



PROYEK AKHIR

**IDS LOG ANALISIS MENGGUNAKAN HADOOP DAN
MAHOUT UNTUK
DATA MINING PADA MATA GARUDA**

**Masfu Hisyam
NRP. 2110145018**

Dosen Pembimbing :

**Ferry Astika Saputra. ST, M.Sc
NIP. 197708232001121002**

**Jauari Ahkmad S.ST
NIP. 2000000052**

**JURUSAN TEKNIK INFORMATIKA
POLITEKNIK ELEKTRONIKA NEGERI SURABAYA
2015**

[Halaman ini sengaja dikosongkan]



PROYEK AKHIR

**IDS LOG ANALISIS MENGGUNAKAN HADOOP
DAN MAHOUT UNTUK
DATA MINING PADA MATA GARUDA**

**Masfu Hisyam
NRP. 2110145018**

Dosen Pembimbing :

**Ferry Astika Saputra. ST, M.Sc
NIP. 197708232001121002**

**Jauari Ahkmad S.ST
NIP. 2000000052**

**JURUSAN TEKNIK INFORMATIKA
POLITEKNIK ELEKTRONIKA NEGERI SURABAYA
2015**

[Halaman ini sengaja dikosongkan]

**IDS LOG ANALISIS MENGGUNAKAN HADOOP DAN
MAHOUT UNTUK
DATA MINING PADA MATA GARUDA**

Oleh :

Masfu Hisyam

7411030030

Proyek Akhir ini Diajukan Sebagai Salah Satu Syarat Untuk
Memperoleh Gelar Ahli Madya (A. Md)

Di

Politeknik Elektronika Negeri Surabaya

Disetujui Oleh:

Dosen Penguji PA :

Dosen Pembimbing PA:

**1. Idris Winarno, S.ST, M.Kom
NIP. 198203082008121001**

**1. Umi Saa'dah S.Kom M.Kom.
NIP. 197404162000032003**

**2. Kholid Fathoni, S.Kom, MT
NIP. 198012262008121003**

**2. Jauari Akhmad S.ST
NIP. 2000000052**

**3. Rizky Yuniar Hakkun, S.Kom, MT
NIP. 198106222008121003**

Mengetahui

**Ketua Program Studi D3 Teknik Informatika Departemen Teknik
Informatika dan Komputer
Politeknik Elektronika Negeri Surabaya**

Arif Basofi, S.Kom, M.T

NIP. 197609212003121002

[Halaman ini sengaja dikosongkan]

ABSTRAK

Seiring berkembangnya teknologi internet saat ini kebutuhan keamanan jaringan sangat diperlukan karena kita internet hari ini digunakan oleh hampir seluruh aspek kehidupan masyarakat modern. Untuk itulah kami merancang suatu sistem matagaruda untuk memonitoring dan menganalisis keamanan jaringan, namun dalam perkembanganya menggunakan sistem *RDBMS* tradisional mengalami keterbatasan dalam segi volume data, jenis data, dan kecepatan pertumbuhan data. Untuk itulah dalam penelitian ini kami mengajukan teknologi *big data hadoop* untuk menyimpan, memproses, dan menganalisa. Dengan sistem terdistribusi menggunakan model pemrograman *map reduce* dan juga *HDFS* sebagai file sistem terdistribusi akan meningkatkan skalabilitas dalam memproses dan menyimpan. Dan juga dengan menggunakan *HIVE* sebagai perangkat *datawarehouse* akan lebih mudah melakukan *query* data dan *mahout* sebagai *machine learning* akan mempermudah proses analisis pada data yang sangat besar jumlahnya

Kata Kunci : keamanan jaringan, matagaruda, *hadoop*, *map reduce*, *HDFS*, *HIVE*, *Mahout*

[Halaman ini sengaja dikosongkan]

ABSTRACT

Internet technology is very important today and also the need of internet security is also important too. Because every aspect of modern society uses internet. That's why we design and implement monitoring and analyzing internet security called matagaruda. As the data more growth and bigger matagaruda has several problems and limitations due to conventional RDBMS technology like data volume, data variety, and data velocity. Based on those problems we propose to use big data technology using Hadoop to store, process, and analyze. Using its distributed parallel computing called map reduce model programming and also HDFS as a distributed file system across computer cluster to increase scalability. And HIVE as data warehouse tool to simplify query from HDFS. And Mahout as machine learning library will help to analyze large data set using map reduce.

Keyword: security network, matagaruda, Hadoop, map reduce, HDFS, HIVE, Mahout.

[Halaman ini sengaja dikosongkan]

KATA PENGANTAR



Assalamu'alaikum Wr. Wb.

Dengan mengucapkan puji syukur kepada Allah, atas limpahan rahmat dan hidayah-Nya sehingga penulis dapat menyelesaikan proyek akhir ini dengan judul :

IDS LOG ANALISIS MENGGUNAKAN HADOOP DAN MAHOUT UNTUK DATA MINING PADA MATA GARUDA

Proyek Akhir ini adalah kewajiban bagi setiap mahasiswa Politeknik Elektronika Negeri Surabaya dengan tujuan untuk memenuhi persyaratan untuk memperoleh kelulusan pada program Diploma IV Jurusan Teknik Informatika di Politeknik Elektronika Negeri Surabaya.

Dengan selesainya buku laporan proyek akhir ini, penulis berharap semoga buku ini dapat bermanfaat bagi pembaca umumnya dan juga bagi penulis pada khususnya serta semua pihak yang berkepentingan. Penulis juga berharap agar proyek akhir ini dapat dikembangkan lebih lanjut sehingga dapat benar-benar digunakan sebaik-baiknya untuk mendukung perkembangan ilmu pengetahuan.

Penulis menyadari bahwa penulis adalah manusia biasa yang tidak luput dari kesalahan dan kekurangan. Untuk itu, kritikan dan saran yang bersifat membangun kami harapkan untuk perbaikan selanjutnya.

Wassalamu'alaikum Wr. Wb.

Surabaya, Januari 2015

Masfu Hisyam

[Halaman ini sengaja dikosongkan]

UCAPAN TERIMA KASIH

Alhamdulillahirobbil ‘alamiin, saya sangat bersyukur atas terselesaikannya proyek akhir ini. Pada kesempatan ini saya ingin menyampaikan rasa hormat dan terima kasih yang sebesar besarnya kepada :

1. Allah SWT atas berkat rahmat serta hidayah-Nya, sehingga saya dapat menyelesaikan proyek akhir ini.
2. Bapak Dr. Zaenal Arif S.T, M.T, selaku Direktur Politeknik Elektronika Negeri Surabaya.
3. Bapak Ferry Astika Saputra. ST, M.Sc, dan Bapak Jauari Akhmad Nur Hasim, S.ST selaku Dosen Pembimbing. Terima kasih atas bimbingannya dan nasehat selama pengerjaan Proyek Akhir ini.
4. Ibu Arna Fariza, S.Kom, M.Kom, selaku Ketua Jurusan Teknologi Informasi
5. Seluruh dosen yang telah memberikan ilmunya pada penulis, dan seluruh karyawan dan teknisi yang telah ikut membantu.
6. Kepada Ayah, Ibu, Mbak Riza, Adekku Maya, serta keluarga besar saya yang senantiasa memberikan doa, perhatian, dorongan, semangat, dan kasih sayang mereka untuk saya.
7. Kepada saudara dan saudariku satu kelas D4 IT LJ 2014, yang memberi warna baru dan kesan saat di Lab TA.

Ucapan terimakasih saja tentu masih jauh dari cukup untuk menggambarkan rasa bersyukur penulis, semoga Allah SWT membalas kebaikan anda semua. Amin.

[Halaman ini sengaja dikosongkan]

DAFTAR ISI

ABSTRAK.....	vii
ABSTRACT.....	ix
KATA PENGANTAR	18
UCAPAN TERIMA KASIH	18
DAFTAR ISI	16
DAFTAR GAMBAR	18i
DAFTAR TABEL	18
BAB I PENDAHULUAN	1
1.1 Latar Belakang	Error! Bookmark not defined.
1.2 Tujuan	Error! Bookmark not defined.
1.3 Rumusan Masalah	Error! Bookmark not defined.
1.4 Batasan Masalah	Error! Bookmark not defined.
1.5 Metodologi	Error! Bookmark not defined.
1.6 Sistematika Pembahasan	Error! Bookmark not defined.
BAB II TEORI PENUNJANG	5
2.1 IDS	5
2.2 SNORT	7
2.3 Big Data	8
2.4 Hadoop	13
2.5 Hive	34
2.6 Apache Mahout	35
2.7 Matagaruda	35
2.8 Oauth2	36
2.9 Web Service	38
2.10 Rest Web Service	38
2.11 JSON	40
2.12 YAML	43
2.13 JAVA	43

BAB 3 PERANCANGAN DAN PEMBUATAN SISTEM	46
3.1. Metodologi Penelitian	46
3.2. Peralatan yang Digunakan	47
3.3. Arsitektur Sistem.....	48
3.4. Rancangan Software	48
3.5. Perancangan Insfrastrutur Server	50
3.6. Perancangan Hadoop Cluster	50
3.7. Data Analisis	66
3.8. Rancangan Sistem dan Implementasi	50
3.5. Perancangan Insfrastrutur Server	50
BAB IV UJI COBA DAN ANALISA DATA.....	80
4.1. Uji Coba	80
4.2. Analisa	74
BAB V PENUTUP	90
DAFTAR PUSTAKA.....	91
BIODATA PENULIS	83
LAMPIRAN.	85

DAFTAR GAMBAR

Gambar 2.1. Contoh Log Shnort.....	7
Gambar 2.2 3V (Volume, Velocity, Variety).....	9
Gambar 2.3 Arsitektur <i>Big Data</i>	10
Gambar 2.4 Konsep <i>Mapreduce</i>	12
Gambar 2.5 inti <i>hadoop</i> dan komponen dari <i>hadoop</i>	13
Gambar 2.6 Arsitektur <i>HDFS</i>	15
Gambar 2.7 komponen <i>HDFS</i>	16
Gambar 2.8 <i>namenode</i> pada <i>HDFS</i>	16
Gambar 2.9 interaksi antara <i>namenode</i> dan <i>datanode</i>	17
Gambar 2.1.0 arsitektur penyimpanan data.....	19
Gambar 2.1.1 arsitektur membaca data.....	20
Gambar 2.1.12 proses <i>mapping</i>	22
Gambar 2.1.13 proses <i>shuffle</i>	23
Gambar 2.1.14 proses <i>reducing</i>	23
Gambar 2.1.14 proses <i>reducing</i>	23
Gambar 2.1.15 keseluruhan proses <i>mapreduce</i>	23
Gambar 2.1.16 kerja <i>jobtracker</i>	25
Gambar 2.1.17 kerja <i>tasktracker</i>	25
Gambar 2.1.18 <i>multinode cluster</i>	26
Gambar 2.1.19 Sistem <i>Multi-Node Cluster</i>	27
Gambar 2.2.20 Proses aliran data Hadoop Mapreduce.....	29
Gambar 2.2.21 Ilustrasi secara detail terhadap proses <i>Mapreduce</i>	30
Gambar 2.2.22 Identifikasi job configuration dalam Mapreduce.....	33
Gambar 2.2.23 Mekanisme Kinerja OAuth 2.....	37
Gambar 3.1 Use Case Diagram.....	49
Gambar 3.2 Infrastruktur Server.....	50
Gambar 3.3 <i>format hadoop file system</i>	56
Gambar 3.4 <i>hadoop binary</i>	57
Gambar 3.5 Status servis lingkungan java.....	58
Gambar 3.6 <i>Hadoop Node</i>	58
Gambar 3.8 <i>Java Proses</i>	62
Gambar 3.9 <i>Node manager</i>	63
Gambar 3.10 <i>Hadoop User Interface</i>	63
Gambar 3.11 <i>Diagram proses</i>	66
Gambar 3.12 <i>Framework Sistem</i>	72
Gambar 3.13 <i>ERD</i>	73
Gambar 4.1 Ping ke sensor.....	80

Gambar 4.2 Log Snort.....	81
Gambar 4.3 Testing agent.....	82
Gambar 4.4 Query dari snort.....	83
Gambar 4.5 testing apache bench.....	83
Gambar 4.6 testing apache bench.....	85
Gambar 4.7 Request per second.....	86
Gambar 4.8 Time per request.....	87
Gambar 4.9 K-means cluster	88

[Halaman ini sengaja dikosongkan]

DAFTAR TABEL

Tabel 1. identifikasi job configuration.....	46
Tabel 2. Metode <i>HTTP</i> dan Penggunaannya dalam <i>REST</i>	39
Tabel 3. Perbandingan query join dan UDTF.....	87

BAB I

PENDAHULUAN

1.1 LATAR BELAKANG

Pertumbuhan internet yang semakin pesat dan penggunaannya yang semakin luas dalam aspek kehidupan, sehingga internet menjadi sasaran bagi para pelaku kejahatan untuk menjalankan aksinya. sehingga penggunaan internet harus diaamankan atau dicegah dengan menggunakan *IDS (Intrusion Detection System)*. *Tools IDS* memonitor lalu lintas data pada jaringan dan ketika terjadi suatu Event maka akan disimpan didalam log file. Dengan menganalisa log file ini maka administrator jaringan dapat menjalankan langkah pencegahan cyber attack. Namun untuk menganalisa log file tersebut dibutuhkan metode khusus menggunakan data mining. Maka untuk tujuan itulah dibuatlah project Mata Garuda untuk memonitor dan menganalisis jaringan dari *cyber attack*. Penggunaan database relasi dalam mata garuda membuat kemampuan dalam menganalisis menjadi terbatas karena data dari log file semakin membesar. Maka dari itulah diperlukan penggunaan *big data* dalam hal ini *hadoop* sebagai *framework big data* dan *apache mahout* sebagai *machine learning library*.

Optimasi pada proses *ETL (Extract, Transform, Load)* juga sangat menentukan hasil pengumpulan data. Karena jumlah sensor yang akan dipasang lebih dari satu dan juga tentunya data yang akan dikirim dari masing-masing sensor akan sedemikian besar. Sehingga dari sisi sensor dalam hal ini *snort* dikumpulkan sedikit demi sedikit menggunakan aplikasi kecil yang bernama *agent*

1.2 PERUMUSAN MASALAH

Berdasarkan uraian diatas, maka permasalahan yang timbul dalam pengerjaan proyek akhir ini adalah :

1. bagaimana melakukan log analisis menggunakan *hadoop* ?
2. bagaimana melakukan *ETL* menggunakan *REST webservice* dan *Oauth2* ?
3. Bagaimana melakukan data mining menggunakan *apache mahout* ?

1.3 BATASAN MASALAH

Batasan-batasan masalah pada proyek akhir ini adalah sebagai berikut :

1. Aplikasi Matagaruda ini hanya berjalan di *linux*
2. Membutuhkan *hadoop* versi 2.6 keatas dan *hive* dengan versi 1.1.0

1.4 TUJUAN DAN SASARAN

1. Mengoptimalkan kinerja analisis Mata Garuda.
2. Memudahkan pengembangan lebih lanjut dengan penggunaan *REST webservice* dan *Oauth2*
3. Membantu para administrator jaringan untuk melakukan analisis terhadap keamanan jaringan.

1.5 METODOLOGI

Metodologi yang digunakan dalam proyek akhir ini meliputi:

1. Arsitektur Sistem Mata Garuda

Mata garuda membutuhkan beberapa layer yang terdiri dari database dan aplikasi server dan client untuk dapat bekerja. Untuk level database yang terdiri dari *hadoop*, *hbase*, *hive*, *metastore*, *postgres*. Sedangkan untuk aplikasi server terdiri dari *oauth2 server*, antar muka *web*, *restfull api*. Dan yang terakhir adalah client yang digunakan untuk mengirim data dari sensor ke mata garuda.

2. Pembuatan desain Aplikasi

Yaitu dalam tahap ini akan dibuat *ERD* atau (*entity relational diagram*).

3. Tahap implementasi

Dalam tahap ini akan dibangun arsitektur dan desain dari server *hadoop* dan *hive*. Dan juga implementasi dari rancangan aplikasi matagaruda berdasarkan *ERD* dan *UML* yang telah dibuat sebelumnya disertasi *testing* dan *debugging*.

4. Tahap pembuatan laporan

Setelah tahap implementasi selesai maka selanjutnya adalah menyusun laporan mengenai penelitian pembuatan aplikasi matagaruda.

1.6 SISTEMATIKA STUDI

Sistematika pembahasan dari proyek akhir ini direncanakan sebagai berikut :

BAB I PENDAHULUAN

Bab ini berisi tentang pendahuluan yang terdiri dari latar belakang, perumusan masalah, batasan masalah, tujuan dan sasaran, metodologi, serta sistematika pembahasan dari proyek akhir ini.

BAB II TEORI PENUNJANG

Bab ini membahas mengenai teori-teori yang berkaitan dengan penyelesaian proyek akhir, yang didapatkan dari berbagai macam buku serta sumber-sumber terkait lainnya yang berhubungan dengan pembuatan proyek akhir ini.

BAB III PERANCANGAN SISTEM

Bab ini membahas mengenai instalasi *hadoop* dan *hive* dan perancangan sistem, meliputi pembuatan *ERD*, implementasi *ERD*, *testing* dan *debugging*.

BAB IV UJI COBA DAN ANALISA

Bab ini menyajikan dan menjelaskan seluruh hasil dan analisa dalam pembuatan proyek akhir ini dan

BAB V PENUTUP

Bab ini berisi kesimpulan dari uji coba perangkat lunak, dan saran untuk pengembangan, perbaikan serta penyempurnaan terhadap aplikasi yang telah dibuat.

BAB II

TEORI PENUNJANG

2. Dasar Teori

Teori-teori yang digunakan dalam penyelesaian proyek akhir akan dibahas dalam bab ini sesuai kaitannya dengan teknik pembuatan *framework*. Serta membahas software - software yang digunakan dalam pembuatan proyek akhir ini.

2.1. IDS

Intrusion Detection System adalah sebuah aplikasi perangkat lunak atau perangkat keras yang dapat mendeteksi aktivitas yang mencurigakan dalam sebuah sistem atau jaringan. IDS dapat melakukan inspeksi terhadap lalu lintas inbound dan outbound dalam sebuah sistem atau jaringan, melakukan analisis dan mencari bukti dari percobaan intrusi (penyusupan). (1) Ada dua jenis IDS, yaitu:

- *Network-based Intrusion Detection System (NIDS)*: Semua lalu lintas yang mengalir ke sebuah jaringan akan dianalisis untuk mencari apakah ada percobaan serangan atau penyusupan ke dalam sistem jaringan. NIDS umumnya terletak di dalam segmen jaringan penting di mana server berada atau terdapat pada “pintu masuk” jaringan. Kelemahan *NIDS* adalah bahwa *NIDS* agak rumit diimplementasikan dalam sebuah jaringan yang menggunakan *switch Ethernet*, meskipun beberapa vendor *switch Ethernet* sekarang telah menerapkan fungsi *IDS* di dalam *switch* buatannya untuk memonitor port atau koneksi.
- *Host-based Intrusion Detection System (HIDS)*: Aktivitas sebuah host jaringan individual akan dipantau apakah terjadi sebuah

percobaan serangan atau penyusupan ke dalamnya atau tidak. *HIDS* seringkali diletakkan pada server-server kritis di jaringan, seperti halnya *firewall*, *web server*, atau server yang terkoneksi ke Internet.

