bar plot using Seaborn**

```
import pandas as pd
import seaborn as sns
from pylab import savefig
```

```
data = pd.read_csv("auto-mpg.csv")

sns.barplot(x="cylinders", y="mpg", data=data)

savefig("bar.png")
```



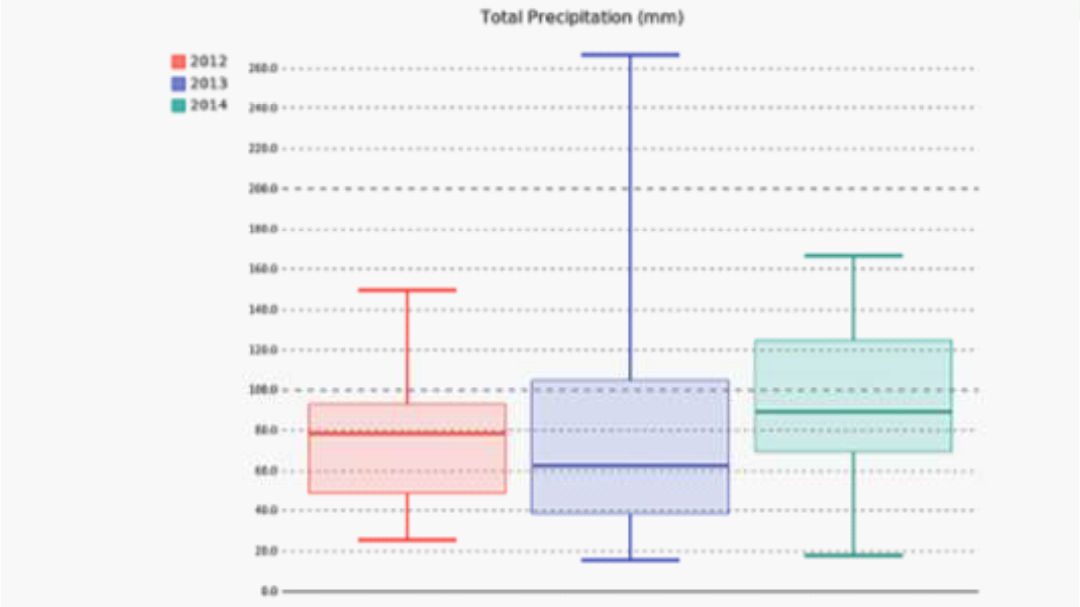

Gambar 12.7: *Plot batang diplot dengan Seaborn*

### 12.2.4 Box Plot

Box plot dapat digunakan untuk menampilkan nilai minimum, medium dan maksimum suatu kumpulan data. Dalam plot box, kumis menunjukkan ekstrem dari kumpulan data dan garis tengah adalah median. Box tersebut beralih dari kuartil pertama ke kuartil ketiga dari kumpulan data. Box 12.8 menunjukkan kode Python untuk membuat plot box dari total curah hujan yang tercatat di Atlanta untuk setiap bulan pada tahun 2012, 2013 dan 2014. Outputnya ditunjukkan pada Gambar 12.8.

**Box 12.8: Python program for plotting box plot using Pygal**

```
import pygal
box_plot = pygal.Box()
box_plot.title = 'Total Precipitation (mm)'

box_plot.add('2012', [67.81, 25.41 , 30.72, 78.49, 87.12,
   119.62, 98.56, 78.23 , 86.11 , 66.53, 25.65 , 149.59])

box_plot.add('2013', [28.68, 120.89, 41.90, 34.80, 81.54,
   266.70, 91.67, 46.73, 56.14 , 15.48, 69.09, 117.35])

box_plot.render_to_png('box.png')
```

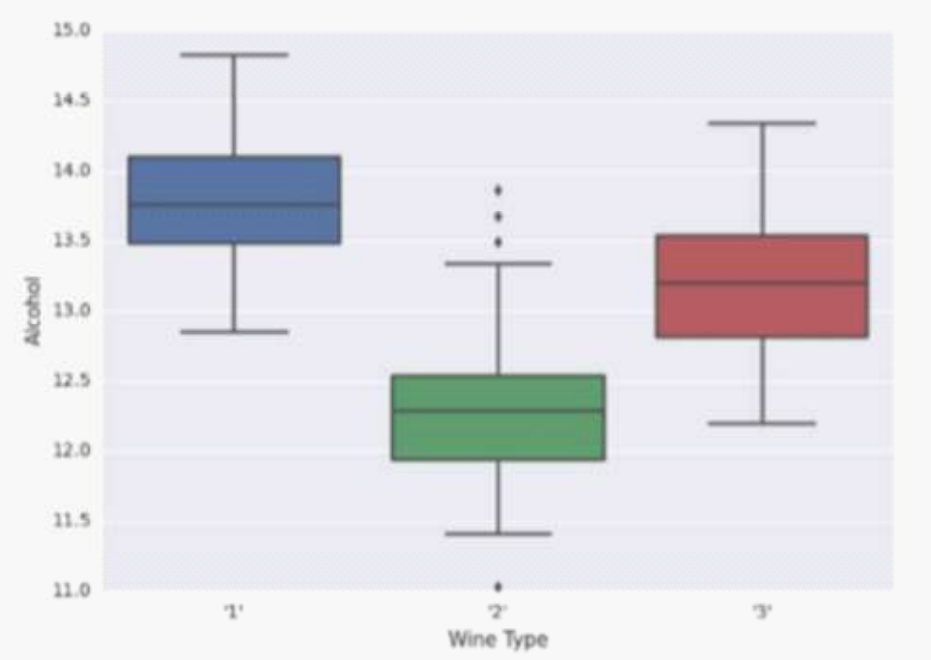



Gambar 12.8: *Plot box diplot dengan Pygal*

Mari kita lihat contoh lain dari plot box yang diplot menggunakan Seaborn. Plot box Seaborn menunjukkan kuartil dari kumpulan data dengan kumis yang menunjukkan ekstrem dan titik yang menunjukkan titik-titik yang ditentukan sebagai pencilan. Untuk memplot plot box kita akan menggunakan kumpulan data Wine dari gudang pembelajaran mesin UCIM [39]. Data ini merupakan hasil analisis kimiawi anggur yang ditanam di Italia. Analisis kimiawi menentukan jumlah 13 konstituen (seperti alkohol, asam malat, magnesium, dsb.) Yang ditemukan dalam tiga jenis anggur. Box 12.9 menunjukkan program Python untuk membuat plot box menggunakan Seaborn dan Gambar 12.9 menunjukkan plot box. Plot ini menunjukkan distribusi kandungan alkohol dalam tiga jenis anggur.

**Box 12.9: Python program for plotting box plot using Seaborn**

```
import pandas as pd
import seaborn as sns
from pylab import savefig

data = pd.read_csv("wine.csv")

sns.boxplot(x="Wine Type", y ="Alcohol", data=data)

savefig("box.png")
```



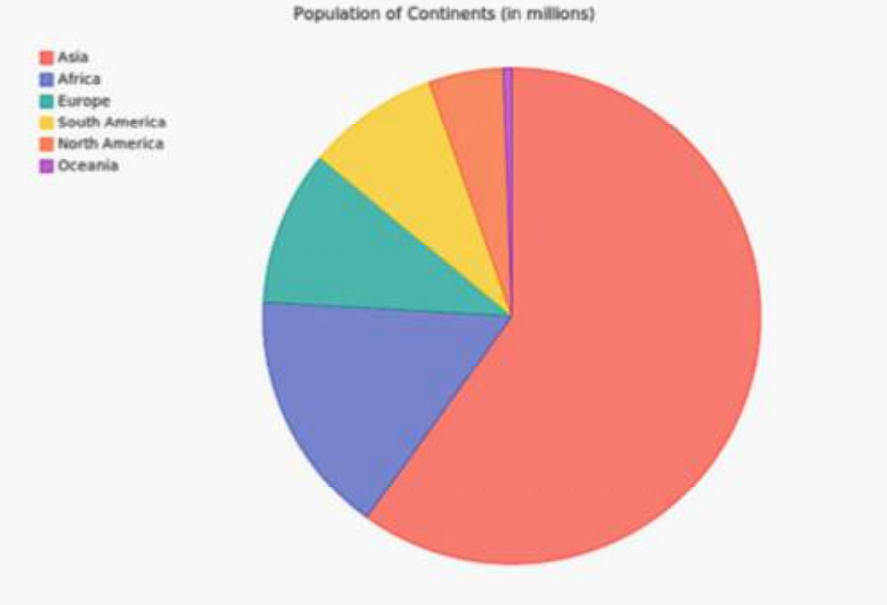

Gambar 12.9: *Plot Box diplot dengan Seaborn*

### 12.2.5 Diagram Lingkaran

Diagram lingkaran digunakan untuk menampilkan proporsi numerik pada lingkaran di mana panjang busur sebanding dengan kuantitas yang diwakili. Box 12.10 menunjukkan kode Python untuk membuat diagram lingkaran dari total populasi benua. Outputnya ditunjukkan pada Gambar 12.10.

**Box 12.10: Python program for plotting pie chart using Pygal**

```
import pygal

pie_chart = pygal.Pie()
pie_chart.title = 'Population of Continents (in millions)'

pie_chart.add('Asia', 4397)
pie_chart.add('Africa', 1171)
pie_chart.add('Europe', 742)
pie_chart.add('South America', 630)
pie_chart.add('North America', 357)
pie_chart.add('Oceania', 40)

pie_chart.render_to_png('pie.png')
```



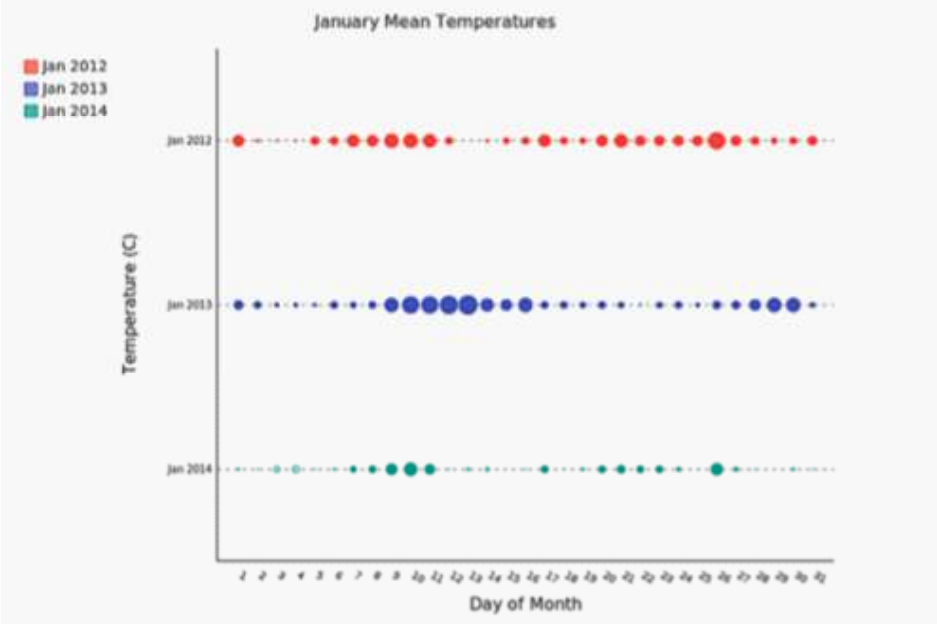

Gambar 12.10: *Diagram lingkaran diplot dengan Pygal*

### 12.2.6 Diagram Titik

Diagram titik digunakan untuk menampilkan kumpulan data yang berbeda di mana ukuran titik sebanding dengan nilai yang diwakili. Box 12.11 menunjukkan kode Python untuk memplot diagram titik dari suhu rata-rata Januari yang tercatat di Atlantaintheyears2012, 2013and2014. Outputnya ditunjukkan pada Gambar 12.11.