2.2. SNORT

SNORT adalah salah satu contoh software dari *NIDS* yang melakukan *matching* paket terhadap rule, dan menentukan apakah sebuah paket atau serangan yang masuk ke dalam jaringan merupakan sebuah intrusi atau bukan. *SNORT* mampu melakukan analisis *traffic* dan *packet logging* secara *real-time* pada jaringan, mampu melakukan analisis *protokol*, *matching content*, dan juga dapat digunakan untuk mendeteksi berbagai jenis serangan dan pemeriksaan seperti halnya *buffer overflows*, *stealth port scan*, *CGI attacks*, *SMB Probes*, *OS Fingerprinting attempts*, dan masih banyak lagi terkait dengan percobaan penyerangan pada jaringan. *SNORT* akan memberikan *alerts* kepada *administrator* jika ada paket yang melalui jaringan dan dianggap sebagai sebuah serangan atau intrusi, berikut ini merupakan contoh *alerts* dari *SNORT*:

```
01/22-08:47:00.846966 10.252.108.16 -> 10.252.108.17
ICMP TTL:64 TOS:0x0 ID:11454 IpLen:20 DgmLen:84 DF
Type:8 Code:0 ID:7019 Seq:1 ECHO
A5 56 C0 54 B2 38 0A 00 08 09 0A 0B 0C 0D 0E 0F .V.T.8.....
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F .....
20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F !"#%&'()*+,-./
30 31 32 33 34 35 36 37 01234567

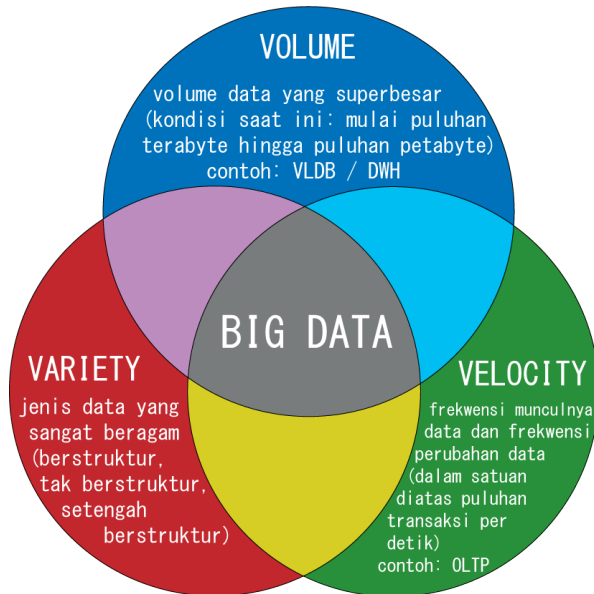
+++++
01/22-08:47:00.846966 10.252.108.16 -> 10.252.108.17
ICMP TTL:64 TOS:0x0 ID:11454 IpLen:20 DgmLen:84 DF
Type:8 Code:0 ID:7019 Seq:1 ECHO
A5 56 C0 54 B2 38 0A 00 08 09 0A 0B 0C 0D 0E 0F .V.T.8.....
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F .....
20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F !"#%&'()*+,-./
30 31 32 33 34 35 36 37 01234567

+++++
01/22-08:47:00.846966 10.252.108.16 -> 10.252.108.17
ICMP TTL:64 TOS:0x0 ID:11454 IpLen:20 DgmLen:84 DF
Type:8 Code:0 ID:7019 Seq:1 ECHO
A5 56 C0 54 B2 38 0A 00 08 09 0A 0B 0C 0D 0E 0F .V.T.8.....
--More--
```

Gambar 2.1 contoh log snort

Pada Gambar 2.1 diatas merupakan contoh dari alerts *SNORT* yang memberikan informasi berupa kategori *traffic* (*Bad Traffic*), *source IP Address* dan *Port*, *destination IP Address* dan *Port*, *Protokol*, *IpLen*, dan informasi lainnya. Informasi tersebut dapat disimpan kedalam database dan dapat digunakan untuk kebutuhan analisa lebih lanjut.

2.3. Big Data

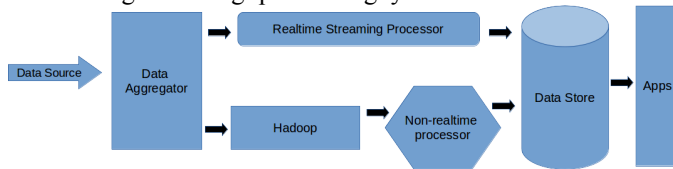


Gambar 2.2 : 3V (*Volume, Velocity, Variety*)

Hingga saat ini, definisi resmi dari istilah big data belum ada. Namun demikian, latar belakang dari munculnya istilah ini adalah fakta yang menunjukkan bahwa pertumbuhan data yang terus berlipat ganda dari waktu ke waktu telah melampaui batas kemampuan media penyimpanan maupun sistem database yang ada saat ini. Kemudian, McKinseyGlobal Institute (MGI), dalam laporannya yang dirilis pada Mei 2011, mendefinisikan bahwa big data adalah data yang sudah sangat sulit untuk dikoleksi, disimpan, dikelola maupun dianalisa dengan menggunakan sistem database biasa karena volumenya yang terus berlipat. Tentu saja definisi ini masih sangat relatif, tidak mendeskripsikan secara eksplisit sebesar apa big data itu. Tetapi, untuk saat sekarang ini, data dengan volume puluhan terabyte hingga beberapa *petabyte* kelihatannya

dapat memenuhi definisi MGI tersebut. Di lain pihak, berdasarkan definisi dari Gartner, big data itu memiliki tiga atribut yaitu : *volume* , *variety* , dan *velocity*. Ketiga atribut ini dipakai juga oleh IBM dalam mendefinisikan big data. Volume berkaitan dengan ukuran, dalam hal ini kurang lebih sama dengan definisi dari MGI. Sedangkan variety berarti tipe atau jenis data, yang meliputi berbagai jenis data baik data yang telah terstruktur dalam suatu database maupun data yang tidak terorganisir dalam suatu database seperti halnya data teks pada web pages, data suara, video, click stream, log file dan lain sebagainya. Yang terakhir, *velocity* dapat diartikan sebagai kecepatan dihasilkannya suatu data dan seberapa cepat data itu harus diproses agar dapat memenuhi permintaan pengguna. Dari segi teknologi, dipublikasikannya *Google Bigtable* pada 2006 telah menjadi moment muncul dan meluasnya kesadaran akan pentingnya kemampuan untuk memproses '*big data*'. Berbagai layanan yang disediakan *Google*, yang melibatkan pengolahan data dalam skala besar termasuk search engine-nya, dapat beroperasi secara optimal berkat adanya *Bigtable* yang merupakan sistem database berskala besar dan cepat. Semenjak itu, teknik akses dan penyimpanan data *KVS (Key-Value Store)* dan teknik komputasi paralel yang disebut *MapReduce* mulai menyedot banyak perhatian. Lalu, terinspirasi oleh konsep dalam *GoogleFile System* dan *MapReduce* yang menjadi pondasi *Google Bigtable*, seorang karyawan Yahoo! bernama Doug Cutting kemudian mengembangkan software untuk komputasi paralel terdistribusi (*distributed parallel computing*) yang ditulis dengan menggunakan *Java* dan diberi nama *Hadoop*. Saat ini *Hadoop* telah menjadi project open source-nya *Apache Software*. Salah satu pengguna *Hadoop* adalah *Facebook*, *SNS (Social Network Service)* terbesar dunia dengan jumlah pengguna yang mencapai 800 juta lebih. *Facebook* menggunakan *Hadoop* dalam memproses *big data* seperti halnya *content sharing*, analisa *access log*, layanan *message / pesan* dan layanan lainnya yang melibatkan pemrosesan *big data*. Jadi, yang dimaksud dengan '*big data*' bukanlah semata-mata hanya soal ukuran, bukan hanya tentang data yang berukuran raksasa. *Big data* adalah data berukuran raksasa yang volumenya terus bertambah, terdiri dari berbagai jenis atau varietas data, terbentuk secara terus menerus dengan kecepatan tertentu dan harus diproses dengan kecepatan tertentu pula. Momen awal ketenaran istilah '*big data*' adalah kesuksesan *Google* dalam memberdayakan '*big data*'

dengan menggunakan teknologi canggihnya yang disebut *Bigtable* beserta teknologi-teknologi pendukungnya.



Gambar 2.3 Arsitektur *Big Data*

Data source adalah sumber data untuk *big Data*. Data umumnya dipompa masuk *Big Data* dengan menggunakan *API* ataupun dengan operasional *file system* seperti transfer file. Ada dua jenis data source yaitu *streaming data source* dan *bulk data source*. Contoh *streaming data source* misalnya adalah *tweets* dari *twitter API*. Sedangkan *Bulk data* misalnya adalah file teks biasa yang sangat besar seperti file log dari suatu aplikasi ataupun file yang berisi data yang di dump dari database.

Data aggregator adalah *tool* atau *software* yang mengumpulkan dan menyalurkan data dari sumber ke beberapa jenis pengolahan data di *Big data*. Ada dua jenis data aggregator berdasarkan cara kerjanya. Jenis pertama adalah *Pull-based data aggregator*. Jenis ini mengumpulkan data dan memberikan data tersebut kepada siapa saja yang meminta tanpa registrasi sebelumnya, mirip seperti *Java Messaging Queue*. Contohnya adalah *Apache Kafka*, *RabbitMQ*. Jenis kedua adalah *Push-based data aggregator*. Jenis kedua ini mengumpulkan data dan mengirim data ke sistem lain yang sudah di set terhubung dan menerima data dari data aggregator. Sistem yang mau mendapatkan data harus ‘terdaftar’ di data *aggregator* dulu dan biasanya diperlukan effort lebih jika ada sistem baru yg ingin mendapatkan data dari data aggregator jenis ini dibanding jenis yang pertama. Contoh *Push-Based Data Aggregator* adalah *Apache Flume* dan *Spring-XD*.

Realtime streaming Processor adalah salah satu sistem pengolahan di *Big Data* yang umum ditemukan. Fungsinya adalah untuk menganalisis data yang bersifat *realtime* dan *streaming*. Contohnya adalah menghitung *hashtag* yang muncul di semua tweet di *twitter*. Sifat dari pemrosesan ini haruslah ringan, dan cepat. Oleh karena itu analisis data secara kompleks jarang sekali dilakukan. Output dari pemrosesan ini adalah gambaran umum dari data yang didapatkan dan tidak terlalu detail. Outputnya pun sebaiknya disimpan di *datastore* sehingga bisa digunakan oleh aplikasi yang

membutuhkan. Untuk hasil analisis data yang sangat detil bisa di lihat di *Non-realtime processor*. Contoh *tool* yang digunakan di realtime streaming misalnya adalah *Apache Storm*, *Apache Spark Streaming* dan *Spring-XD*. Meskipun hasilnya tidal detil, tetapi pemrosesan ini diperlukan mengingat pemrosesan secara bulk / non-realtime membutuhkan waktu yang cukup lama. Dengan demikian user bisa melihat secara garis besar data yang diolah meskipun tidak detil sembari menunggu pemrosesan *non realtime* selesai..

Hadoop disini yang saya maksud adalah *HDFS*. Disini *hadoop* lebih ditekankan sebagai tempat penyimpanan data yang sangat besar. *Hadoop* menjadi tempat semua data sehingga bisa dianalisis oleh berbagai tools untuk berbagai kepentingan sehingga bisa didapatkan hasil yang cukup detil dan bisa memenuhi kebutuhan dari user.

Non-realtime processor adalah proses pemrosesan data di *Big Data* untuk data besar yang terdapat di *HDFS*. pemrosesan ini menggunakan berbagai jenis tool sesuai kebutuhan. sebuah data bisa dianalisis lebih dari satu tools. Contoh tool yang sering digunakan antara lain *Hive* dan *Pig* untuk *Map Reduce*, *Apache Mahout* dan *Apache Spark* untuk *machine learning* dan *artificial intelligence*. Hasil dari pemrosesan ini dimasukkan ke dalam data store untuk kemudian bisa di lihat di level aplikasi. Sistem pemrosesan ini umumnya memerlukan waktu yang relatif lebih lama mengingat data yang diproses relatif sangat besar.

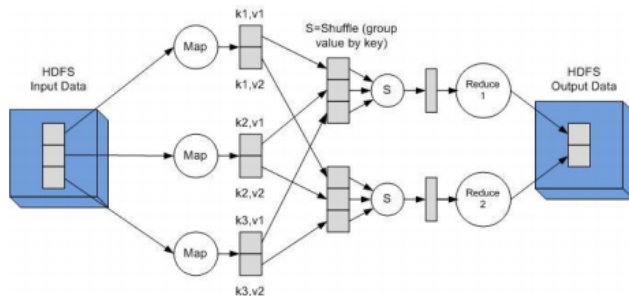
Data store adalah tools untuk menyimpan data hasil pemrosesan baik *realtime* maupun *on-realtime*. *Datastore* disini bisa berupa *RDBMS* ataupun jenis *NoSQL* lainnya. *RDBMS* sangat jarang digunakan sebagai data store mengingat keterbatasan dalam sisi ukuran yang bisa ditampung tanpa kehilangan kinerja. *Datastore* yang umumnya dipakai adalah *NoSQL* yang berbasis *Document* (mis. *MongoDB*), *Column-oriented* seperti *HBase* dan *Cassandra*, dan juga *key-value pair* seperti *couchDB*. Beberapa *data store* yang jarang kedengaran juga dipakai seperti misalnya *Voldemort* dan *Druid*.

Apps adalah aplikasi yang berinteraksi langsung dengan user. Aplikasi disini mengakses data yang berada di data store untuk kemudian disajikan kepada user. Jenis aplikasi disini sangat bervariasi bisa berupa web, desktop ataupun mobile. Pada umumnya aplikasi disini hanyalah untuk melakukan visualisasi dari data yang sudah dianalisis sebelumnya. insert data tidak saya temui

untuk jenis aplikasi ini. Karena memang ditujukan untuk user, maka data yang disajikan harus sesuai dengan kebutuhan user.

Konsep *Mapreduce*

MapReduce adalah model pemrograman yang digunakan untuk mendukung *distributed computing* yang dijalankan di atas data yang sangat besar dan dijalankan secara paralel di banyak komputer. *Mapreduce* memungkinkan programmer untuk melakukan komputasi yang sederhana dengan menyembunyikan kompleksitas dan detail dari paralelisasi, distribusi data, *load balancing* dan *fault tolerance*. Program *Mapreduce* ini dapat dijalankan pada berbagai bahasa pemrograman termasuk *Java*, *Python*, dan *C++* (2). Konsep *Mapreduce* membagi proses menjadi dua tahapan, yaitu *Map* dan *Reduce*. *Map* merupakan proses yang berjalan secara *parallel*, sedangkan *reduce* merupakan penggabungan hasil dari proses *Map*. *Map* memiliki fungsi yang dipanggil untuk setiap input yang menghasilkan output pasangan $\langle \text{key}, \text{value} \rangle$. Pada kondisi ini *Map* melakukan *transformasi* setiap data elemen input menjadi data elemen output. *Reduce* adalah tahapan yang dilakukan setelah *mapping* selesai. *Reduce* akan memeriksa semua value input dan mengelompokkannya menjadi satu value output. Sebelum memasuki tahap *reduce*, pasangan $\langle \text{key}, \text{value} \rangle$ dikelompokkan berdasarkan *key*. Tahap ini dinamakan tahap *shuffle*. Tahap *shuffle* dilakukan sebagai persiapan menuju tahap *reduce*. *Reduce* juga memiliki fungsi yang dipanggil untuk setiap *key*. Fungsi *reduce* memperhatikan setiap value dari *key* bersangkutan. Untuk setiap *key* akan dihasilkan sebuah ringkasan value-nya. Proses skema *Mapreduce* dapat dilihat pada gambar 2.4.



Gambar 2.4 Konsep *Mapreduce*

Pada gambar 2.3 dapat menunjukkan bagaimana proses mapping menghasilkan suatu pasangan *key* dan *value*, bagaimana *key* dan *value* tersebut dikelompokkan berdasarkan *key* yang sama kemudian dilakukan proses reducing, sehingga mendapatkan suatu output.

2.4. HADOOP

Hadoop adalah *framework* perangkat lunak berbasis *Java* dan *opensource* yang berfungsi untuk mengolah data yang besar secara terdistribusi dan berjalan di atas *Cluster* yang terdiri dari beberapa komputer yang saling terhubung. *Hadoop* dibuat oleh *Doug Cutting* yang pada awalnya *Hadoop* ini adalah *sub project* dari *Nutch* yang digunakan untuk *search engine*. *Hadoop* bersifat *open source* dan berada di bawah

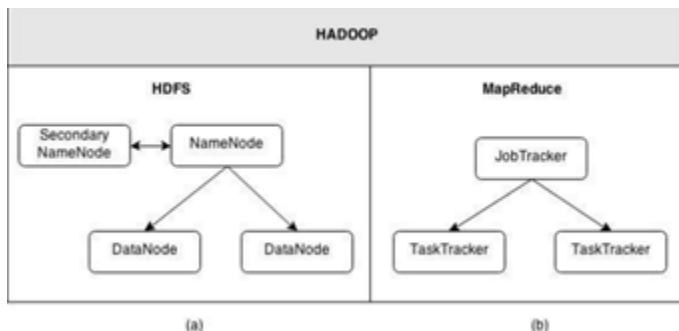
bendera *Apache Software Foundation*.

2.4.1. Arsitektur Hadoop

Hadoop terdiri dari *common Hadoop* yang berguna dalam menyediakan akses ke dalam file system yang didukung oleh *Hadoop*. *Common Hadoop* ini berisi paket yang diperlukan oleh *JAR file*, skrip yang dibutuhkan untuk memulai *Hadoop* dan dokumentasi pekerjaan yang telah dilakukan oleh *Hadoop*.

Berdasarkan dua inti dari *Hadoop* adalah terdiri dari:

- *Hadoop Distributed File System (HDFS)* Untuk data yang terdistribusi.
- *MapReduce Framework* untuk aplikasi dan programming yang terdistribusi.



Gambar 2.5 inti *hadoop* dan komponen dari *hadoop*

Gambar 2.5 menggambarkan bagian inti *Hadoop* yang terdiri dari *HDFS* dan *MapReduce*. Pada Gambar 2.5 (a) menggambarkan komponen dari *HDFS* yang terdiri dari *NameNode*, *DataNode*, dan *Secondary NameNode* dan Gambar 2.4 (b) menggambarkan komponen dari *MapReduce* yang terdiri dari *JobTracker* dan *TaskTracker*. Sebuah *cluster* kecil pada *Hadoop* dapat terdiri dari satu *master node* dan

beberapa *slave node*. *Master node* ini terdiri dari *NameNode* dan *JobTracker*, sedangkan *slave node* terdiri dari *DataNode* dan *TaskTracker*. *Hadoop* membutuhkan *JRE 1.6* atau *JRE* dengan versi yang lebih tinggi. Dalam menjalankan dan menghentikan sistem pada *Hadoop* dibutuhkan *ssh* yang harus dibentuk antar node pada sebuah *cluster* (3).

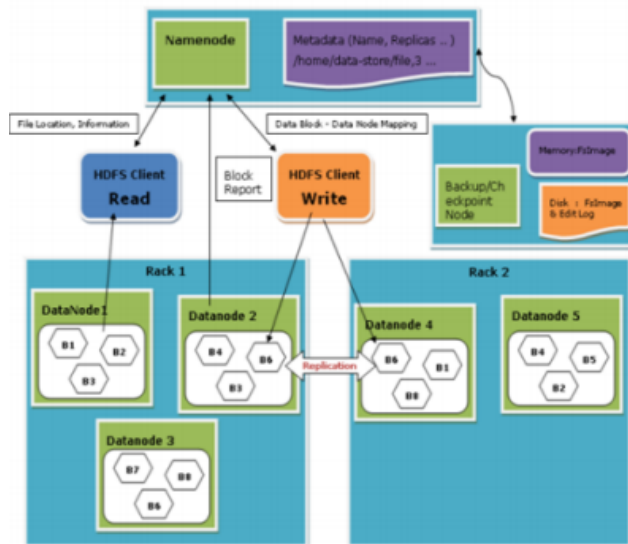
2.4.2. Kelebihan *Hadoop*

Komputasi terdistribusi merupakan bidang yang sangat beragam dan luas, namun *Hadoop* memiliki beberapa kelebihan yang dapat membedakannya dengan yang lain, berdasarkan (2) kelebihan *Hadoop* adalah sebagai berikut:

- Mudah untuk di akses *Hadoop* dapat berjalan pada jumlah *cluster* yang besar ataupun pada layanan komputasi awan seperti *Amazon Elastic Compute Cloud (EC2)*.
- Stabil
Hadoop sangat baik dalam menangani sebuah masalah yang muncul ketika sedang memproses sebuah pekerjaan, hal ini dikarenakan dari awalnya *Hadoop* memang ditunjukkan untuk di jalankan pada komuditas perangkat keras.
- Memiliki skala yang besar
Hadoop memiliki jangkauan skala yang besar, sehingga dapat *menghandle* ketika adanya penambahan jumlah node dalam sebuah *cluster*.
- Mudah digunakan
Hadoop sangat mudah dijalankan dan digunakan pada *single node* maupun *multi node*.

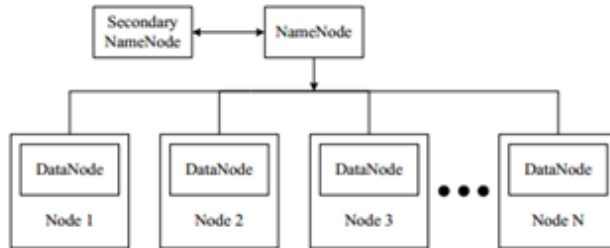
2.4.3. Hadoop Distributed File System (HDFS)

HDFS adalah *distributed file* sistem berbasis *Java* yang menyimpan file dalam jumlah yang besar dan disimpan secara terdistribusi di dalam banyak komputer yang saling berhubungan (2). File sistem ini membutuhkan server induk yang dinamakan *Namenode* yang berfungsi untuk menyimpan *Metadata* dari data yang ada di dalam *HDFS*. *Namenode* juga melakukan pemecahan (*splitting*) file menjadi *block file* dan mendistribusikannya pada komputerkomputer dalam *Cluster*. Hal ini bertujuan untuk replikasi dan *fault tolerance*. *Namenode* juga berfungsi sebagai antar muka yang memberikan informasi melalui *Metadata* terhadap file yang tersimpan agar data *block file* tersebut terlihat seperti file yang sesungguhnya. Sedangkan block data disimpan di dalam komputer-komputer yang dinamakan *Datanode* yang merupakan *slave* dari *HDFS*. *Datanode* dapat berkomunikasi satu sama lain untuk menjaga konsistensi data dan memastikan proses replikasi data berjalan dengan baik.



Gambar 2.6 Arsitektur *HDFS*

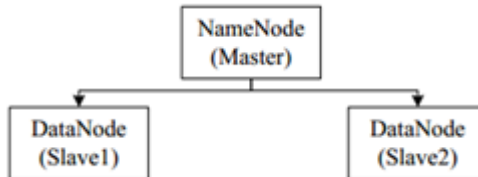
Suatu arsitektur *HDFS* terdiri dari satu *Namenode* yang menyimpan *Metadata* dari file dan satu atau banyak *Datanode* yang menyimpan blok-blok file dan hasil replikasi blok file sebagaimana terlihat pada gambar 2.7.



Gambar 2.7 komponen HDFS

a) *NameNode*

NameNode terdapat pada komputer yang bertindak sebagai *master* yang mengkoordinasi *DataNode* untuk melakukan beberapa tugas (jobs) [5]. *NameNode* ini adalah pusat dari sistem berkas pada *HDFS*. Gambaran *NameNode* yang berada pada master sebagai pusat sistem berkas HDFS dapat dilihat pada Gambar 2.3.

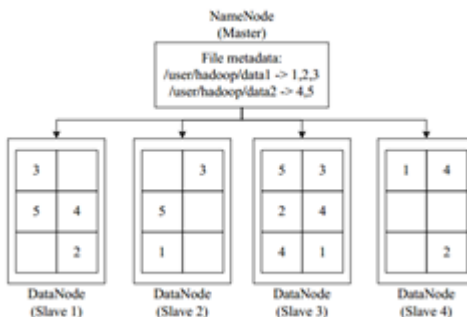


Gambar 2.8 *namenode* pada *HDFS*

NameNode membuat sistem direktori dari semua file yang ada di dalam sistem dan dapat mengetahui bagaimana file tersebut di pecah-pecah menjadi beberapa *blocks* data serta mengetahui nodes yang menyimpan *blocks* data tersebut (2).

b) *DataNode*

Berdasarkan *DataNode* adalah salah satu komponen dari *HDFS* yang berfungsi untuk menyimpan dan mengambil kembali data pada slave node pada setiap permintaan yang dilakukan oleh *NameNode*. *DataNode* berada pada setiap slave node pada sebuah cluster yang telah dibuat. *DataNode* juga berfungsi untuk membaca dan menulis block pada *HDFS* ke file yang sebenarnya pada local file system. Sebagai contoh apabila user ingin membaca atau menulis file ke *HDFS*, file tersebut akan dipecah menjadi beberapa *block*, kemudian *NameNode* akan memberitahu dimana *blocks* tersebut berada sehingga *DataNode* dapat membaca dan menulis *blocks* tersebut ke file yang sebenarnya pada *file system* (2).



Gambar 2.9 interaksi antara *namenode* dan *datanode*

Pada Gambar 2.9 terlihat bahwa *NameNode* menjaga jalur dari *file metadata* dimana setiap file tersebut adalah sebuah sistem yang dipecah-pecah menjadi beberapa *block* (2). *DataNode* menyimpan *backup* dari pecahan-pecahan *block* tersebut dan secara berkala memberitahu kepada *NameNode* untuk tetap menjaga

jalur dari file metadata. Selama sistem berjalan, *DataNode* terhubung dengan *NameNode* dan melakukan sebuah *handshake*. Berdasarkan *handshake* ini bertujuan untuk melakukan verifikasi terhadap *namespace ID* dan juga *software version* pada sebuah *DataNode*.

Namespace ID adalah sebuah *ID* yang muncul ketika pertama kali melakukan format pada *NameNode*. *Namespace ID* ini disimpan pada semua node yang ada pada sebuah cluster. Jika ada node yang memiliki namespace *ID* yang berbeda maka node tersebut tidak akan dapat bergabung pada sebuah cluster. Tujuan adanya namespace *ID* ini adalah untuk menjaga integritas dari *HDFS*. Software version adalah versi software yang digunakan oleh *Hadoop* [5]. Konsistensi pada software version ini sangat penting, karena jika software version yang digunakan berbeda maka akan menyebabkan file corrupt pada sebuah sistem.

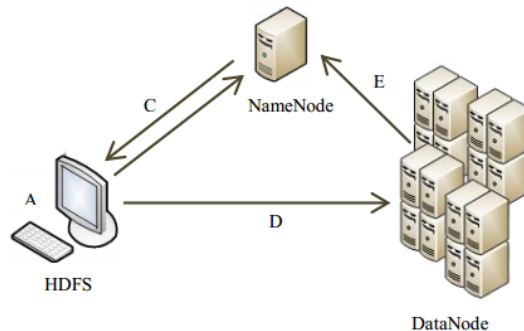
Jika salah satu node memiliki namespace *ID* dan juga software version tidak sama dengan nodes yang lain, maka node tersebut tidak akan terdaftar pada sistem cluster yang ada [5].

c) *Secondary NameNode*

Bedasarkan *Secondary NameNode* adalah *daemon* yang berfungsi melakukan *monitoring* keadaan dari cluster *HDFS*. Sama seperti *NameNode*, pada setiap cluster yang ada terdapat satu *Secondary NameNode*, yang berada pada *master node*. *Secondary NameNode* ini juga berfungsi untuk membantu dalam meminimalkan *down time* dan hilangnya data yang terjadi pada *HDFS* (2). *Secondary NameNode* ini sering menimbulkan kesalahpahaman pengertian bahwa apabila *NameNode* *down* maka akan langsung digantikan oleh *Secondary NameNode* padahal *Secondary NameNode* ini hanya menyimpan informasi terbaru dari struktur direktori pada *NameNode*. Jadi jika terjadi kegagalan yang dilakukan oleh *NameNode* maka dibutuhkan konfigurasi yang dilakukan oleh user untuk menjadikan *Secondary NameNode* sebagai *NameNode* yang utama.

2.4.4. Prosedur Menyimpan Data

Untuk prosedur menyimpan data harus ada sebuah komputer client yang terhubung dengan sebuah Hadoop cluster.



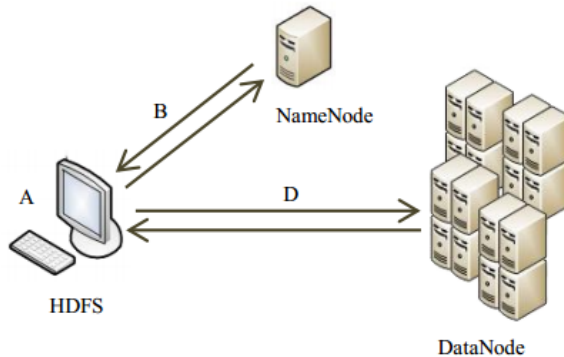
Gambar 2.10 arsitektur penyimpanan data

Langkah-Langkah prosedur menyimpan data sebagai berikut:

- *User* memasukan perintah masukan pada komputer client
- Komputer client berkomunikasi dengan *NameNode* memberitahu bahwa ada data yang akan disimpan dan menanyakan lokasi blok-blok tempat menyimpan data.
- Komputer client mendapat jawaban dari *NameNode* berupa lokasi blok-blok untuk penyimpanan data.
- Komputer client langsung berkomunikasi dengan *DataNode* untuk memasukan data pada lokasi blok-blok yang sudah diatur *NameNode*. Data sudah otomatis terbelah-belah sesuai dengan ukuran yang di setting sehingga dapat langsung menempati blok-blok yang sudah ditentukan.
- *DataNode* memberikan laporan kepada *NameNode* bahwa data-data telah masuk dan menempati blok-blok yang sudah ditentukan oleh *NameNode*.

2.4.5. Prosedur Membaca Data

Untuk prosedur membaca data harus ada sebuah komputer client yang terhubung dengan sebuah Hadoop cluster.



Gambar 2.11 arsitektur membaca data

Langkah-Langkah prosedur membaca data sebagai berikut:

- User memasukan perintah untuk mengambil data pada komputer client.
- Komputer client berkomunikasi dengan *NameNode* untuk menanyakan alamat *DataNode* penyimpan data yang diinginkan.
- Komputer client mendapat jawaban dari *NameNode* berupa lokasi blok-blok tempat penyimpanan data yang diinginkan.
- Komputer client secara langsung berhubungan dengan *DataNode* untuk mengakses lokasi blok-blok tempat penyimpanan data yang diinginkan.
- *DataNode* akan memberikan data yang diinginkan dan data secara otomatis akan ditampilkan pada layar komputer client.

2.4.6. Hadoop Mapreduce

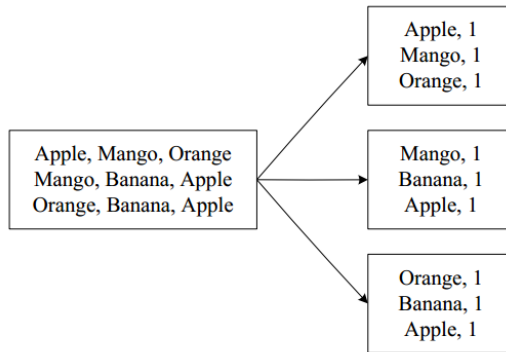
Di *Hadoop MapReduce engine* ini terdiri dari satu *Jobtracker* dan satu atau banyak *Tasktracker*. *Jobtracker* adalah *Mapreduce master*. *Jobtracker* meminta kepada *Namenode* tentang lokasi dari blok data sekaligus akan mendistribusikan blok data tersebut ke *Tasktracker* yang lokasinya paling dekat dengan data. *Namenode* menerima input, kemudian input tersebut dipecah menjadi beberapa *sub-problem* yang kemudian didistribusikan ke

Datanode ini akan memproses *sub-problem* yang diterimanya, setelah *sub-problem* tersebut sudah diselesaikan maka akan dikembalikan ke-*Namenode*. *Namenode* menerima jawaban dari semua *sub-problem* dari banyak *Datanode*, kemudian menggabungkan jawaban-jawaban tersebut menjadi satu jawaban untuk mendapatkan penyelesaian dari permasalahan utama. Keuntungan dari *Mapreduce* ini adalah proses *map* dan *reduce* dijalankan secara *terdistribusi*. Dalam setiap proses *mapping* bersifat *independent*, sehingga proses ini dapat dijalankan secara *simultan* dan *paralel*. Demikian pula dengan proses *reduce* dapat dilakukan secara paralel di waktu yang sama, selama output dari operasi *mapping* mengirimkan *key-value* yang sesuai dengan proses *reducernya*.

2.4.7. Konsep Dasar MapReduce

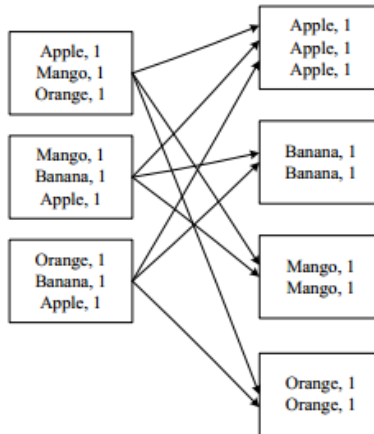
Hadoop menyediakan dua jenis slot untuk melakukan *MapReduce* yaitu *slot map* dan *slot reduce*. Secara default Hadoop telah menentukan jumlah *slot map* dan *slot reduce* untuk setiap node yaitu dua *slot map* dan satu *slot reduce*. Pada saat memproses data, *Hadoop* terlebih dahulu melakukan proses *mapping* pada task yang terdapat pada slot map sampai selesai kemudian dilanjutkan dengan proses *reduce* pada *slot reduce*. Proses *mapping*: pertama *WordCount* menginput *file plaintext* yang tersimpan pada direktori *HDFS*. Kemudian *WordCount* akan membagi *file plaintext* tersebut menjadi beberapa bagian yang berisikan kata yang muncul pada file input dan nilai 1 pada setiap kata yang

ada. Gambaran pada saat *WordCount* melakukan proses *mapping* ini dapat dilihat pada Gambar 2.5.



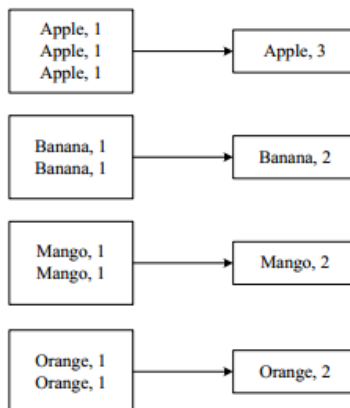
Gambar 2.12 proses *mapping*

Pada Gambar 2.12 terlihat sebuah file input yang berisikan kata-kata yang dibagi menjadi beberapa bagian yang berisikan kata dan nilai 1 pada setiap kata yang ada. Setelah proses *mapping* ini selesai maka akan dilanjutkan dengan proses *shuffle* yang berfungsi untuk menggabungkan kata-kata yang sama untuk mempersiapkan proses *reducing*. Gambaran dari proses *shuffle* ini dapat dilihat pada Gambar 2.13.



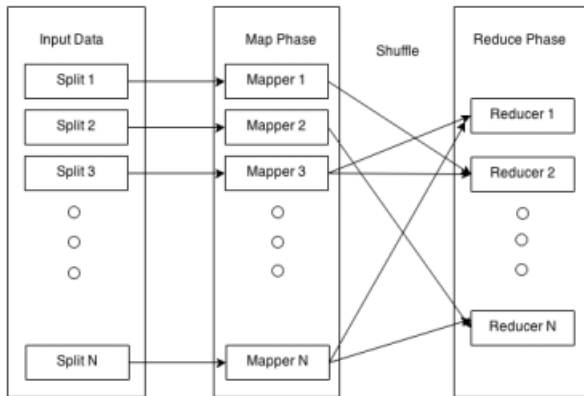
Gambar 2.13 proses shuffle

Proses reducing: pada proses ini terjadi penggabungan kata yang sama setelah proses *shuffle* dan menghitung jumlah kata yang sama tersebut. Gambaran proses reducing ini dapat dilihat pada Gambar 2.14.



Gambar 2.14 proses reducing

Gambaran proses *MapReduce* yang terjadi secara keseluruhan dapat dilihat pada Gambar 2.15.



Gambar 2.15 keseluruhan proses *mapreduce*

Gambar 2.15 menggambarkan sebuah data yang dibagi menjadi beberapa bagian yang kemudian pada setiap bagian dilakukan proses mapping, dan setelah proses mapping selesai bagian-bagian data tersebut di acak (*shuffle*) untuk melalui proses *reducing*.

Keuntungan dari *MapReduce* adalah proses map dan reduce yang dapat diterapkan secara terdistribusi. Pada setiap proses mapping dan proses reducing bersifat independent sehingga proses dapat dijalankan secara paralel pada waktu yang sama, selama output dari proses mapping mengirimkan key value yang sesuai dengan proses reducingnya. Didalam *Hadoop*, *MapReduce* ini terdiri dari satu *JobTracker* dan beberapa *TaskTracker* pada sebuah *cluster*.