**Box 12.11: Python program for plotting dot chart using Pygal**

```
import pygal
dot_chart = pygal.Dot(x_label_rotation=30)
dot_chart.title = 'January Mean Temperatures'
dot_chart.x_title = 'Day of Month'
dot_chart.y_title = 'Temperature (C)'

dot_chart.x_labels = ['1','2','3','4','5','6','7', '8','9','10',
   '11','12','13','14','15', '16','17','18','19','20','21','22',
   '23', '24','25','26','27','28','29','30','31']

dot_chart.add('Jan 2012', [11, 3, -2, 2, 8, 8, 12, 11, 14, 14, 13,
    7, 0, 3, 6, 7, 12, 7, 6, 11, 13, 10, 10, 10, 10, 17, 10, 8, 6, 7, 9]

dot_chart.add('Jan 2013', [9, 7, 4, 4, 3, 7, 6, 7, 14, 17, 17,
   18, 19, 13, 11, 14, 7, 7, 6, 7, 6, 2, 6, 7, 4, 8, 8, 11, 14, 14, 5])

dot_chart.add('Jan 2014', [2, -1, -6, -7, -1, -2, 6, 7, 11, 13, 10,
   -1, -3, -4, 0, -1, 7, 1, -3, 7, 8, 7, 7, 4, 1, 12, 4, -1, 0, -3, -1])

dot_chart.render_to_png('dot.png')
```




Gambar 12.11: *Diagram titik diplot dengan Pygal*

### 12.2.7 Diagram Map

Mari kita lihat beberapa contoh pembuatan grafik map dunia menggunakan Lightning dan Pygal. Box 12.12 menunjukkan kode Python untuk membuat grafik map dunia menggunakan Lightning yang menunjukkan 20 negara terpadat. Outputnya ditunjukkan pada Gambar 12.12.

**Box 12.12: Python program for plotting map using Lightning**

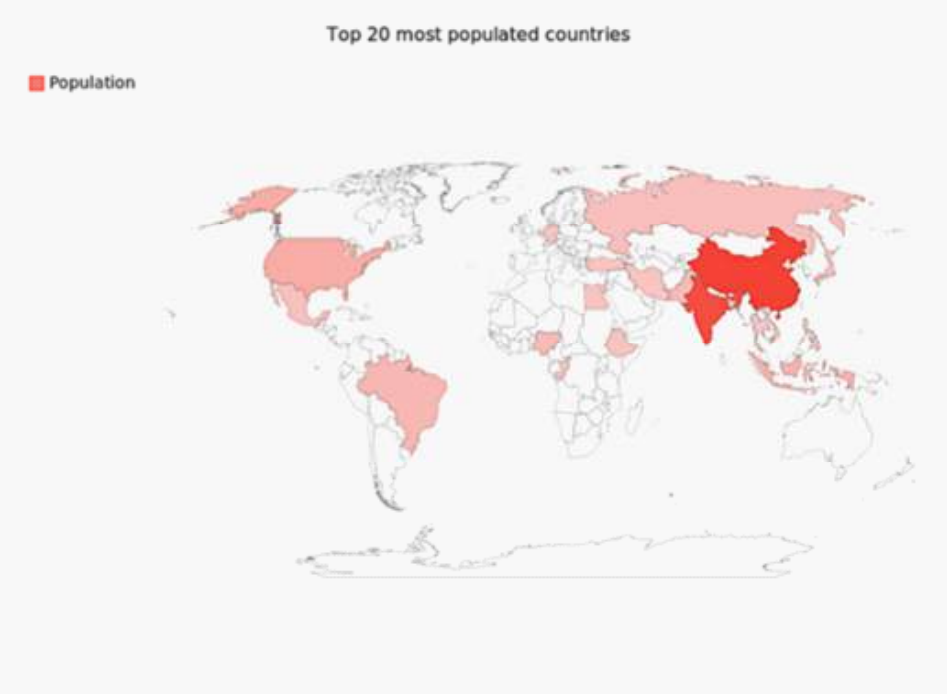
```
from lightning import Lightning

from numpy import random

lgn = Lightning(ipython=True,local=True)

countries = ['CHN','IND','USA','IDN','BRA',
   'PAK','NGA','BGD','RUS','JPN',
   'MEX','PHL','ETH','VNM','EGY',
   'DEU','IRN','TUR','COG','THA']

values = [1393783836,1267401849,322583006,
252812245,202033670,185132926,178516904,
158512570,142467651,126999808,123799215,
100096496,96506031,92547959,83386739,82652256,
78470222,75837020,69360118,67222972]

lgn.map(countries,values,colormap='Pastel1',width=900)
```

Box 12.13 menunjukkan kode Python untuk membuat grafik map dunia menggunakan Pygal yang menunjukkan 20 negara terpadat. Outputnya ditunjukkan pada Gambar 12.13.

Gambar 12.12: *Map diplot dengan Lightning*

**Box 12.13: Python program for plotting map chart using Pygal**

```
import pygal

worldmap_chart = pygal.maps.world.World()
worldmap_chart.title = 'Top 20 most populated countries '
worldmap_chart.add('Population',
'cn':  1393783836,
'in':  1267401849,
'us':  322583006,
'id':  252812245,
'br':  202033670,
'pk':  185132926,
'ng':  178516904,
'bd':  158512570,
'ru':  142467651,
'jp':  126999808,
'mx':  123799215,
'ph':  100096496,
'et':  96506031,
'vn':  92547959,
'eg':  83386739,
'de':  82652256,
'ir':  78470222,
'tr':  75837020,
'cg':  69360118,
'th':  67222972
)
worldmap_chart.render_to_png('map.png')
```



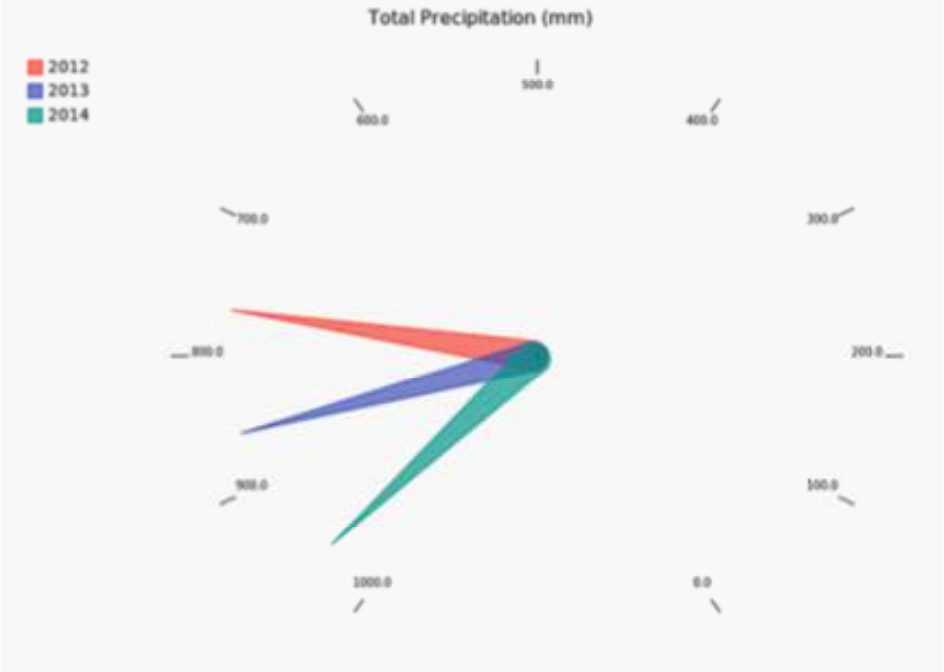

Gambar 12.13: *Map diplot dengan Pygal*

### 12.2.8 Diagram Pengukur

Diagram pengukur menampilkan data pada pengukur melingkar (seperti pada spedometer seluler otomatis) di mana jarum yang berbeda mewakili nilai yang berbeda. Box 12.14 menunjukkan kode Python untuk membuat grafik pengukur dari total curah hujan yang tercatat di Atlanta pada tahun 2012, 2013 dan 2014. Outputnya ditunjukkan pada Gambar 12.14.