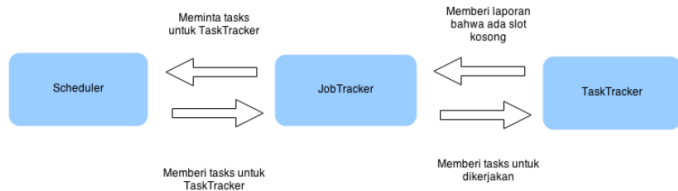
2.4.8. Komponen *MapReduce*

MapReduce yang terdapat pada *Hadoop* memiliki 2 komponen utama penting yaitu:

a. *JobTracker*

JobTracker adalah sebuah komponen dari *MapReduce* yang berfungsi untuk memecah pekerjaan (job) yang diberikan

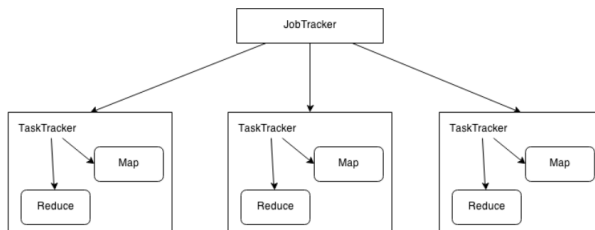
ke *HDFS* menjadi beberapa tasks yang lebih kecil berdasarkan jumlah slave yang ada (2). Setelah pekerjaan (job) tersebut dipecah-pecah menjadi beberapa tasks, *JobTracker* akan memberikan pekerjaan-pekerjaan tersebut kepada setiap slave node yang terdapat di dalam cluster tersebut. *JobTracker* secara berkala mengkoordinasi semua tasks yang diberikan kepada *TaskTracker* menggunakan scheduler task (pengatur tugas), kemudian *TaskTracker* akan mengerjakan tasks tersebut. Setelah *TaskTracker* menyelesaikan task yang diberikan, maka *TaskTracker* akan meminta task yang baru kepada *JobTracker*. Gambaran kerja dari *JobTracker* dapat dilihat pada Gambar 2.16.



Gambar 2.16 kerja *jobtracker*

b. *TaskTracker*

TasTracker adalah sebuah daemon yang berfungsi untuk menerima tugas (task) yang diberikan oleh *JobTracker* dan kemudian mengerjakan task tersebut ke dalam *Java Virtual Machine (JVM)* yang terpisah, dengan menjalankan task tersebut ke dalam *Java Virtual Machine (JVM)* yang terpisah, maka hal ini akan mengurangi beban pekerjaan yang dilakukan secara paralel yang diberikan oleh *JobTracker*.

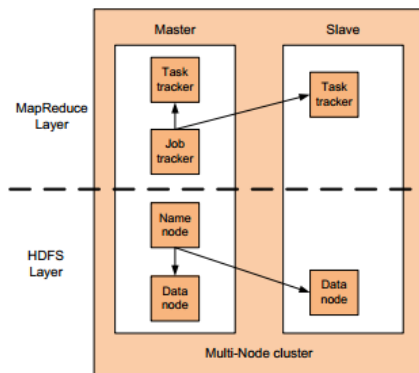


Gambar 2.17 kerja *tasktracker*

Gambar 2.17 menggambarkan bagaimana sebuah *JobTracker* yang berkomunikasi dengan beberapa *TaskTracker* yang melakukan proses *MapReduce*. Secara konstan *TaskTracker* terus berkomunikasi dengan *JobTracker* dengan memberikan laporan setiap proses yang telah dilakukan. Jika *JobTracker* gagal menerima hasil task yang dikerjakan oleh *TaskTracker*, maka *JobTracker* akan mengirimkan kembali task tersebut kepada nodes lain pada cluster tersebut untuk dikerjakan ulang.

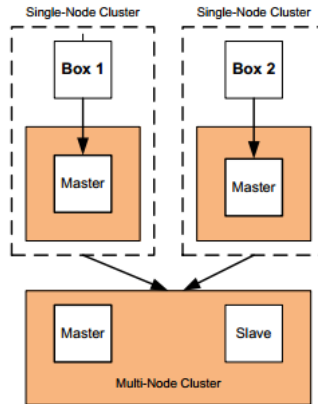
2.4.9. Hadoop Single-Node Cluster dan Multi-Node Cluster

Hadoop single-node diimplementasikan pada satu mesin. Mesin tersebut didesain menjadi master tetapi dapat bekerja juga sebagai slave dan semua proses distribusi dilakukan dalam satu mesin tersebut. Seperti pada Gambar 2.5, dalam *Hadoop* terbagi menjadi dua layer yaitu layer HDFS yang menjalankan *Namenode* dan *Datanode* dan layer Mapreduce yang menjalankan *Jobtracker* dan *Tasktracker*. Kedua layer ini sangat penting terutama *Namenode* dan *Jobtracker*, karena apabila dua bagian ini tidak berjalan maka kerja *HDFS* dan *Mapreduce* tidak bisa dijalankan. Pada mesin single node, *Datanode* dan *Tasktracker* hanya ada satu, jika memiliki mesin yang banyak maka kedua bagian ini akan terbentuk pada setiap mesin (multi-node).



Gambar 2.18 *multinode cluster*

Gambar 2.5 Ilustrasi layer *HDFS* dan *Mapreduce* dalam *Hadoop*



Gambar 2.19 Sistem *Multi-Node Cluster*

Identifikasi pada sebuah Cluster *Hadoop multi-node* menggunakan dua mesin atau lebih, adalah satu untuk master dan satu atau yang lain sebagai *slave*. Untuk mengkonfigurasi mesin tersebut adalah berupa mesin-mesin single-node yang akan digabung menjadi satu multi-node dimana satu mesin akan didesain menjadi master tapi dapat bekerja juga sebagai *slave*, sedangkan komputer yang lain akan menjadi *slave*. Pada gambar 2.19 merupakan sistem *Multi-Node Cluster* yang menggunakan dua mesin.

2.4.10. Aliran Data *Hadoop MapReduce*

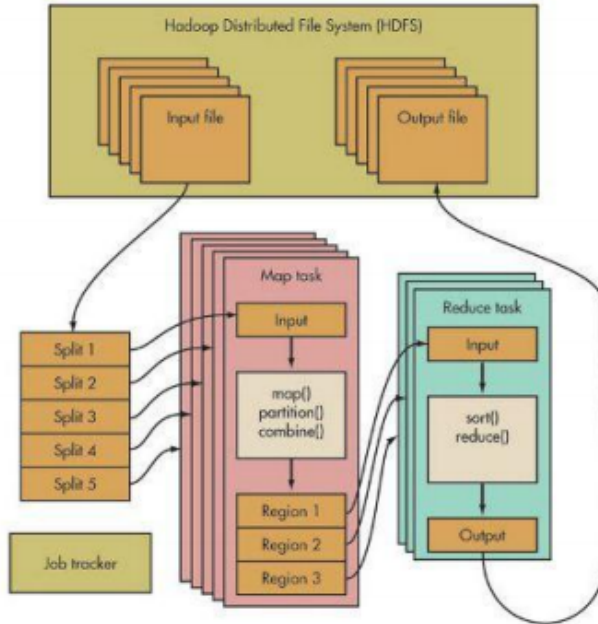
Input Mapreduce berasal dari file-file yang akan diproses dalam cluster *HDFS*. File-file ini akan didistribusikan pada semua node yang ada. Jika menjalankan sebuah program Mapreduce maka akan menjalankan mapping task pada semua node. Semua mapping task adalah sama dan setiap mapping tidak memiliki identitas tertentu dalam mengeksekusi task, oleh karenanya semua mapping dapat memproses semua input file yang manapun. Proses mapping tersebut tersebar dalam semua node yang ada dalam Cluster. Pasangan *intermediate (key, value)* akan didistribusikan untuk

mendapatkan semua value dengan key yang sama saat semua proses mapping selesai dilakukan. Semua pasangan intermediate (key, value) dengan key yang sama tersebut akan diproses oleh satu reducer. Reducing task juga tersebar pada semua node yang terdapat dalam Cluster sebagaimana mapping task. Masing-masing mapping task mengabaikan mapping task yang lain dan tidak saling bertukar informasi mengenai proses mapping, demikian juga dengan reducing juga mengabaikan reducing task yang lain dan tidak bertukar informasi mengenai proses reducing. Salah satu kelebihan *Hadoop Mapreduce*, user tidak perlu memberikan informasi mekanisme transfer data dari node yang satu ke node yang lain karena telah dilakukan oleh *Hadoop Mapreduce* itu sendiri secara implisit menggunakan key dan value sebagai informasi. Di dalam *Mapreduce* terdapat beberapa proses yang terjadi hingga suatu output dapat dihasilkan sesuai dengan yang diinginkan oleh user.

Mapreduce job adalah sebuah unit kerja yang ingin diimplementasikan. *Mapreduce job* ini terdiri dari input data, program Mapreduce dan informasi tentang konfigurasi. Hadoop menjalankan job tersebut dengan membaginya menjadi task-task. Task-task tersebut dibagi lagi menjadi dua yaitu map task dan reduce task. Ada dua tipe node yang akan mengontrol proses eksekusi job tersebut yaitu sebuah Jobtracker dan sejumlah *Tasktracker*. *Jobtracker* mengkoordinasi semua job yang akan dijalankan di sistem dengan menjadwalkan eksekusi task-task tersebut pada *Tasktracker*. *Tasktracker* menjalankan task dan memberikan laporan progres kepada *Jobtracker*.

Jobtracker menyimpan record dari semua progress dari masing-masing job. Jika sebuah task gagal dieksekusi maka *Jobtracker* akan mengatur kembali eksekusi task tersebut pada *Tasktracker* yang lain. *Hadoop* membagi input untuk *Mapreduce* job menjadi bagian-bagian yang memiliki besaran tetap yang disebut input split.

Hadoop membuat satu map task untuk setiap input split. Input split ini dijalankan oleh fungsi map yang telah dibuat oleh user untuk setiap record dalam input split.



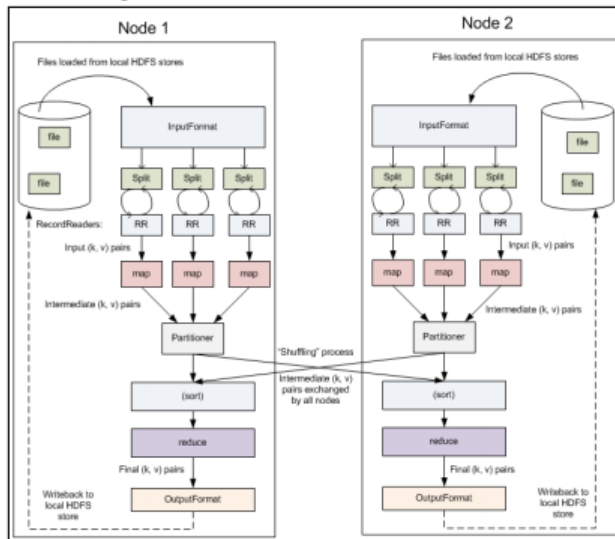
Gambar 2.20 Proses aliran data Hadoop Mapreduce

Seperti pada Gambar 2.20, Aliran data dari suatu proses *Mapreduce* dimulai dari suatu input data. Input data tersebut kemudian dipecah menjadi beberapa blok bagian. Data tersebut direplikasi dan disimpan dalam node-node (*Datanode*) sedangkan *Metadata* (nama file, jumlah replikasi, jumlah blok file) dari input tersebut tersimpan dalam

satu *Namenode*. Untuk melakukan proses mapping, input tersebut dipanggil dari *Datanode* kemudian dilakukan proses splitting. Dari file input yang telah di-split tersebut akan dilakukan proses mapping. Proses mapping akan menghasilkan suatu output pasangan intermediate key dan value, dari semua key yang sama akan dikelompokkan menjadi satu. Pasangan intermediate key-value tersebut kemudian di-*shuffle* untuk dilakukan

proses reducing yang tersebar di sejumlah komputer. Dari proses reducing tersebut akan dihasilkan suatu output

yang diinginkan. Berikut proses detail yang terjadi dalam proses *Mapreduce* terangkum dalam gambar 2.21.



Gambar 2.21 Ilustrasi secara detail terhadap proses *Mapreduce*

Sebagaimana pada gambar 2.21 bagian-bagian aliran data *Mapreduce* dapat dijelaskan sebagai berikut.

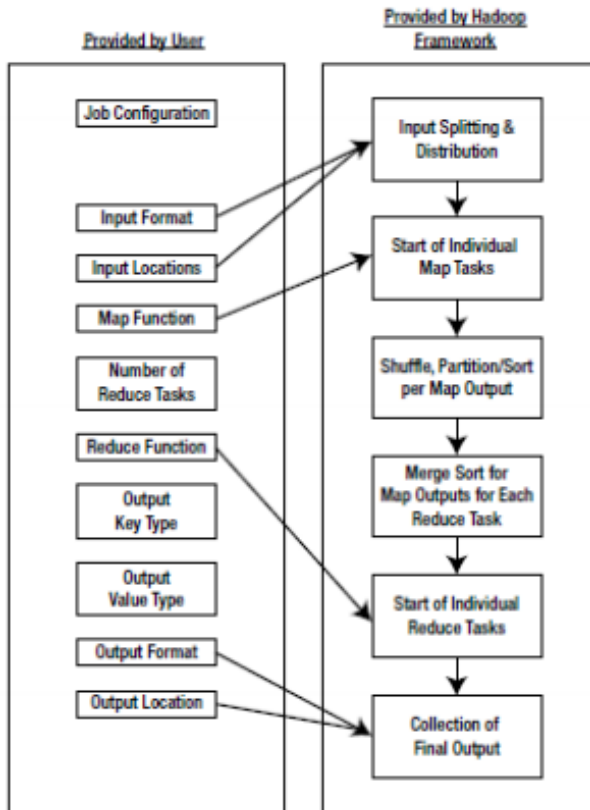
- *Input File*
Input file adalah tempat dimana data sebagai input dari *Mapreduce* pertama kali disimpan. Input file ini tersimpan dalam *HDFS*.
- *Input format*
adalah cara bagaimana input tersebut akan dipecah (split) dan dibaca. Dalam *Hadoop InputFormat* memiliki class *abstract* yang dinamakan *FileInputFormat*. Saat mengeksekusi sebuah job, *FileInputFormat* menyediakan path yang berisi file-file yang akan dibaca. *FileInputFormat* akan membagi file-file ini menjadi satu atau lebih untuk masing-masing *InputSplit*. *InputFormat* membaca *record* melalui implementasi dari *RecordReader*. *InputFormat* mendefinisikan daftar task yang akan dieksekusi pada tahap

mapping. Setiap task sesuai untuk sebuah satu input split. Standar InputFormat yang telah disediakan oleh Hadoop sebagai berikut:

- *Text Input Format*
TextInputFormat adalah default dari *InputFormat*, membaca baris-baris dari file teks. *TextInputFormat* sangat berguna untuk data yang tidak terformat seperti *logfile*. *TextInputFormat* mengambil nilai *byte (byte offset of the line)* dari setiap baris sebagai key dan isi dari baris tersebut sebagai value.
- *Key Value Input Format*
KeyInputFormat mem-parsing baris-baris menjadi pasangan key dan value berdasarkan karakter tab. Key yang diambil adalah semua karakter sampai ditemukannya karakter tab pada suatu baris dan sisanya adalah sebagai value.
- *Sequence File Input Format*
SequenceFileInputFormat membaca file biner yang spesifik untuk *Hadoop*. Key dan value pada *SequenceFileInputFormat* ini ditentukan oleh user.
- *Input Split*
Input split mendeskripsikan sebuah unit kerja yang meliputi sebuah task dalam sebuah program *Mapreduce*. File yang akan diproses dalam task sebelumnya melalui proses pemecahan (splitting) file menjadi beberapa bagian. Besaran pemecahan file ini mencapai 64 MB, nilai ini sama dengan besaran block data dalam *HDFS*.
- *Record Reader*
Record Reader mengatur bagaimana cara mengakses data, menampung data dari sumber, dan mengubah data-data tersebut menjadi pasangan *<key, value>* yang dapat dibaca oleh mapper. Instance dari *RecordReader* disediakan oleh *InputFormat*.
- *Mapper*
Mapper sebagai tahap pertama dari *Mapreduce* yang didefinisikan oleh user. Saat diberikan sebuah key dan sebuah value maka *method map()* akan memberikan pasangan *<key, value>* yang akan diteruskan kepada *Reducer*.

- *Partition dan Shuffle*
Setelah map task pertama selesai dieksekusi, ada node-node yang masih memproses map task yang lain, disamping itu pula terjadi pertukaran intermediate output dari map task yang dibutuhkan oleh reducer. Proses pertukaran output map task dan diteruskannya output map task tersebut kepada reducer dinamakan proses *shuffle*. Ada sebagian dataset yang telah dieksekusi melalui map task namun ada juga yang masih dalam proses mapping oleh map task. Bagian data set yang telah melalui map task tersebut menghasilkan intermediate key value yang akan menjadi *input* dari *reduce tasks*. Sebagian data set inilah yang dinamakan *partition*.
- *Sort*
Setiap *reduce task* bertanggung jawab melakukan reducing terhadap value yang sesuai dengan *intermediate key*. Kumpulan *intermediate key* pada *single-node* secara otomatis akan di sorting oleh *Hadoop* sebelum akhirnya dilakukan proses reduce.
- *Reducer*
Reducer adalah tahap kedua dari *Mapreduce* yang didefinisikan oleh user. *Instance reducer* akan dibuat untuk setiap *reduce task*, *instance reducer* ini sesuai dengan program yang dibuat oleh user. Untuk setiap key yang ada *method reduce()* akan dipanggil sekali.
- *Output Format*
Outputformat berfungsi seperti *InputFormat* hanya saja *OutputFormat* berfungsi memberikan output file. *Instance* dari *OutputFormat* disediakan oleh *Hadoop* akan memberikan output file pada lokal disk *HDFS*.
- *RecordWriter*
Sebagaimana *RecordReader*, *RecordWriter* berguna untuk menuliskan record ke file sebagaimana diinstruksikan oleh *OutputFormat*. User perlu mengkonfigurasi dan menentukan *Mapreduce job* yang akan dieksekusi, menentukan input, menentukan lokasi input. *HDFS* akan

mendistribusikan *job* tersebut kepada *Tasktracker* dan memecah *job* tersebut ke dalam sejumlah *map task*, *shuffle*, *sort* dan *reduce task*. HDFS akan menempatkan output pada output directory dan memberitahu user jika *job* telah selesai dieksekusi.



Gambar 2.22 Identifikasi job configuration dalam Mapreduce

Pada gambar 2.22 di atas memberitahukan bagian-bagian *Mapreduce job* yang harus dispesifikasikan dan dibuat oleh user dan yang telah disediakan oleh *Hadoop*. Berikut adalah tabel penjelasan mengenai gambar tersebut.

Tabel 2.1 *Identifikasi job configuration* antara *user* dan *hadoop* framework

<i>Configuration</i>	<i>Handled By</i>
Konfigurasi <i>Job</i>	<i>User</i>
<i>Input split</i> dan <i>distribution</i>	<i>Hadoop Framework</i>
<i>Map</i> and <i>Reduce Function</i> < <i>key</i> , <i>value</i> >	<i>User</i>
Proses <i>shuffle</i> , <i>partition</i> , dan <i>sort</i>	<i>Hadoop Framework</i>
Lokasi <i>distributed file system</i> untuk <i>input job</i>	<i>User</i>
Lokasi <i>distributed file system</i> untuk <i>output job</i>	<i>User</i>
<i>Input format</i> dan <i>output format</i>	<i>User</i>
<i>Class fungsi Mapper</i> dan <i>Reducer</i>	<i>User</i>
<i>File jar</i> yang berisi fungsi <i>map</i> dan <i>reduce</i>	<i>User</i>
Informasi <i>Job</i>	<i>Hadoop Framework</i>

(4).

2.5. *Apache Hive*

Apache hive adalah tool selain untuk membentuk program *Map Reduce*. *Apache Hive* pertama kali dikembangkan oleh Facebook untuk melakukan *data warehouse* pada *cluster Hadoop* mereka yang sangat banyak. Selanjutnya *Hive* disumbangkan ke *Apache Foundation* untuk dikembangkan oleh komunitas open source. *Hive* lebih ditujukan untuk proses data warehouse diatas *HDFS*.

pada *Apache Hive* proses *Map Reduce* dituliskan dengan gaya yang sangat mirip dengan *SQL* yang pada umumnya ada di *RDBMS*.

Contoh *Script Hive* yang menghitung kemunculan huruf

```
CREATE TABLE word_text(word STRING)
COMMENT 'This is the word table'
ROW FORMAT DELIMITED FIELDS TERMINATED BY ' '
LINES TERMINATED BY '\n';
LOAD DATA INPATH 'hdfs:/user/hue/word_count_text.txt'
INTO TABLE word_text;
SELECT word, count(*) as count FROM word_text GROUP BY
word;
```

Seperti terlihat pada script *Hive* diatas, tampak sekali kemiripan antara Script *Hive* dengan *SQL*. *Script Hive* diatas terdiri dari tiga statement yang masing-masing diakhiri dengan semi-colon (;). Bagian pertama adalah membentuk tabel yang akan menampung semua kata. Bagian kedua adalah menarik semua kata dari file di *HDFS* ke dalam tabel. Sedangkan bagian ketiga adalah

SQL query yang bisa dilakukan terhadap data yang sudah dimasukkan ke tabel. Kita bisa melakukan berbagai query seperti *SQL* pada tabel yang sudah kita bentuk sehingga tidak hanya terbatas pada satu query saja. Output dari query tersebut bisa langsung ke layar, bisa ke file atau ke sistem eksternal menggunakan tool tambahan seperti *Apache Thrift* atau *Apache Avro*.

Hive cukup modular sehingga bisa di gabungkan dengan banyak tool lain seperti *Spring*, *Apache Thrift* dan *Apache Avro*. *Pig* (5).

2.6. *Apache Mahout*

Apache Mahout adalah sebuah *library machine learning* untuk melakukan *data mining* dengan menggunakan beberapa metode seperti *clasification*, *clustering* dsb. Library ini ditulis dengan menggunakan bahasa *Java* dan bersifat *opensource*. Dan dapat digunakan untuk bersama dengan *apache hadoop*. (6).

2.7. *Matagaruda*

Mata Garuda adalah *IDS* dengan basis *snort IDS*. Dengan *IDS* ini maka lalu lintas jaringan internet di indonesia bisa dipantau. Dan dengan Mata Garuda bisa mengenali serangan yang terjadi misalnya *DDOS* dan menampilkanya melalui peta dunia (7).

2.8. OAuth2

OAuth2 (Open Authorization) adalah sebuah kerangka kerja yang berguna agar aplikasi pihak ketiga bisa mengakses sebuah *resource* atau *HTTP service* yang telah dibatasi hak aksesnya (8).

OAuth 2 adalah pengembangan dari protokol OAuth yang asal mulanya diciptakan pada akhir tahun 2006. OAuth 2 lebih menekankan pada kemudahan client sebagai pemilik dan pengembang aplikasi dengan memberikan otorisasi khusus di berbagai aplikasi (9).

Terdapat 4 peran utama dalam mekanisme kerja OAuth 2 yakni (8):

1. *Resource Server*

Resource server (Sumber Daya Server) yang digunakan oleh pengguna yang memiliki *API (Application Programming Interface)* dan dilindungi oleh *OAuth2*. *Resource server* merupakan penerbit API yang memegang dan memiliki kekuasaan pengaturan data seperti foto, video, kontak, atau kalender.

2. *Resource Owner*

Memposisikan diri sebagai pemilik sumber daya (*Resource Owner*), yang merupakan pemilik dari aplikasi. Pemilik sumber daya memiliki kemampuan untuk mengakses sumber daya server dengan aplikasi yang sudah tersedia.

3. *Client*

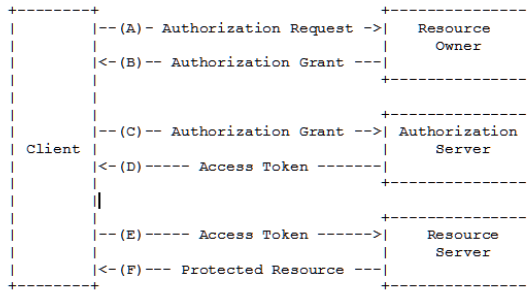
Sebuah aplikasi yang membuat permintaan API pada *Resource Server* yang telah diproteksi untuk kepentingan pemilik *Resource Owner* dengan melakukan otorisasi.

4. *Authorization Server*

Authorization Server (Otorisasi Server) mendapat persetujuan dari pemilik sumber daya (*Resource Owner*) dengan melakukan dan memberikan akses token kepada client

untuk mengakses sumber daya yang diproteksi yang sudah tersedia pada *Resource Server*.

Berikut adalah protokol *flow* dari *oauth2*



Gambar 2.23 Mekanisme Kinerja OAuth 2

1. *Client* melakukan permintaan otorisasi dari *Resource Owner*. Permintaan otorisasi dapat dilakukan langsung menuju *Resource Owner*, atau jika tidak langsung melalui perantara *Authorization Server*.
2. *Client* mendapatkan persetujuan otorisasi yang merupakan *credential* mewakili otorisasi kepemilikan client. Pemberian otorisasi ini tergantung pada metode yang digunakan oleh client dan jenis yang didukung oleh *Authorization Server*.
3. *Client* melakukan permintaan akses token dengan otentikasi kepada *Authorization Server*, client mendapatkan penyajian hibah dan bentuk otorisasi dari *Authorization Server*.
4. Otorisasi Server (*Authorization Server*) melakukan otentikasi kepada client dan memvalidasi pemberian otorisasi kepada client, jika sesuai dan berlaku, otorisasi server membagikan akses token.
5. Client melakukan permintaan sumber daya yang sudah diproteksi dari *Resource Server*,

melakukan tindakan otentikasi dengan menghadirkan akses token.

6. *Resource Server* memvalidasi akses token, jika *valid* dan sesuai, maka permintaan terhadap resource akan dilayani.

2.9. Web Service

Web service adalah sebuah software yang dirancang untuk mendukung interoperabilitas interaksi mesin-ke-mesin melalui sebuah jaringan. Web service secara teknis memiliki mekanisme interaksi antar sistem sebagai penunjang interoperabilitas, baik berupa agregasi (pengumpulan) maupun sindikasi (penyatuan). Web service memiliki layanan terbuka untuk kepentingan integrasi data dan kolaborasi informasi yang bisa diakses melalui internet oleh berbagai pihak menggunakan teknologi yang dimiliki oleh masing-masing pengguna.

Alasan menggunakan web service adalah kemudahan dalam penggunaan kembali (reuse) dan berbagi (share) logika yang sama dengan klien yang beragam seperti mobile, desktop, dan aplikasi web. Jangkauan web service yang luas karena web service bergantung pada standar yang terbuka, dapat beroperasi pada platform yang berbeda, serta tidak bergantung pada teknologi eksekusi yang mendasarinya. Semua web service setidaknya menggunakan *HTTP* dan format pertukaran data standar berupa *XML*, *JSON*, atau media lain. Selain itu, web service menggunakan *HTTP* dalam dua cara yang berbeda yaitu sebagai protokol standar untuk menentukan perilaku standar pelayanan serta sebagai media transportasi untuk menyampaikan data (10).

2.10. Rest Web Service

REST web service terdiri dari dua istilah yaitu “*REST*” dan “*web service*”. Menurut Roy Thomas Fielding, *REST (REpresentational State Transfer)* adalah model arsitektur yang pada dasarnya memanfaatkan teknologi dan protokol yang sudah ada seperti *HTTP (Hypertext Transfer Protocol)* dan *XML*. *REST web service* merupakan web service yang dibangun dengan memanfaatkan teknologi dan protokol yang sudah ada. Desain *REST web service* lebih mirip seperti pekerjaan seni daripada

sains, hal ini dikarenakan desain *REST* harus bisa menjawab permasalahan yang dihadapi.

Penggunaan metode-metode *HTTP* dalam *REST* adalah sebagai berikut:

Tabel 2.2 Metode *HTTP* dan Penggunaannya dalam *REST*

Metode	Deskripsi
GET	Mendapatkan (<i>read</i>) sebuah sumber daya (<i>resource</i>) yang diidentifikasi dengan URI (<i>Uniform Resource Identifier</i>)
POST	Mengirimkan sumber daya (<i>resource</i>) ke server. Digunakan untuk membuat (<i>create</i>) sumber daya baru.
PUT	Mengirimkan sumber daya (<i>resource</i>) ke server. Digunakan untuk memasukkan (<i>insert</i>) atau memperbarui (<i>update</i>) sumber daya yang tersimpan.
DELETE	Menghapus (<i>delete</i>) sumber daya (<i>resource</i>) yang diidentifikasi dengan URI.
HEAD	Mendapatkan <i>metadata</i> (<i>response header</i>) dari sumber daya (<i>resource</i>) yang diidentifikasi dengan URI.

Sampai sekarang, *REST best practice* dipilih oleh para pengembang dalam melakukan desain *REST web service* sehingga desain antara yang satu dengan yang lain bisa berbeda-beda tergantung masalah yang dihadapi. Beberapa aturan *best practice* tersebut adalah *REST* menggunakan *URI* sebagai identifier, interaksi menggunakan *HTTP*, serta format balasan harus valid dan konsisten (10)..

2.11. JSON

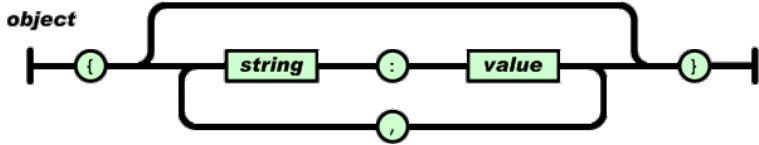
JSON (*JavaScript Object Notation*) adalah format pertukaran data yang ringan, mudah dibaca dan ditulis oleh manusia, serta mudah diterjemahkan dan dibuat (*generate*) oleh komputer. Format ini dibuat berdasarkan bagian dari *Bahasa Pemrograman JavaScript, Standar ECMA-262 Edisi ke-3 - Desember 1999*. JSON merupakan format pertukaran data yang independen atau tidak tergantung dari bahasa pemrograman yang digunakan dan juga bisa digunakan oleh antar bahasa untuk berkomunikasi beberapa contoh bahasa seperti C termasuk C, C++, C#, Java, JavaScript, Perl, Python dll. Oleh karena sifat-sifat tersebut, menjadikan JSON ideal sebagai bahasa pertukaran-data.

JSON terbuat dari dua struktur:

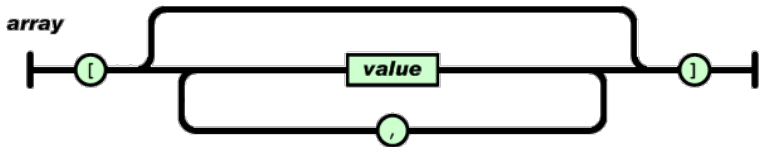
- Kumpulan pasangan nama/nilai. Pada beberapa bahasa, hal ini dinyatakan sebagai objek (*object*), rekaman (*record*), struktur (*struct*), kamus (*dictionary*), tabel hash (*hash table*), daftar berkunci (*keyed list*), atau *associative array*.
- Daftar nilai terurutkan (*an ordered list of values*). Pada kebanyakan bahasa, hal ini dinyatakan sebagai larik (*array*), vektor (*vector*), daftar (*list*), atau urutan (*sequence*).

Struktur-struktur data ini disebut sebagai struktur data universal. Pada dasarnya, semua bahasa pemrograman moderen mendukung struktur data ini dalam bentuk yang sama maupun berlainan. Hal ini pantas disebut demikian karena format data mudah dipertukarkan dengan bahasa-bahasa pemrograman yang juga berdasarkan pada struktur data ini. JSON menggunakan bentuk sebagai berikut:

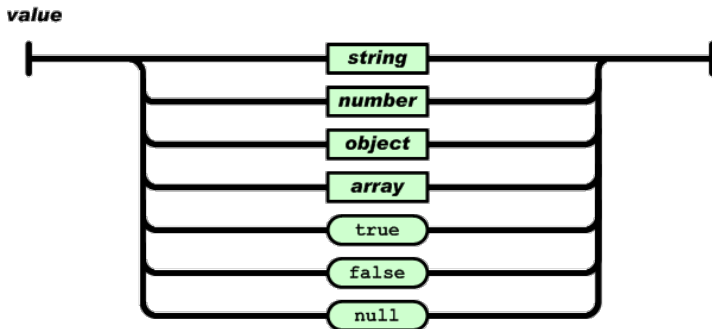
Objek adalah sepasang nama/nilai yang tidak terurutkan. Objek dimulai dengan { (kurung kurawal buka) dan diakhiri dengan } (kurung kurawal tutup). Setiap nama diikuti dengan : (titik dua) dan setiap pasangan nama/nilai dipisahkan oleh , (koma).



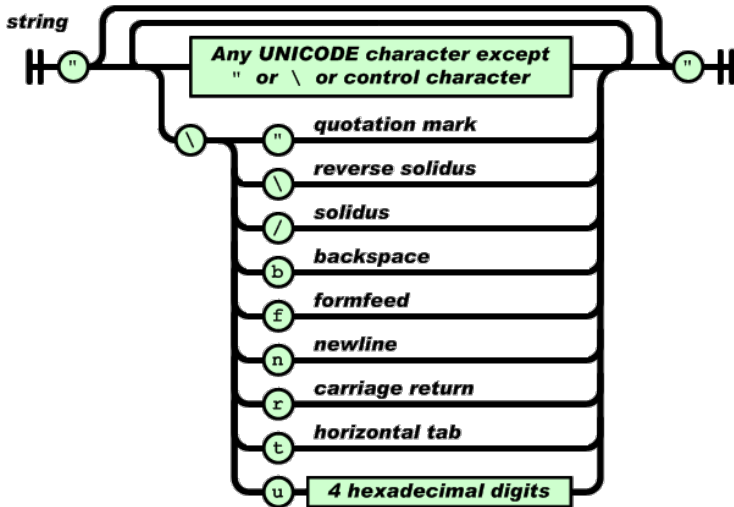
Larik adalah kumpulan nilai yang terurutkan. Larik dimulai dengan [(kurung kotak buka) dan diakhiri dengan] (kurung kotak tutup). Setiap nilai dipisahkan oleh , (koma).



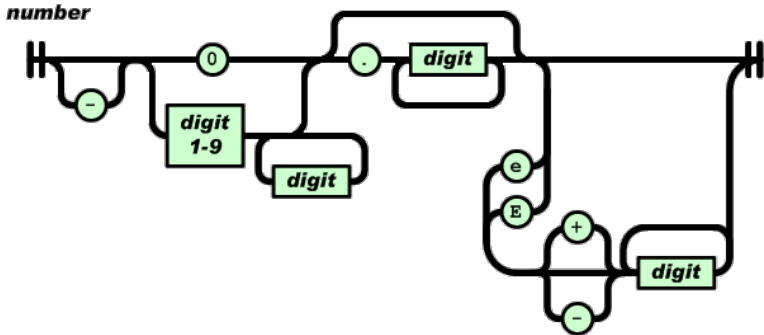
Nilai (value) dapat berupa sebuah **string** dalam tanda kutip ganda, atau *angka*, atau `true` atau `false` atau `null`, atau sebuah *objek* atau sebuah *larik*. Struktur-struktur tersebut dapat disusun bertingkat.



String adalah kumpulan dari nol atau lebih karakter Unicode, yang dibungkus dengan tanda kutip ganda. Di dalam string dapat digunakan *backslash escapes* "\" untuk membentuk karakter khusus. Sebuah karakter mewakili karakter tunggal pada string. String sangat mirip dengan string C atau Java.



Angka adalah sangat mirip dengan angka di C atau Java, kecuali format oktal dan heksadesimal tidak digunakan.



Spasi kosong (*whitespace*) dapat disisipkan di antara pasangan tanda-tanda tersebut, kecuali beberapa detail *encoding* yang secara lengkap dipaparkan oleh bahasa pemrograman yang bersangkutan. (11).

2.12. YAML

YAML adalah format serialisasi data terbaca-manusia yang mengambil konsep dari bahasa-bahasa seperti XML, C, Python, Perl, serta format surat elektronik seperti yang tercantum dalam RFC 2822. YAML pertama kali diusulkan oleh Clark Evans pada tahun 2001 . yang merancang format ini bersama dengan Ingy döt Net dan Oren Ben-Kiki. YAML tersedia bagi beberapa bahasa dan skrip pemrograman dan khususnya digunakan pada bahasa pemrograman ruby meski juga digunakan di bahasa atau framework lain sebagai file konfigurasi. Pada awal pengembangannya, YAML dimaksudkan sebagai singkatan dari "*Yet Another Markup Language*" Dalam perkembangannya, untuk menegaskan tujuannya yang terfokus pada data dan bukan markah dokumen, YAML diubah menjadi singkatan rekursif dari "*YAML Ain't a Markup Language*." (12).

2.13. JAVA

Java adalah bahasa pemrograman yang multi platform dan multi device. Sekali anda menuliskan sebuah program dengan menggunakan Java, anda dapat menjalankannya hampir di semua komputer dan perangkat lain yang support Java, dengan sedikit perubahan atau tanpa perubahan sama sekali dalam kodenya. Aplikasi dengan berbasis Java ini dikompulasikan ke dalam p-code dan bisa dijalankan dengan Java Virtual Machine. Fungsionalitas dari Java ini dapat berjalan dengan platform sistem operasi yang berbeda karena sifatnya yang umum dan non-spesifik.

Slogan Java adalah "Tulis sekali, jalankan di manapun". Sekarang ini Java menjadi sebuah bahasa pemrograman yang populer dan dimanfaatkan secara luas untuk pengembangan perangkat lunak. Kebanyakan perangkat lunak yang menggunakan Java adalah ponsel feature dan ponsel pintar atau smartphone.

Kelebihan dan kekurangan Java
Setelah membahas mengenai pengertian java, selanjutnya kita membahas mengenai kelebihan dan kekurangan java. Kelebihan Java yang pertama tentu saja multiplatform. Java dapat dijalankan dalam beberapa platform komputer dan sistem operasi yang berbeda. Hal ini sesuai dengan slogannya yang sudah dibahas

sebelumnya. Yang kedua adalah OOP atau Object Oriented Programming. Java memiliki library yang lengkap. Library disini adalah sebuah kumpulan dari program yang disertakan dalam Java. Hal ini akan memudahkan pemrograman menjadi lebih mudah. Kelengkapan library semakin beragam jika ditambah dengan karya komunitas Java.