**Box 12.14: Python program for plotting gauge chart using Pygal**

```
import pygal

gauge_chart = pygal.Gauge()
gauge_chart.title = 'Total Precipitation (mm)'
gauge_chart.range = [0, 1000]
gauge_chart.add('2012', 765)
gauge_chart.add('2013', 860)
gauge_chart.add('2014', 962)

gauge_chart.render_to_png('gauge.png')
```



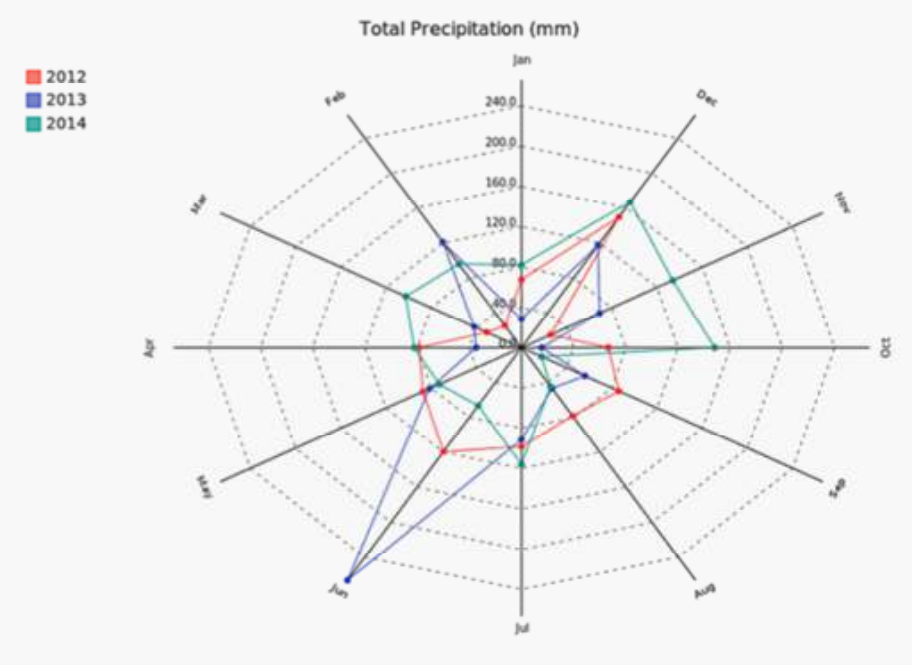

Gambar 12.14: *Diagram pengukur diplot dengan Pygal*

### 12.2.9 Diagram Radar

Diagram radar (disebut juga diagram Kiviat) digunakan untuk menampilkan data multivariat pada diagram dua dimensi dengan titik nol di tengahnya. Box 12.15 menunjukkan kode Python untuk memplot diagram radar dari total curah hujan yang tercatat di Atlanta di setiap bulan pada tahun 2012, 2013 dan 2014. Outputnya ditunjukkan pada Gambar 12.15.

**Box 12.15: Python program for plotting radar chart using Pygal**

```
import pygal

dot_chart = pygal.Radar()
dot_chart.title = 'Total Precipitation (mm)'

dot_chart.x_labels = ['Jan','Feb','Mar','Apr','May',
   'Jun','Jul','Aug','Sep','Oct','Nov','Dec']

dot_chart.add('2012', [67.81, 25.41 , 30.72, 78.49, 87.12,
    119.62, 98.56, 78.23 , 86.11 , 66.53, 25.65 , 149.59])

dot_chart.add('2013', [28.68, 120.89, 41.90, 34.80, 81.54,
    266.70, 91.67, 46.73, 56.14 , 15.48, 69.09, 117.35])

dot_chart.add('2014', [82.31, 96.01, 102.36, 82.30, 72.63,
   66.55, 116.07, 44.44, 17.77, 148.06, 133.86, 166.61 ])

dot_chart.render_to_png('radar.png')
```




Gambar 12.15: *Diagram Radar diplot dengan Pygal*

### 12.2.10 Diagram Matriks

Diagram matriks dapat digunakan untuk menampilkan data dalam format grid. Box 12.16 menunjukkan kode Python untuk memplot diagram matriks menggunakan Lightning yang menunjukkan matriks peringkat yang diberikan oleh pengguna untuk item yang berbeda. Outputnya ditunjukkan pada Gambar 12.16.

**Box 12.16: Python program for plotting matrix using Lightning**

```
from lightning import Lightning
from numpy import random

lgn = Lightning(ipython=True, local=True)

rows = ['User-' + str(x) for x in range(1,11)]
columns = ['Item-' + str(x) for x in range(1,11)]

mat = (random.rand(10,10) * 10).astype('int')
lgn.matrix(mat, row_labels=rows,
   column_labels=columns, colormap='Reds', numbers=True)
```


Gambar 12.16: *Matriks diplot dengan Lightning*

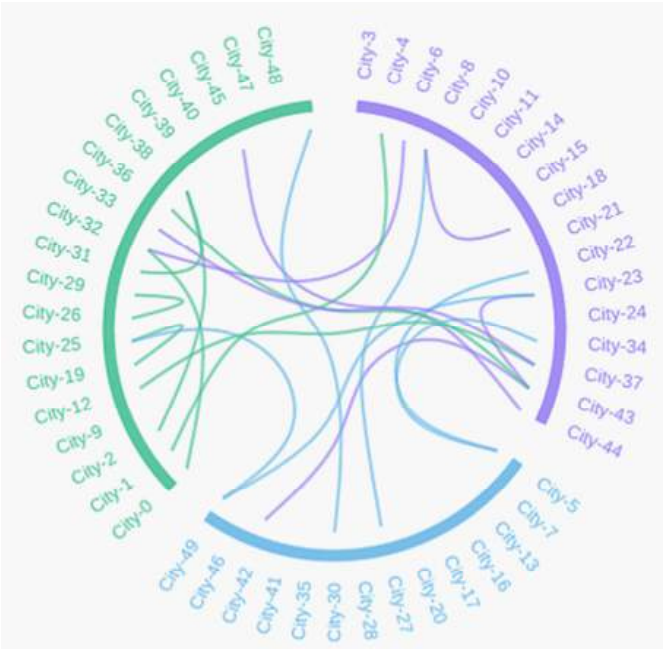
#### *Plot Lingkaran*

Plot lingkaran menunjukkan kelompok node sebagai titik di sekitar lingkaran dengan koneksi antar node diwakili oleh garis antar titik. Box 12.17 menunjukkan kode Python untuk merencanakan plot lingkaran menggunakan Lightning yang menunjukkan penerbangan

langsung antar kota (untuk kumpulan data sintetis acak) di tiga negara berbeda (ditampilkan sebagai tiga grup dengan warna berbeda). Outputnya ditunjukkan pada Gambar 12.17.

**Box 12.17: Python program for plotting circle chart using Lightning**

```
from lightning import Lightning
from random import randint
from numpy import random

lgn = Lightning(ipython=True, local=True)

#Nodes denoting cities
nodes = range(50)

#Links denoting direct flights between cities
links = [ ]
for i in range(20):
   x= [randint(0,50),randint(0,50)]
   links.append(x)

labels = ['City-' + str(x) for x in range(50)]

#Groups denoting countries
groups = [ ]
for i in range(50):
   groups.append(randint(0,2))

lgn.circle(links, group=groups, labels=labels)
```




Gambar 12.17: *Diagram lingkaran yang diplot menggunakan Lightning*

### 12.2.11 Grafik Force-directed

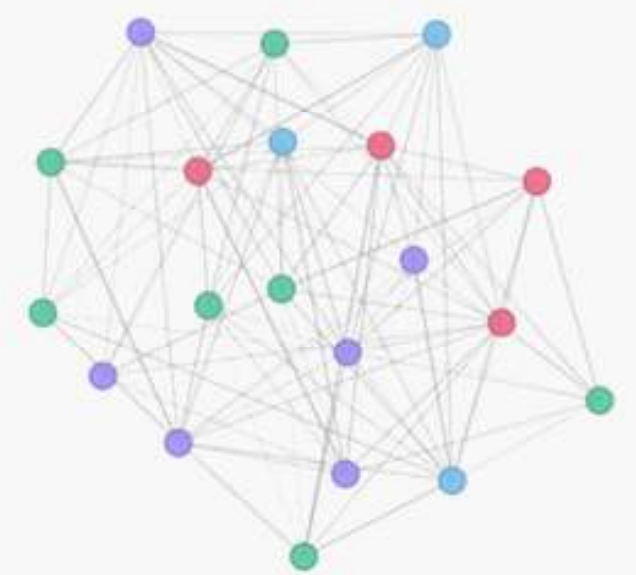
Grafik force-directed oleh gaya digunakan untuk menampilkan data dalam grafik yang menyenangkan secara estetika. Tepi mewakili koneksi antara node dan memiliki panjang yang kurang lebih sama. Tata letak node dalam grafik dibuat sedemikian rupa sehingga terdapat sesedikit mungkin cross edge. Box 12.18 menunjukkan kode Python untuk memplot grafik gaya diarahkan menggunakan Lightning yang menunjukkan penerbangan langsung antar kota (untuk kumpulan data sintetis acak) di tiga negara berbeda (ditampilkan sebagai node dalam warna berbeda). Outputnya ditunjukkan pada Gambar 12.18.

**Box 12.18: Python program for plotting force-directed graph using Lightning**

```
from lightning import Lightning
from numpy import random
from random import randint

lgn = Lightning(ipython=True, local=True)

#Nodes denoting cities
nodes = range(50)
```

```
#Links denoting direct flights between cities
links = [ ]
for i in range(20):
   x= [randint(0,50),randint(0,50)]
   links.append(x)

labels = ['City-' + str(x) for x in range(50)]

#Groups denoting countries
groups = [ ]
for i in range(50):
   groups.append(randint(0,2))


lgn.force(links, group=groups)
```

Gambar 12.18: *Grafik arah-gaya yang diplot menggunakan Lightning*



Gambar 12.19: *Grafik arah-gaya yang diplot menggunakan Lightning*

Box 12.19 menunjukkan contoh Python lain untuk merencanakan grafik yang diarahkan secara paksa dengan kumpulan data acak. Outputnya ditunjukkan pada Gambar 12.19.

**Box 12.19: Python program for plotting force-directed graph using Lightning**

```
from lightning import Lightning
from numpy import random
from random import randint


lgn = Lightning(ipython=True, local=True)

mat = random.rand(20,20) mat[mat>0.40] = 0
group = (random.rand(20) * 4).astype('int')

lgn.force(mat, group=group)
```

### 12.2.12 Grafik Spasial

Grafik spasial dapat digunakan untuk menampilkan kode dengan posisi spasial tetap, dan hubungan di antara keduanya. Box 12.20 menunjukkan contoh Python untuk memplot grafik spasial dengan kumpulan data acak. Outputnya ditunjukkan pada Gambar 12.20.

**Box 12.20: Python program for plotting spatial graph using Lightning**

```
from lightning import Lightning
from numpy import random
from random import randint

lgn = Lightning(ipython=True, local=True)

x = random.randn(20)
y = random.randn(20)
mat = random.rand(20,20)
mat[mat>0.50] = 0

lgn.graphbundled(x, y, mat)
```

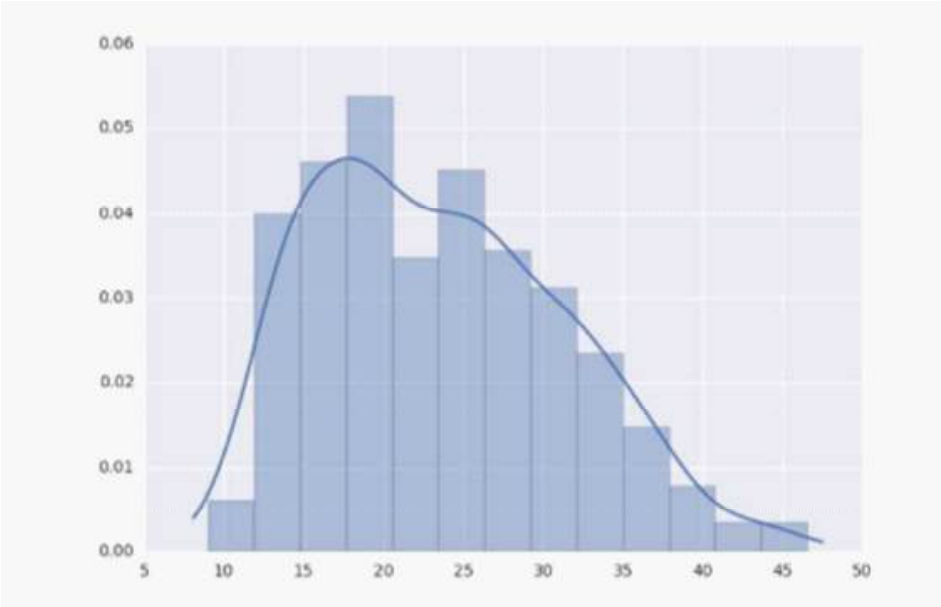
Gambar 12.20: *Grafik spasial pot menggunakan Lightning*

### 12.2.13 Plot Distribusi

Plot distribusi digunakan untuk memvisualisasikan distribusi observasi univariat. Box 12.21 menunjukkan program Python untuk membuat plot distribusi menggunakan Seaborn dan Gambar 12.21 menunjukkan plot distribusi. Plot ini menunjukkan estimasi kepadatan kernel dan histogram untuk nilai konsumsi bahan bakar (mpg) dalam dataset Auto MPG.

**Box 12.21: Python program for plotting distribution plot using Seaborn**

```
import pandas as pd
import seaborn as sns
from pylab import savefig
import numpy as np

data = pd.read_csv("auto-mpg.csv")

x=np.array(data.ix[0:,0])

g = sns.distplot(x)

savefig("dist.png")
```




Gambar 12.21 *: Plot Distribusi*

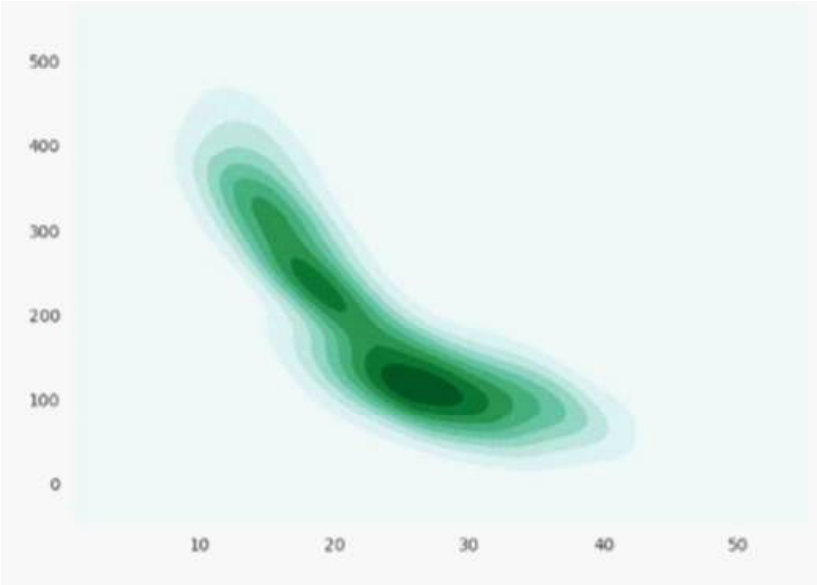
### 12.2.14 Plot Kernel Density Estimate (KDE)

Plot KDE dapat digunakan untuk memplot estimasi kepadatan kernel univariat atau bivariat. Box 12.22 menunjukkan program Python untuk membuat plot KDE menggunakan Seaborn dan Gambar 12.22 menunjukkan plot KDE. Plot ini menunjukkan kepadatan

bivariat untuk konsumsi bahan bakar (mpg) dan variabel perpindahan dalam dataset Auto MPG.

**Box 12.22: Python program for plotting KDE plot using Seaborn**

```
import pandas as pd
import seaborn as sns
from pylab import savefig
import numpy as np

data = pd.read_csv("auto-mpg.csv")

x=np.array(data.ix[0:,0])
y=np.array(data.ix[0:,2])

g = sns.kdeplot(x, y, shade=True)

savefig("kde.png")
```




Gambar 12.22: *Plot KDE*

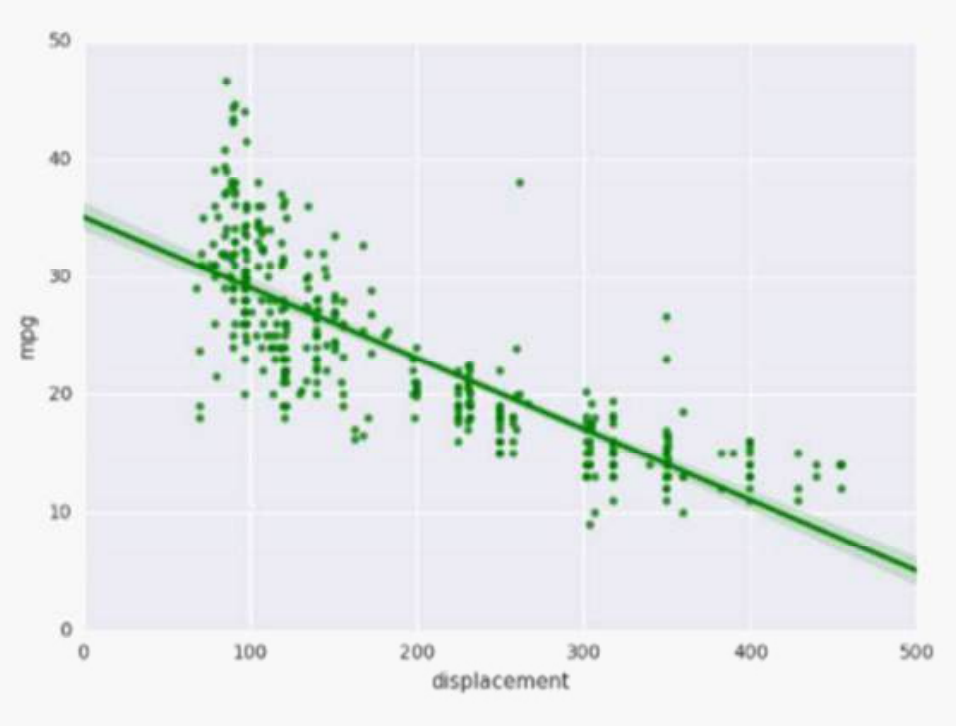
### 12.2.15 Plot Regresi

Plot regresi dapat digunakan untuk memplot data dan membuat model regresi linier. Box 12.23 menunjukkan program Python untuk membuat plot regresi menggunakan Seaborn dan Gambar 12.23 menunjukkan plot regresi. Plot ini menunjukkan perpindahan versus

data konsumsi bahan bakar (mpg) dalam dataset Auto MPG dan garis fit model regresi linier.

**Box 12.23: Python program for plotting regression plot using Seaborn**

```
import pandas as pd
import seaborn as sns
from pylab import savefig
import numpy as np

data = pd.read_csv("auto-mpg.csv")

#extract mpg column
x=np.array(data.ix[0:,0])

#extract displacement column
y=np.array(data.ix[0:,2])

sns.regplot(x="displacement", y="mpg", data=data, color="g")

savefig("regplot.png")
```



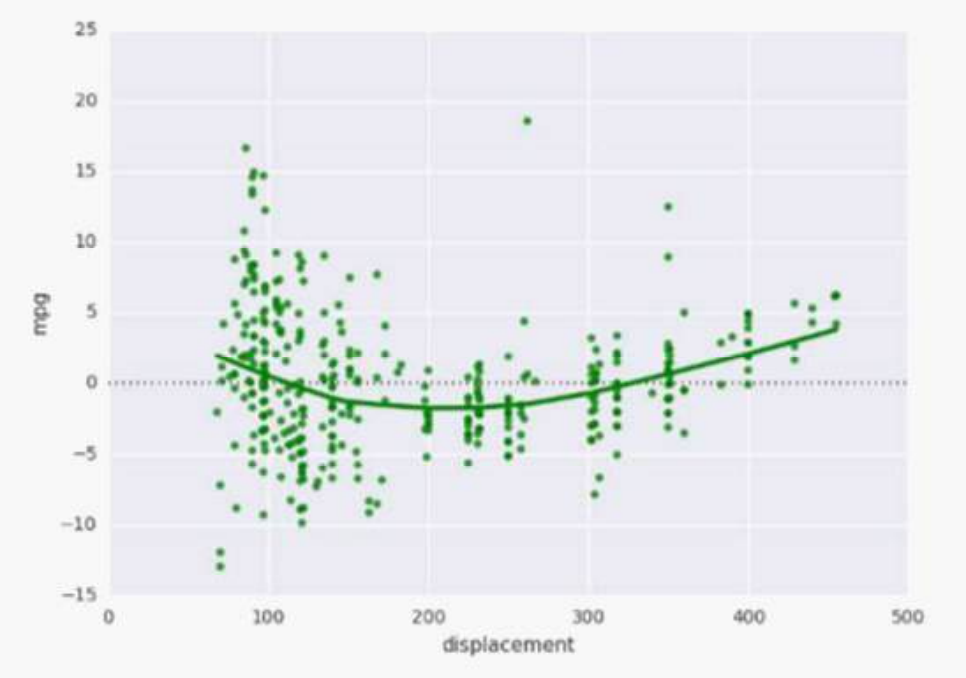

Gambar 12.23 *: Plot Regresi*

### 12.2.16 Plot Residual

Plot residual dapat digunakan untuk memplot residual regresi linier. Box 12.24 menunjukkan program Python untuk membuat plot sisa menggunakan Seaborn dan Gambar 12.24 menunjukkan plot sisa. Plot ini menunjukkan residual regresi linier antara variabel perpindahan dan konsumsi bahan bakar (mpg) dalam dataset Auto MPG.