Setiap hal pasti memiliki kelebihan dan kekurangan. Kekurangan yang dimiliki oleh Java adalah pada satu slogannya, yakni “Tulis sekali dan jalankan dimana saja” ternyata tidak sepenuhnya benar. Beberapa hal harus disesuaikan jika dijalankan pada platform yang berbeda. Misalnya untuk *J2SE* dengan platform *SWT-AWT bridge* tidak dapat berfungsi di *Mac OS X*. Kekurangan lainnya adalah kemudahan aplikasi Java didekompilasi. Dekompilasi adalah suatu proses membalikkan sebuah aplikasi menjadi kode sumbernya. Hal ini memungkinkan terjadi pada Java karena berupa *bytecode* yang menyimpan bahasa tingkat tinggi. Hal ini terjadi pula pada platform *.NET* dari *Microsoft* sehingga program yang dihasilkan mudah dibajak kodenya karena sulit untuk disembunyikan.

Kekurangan *Java* yang lain adalah penggunaan memori yang cukup banyak, lebih besar daripada bahasa tingkat tinggi sebelum generasi Java. Namun hal ini memang sesuai dengan fitur beragam yang dimiliki oleh *Java*. Masalah memori ini juga tidak dialami oleh semua pengguna aplikasi *Java*. Mereka yang sudah menggunakan perangkat keras dengan teknologi terbaru tidak merasakan kelambatan dan konsumsi memori Java yang tinggi. Lain halnya dengan mereka yang menggunakan teknologi lama atau komputer yang sudah berumur tua lebih dari empat tahun akan merasakan adanya kelambatan. Namun apapun kelemahan yang dimiliki Java, faktanya adalah *Java* merupakan bahasa pemrograman yang populer dan digunakan di seluruh dunia saat ini.

**** *Halaman ini sengaja dikosongkan*****

BAB III

PERANCANGAN SISTEM

3. METODE PROYEK AKHIR

Dalam bab ini dibahas tentang langkah-langkah perencanaan beserta pembuatan sistem dan perangkat lunak secara keseluruhan yang merupakan pokok dari bahasan utama tugas akhir.

3.1. Metodologi Penelitian

Sistem ini dikerjakan melalui beberapa tahap. Adapun metodologi yang digunakan dalam pengerjaan proyek akhir ini adalah sebagai berikut:



3.2. Peralatan yang Digunakan

a. 3 PC Server dengan spesifikasi sebagai berikut :

Table 3.1 table spesifikasi perangkat keras

PC 1 (Master)	PC 2 (Slave)	PC 3 (App)
Debian GNU/Linux 8	Debian GNU/Linux 8	Debian GNU/Linux 8
Intel(R) Core(TM)2 Quad CPU Q8400 @ 2.66GHz	Intel(R) Core(TM)2 Quad CPU Q8200 @ 2.33GHz	AMD Athlon(tm) 64 Processor 3200+
Memori 6 GB	Memori 5 GB	Memori 2 GB
HD 1 TB	HD 500 GB	HD 150 GB

b. *Cisco Catalyst Express 500 Series*

c. Perangkat lunak yang digunakan dalam *OS Debian 8 64 Bit*

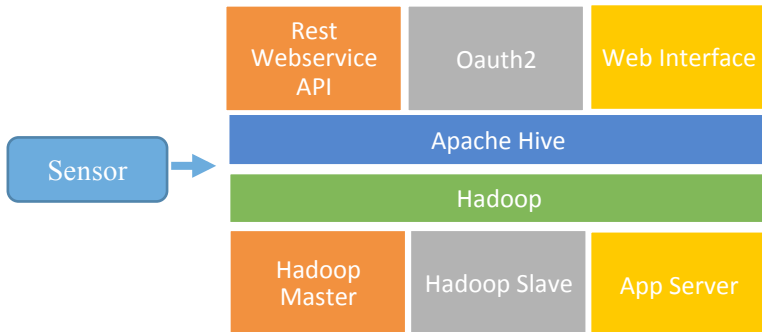
- *Hadoop*
- *Tomcat*
- *PostgreSQL*
- *Java Environment*
- *OpenSSH-Server*

d. Perangkat lunak yang digunakan dalam pembuatan aplikasi

- *Netbeans*
- *Navicat*
- *Mozilla Firefox*

3.3. Arsitektur Sistem

Perancangan yang dibuat merupakan hasil studi literatur yang telah dilakukan sehingga menghasilkan blok sistem yang terlihat pada gambar 3.1 akan digunakan sebagai prosedur dalam pembuatan sistem ini. Secara umum konfigurasi dari blok diagram cyber system sebagai berikut (13).



Aplikasi matagaruda membutuhkan beberapa layer yang terdiri dari database dan aplikasi server dan client untuk dapat bekerja. Untuk level database yang terdiri dari *hadoop*, *hive*, *metastore*, *postgresql*. Sedangkan untuk aplikasi server terdiri dari *oauth2 server*, antar muka web, *restfull api*. Dan yang terakhir adalah client yang digunakan untuk mengirim data dari sensor ke matagaruda

3.4. Rancangan Software :

Berbagai fitur dasar yang ada dalam sebuah framework meliputi :

1. Superuser

- Login
- Admin dashboard
- Mengatur akun
- Mengatur User
- Mengatur database
- Mengatur table hive
- Membuat reporting
- Membuat dan mendesain query

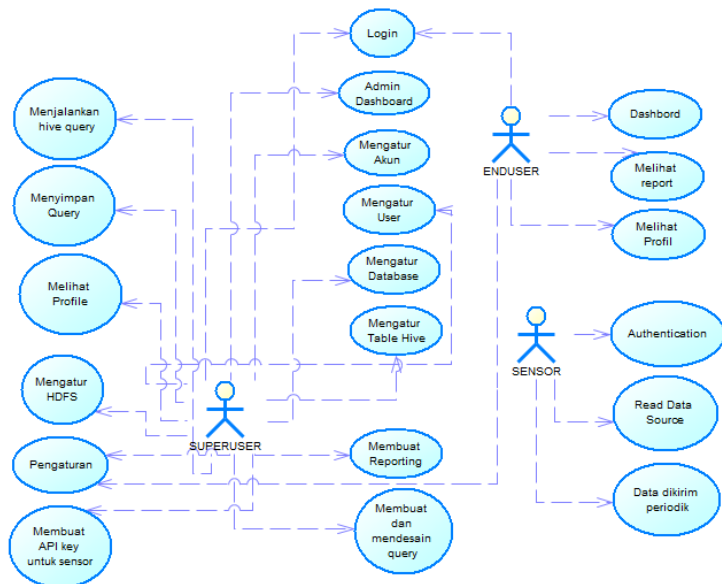
- Menjalankan hive query
- Menyimpan query
- Mengatur HDFS
- Pengaturan
- Membuat API key untuk sensor
- Melihat Profile

2. Enduser

- Login
- Dashboard
- Melihat Report
- Pengaturan
- Melihat Profil

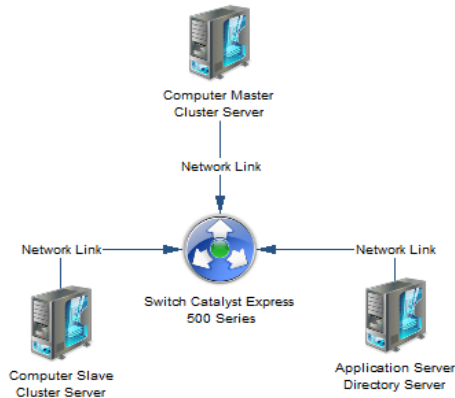
3. Sensor

- Authentication using client_id and client_secret
- Read from data source, csv, tsv, log dsb
- Data dikirim secara periodik



Gambar 3.1 Use Case Diagram

3.5. Perancangan Insfrastruktur Server



Gambar 3.2 infrastruktur server

Pada pembangunan insfrastruktur server memakai virtualisasi komputer dengan menggunakan multi node cluster. Hadoop single-node diimplementasikan pada satu mesin. Mesin tersebut didesain menjadi master tetapi dapat bekerja juga sebagai slave dan semua proses distribusi dilakukan dalam satu mesin tersebut. dalam Hadoop terbagi menjadi dua layer yaitu layer *HDFS* yang menjalankan *Namenode* dan *Datanode* dan layer *Mapreduce* yang menjalankan *Jobtracker* dan *Tasktracker*. Kedua layer ini sangat penting terutama *Namenode* dan *Jobtracker*, karena apabila dua bagian ini tidak berjalan maka kerja HDFS dan *Mapreduce* tidak bisa dijalankan. Pada mesin single node, *Datanode* dan *Tasktracker* hanya ada satu, jika memiliki mesin yang banyak maka kedua bagian ini akan terbentuk pada setiap mesin (multinode).

3.6. Perancangan Hadoop Cluster

3.6.1. Konfigurasi Hadoop Framework

Berikut merupakan tahapan yang dilakukan untuk proses instalasi dan konfigurasi *Hadoop* di 10.252.108.33. Pertama lakukan instalasi *java open-jdk* pada *linux*, karena java merupakan core dari *hadoop* itu sendiri. Dan instalasi *SSH Install Open SSH Server*.

```
root@hadoop-master:/home/jarkom# apt-get install openjdk-7-jdk
```

K

kemudian cek versi java dengan perintah pada terminal `java -version`

Install *Open SSH Server* dengan perintah pada terminal `apt-get install openssh-server`

```
root@hadoop-master:/home/jarkom# apt-get install openssh-  
-----
```

Selanjutnya menambahkan user baru untuk dilakukan pengujian terhadap hadoop single node cluster :

- `sudo addgroup hadoop` (membuat grup bernama hadoop)
- `sudo adduser -ingroup hadoop hduser` (menambah user baru bernama hduser)
- `sudo adduser hduser sudo` (menambahkan hduser pada grup sudo)

```
root@hadoop-master:/home/jarkom# adduser hduser
```

Lakukan konfigurasi pada SSH access.

- Switch ke user hduser pada terminal menggunakan `su - hduser`
- `Type ssh-keygen -t rsa -P ""` (tekan enter untuk melanjutkan)
- Untuk mengaktifkan SSH access, copy keys ke home folder menggunakan perintah.

```
cat $HOME/.ssh/id_rsa.pub >>  
$HOME/.ssh/authorized_keys
```

```
root@hadoop-master:/home/jarkom# su hduser  
hduser@hadoop-master:/home/jarkom$ ssh-keygen -t rsa  
-P ""
```

coba tes ssh pada terminal dengan menggunakan perintah `ssh hduser@10.252.108.33`

Lakukan Disable pada IPv6

- Buka config file: `sudo gedit /etc/sysctl.conf`
- Tambahkan 3 lines pada line terakhir:

```
#disable ipv6;  
net.ipv6.conf.all.disable_ipv6 = 1  
net.ipv6.conf.default.disable_ipv6 = 1  
net.ipv6.conf.lo.disable_ipv6 = 1
```

Jika semuanya sudah selesai dan berhasil, sekarang lakukan instalasi Hadoop

- Extract `Hadoop.2.6.0.tar.gz` , rename foldernya dengan nama `hadoop`.
- Change directory pada Terminal ke path `hadoop` yang sudah di ekstrak tadi. `cd /home/jarkom/downloads/`
- Pindah folder `hadoop` ke dalam folder `local` -
`mv hadoop /usr/local/`
- Lalu ganti permission pada user `hduser` untuk akses folder `hadoop`
`sudo chown -R hduser:hadoop /usr/local/hadoop`
- Lalu ganti permission untuk semua excute -
`chmod +x -R /usr/local/hadoop`

3.6.2. Setting Global Variables.

- Pada file `.bashrc`, dan profile file. Lakukan perintah ini untuk kedua user. Salin ke kedua file sekali dari user biasa di terminal dan sekali lagi dari `hduser`)
`gedit ~/.bashrc` and `sudo gedit ~/.profile`
- Buka file tersebut dan tambahkan baris berikut di akhir.

Set Hadoop-related environment variables

```
export HADOOP_PREFIX=/usr/local/hadoop  
export HADOOP_HOME=/usr/local/hadoop  
export HADOOP_MAPRED_HOME=${HADOOP_HOME}  
export HADOOP_COMMON_HOME=${HADOOP_HOME}  
export HADOOP_HDFS_HOME=${HADOOP_HOME}  
export YARN_HOME=${HADOOP_HOME}  
export HADOOP_CONF_DIR=${HADOOP_HOME}/etc/hadoop
```

Native Path

```
export
HADOOP_COMMON_LIB_NATIVE_DIR=${HADOOP_PREFIX}/lib/native
export
HADOOP_OPTS="Djava.library.path=$HADOOP_PREFIX/lib"
```

#Java path

```
export JAVA_HOME='/usr/lib/jvm/java-7-openjdk-amd64'#
Add Hadoop bin/ directory to PATH
export
PATH=$PATH:$HADOOP_HOME/bin:$JAVA_PATH/bin:$HADOOP_HOME/sbin
```

dan juga set java home pada file `hadoop-env.sh`

`gedit /usr/local/hadoop/conf/hadoop-env.sh`

tambahkan :

```
export JAVA_HOME="/usr/lib/jvm/java-7-openjdk-amd64"
```

Setelah itu tambahkan, reload pada seting menggunakan –

`source ~/.bashrc` dan `source ~/.profile`

lakukan pengetesan dengan perintah

```
echo $HADOOP_HOME dan echo $JAVA_HOME
```

Konfigurasi pada file Hadoop (bisa menggunakan Eclipse atau Text Editor)

1. Ganti direktori `cd /usr/local/hadoop/etc/hadoop`
Buka `yarn-site.xml` file `gedit yarn-site.xml` dan replace dengan menambahkan : Konfigurasi dibawah :

```
<?xml version="1.0"?>
<configuration>
  <property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
  </property>
  <property>
    <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
    <value>org.apache.hadoop.mapred.ShuffleHandler</value>
  </property>
  <property>
```



```

        <name>yarn.resourcemanager.resource-
tracker.address</name>
        <value>hadoop-master:8025</value>
    </property>
</property>
    <name>yarn.resourcemanager.scheduler.add
ress</name>
    <value>hadoop-master:8035</value>
</property>
</property>
    <name>yarn.resourcemanager.address</name
>
    <value>hadoop-master:8050</value>
</property>
</configuration>

```

- Buka core-site.xml file gedit core-site.xml dan replace dengan konfigurasi dibawah

```

<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl"
href="configuration.xsl"?>
<configuration>
    <property>
        <name>fs.default.name</name>
        <value>hdfs://hadoop-master:9000</value>
    </property>
    <property>
        <name>hadoop.proxyuser.hive.hosts</name>
        <value>*</value>
    </property>
    <property>
        <name>hadoop.proxyuser.hive.groups</name>
        <value>*</value>
    </property>
</configuration>

```

- Buka mapred-site.xml file - gedit mapred-site.xml dan replace dengan konfigurasi dibawah

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl"
href="configuration.xsl"?>
<configuration>
  <property>
    <name>mapreduce.job.tracker</name>
    <value>hadoop-master:5431</value>
  </property>
  <property>
    <name>mapred.framework.name</name>
    <value>yarn</value>
  </property>
</configuration>
```

- Buka hdfs-site.xml file - gedit hdfs-site.xml dan replace dengan konfigurasi dibawah

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl"
href="configuration.xsl"?>
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>2</value>
  </property>
  <property>
    <name>dfs.namenode.name.dir</name>
    <value>file:/usr/local/hadoop_tmp/hdfs/namenode</value>
  </property>
  <property>
    <name>dfs.datanode.data.dir</name>
    <value>file:/usr/local/hadoop_tmp/hdfs/datanode</value>
  </property>
</configuration>
```

Lalu buat beberapa folder baru

```
sudo mkdir -p $HADOOP_HOME/yarn_data/hdfs/namenode
sudo mkdir -p $HADOOP_HOME/yarn_data/hdfs/datanode
```

Sekarang lakukan testing , perlu diperhatikan saat melakukan perintah pada kali akan menggunakan user yang sudah dibuat yaitu hduser unuk melakukan testing.

- Ganti ke bin folder (`cd /usr/local/hadoop/bin`) dan format pada namenode (bekerja dari mana saja karena tadi kita telah menambahkan folder bin Hadoop untuk Sistem jalur sebelumnya).

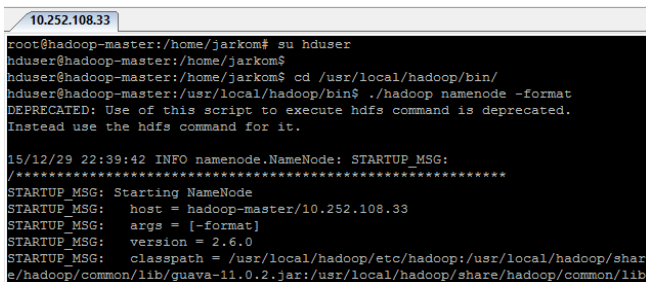
- Untuk mengedit atau menghapus data sementara di folder / app, pertama kita buat direktori / app dan mengubah izin akses sehingga kita dapat melakukan operasi apapun.

- `sudo mkdir /app`
- `sudo chown hduser:hadoop -R /app`
- `sudo chown -R hduser:hadoop /usr/local/hadoop`
- `./hadoop namenode -format`

- Exit status should be 0.

jika folder tidak dapat dibuat, Anda harus memiliki folder itu lagi.

- Lihat gambar bawah untuk contoh output dari perintah Hadoop namenode-format



```
10.252.108.33
root@hadoop-master:/home/jarkom# su hduser
hduser@hadoop-master:/home/jarkom$
hduser@hadoop-master:/home/jarkom$ cd /usr/local/hadoop/bin/
hduser@hadoop-master:/usr/local/hadoop/bin$ ./hadoop namenode -format
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.

15/12/29 22:39:42 INFO namenode.NameNode: STARTUP_MSG:
/*****
STARTUP_MSG: Starting NameNode
STARTUP_MSG:  host = hadoop-master/10.252.108.33
STARTUP_MSG:  args = [-format]
STARTUP_MSG:  version = 2.6.0
STARTUP_MSG:  classpath = /usr/local/hadoop/etc/hadoop:/usr/local/hadoop/share/hadoop/common/lib/guava-11.0.2.jar:/usr/local/hadoop/share/hadoop/common/lib
```

Gambar 3.3 *format hadoop file system*

- Buka folder sbin run semua daemon. `cd /usr/local/hadoop/sbin`
- Eksekusi dengan perintah `./start-all.sh`.
- Untuk memeriksa apakah daemon berjalan, gunakan perintah `jps` (dari hduser).

```

10.252.108.33
hduser@hadoop-master:/usr/local/hadoop/sbin$ ls
distribute-exclude.sh  start-all.cmd  stop-balancer.sh
hadoop-daemon.sh       start-all.sh   stop-dfs.cmd
hadoop-daemons.sh     start-balancer.sh stop-dfs.sh
hdfs-config.cmd        start-dfs.cmd  stop-secure-dns.sh
hdfs-config.sh         start-dfs.sh   stop-yarn.cmd
httpfs.sh              start-secure-dns.sh stop-yarn.sh
kms.sh                 start-yarn.cmd  yarn-daemon.sh
mr-jobhistory-daemon.sh start-yarn.sh   yarn-daemons.sh
refresh-namenodes.sh  stop-all.cmd
slaves.sh              stop-all.sh
hduser@hadoop-master:/usr/local/hadoop/sbin$ ./start-all.sh
This script is deprecated. Instead use start-dfs.sh and start-yarn.sh
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
15/12/29 22:43:02 WARN util.NativeCodeLoader: Unable to load native-hadoop library
for your platform... using builtin-java classes where applicable
Starting namenodes on [hadoop-master]
hduser@hadoop-master's password:
hadoop-master: starting namenode, logging to /usr/local/hadoop/logs/hadoop-hduser-namenode-hadoop-master.out
hduser@hadoop-master's password:
hadoop-master: starting datanode, logging to /usr/local/hadoop/logs/hadoop-hduser-datanode-hadoop-master.out

```

Gambar 3.4 *hadoop binary*

• Jika daemon tidak mau jalan, gunakan cara secara manual, dengan perintah dibawah.