**Box 12.24: Python program for plotting residual plot using Seaborn**

```
import pandas as pd
import seaborn as sns
from pylab import savefig
import numpy as np

data = pd.read_csv("auto-mpg.csv")

#extract mpg column
x=np.array(data.ix[0:,0])

#extract displacement column
y=np.array(data.ix[0:,2])

# plot the residuals after fitting a linear model
sns.residplot(x="displacement", y="mpg", data=data, lowess=True,
color="g")

savefig("resi.png")
```



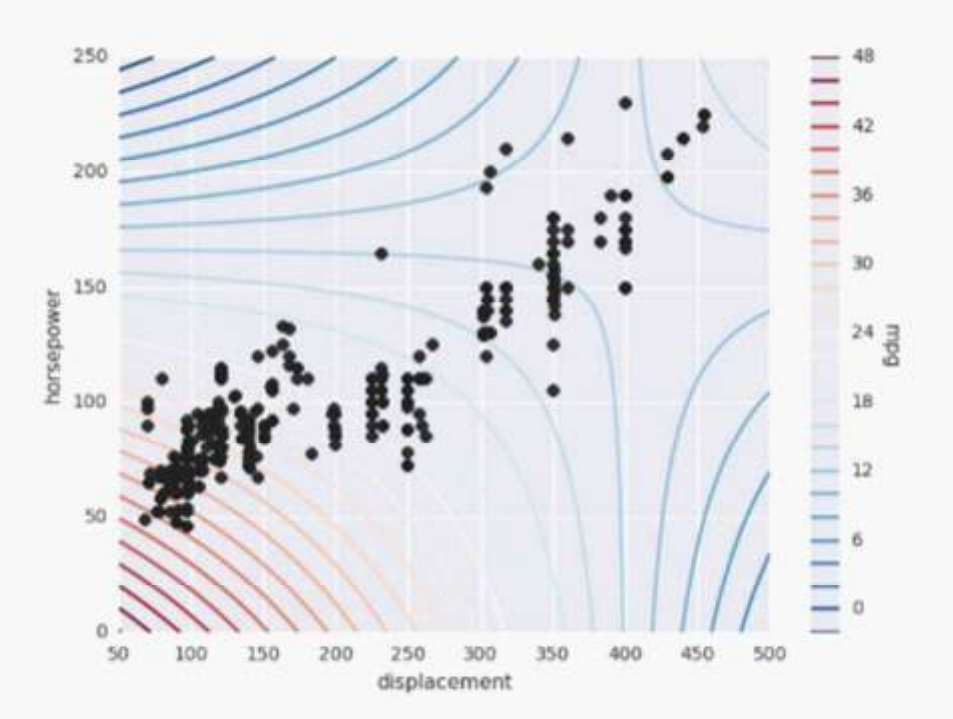

Gambar 12.24 *:Plot Residual*

### 12.2.17 Plot Interaksi

Plot interaksi dapat digunakan untuk memvisualisasikan interaksi dua arah yang berkelanjutan dengan plot kontur. Box 12.25 menunjukkan program Python untuk membuat plot interaksi menggunakan Seaborn dan Gambar 12.25 menunjukkan plot interaksi. Plot ini menunjukkan interaksi dua arah antara dua variabel independen (perpindahan dan tenaga kuda) dan variabel dependen (mpg) dalam dataset Auto MPG.

**Box 12.25: Python program for plotting interaction plot using Seaborn**

```
import pandas as pd
import seaborn as sns
from pylab import savefig
import numpy as np

data = pd.read_csv("auto-mpg.csv")

sns.interactplot("displacement", "horsepower", "mpg", data)

savefig("interactplot.png")
```




Gambar 12.25: *Plot Interaksi*

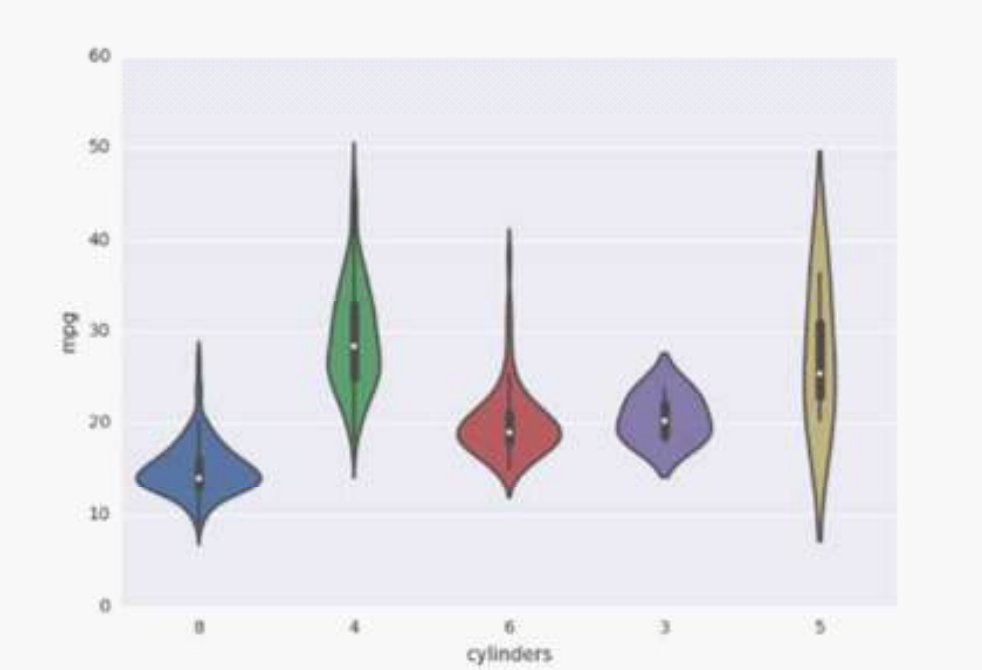
### 12.2.18 Plot Violin (Biola)

Plot biola dapat digunakan untuk memplot kombinasi plot box dan perkiraan kepadatan kernel. Box 12.26 menunjukkan program Python untuk membuat plot biola menggunakan

Seaborn dan Gambar 12.26 menunjukkan plot biola. Plot ini menunjukkan data distribusi konsumsi bahan bakar (mpg) untuk jumlah silinder yang berbeda (yang merupakan variabel kategori).

**Box 12.26: Python program for plotting violin plot using Seaborn**

```
import pandas as pd
import seaborn as sns
from pylab import savefig

data = pd.read_csv("auto-mpg.csv")

g = sns.violinplot(y="mpg", x ="cylinders", data=data)

savefig("violin.png")
```



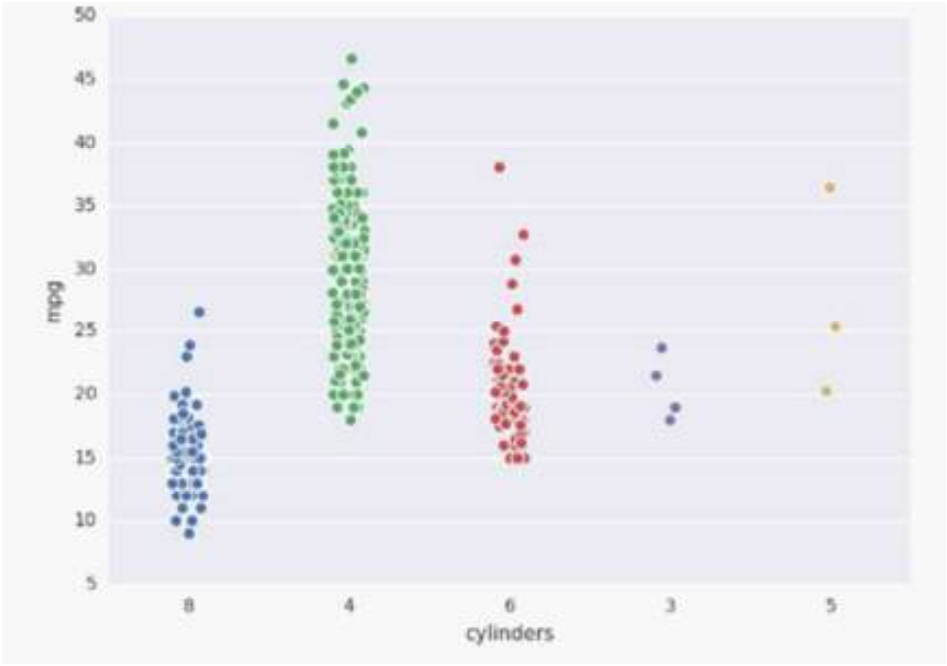

Gambar 12.26 *: Plot Biola*

### 12.2.19 Plot Strip (jalur)

Plot jalur dapat digunakan untuk memplot plot pencar di mana satu variabel dikategorikan. Box 12.27 menunjukkan program Python untuk membuat plot strip menggunakan Seaborn dan Gambar 12.27 menunjukkan plot strip. Plot ini menunjukkan data scatter plot konsumsi bahan bakar (mpg) untuk jumlah silinder yang berbeda (yang merupakan variabel kategori).

**Box 12.27: Python program for plotting strip plot using Seaborn**

```
import pandas as pd
import seaborn as sns
from pylab import savefig

data = pd.read_csv("auto-mpg.csv")

sns.stripplot(x="cylinders", y="mpg", data=data, jitter=True)

savefig("strip.png")
```



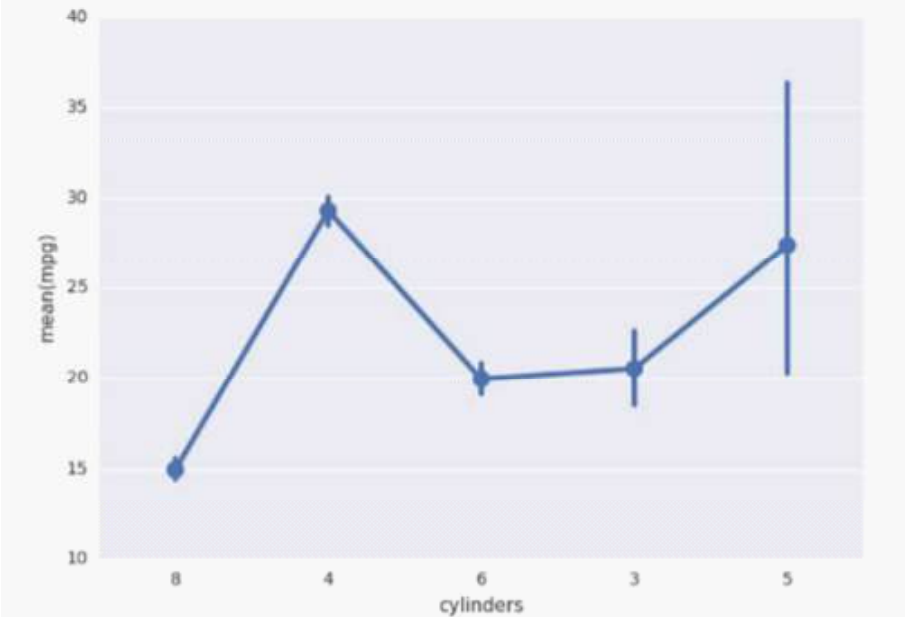

Gambar 12.27 *: Plot Jalur*

### 12.2.20 Plot Point (titik)

Plot titik dapat digunakan untuk memplot perkiraan tendensi sentral (misalnya mean) untuk variabel numerik (sebagai titik plot pencar) dan ketidakpastian di sekitar estimasi tersebut (menggunakan bilah kesalahan). Box 12.28 menunjukkan program Python untuk membuat plot titik menggunakan Seaborn dan Gambar 12.28 menunjukkan plot titik. Plot ini menunjukkan konsumsi bahan bakar rata-rata (mpg) untuk jumlah silinder yang berbeda untuk dataset MPG Otomatis.

**Box 12.28: Python program for plotting point plot using Seaborn**

```
import pandas as pd
import seaborn as sns
from pylab import savefig

data = pd.read_csv("auto-mpg.csv")

sns.pointplot(x="cylinders", y="mpg", data=data)

savefig("pointplot.png")
```



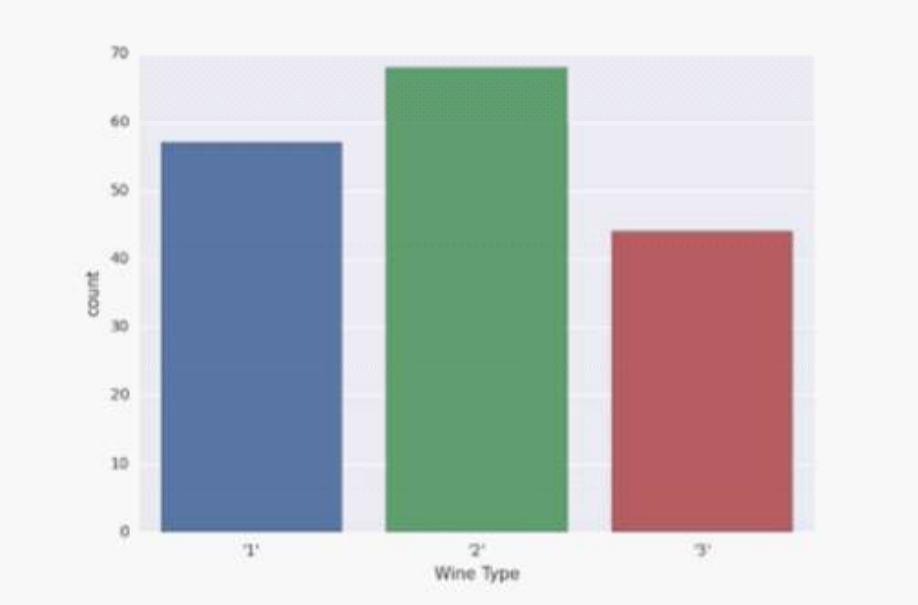

Gambar 12.28 *: Plot Titik*

### 12.2.21 Plot Count

Plot Count dapat digunakan untuk memplot jumlah pengamatan di setiap bin kategori menggunakan batang. Box 12.29 menunjukkan program Python untuk membuat plot hitungan menggunakan Seaborn dan Gambar 12.29 menunjukkan plot hitungan. Plot ini menunjukkan jumlah pengamatan untuk tiga jenis anggur dalam kumpulan data Wines.

**Box 12.29: Python program for plotting count plot using Seaborn**

```
import pandas as pd
import seaborn as sns
from pylab import savefig

data = pd.read_csv("wine.csv")

sns.countplot(x="Wine Type", data=data)

savefig("count.png")
```



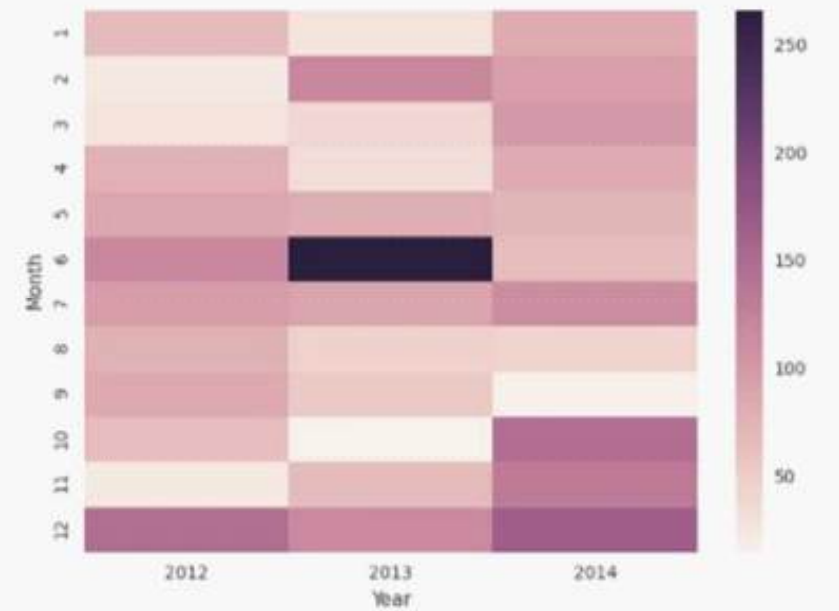

Gambar 12.29 : *Plot Count*

### 12.2.22 Heatmap

Map panas dapat digunakan untuk memplot matriks berkode warna. Box 12.30 menunjukkan program Python untuk membuat map panas menggunakan Seaborn dan Gambar 12.30 menunjukkan map panas. Dalam map panas ini, informasi indeks (bulan) dan kolom (tahun) digunakan untuk memberi label pada baris dan kolom dan nilai (curah hujan total) digunakan untuk kode warna. Dalam contoh ini, kami menggunakan fungsi pivot dari pustaka pandas Python untuk membentuk kembali data.

**Box 12.30: Python program for plotting heatmap using Seaborn**

```
data = pd.read_csv("data.csv")
data_to_plot = data.pivot(index="Month", columns="Year",
values="Total Precipitation")
sns.heatmap(data_to_plot)
```



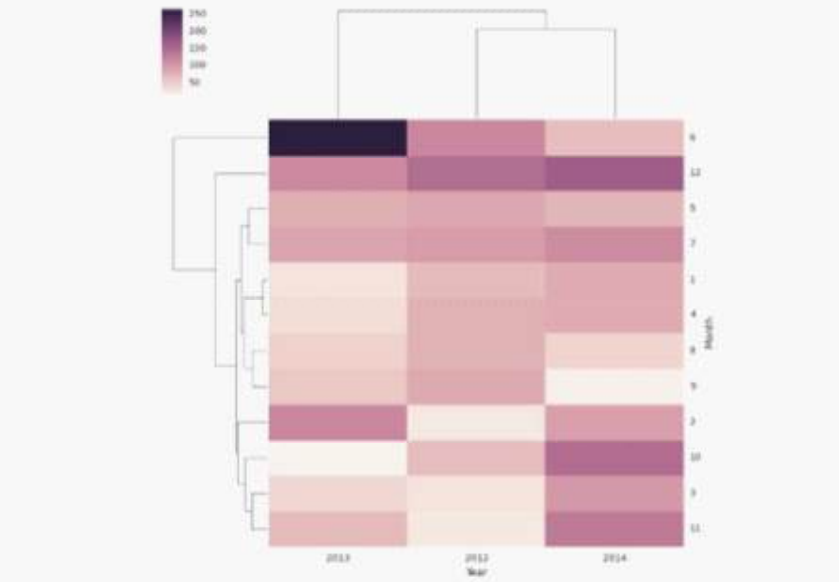

Gambar 12.30: *Heatmap*

### 12.2.23 Clustered Heatmap

Clustered Hipmap dapat digunakan untuk memplot map panas yang dikelompokkan secara hierarki. Box 12.31 menunjukkan program Python untuk merencanakan map panas berkerumun menggunakan Seaborn dan Gambar 12.31 menunjukkan map panas berkerumun.

**Box 12.31: Python program for plotting clustered heatmap using Seaborn**

```
import pandas as pd
import seaborn as sns
from pylab import savefig

data = pd.read_csv("data.csv")
data_to_plot = data.pivot(index="Month", columns="Year",
values="Total Precipitation")
sns.clustermap(data_to_plot)

savefig("clustermap.png")
```