- `hadoop-daemon.sh start namenode`
- `hadoop-daemon.sh start datanode`
- `yarn-daemon.sh start resourcemanager`
- `yarn-daemon.sh start nodemanager`
- `mr-jobhistory-daemon.sh start history server`

Daemon Hadoop memiliki beberapa port melalui TCP. Beberapa port ini digunakan oleh daemon Hadoop untuk berkomunikasi di antaranya berfungsi sebagai (untuk menjadwalkan pekerjaan, meniru blok, dll). port Lainnya mendengarkan langsung ke user, baik melalui klien yang berkomunikasi melalui protokol internal atau melalui HTTP biasa.

Table 3.1 Default Hadoop port sebagai berikut :

	Daemon	Default Port	Configuration Parameter
HD	Namenode	50070	dfs.http.address
FS	Datanodes	50075	dfs.datanode.http.address

MR	Secondarynamenode	50090	dfs.secondary.http.address
	Backup/Checkpoint node?	50105	dfs.backup.http.address
	Jobtracker	50030	mapred.job.tracker.http.address
	Tasktracker	50060	mapred.task.tracker.http.address

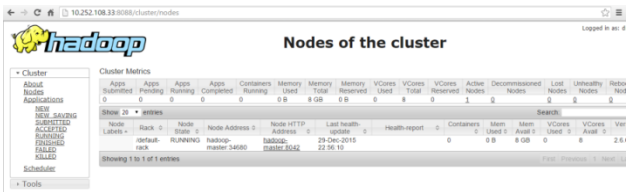
Pengujian yang paling penting adalah task jps. Gunakan task jps untuk memeriksa daemon yang berjalan.

```

nduser@hadoop-master:/usr/local/hadoop/sbin$ jps
2179 NodeManager
1260 NameNode
2335 Jps
1931 ResourceManager
1379 DataNode

```

Gambar 3.5. Status servis lingkungan java



Gambar 3.6. Hadoop Node

10.252.108.33:50070/dfshealth.html#tab=overview

Hadoop
Overview
Datanodes
Snapshot
Startup Progress
Utilities

Overview 'hadoop-master:9000' (active)

Started:	Tue Dec 29 22:43:14 WIB 2015
Version:	2.6.0, re3496499ecb8d2207ba99dc5ed4c99c89e33bb1
Compiled:	2014-11-13T21:10Z by jenkins from (detached from e349649)
Cluster ID:	CID-e463a1c9-6557-481a-be19-dec56e59a1e1
Block Pool ID:	BP-220695761-10.252.108.33-1450811879813

Summary

Security is off.
Safemode is off.
467 files and directories, 407 blocks = 874 total filesystem object(s).
Heap Memory used 53.09 MB of 158.5 MB Heap Memory. Max Heap Memory is 889 MB.
Non Heap Memory used 28.48 MB of 29.94 MB Committed Non Heap Memory. Max Non Heap Memory is 214 MB.

Gambar 3.7. Hadoop User Interface

3.6.3. Instalasi dan Konfigurasi Multi-Node Cluster

Proses instalasi hampir sama dengan konfigurasi single node sebelumnya,

Berikut, Multi Node Hadoop cluster terdiri dari Master-Slave Arsitektur untuk pemenuhan pengolahan BigData yang berisi beberapa node. Jadi, dalam (satu komputer sebagai Master Node dan satu komputer lain sebagai Slave Nodes) untuk menyiapkan Hadoop Cluster. Berikut ada korelasi antara jumlah komputer di cluster dengan ukuran data dan data teknik pengolahan. Oleh karena itu semakin berat dataset (serta berat teknik pengolahan data) membutuhkan jumlah yang lebih besar dari komputer / node dalam Hadoop klaster.

Dasar instalasi dan konfigurasi:

Langkah pertama identifikasi Hostname node harus dikonfigurasi dalam langkah-langkah lebih lanjut. Untuk Masternode diberi nama HadoopMaster dan Slave diberi HadoopSlave1 masing-masing dalam direktori / etc / hosts. Setelah memutuskan nama host dari semua node,

```
10.252.108.33  hadoop-master database
10.252.108.34  hadoop-slave1
```

Buat Hadoop sebagai kelompok dan pengguna sebagai pengguna di semua Mesin

```
hduser@hadoop-master:/$ sudo addgroup hadoop
hduser@hadoop-master:/$ sudo adduser -ingroup hadoop
hduser
```

Lalu tambahkan hdusers ke sudoers, dengan perintah berikut

```
hduser@hadoop-master:/$ sudo usermod -a -G sudo hduser
```

install rsync untuk berbagi sumber Hadoop pada semua Mesin, setelah terinstal restart komputer

```
hduser@hadoop-master:/$ sudo apt-get install rsync
```

langkah selanjutnya update core-site.xml, hdfs-site.xml, yarn-site.xml, mapred-site.xml

Update file master pada /usr/local/hadoop/etc/hadoop/masters

```
hadoop-master
```

```
hadoop-master
hadoop-slave1
```

ga file slaves /usr/local/hadoop/etc/hadoop/slaves

M

menyalin / Sharing / Mendistribusikan Hadoop file konfigurasi untuk semua node - master / slave

Gunakan rsync untuk mendistribusikan dikonfigurasi sumber Hadoop antara node melalui jaringan.

```
sudo rsync -avxP /usr/local/hadoop/ hduser@hadoop-
slave1:/usr/local/hadoop
```

Perintah di atas akan berbagi file yang tersimpan dalam folder Hadoop ke Slave node dengan lokasi - /usr / local / Hadoop. Jadi, tidak perlu lagi men-download serta pengaturan konfigurasi di atas di sisa semua node. hanya perlu Java dan rsync harus diinstal atas semua node. Dan jalur JAVA_HOME ini perlu dicocokkan dengan \$ HADOOP_HOME / etc Hadoop file // hadoop-env.sh distribusi Hadoop yang sudah dikonfigurasi dalam konfigurasi single node Hadoop.

Lalu menerapkan node master konfigurasi Hadoop spesifik:

(Hanya untuk node master) Ini adalah beberapa konfigurasi untuk diterapkan lebih Hadoop master Nodes (satu node master itu akan diterapkan hanya satu node master.)

```
hduser@hadoop-master:/$ sudo rm -rf
/usr/local/hadoop_tmp/
```

Membuat yang sama direktori (usr/local/hadoop_tmp/) dan menciptakan NameNode pada direktori (usr / local / hadoop_tmp / hdfs / namenode). Dan menjadikan hduser sebagai owner direktori tersebut.

```
sudo mkdir -p /usr/local/hadoop_tmp/
sudo mkdir -p /usr/local/hadoop_tmp/hdfs/namenode
sudo chown hduser:hadoop -R /usr/local/hadoop_tmp/
```

Menerapkan node konfigurasi Hadoop Slave spesifik:

(Hanya untuk node slave). Karena memiliki 1 node slave, lakukan perubahan berikut lebih hadoop-slave1.

Hapus folder yang ada hadoop_data (pengaturan Hadoop single node)

```
hduser@hadoop-slave1:/$ sudo rm -rf
/usr/local/hadoop_tmp/hdfs/
```

Kemudian buat direktori

/usr/local/hadoop_tmp/hdfs/datanode

```
sudo mkdir -p /usr/local/hadoop_tmp/hdfs/datanode
```


Jadikan hduser sebagai owner dari direktori `hadoop_tmp`

```
sudo chown hduser:hadoop -R /usr/local/hadoop_tmp/
```

e

nyalin kunci ssh untuk Menyiapkan akses ssh passwordless dari master untuk Slave node:

Untuk mengelola (start / stop) semua node dari Master-Slave arsitektur, hduser (Hadoop pengguna dari Masternode) perlu login pada semua Slave serta semua node Guru yang dapat dimungkinkan melalui login pengaturan SSH passwordless. (Jika tidak menetapkan ini maka perlu memberikan password saat mulai dan stoping daemon pada node Slave dari Master node).

perintah berikut untuk berbagi kunci SSH publik - `$ HOME / .ssh / id_rsa.pub` (dari `hadoop-master` node) untuk `authorized_keys` file `hduser @ HadoopSlave1` dan juga pada `hduser @ hadoop-slave1` (di `$ HOME / .ssh / authorized_keys`)

```
ssh-copy-id -i /home/hduser/.ssh/id_rsa.pub hduser@hadoop-master
ssh-copy-id -i /home/hduser/.ssh/id_rsa.pub hduser@hadoop-slave1
```

f

Format namenode (jalankan pada master-node)

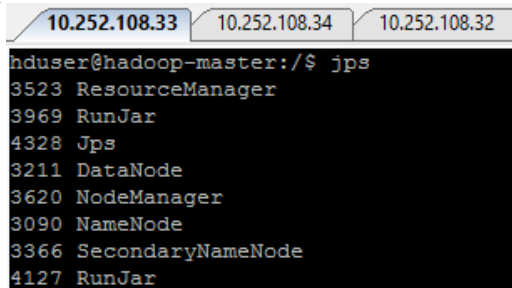
```
hduser@hadoop-master:/usr/local/hadoop$ hdfs namenode -format
```

```
hduser@hadoop-master:/$ start-all.sh
```

a

lankan semua service hadoop dengan perintah `start-all.sh`

Track/Monitor/Verifikasi `hadoop-master` cluster dengan perintah `jps`



```
10.252.108.33 10.252.108.34 10.252.108.32
hduser@hadoop-master:/$ jps
3523 ResourceManager
3969 RunJar
4328 Jps
3211 DataNode
3620 NodeManager
3090 NameNode
3366 SecondaryNameNode
4127 RunJar
```

Gambar 3.8 java proses pada master

Track/Monitor/Verifikasi hadoop-slave1 cluster dengan perintah jps

```

10.252.108.33 10.252.108.34 10.252.108.32
hduser@hadoop-slave1:/$ jps
2116 NodeManager
2256 Jps
2013 DataNode

```

Gambar 3.8 java proses pada slave1

Web Interface Hadoop dengan Multi-node Cluster (Master & Slave)



Gambar 3.9 node manager

3.6.4. Instalasi dan Konfigurasi Apache Hive

Download dan Ekstrak file Apache Hive versi 1.1.0

```
sudo tar -zxzf apache-hive-1.1.0-bin.tar.gz
```

Pindahkan apache hive ke direktori /usr/local/hive

```
sudo mv apache-hive-1.1.0-bin /usr/local/hive
```

Set Path Environment

```
sudo vim ~/.bashrc
```

```
# --- HIVE CONFIG --- #
export HIVE_HOME=/usr/local/hive
export PATH=$PATH:$HIVE_HOME/bin
export CLASSPATH=$CLASSPATH:/usr/local/hadoop/hadoop/lib/*:.
export CLASSPATH=$CLASSPATH:/usr/local/hadoop/hive/lib/*:.
```

```
hduser@hadoop-master:/$ source ~/.bashrc
```

Instalasi dan Konfigurasi PostgreSQL

```
Sudo apt-get install postgresql postgresql-contrib
```

beberapa langkah untuk menjalankan hive :

Pertama adalah menyiapkan metastore atau didalam oracle disebut sebagai data dictionary, kita harus menyiapkan database RDBMS dalam hal ini kita memakai postgres sebagai metastore untuk hive. Berikut konfigurasinya:

```
<property>
<name>javax.jdo.option.ConnectionURL</name>
<value>jdbc:postgresql://myhost/metastore</value>
</property>
<property>
<name>javax.jdo.option.ConnectionDriverName</name>
  <value>org.postgresql.Driver</value>
</property>
<property>
<name>javax.jdo.option.ConnectionUserName</name>
  <value>hiveuser</value>
</property>
<property>
<name>javax.jdo.option.ConnectionPassword</name>
  <value>mypassword</value>
</property>
<property>
  <name>datanucleus.autoCreateSchema</name>
  <value>false</value>
</property>
<property>
  <name>hive.metastore.uris</name>
  <value>thrift://hadoop-master:9083</value>
  <description>IP address (or fully-qualified
domain name) and port of the metastore
host</description>
</property>
<property>
<name>hive.metastore.schema.validation</name>
<value>true</value>
</property>
```

Dan setelah itu kita start metastore nya

```
$ hive -service metastore
```

Agar sebuah client bisa terkoneksi ke hive maka kita perlu menjalankan hiveserver2 dengan port 10000, berikut perintah untuk menjalankan hiveserver2 :

```
#hive -service hiveserver2
```

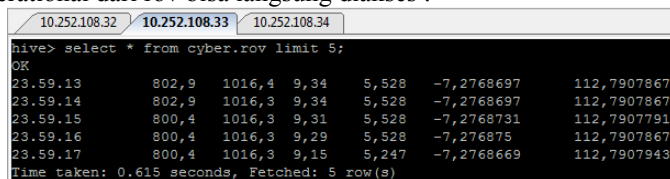
Setelah itu kita bisa langsung terkoneksi dengan jdbc connection dengan beeline untuk mode command line :

```
hduser@debian:~$ beeline
Beeline version 1.2.0 by Apache Hive
beeline>!connect jdbc:hive2://localhost:10000/default
Connecting to jdbc:hive2://localhost:10000/default
Enter username for
jdbc:hive2://localhost:10000/default:
Enter password for
jdbc:hive2://localhost:10000/default:
Connected to: Apache Hive (version 1.2.0)
Driver: Hive JDBC (version 1.2.0)
0: jdbc:hive2://localhost:10000/default>
```

Percobaan untuk pembuatan table dan melakukan query di hive :

```
CREATE EXTERNAL TABLE rov (
    timestamp STRING,
    ec STRING,
    orp STRING,
    do STRING,
    ph STRING,
    latitude STRING,
    longitude STRING
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
STORED AS TEXTFILE;
```

Dan setelah table terbentuk maka kita bisa memasukkan data kedalam hive dengan menggunakan tool scoop jika database operational dari rov bisa langsung diakses :

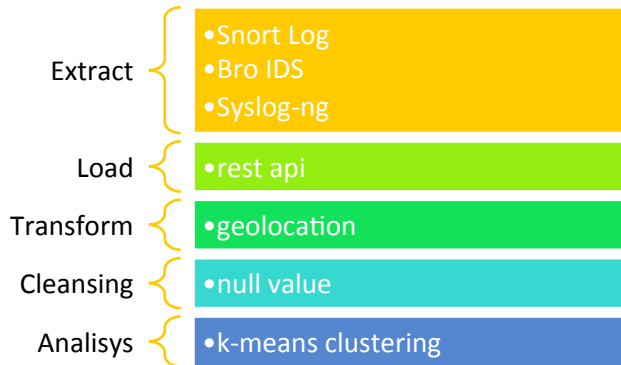


```
hive> select * from cyber.rov limit 5;
OK
23.59.13      802,9      1016,4      9,34      5,528      -7,2768697      112,7907867
23.59.14      802,9      1016,3      9,34      5,528      -7,2768697      112,7907867
23.59.15      800,4      1016,3      9,31      5,528      -7,2768731      112,7907791
23.59.16      800,4      1016,3      9,29      5,528      -7,276875      112,7907867
23.59.17      800,4      1016,3      9,15      5,247      -7,2768669      112,7907943
Time taken: 0.615 seconds, Fetched: 5 row(s)
```

Gambar 3.10 menampilkan data hive

3.7. Data Analisis

Pada tahap selanjutnya yaitu proses untuk analisa data dimana pada tahap ini terbagi menjadi beberapa sub proses seperti pada berikut:



Gambar 3.11 diagram proses analisis data

Pada gambar diatas proses pertama yaitu mengextract dari beberapa sumber data seperti snort, bro, syslog-ng dbs. Selanjutnya yaitu proses load yaitu memasukkan data ke dalam hive table. Proses selanjutnya yaitu transformasi data dari IP address menjadi data geolocation melalui fungsi UDTF (user defined table function) yaitu sebuah fitur di hive dimana kita bisa menulis suatu store function untuk dijalankan di query.

Unk membuat sebuah UDTF atau function di hive kita bisa membuatnya dengan bahasa pemrograman java

```

@UDFType(deterministic = true)
@Description(
    name = "geoip2",
    value = "_FUNC_(ip, database) - looks a property
for an IP address from"
    + "a library loaded\n"
    + "The GeoIP2 database comes separated. To load
the GeoIP2 use ADD FILE.\n"
    + "Usage:\n"
    + " > _FUNC_(\"8.8.8.8\", \"database\")")
public class GenericUDTFGeoIP extends GenericUDTF {

    private PrimitiveObjectInspector stringOI;
    private PrimitiveObjectInspector databaseOI;

    @Override
    public StructObjectInspector
    initialize(ObjectInspector[] args) throws
    UDFArgumentException {

        if (args.length != 2) {
            throw new
            UDFArgumentException("GeoIP2GenericUDTF() takes exactly
two argument");
        }

        if (args[0].getCategory() !=
ObjectInspector.Category.PRIMITIVE
            && ((PrimitiveObjectInspector)
args[0]).getPrimitiveCategory() !=
PrimitiveObjectInspector.PrimitiveCategory.STRING) {
            throw new
            UDFArgumentException("GeoIP2GenericUDTF() takes a string
as a parameter");

```

```

    }

    // input inspectors
    stringOI = (PrimitiveObjectInspector) args[0];
    databaseOI = (PrimitiveObjectInspector) args[1];
    List<String> fieldNames = new ArrayList<>(7);
    List<ObjectInspector> fieldOIs = new
ArrayList<>(7);
        fieldNames.add("country_name");
        fieldNames.add("country_iso_code");
        fieldNames.add("subdivision_name");
        fieldNames.add("city");
        fieldNames.add("postal_code");
        fieldNames.add("latitude");
        fieldNames.add("longitude");

fieldOIs.add(PrimitiveObjectInspectorFactory.javaStringO
bjectInspector);

fieldOIs.add(PrimitiveObjectInspectorFactory.javaStringO
bjectInspector);

fieldOIs.add(PrimitiveObjectInspectorFactory.javaStringO
bjectInspector);

fieldOIs.add(PrimitiveObjectInspectorFactory.javaStringO
bjectInspector);

fieldOIs.add(PrimitiveObjectInspectorFactory.javaStringO
bjectInspector);

fieldOIs.add(PrimitiveObjectInspectorFactory.javaDoubleO
bjectInspector);

```

```

fieldOIs.add(PrimitiveObjectInspectorFactory.javaDoubleObjectInspector);

        return
ObjectInspectorFactory.getStandardStructObjectInspector(
fieldNames, fieldOIs);
    }

    public ArrayList<Object[]> processInputRecord(String
ip, String databaseFile) {
        ArrayList<Object[]> result = new ArrayList<>();

        // ignoring null or empty input
        if (ip == null || ip.isEmpty()) {
            return result;
        }

        File database = new File(databaseFile);

        try {
            // This creates the DatabaseReader object,
which should be reused across
            // lookups.
            DatabaseReader reader = new
DatabaseReader.Builder(database).build();
            InetAddress ipAddress =
InetAddress.getByName(ip);

            CityResponse response =
reader.city(ipAddress);

            result.add(new Object[]{
                response.getCountry().getName(),
                response.getCountry().getIsoCode(),