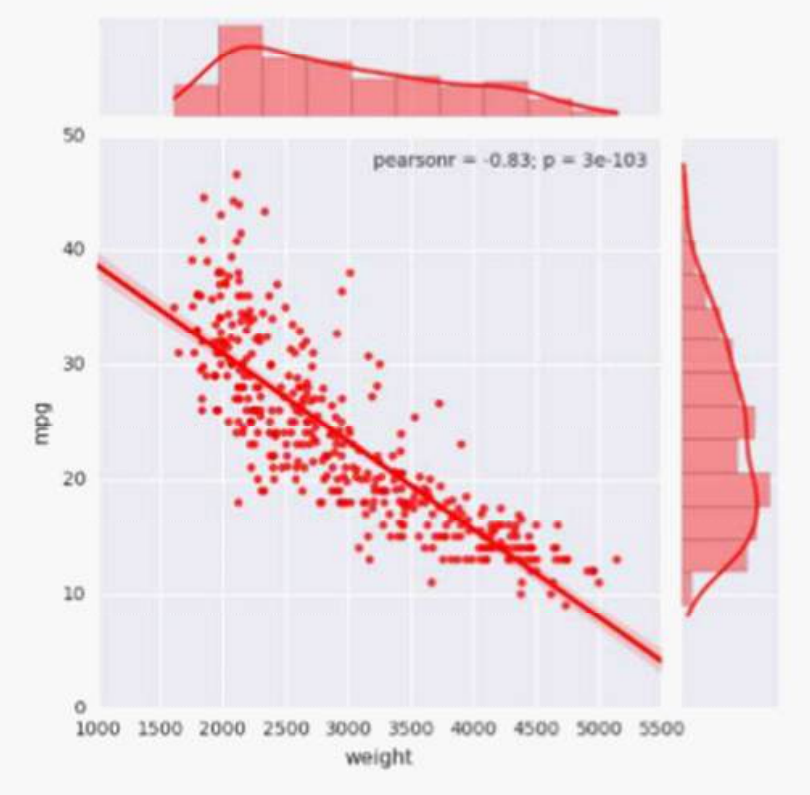
Gambar 12.31: *Clustered Heatmap*

### 12.2.24 Joint Plot

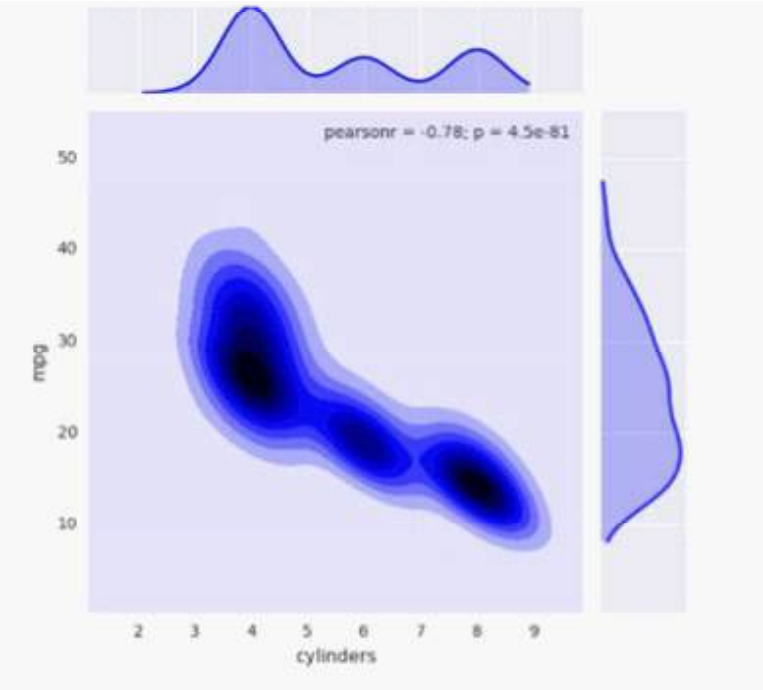
Plot gabungan dapat digunakan untuk memplot dua variabel dengan grafik bivariat dan univariat. Box 12.32 menunjukkan program Python untuk membuat plot bersama menggunakan Seaborn. Plot gabungan pada Gambar 12.32 menunjukkan sebar bobot vs mpg (untuk dataset Auto MPG) dan regresi dan cocok kepadatan kernel. Plot sambungan pada Gambar 12.33 menunjukkan perkiraan kepadatan untuk silinder dan mpg.

**■ Box 12.32: Python program for plotting joint plot using Seaborn**

```
import pandas as pd
import seaborn as sns
from pylab import savefig

data = pd.read_csv("auto-mpg.csv")
#g = sns.jointplot("weight", "mpg", data=data, kind="scatter", color="r")
g = sns.jointplot("cylinders", "mpg", data=data, kind="kde", color="b")

savefig("joint.png")
```



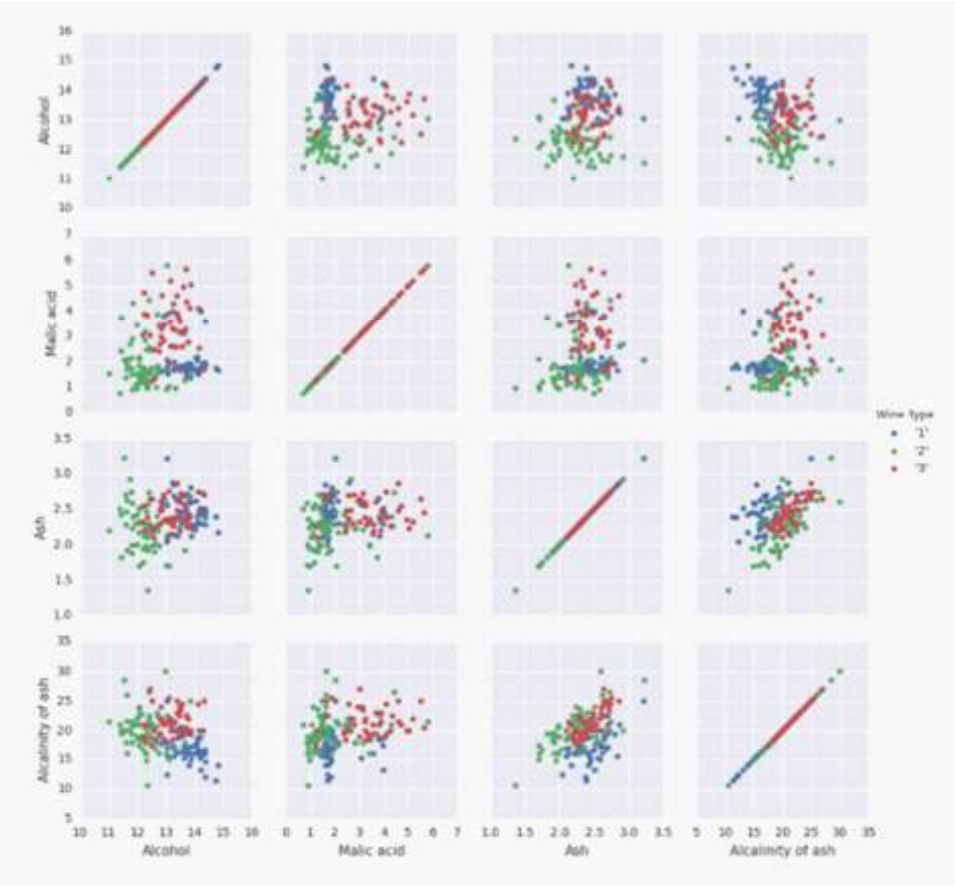

Gambar 12.32: *Joint Plot*




Gambar 12.33: *Joint Plot*

### 12.2.25 Pair Grid

Pair Grid digunakan untuk memplot hubungan berpasangan dalam kumpulan data. Box 12.33 menunjukkan program Python untuk membuat plot pasangan menggunakan Seaborn dan Gambar 12.34 menunjukkan plot pasangan. Plot ini menunjukkan plot pencar untuk setiap hubungan berpasangan antara variabel Alkohol, asam Malat, Abu dan Alkalinitas abu dalam dataset Wines.




Gambar 12.34 *: Pair grid*

**■ Box 12.33: Python program for plotting pair grid using Seaborn**

```
import pandas as pd
import seaborn as sns
from pylab import savefig
import matplotlib.pyplot as plt

data = pd.read_csv("wine.csv")
features_to_plot = ['Wine Type','Alcohol','Malic acid','Ash','Alkalinity
```

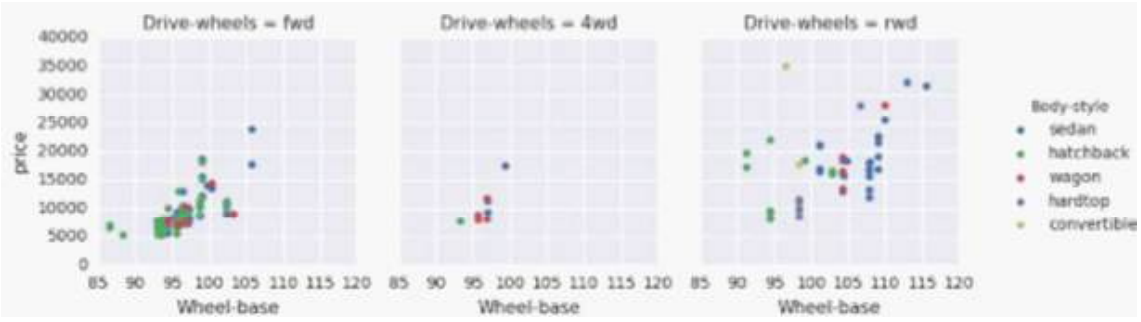
```
of ash']
df_to_plot = data.ix[:, features_to_plot]

g = sns.PairGrid(df_to_plot, hue="Wine Type")
g = g.map(plt.scatter)
g = g.add_legend()

savefig("pair.png")
```

### 12.2.26 Facet Grid

Facet Grid digunakan untuk menggambar grid plot hingga tiga dimensi di mana variabel baris dan kolom menghasilkan array sumbu dan variabel hue bertindak sebagai dimensi ketiga. Box 12.34 menunjukkan program Python untuk memplot grid facet untuk dataset Automobile dari repositori pembelajaran mesin UCI [33]. Gambar 12.35 menunjukkan kisi faset yang menggunakan dua dimensi (Roda penggerak untuk kolom dan Gaya bodi untuk corak). Grid facet menunjukkan plot sebar dari wheel-base dan harga.

**■ Box 12.34: Python program for plotting facet plot using Seaborn**

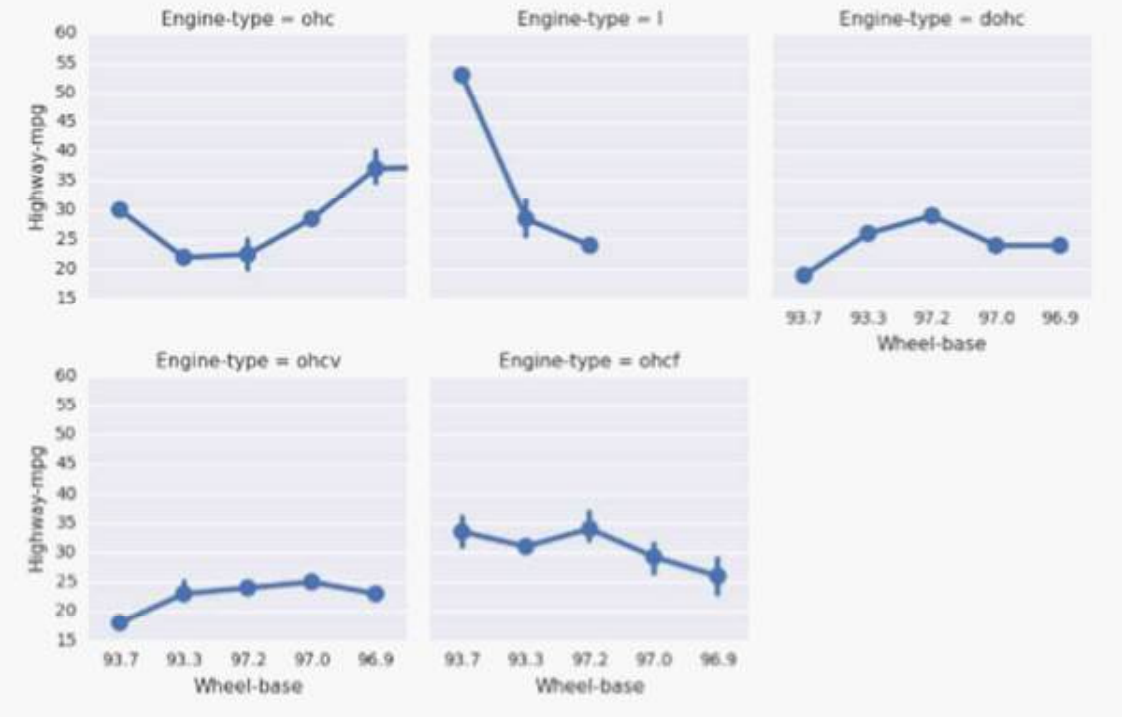
```
import pandas as pd
import seaborn as sns
from pylab import savefig
import matplotlib.pyplot as plt

data = pd.read_csv("imports-85.data.csv")

g = sns.FacetGrid(data, col="Drive-wheels",
hue="Body-style", margin_titles=True)
g = g.map(plt.scatter, "Wheel-base", "price")
g.add_legend()

savefig("facet.png")
```



Box 12.35 menunjukkan program Python untuk memplot kisi segi yang menunjukkan plot titik untuk alas roda dan jalan raya-mpg dan Gambar 12.36 menunjukkan plot.

**Box 12.35: Python program for plotting facet plot using Seaborn**

```
import pandas as pd
import seaborn as sns
from pylab import savefig

data = pd.read_csv("imports-85.data.csv")
g = sns.FacetGrid(data, col="Engine-type",
col_wrap=3, margin_titles=True)
g = g.map(sns.pointplot, "Wheel-base", "Highway-mpg")
g.add_legend()

savefig("facet.png")
```




Gambar 12.36: *Facet Plot*

## Ringkasan

Dalam bab ini, kami menjelaskan kerangka kerja Lightning, Pygal, dan Seaborn untuk visualisasi data. Visualisasi ini dapat digunakan baik secara mandiri atau di dalam aplikasi web yang dibangun dengan kerangka kerja web seperti Django.

# Bibliografi


[1] NationalResearchCouncil,FrontiersinMassiveDataAnalysis,TheNationalAcademies Press, 2013.

[2] R. Taft,M. Vartak,N.R. Satish, N. Sundaram, S. Madden, M. Stonebraker, GenBase: a complex analytics genomeics benchmark, SIGMOD '14 Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data, 2014.

[3] OpenRefine, http://openrefine.org/ , Retrieved 2016.

[4] Stanford DataWrangler, http://vis.stanford.edu/wrangler/ , Retrieved 2016.

[5] A. Bahga, V. Madisetti, A Cloud-Based Approach to Interoperable Electronic Health Records (EHRs), IEEE Journal of Biomedical and Health Informatics, Vol. 17, Iss. 5, Sep 2013.

[6] A. Bahga, V. Madisetti, Cloud-Based Information Integration & Informatics Framework for Healthcare Applications, IEEE Computer, 2013.

[7] Hortonworks HDP, http://hortonworks.com/hdp , Retrieved 2016.

[8] Cloudera CDH 5 Installation Guide, http://www.cloudera.com/documentation/cdh/5-1-x/CDH5-Installation-Guide/CDH5-Installation-Guide.html , Retrieved 2016.

[9] What is Big Data, https://www-01.ibm.com/software/in/data/bigdata/, IBM, Retrieved 2016. [10] Josh James, Data Never Sleeps 3.0, https://www.domo.com/blog/2015/08/data-never-sleeps-3-0/, DOMO, 2015.

[11] Amazon Elastic Compute Cloud, http://aws.amazom.com/ec2 , 2012.

[12] Google Compute Engine, https://developers.google.com/compute/ , Retrieved 2016.

[13] Windows Azure, https://azure.microsoft.com , Retrieved 2016.

[14] Google App Engine, http://appengine.google.com , 2012.

[15] Apache Ambari, https://ambari.apache.org/ , Retrieved 2016.

[16] Apache Ranger, http://ranger.apache.org/ , Retrieved 2016.

[17] Apache Knox, https://knox.apache.org/ , Retrieved 2016.

[18] AFINN Sentiment Lexicon, Finn Arup Nielsen, http://www2.imm.dtu.dk/pubdb/views/publication_details.php?id=6010 , Retrieved 2016.

[19] Leslie Lamport, The Part-Time Parliament, ACM Transactions on Computer Systems 16, 2 (May 1998), 133-169.

[20] Leslie Lamport, Paxos Made Simple, http://research.microsoft.com/en-us/um/people/lamport/pubs/paxos-simple.pdf , 2001.

[21] S. Ghemawat, H. Gobioff, S. Leung, The Google File System, SOSP 2003.

[22] J. Dean, S. Ghemawat, MapReduce: Simplified Data Processing on Large Clusters, OSDI 2004.

[23] Apache Storm, http://storm-project.net, Retrieved 2016.

[24] The Python Standard Library, http://docs.python.org/2/library/ , Retrieved 2016.

[25] Roy T. Fielding, Richard N. Taylor, Principled Design of the Modern Web Architecture, ACM Transactions on Internet Technology (TOIT), 2002.

[26] Mark Devaney, Bill Cheetham, Case-Based Reasoning for Gas Turbine Diagnostics, 18th International FLAIRS Conference, 2005.

[27] Harry Timmerman, SKF WindCon Condition Monitoring System for Wind Turbines, New Zealand Wind Energy Conference, 2009.

[28] Python-RecSys, http://ocelma.net/software/python-recsys/build/html/quickstart.html

[29] Google NGram Dataset, http://storage.googleapis.com/books/ngrams/books/datasetsv2 .html, Retrieved 2016.

[30] Employees Database, https://launchpad.net/test-db/ , Retrieved 2016.

[31] Weather Underground, http://www.wunderground.com/ , Retrieved 2016.

[32] Auto MPG Data Set, http://archive.ics.uci.edu/ml/datasets/Auto+MPG , Retrieved 2016.

[33] Automobile Data Set, http://archive.ics.uci.edu/ml/datasets/Automobile , Retrieved 2016.

[34] E.F. Codd, A Relational Model of Data for Large Shared Data Banks, Communications of the ACM 13 (6): 377–387, 1970.

[35] OpenWeatherMap API, http://openweathermap.org/api , Retrieved 2016.

[36] Seaborn, https://stanford.edu/mwaskom/software/seaborn/ Retrieved 2016.

[37] Corinna Cortes, Vladimir N. Vapnik, "Support-Vector Networks", Machine Learning, 20, 1995.

[38] Leo Breiman, "Random Forests", Machine Learning 45 (1): 5–32, 2001.

[39] Wine Data Set, https://archive.ics.uci.edu/ml/datasets/Wine , Retrieved 2016.

[40] Wine Quality Dataset, https://archive.ics.uci.edu/ml/datasets/Wine+Quality , Retrieved 2016.

[41] Adult Dataset, https://archive.ics.uci.edu/ml/datasets/Adult , Retrieved 2016.

[42] UCI Breast Cancer dataset, http://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic) ,Retrieved 2016.

[43] UCI Parkinsons Data Set, https://archive.ics.uci.edu/ml/datasets/Parkinsons , Retrieved 2016.

[44] Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. The Million Song Dataset. In Proceedings of the 12th International Society for Music Information Retrieval Conference (ISMIR 2011), 2011.

[45] Hastie,Trevor,RobertTibshirani,andJJeromeHFriedman.TheElementsofStatistical Learning. Vol.1. N.p., page 339: Springer New York, 2001.

[46] MNIST Dataset, http://yann.lecun.com/exdb/mnist/ , Retrieved 2016.

[47] MNIST Dataset File,https://s3.amazonaws.com/h2o-public-test-data/smalldata/ flow_examples/mnist/train.csv.gz, Retrieved 2016.

[48] MovieLens dataset, http://movielens.org , Retrieved 2016.

[49] IMDbPY, http://imdbpy.sourceforge.net , Retrieved 2016.

[50] Apache Hadoop, Fair Scheduler, http://hadoop.apache.org/docs/r1.1.1/fair_scheduler.html , 2013.

[51] Arun C. Murthy, The Hadoop Map-Reduce Capacity Scheduler, http://developer.yahoo.com/blogs/hadoop/posts/2011/02/capacity-scheduler/ , 2011.

[52] National Climatic Data Center (NCDC) Weather Dataset, ftp://ftp.ncdc.noaa.gov/pub/data/uscrn/products/daily01 , Retrieved 2016.

[53] A. Bahga, V. Madisetti, Rapid Prototyping of Advanced Cloud-Based Systems, IEEE Computer, vol. 46, iss. 11, Nov 2013.

[54] AutoBahn, http://autobahn.ws/ , Retrieved 2016.

[55] Amazon Web Services, http://aws.amazon.com , Retrieved 2016.

[56] Google Cloud Platform, https://cloud.google.com Retrieved 2016.

[57] Microsoft Windows Azure, http://www.windowsazure.com , Retrieved 2016.

[58] boto, http://boto.readthedocs.org/en/latest/ Retrieved 2016.

[59] Scikit-learn, http://scikit-learn.org/stable/ , Retrieved 2016.

[60] Apache Tez, http://tez.apache.org/ , Retrieved 2016.

[61] Django, https://docs.djangoproject.com/en/1.5/ , Retrieved 2016.

[62] Apache Spark, http://spark.apache.org , Retrieved 2016.

[63] http://code.google.com/p/modwsgi/ , Retrieved 2016.

[64] Apache Hadoop, http://hadoop.apache.org/ , Retrieved 2016.

[65] Storm, http://storm.incubator.apache.org/ , Retrieved 2016.

[66] Zookeeper, http://zookeeper.apache.org/ , Retrieved 2016.

[67] Lightning framework, http://lightning-viz.org/ , Retrieved 2016.