```



```

response.getMostSpecificSubdivision().getName(),
            response.getCity().getName(),
            response.getPostal().getCode(),
            response.getLocation().getLatitude(),
            response.getLocation().getLongitude()
        ));

        } catch (Exception e) {
            throw new
UnsupportedOperationException(e.getMessage());
        }

        return result;
    }

    @Override
    public void process(Object[] record) throws
HiveException {

        final String name =
stringOI.getPrimitiveJavaObject(record[0]).toString();
        final String databaseName =
databaseOI.getPrimitiveJavaObject(record[1]).toString();

        ArrayList<Object[]> results =
processInputRecord(name, databaseName);

        Iterator<Object[]> it = results.iterator();

        while (it.hasNext()) {
            Object[] r = it.next();
            forward(r);
        }
    }

```

```

    }

    @Override
    public void close() throws HiveException {
        // do nothing
    }
}

```

Dan untuk memasukkan ke dalam hive kita perlu membuat function

```

CREATE      TEMPORARY      FUNCTION      geoip2      as
'com.matagaruda.udf.GenericUDTFGeoIP';

```

dan kita langsung memanggil query dengan function yang telah kita buat :

```

select  ip.country_name,  ip.latitude,  ip.longitude
from    snort_lateral_view  geoip2('203.34.119.52',
'/home/hduser/GeoLite2-City.mmdb') ip as country_name,
country_iso_code,  subdivision_name,  city,  postal_code,
latitude, longitude;

```

dan untuk melakukan analisis lebih lanjut kita bisa menggunakan mahout untuk machine learning dan dengan integrasi dengan hive :

```

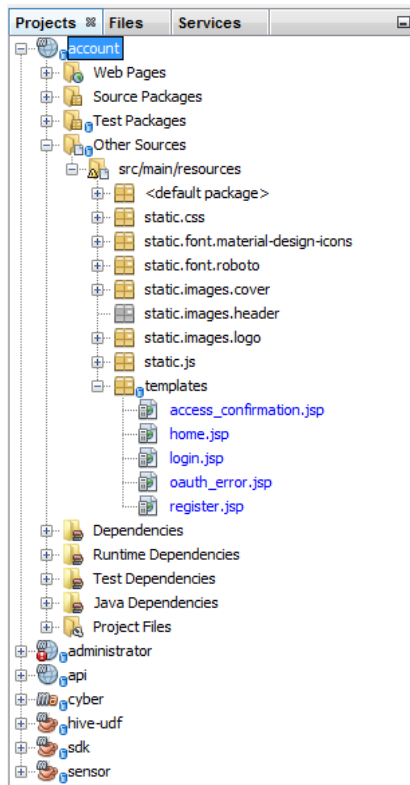
Path    directoryContainingConvertedInput  =    new
Path(output, DIRECTORY_CONTAINING_CONVERTED_INPUT);
InputDriver.runJob(input,
directoryContainingConvertedInput,
"org.apache.mahout.math.RandomAccessSparseVector");
// Get initial clusters randomly
Path clusters = new Path(output, "random-seeds");
Clusters=      RandomSeedGenerator.buildRandom(conf,
directoryContainingConvertedInput,      clusters,      k,
measure);
// Run K-means with a given K
KMeansDriver.run(conf,
directoryContainingConvertedInput,      clusters,      output,
convergenceDelta,
maxIterations, true, 0.0, true);

```

dan kmean diperlukan untuk melakukan cluster terhadap serangan yang terjadi di dunia berdasarkan negara asal dengan centroid adalah negara dan ip address sebagai atribut.

3.8. Rancangan sistem dan Implementasi

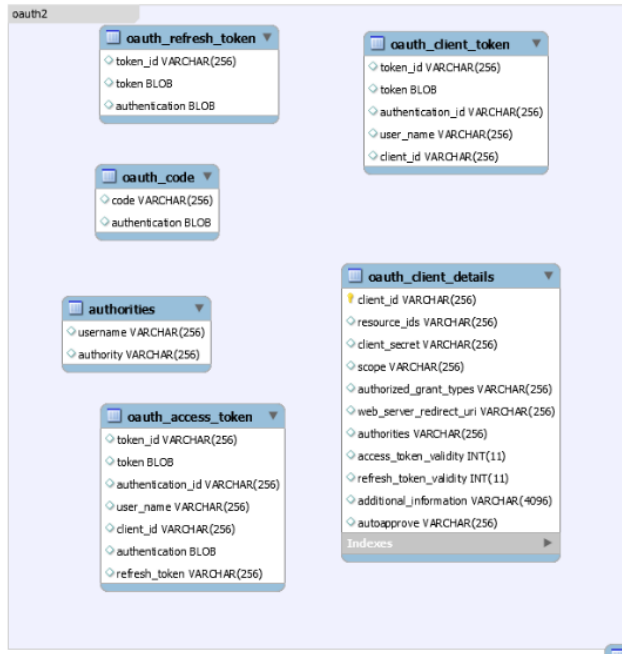
Dalam tahap ini akan dibangun platform menurut desain yang telah dibuat sebelumnya, selain itu juga dalam tahap ini akan ada proses testing. Berikut struktur directory dari framework spring pada java yang digunakan sistem :



Gambar 3.12. Framework Sistem

3.8.1. ERD

Berikut ini adalah diagram ERD dari sistem yang kami buat :



Gambar 3.13. *entity relationship diagram*

Gambar diatas adalah erd untuk schema dari oauth2 server dimana terdapat tabel untuk menyimpan client device atau sensor dan juga terdapat tabel untuk menyimpan token dari aplikasi client.

Berbagai fitur dasar yang ada dalam sebuah framework meliputi :

3.8.2. **Api**

- **/oauth/token**

/oauth/token

POST /oauth/token

Summary

get access token

Description

mengambil access token

Parameters

Name	Located in	Description	Required	Schema
grant_type	query	jenis grant type	Yes	number (double)
scope	query	scope yang ingin diakses	Yes	number (string)

Responses

Code	Description	Schema
200	OK	{ AccessToken { }

Try this operation

- **Membuat database**

/databases

POST /databases

Summary

membuat database

Description

membuat database

Parameters

Name	Located in	Description	Required	Schema
databaseName	query	nama database yang ingin dibuat	Yes	string

Responses

Code	Description	Schema
200	Unexpected error	undefined

Try this operation

- **Mengatur HDFS**

/hdfs-manager

POST /hdfs-manager

Summary
mengakses hdfs file system

Description
mengambil hdfs file system

Parameters

Name	Located in	Description	Required	Schema
path	query	path atau lokasi file atau directory	Yes	⇔ number (double)

Responses

Code	Description	Schema
200	OK	⇔ { ▾ [▸HDFS { }] }

[Try this operation](#)

- **Daftar table hive**

/tables

GET /tables

Summary
mengakses daftar table di hive

Description
mengakses daftar table di hive

Parameters

Name	Located in	Description	Required	Schema
database	query	database dari tabel yang ingin diakses	Yes	⇔ undefined

Responses

Code	Description	Schema
200	Unexpected error	⇔ undefined

[Try this operation](#)

- **Menjalankan hive query**

/query

POST /query		
<h3>Summary</h3> <p>menjalankan query sql di hive</p>		
<h3>Description</h3> <p>menjalankan query sql di hive</p>		
<h3>Responses</h3>		
Code	Description	Schema
200	OK	<pre>{ "query": { "sql": "SELECT * FROM table" } }</pre>

- **Daftar devices**

```
/devices
```

GET /devices

Summary

mengakses daftar device yang sudah dibuat

Description

mengambil daftar device yang sudah dibuat

Parameters

Name	Located In	Description	Required	Schema
path	query	path atau lokasi file atau direktory	Yes	<code>number (double)</code>

Responses

Code	Description	Schema
200	OK	<pre> [Device { }] </pre>

Try this operation

- **Membuat device**

/device

POST /device

Summary
membuat device atau agent

Description
membuat device atau agent dengan grant type client credentials

Parameters

Name	Located in	Description	Required	Schema
grant_type	query	jenis grant type	Yes	<code>number</code> (double)
scope	query	scope yang ingin diakses	Yes	<code>number</code> (string)

Responses

Code	Description
200	OK

Try this operation

3.8.3. Konfigurasi Agent

Untuk memudahkan proses *deployment* maka penggunaan konfigurasi data source yang dinamis sangat diperlukan. Dalam hal ini definisi dari *log file* serta *column* yang dapat dirubah dan diganti sesuai dengan keperluan tanpa merubah *source code* dari aplikasi. Untuk itulah penggunaan dipilih *format YAML* sebagai format untuk definisi dari file konfigurasi di *agent*. Pemilihan dari *YAML* sebagai format konfigurasi karena *YAML* sangat *human readable* atau mudah dibaca struktur dari definis *YAML*. (14) Berikut rancangan atau definisi dari konfigurasi agent menggunakan *YAML*:

```
1  tableName: snort
2  columnSeparator: ","
3  - columns:
4    - {name: id, nullValue: '', type: String}
5    - {name: name, nullValue: '', type: String}
6    - {name: descriptions, nullValue: '', type: String}
7  database: matagaruda
8  driver: csv
9  filename: log.csv
10 lineSeparator: \n
```

Gambar 3.14.definisi yaml

Dari konfigurasi di gambar .. penggunaan *YAML* lebih mudah dibaca. Dan untuk konfigurasi cukup nama table, pemisah kolom, definisi skema kolom, database, driver, filename, line separator.

*** *Halaman ini sengaja dikosongkan* ***

BAB IV

UJI COBA DAN ANALISA DATA

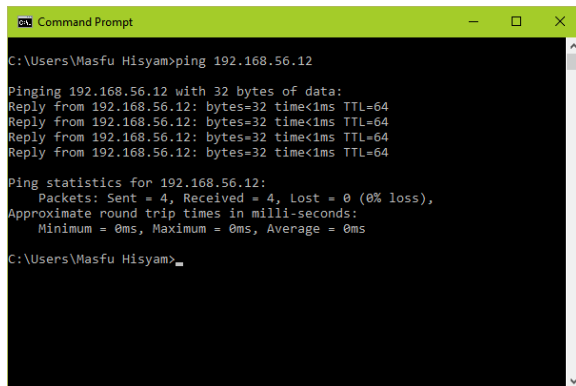
4. Pada bab ini akan dilakukan pengujian dan analisa terhadap beberapa algoritma yang telah dirancang dan dibuat pada bab sebelumnya. Untuk tahap-tahap pengujian yang akan dilakukan adalah sebagai berikut.

4.1. Uji Coba Program

Untuk melakukan pengujian terhadap terhadap aplikasi ini adalah melakukan uji coba pemasangan sensor dengan snort IDS pada tahap ini snort akan melakukan capture terhadap paket data dan jika sesuai dengan rule maka akan disimpan disebuah log file dalam hal ini berupa data csv. Kemudian dari data log tersebut akan dibaca oleh aplikasi sensor matagaruda dan kemudian akan dikirim ke hadoop melalui Rest API dengan Oauth2 Authentication.

4.1.1. Testing snort sensor dan agent

Percobaan pertama yaitu pengujian sensor dari snort untuk mendeteksi serangan yaitu dengan menggunakan deteksi *ping* sederhana seperti pada gambar dibawah ini.



```
Command Prompt
C:\Users\Masfu Hisyam>ping 192.168.56.12

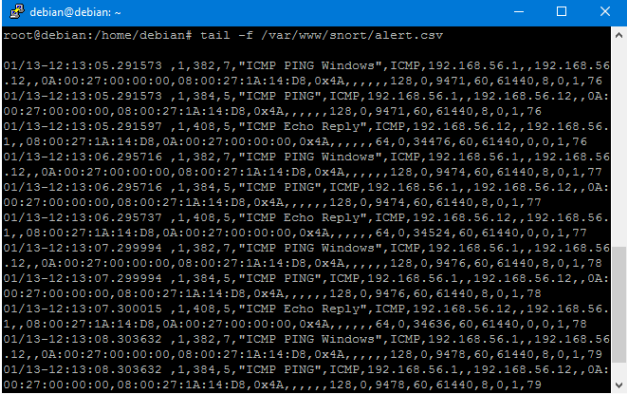
Pinging 192.168.56.12 with 32 bytes of data:
Reply from 192.168.56.12: bytes=32 time<1ms TTL=64
Reply from 192.168.56.12: bytes=32 time<1ms TTL=64
Reply from 192.168.56.12: bytes=32 time<1ms TTL=64
Reply from 192.168.56.12: bytes=32 time<1ms TTL=64

Ping statistics for 192.168.56.12:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 0ms, Average = 0ms

C:\Users\Masfu Hisyam>
```

Gambar 4.1 ping ke sensor

Pada gambar diatas dari komputer client akan mengirim packet icmp ke komputer host dimana terdapat snort yang sudah terinstall. Sedangkan pada komputer host akan terlihat *ping request* yang sudah dicatat oleh *snort*.



```

root@debian:/home/debian# tail -f /var/www/snort/alert.csv
01/13-12:13:05.291573 ,1,382,7,"ICMP PING Windows",ICMP,192.168.56.1,,192.168.56.12,,0A:00:27:00:00:00,08:00:27:1A:14:D8,0x4A,,,,,128,0,9471,60,61440,8,0,1,76
01/13-12:13:05.291573 ,1,384,5,"ICMP PING",ICMP,192.168.56.1,,192.168.56.12,,0A:00:27:00:00:00,08:00:27:1A:14:D8,0x4A,,,,,128,0,9471,60,61440,8,0,1,76
01/13-12:13:05.291597 ,1,408,5,"ICMP Echo Reply",ICMP,192.168.56.12,,192.168.56.1,,08:00:27:1A:14:D8,0A:00:27:00:00:00,0x4A,,,,,64,0,34476,60,61440,0,0,1,76
01/13-12:13:06.295716 ,1,382,7,"ICMP PING Windows",ICMP,192.168.56.1,,192.168.56.12,,0A:00:27:00:00:00,08:00:27:1A:14:D8,0x4A,,,,,128,0,9474,60,61440,8,0,1,77
01/13-12:13:06.295716 ,1,384,5,"ICMP PING",ICMP,192.168.56.1,,192.168.56.12,,0A:00:27:00:00:00,08:00:27:1A:14:D8,0x4A,,,,,128,0,9474,60,61440,8,0,1,77
01/13-12:13:06.295737 ,1,408,5,"ICMP Echo Reply",ICMP,192.168.56.12,,192.168.56.1,,08:00:27:1A:14:D8,0A:00:27:00:00:00,0x4A,,,,,64,0,34524,60,61440,0,0,1,77
01/13-12:13:07.299994 ,1,382,7,"ICMP PING Windows",ICMP,192.168.56.1,,192.168.56.12,,0A:00:27:00:00:00,08:00:27:1A:14:D8,0x4A,,,,,128,0,9476,60,61440,8,0,1,78
01/13-12:13:07.299994 ,1,384,5,"ICMP PING",ICMP,192.168.56.1,,192.168.56.12,,0A:00:27:00:00:00,08:00:27:1A:14:D8,0x4A,,,,,128,0,9476,60,61440,0,0,1,78
01/13-12:13:07.300015 ,1,408,5,"ICMP Echo Reply",ICMP,192.168.56.12,,192.168.56.1,,08:00:27:1A:14:D8,0A:00:27:00:00:00,0x4A,,,,,64,0,34636,60,61440,0,0,1,78
01/13-12:13:08.303632 ,1,382,7,"ICMP PING Windows",ICMP,192.168.56.1,,192.168.56.12,,0A:00:27:00:00:00,08:00:27:1A:14:D8,0x4A,,,,,128,0,9478,60,61440,8,0,1,79
01/13-12:13:08.303632 ,1,384,5,"ICMP PING",ICMP,192.168.56.1,,192.168.56.12,,0A:00:27:00:00:00,08:00:27:1A:14:D8,0x4A,,,,,128,0,9478,60,61440,0,0,1,79

```

Gambar 4.2 log snort

Lalu setelah itu adalah mencoba mengirimkan data *log* tersebut ke *hadoop* menggunakan *agent* atau aplikasi kecil. berikut adalah konfigurasi dari agent yang akan dijalankan:

```

tableName: snort
columnSeparator: ","
columns:
- {name: timestamp, nullValue: '', type: String}
- {name: sig_generator, nullValue: '', type: String}
- {name: sig_id, nullValue: '', type: String}
- {name: sig_rev, nullValue: '', type: String}
- {name: msg, nullValue: '', type: String}
- {name: proto, nullValue: '', type: String}
- {name: src, nullValue: '', type: String}
- {name: srcport, nullValue: '', type: String}
- {name: dst, nullValue: '', type: String}
- {name: dstport, nullValue: '', type: String}

```

```

- {name: ethsrc, nullValue: '', type: String}
- {name: ethdst, nullValue: '', type: String}
- {name: ethlen, nullValue: '', type: String}
- {name: tcpflags, nullValue: '', type: String}
- {name: tcpseq, nullValue: '', type: String}
- {name: tcplen, nullValue: '', type: String}
- {name: tcpwindow, nullValue: '', type: String}
- {name: ttl, nullValue: '', type: String}
- {name: tos, nullValue: '', type: String}
- {name: id, nullValue: '', type: String}
- {name: dgmlen, nullValue: '', type: String}
- {name: ipplen, nullValue: '', type: String}
- {name: icmpitype, nullValue: '', type: String}
- {name: icmpcode, nullValue: '', type: String}
- {name: icmpid, nullValue: '', type: String}
- {name: icmpseq, nullValue: '', type: String}
database: matagaruda
driver: csv
filename: snort.csv
lineSeparator: \n

```

konfigurasi lokasi dari log file snort dan juga format log file yang akan dibaca oleh agentnya, seperti *column separator* dan *line separator*. Berikut adalah percobaan agent yang telah kita buat:

```

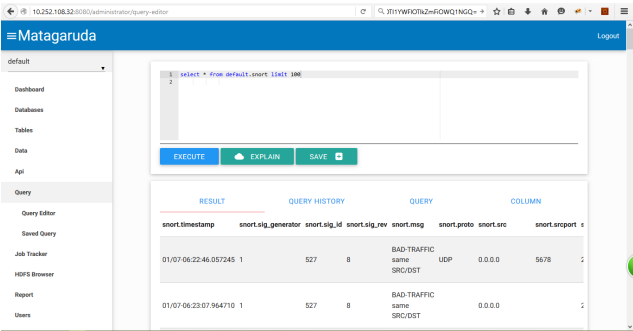
debian@debian: ~
root@debian:/var/www/snort# java -jar sensor.jar
Jan 13, 2016 1:09:37 PM org.springframework.context.annotation.AnnotationConfigApplicationContext prepareRefresh
INFO: Refreshing org.springframework.context.annotation.AnnotationConfigApplicationContext@23f3329a: startup date [Wed Jan 13 13:09:37 WIB 2016]; root of context hierarchy
POST http://10.252.108.32:8080/api/v1/datacollector/default/snort HTTP/1.1
refresh token 1452665424063 1452665380482 43581
POST http://10.252.108.32:8080/api/v1/datacollector/default/snort HTTP/1.1
refresh token 1452665424063 145266562393 -228330

```

Gambar 4.3 testing agent

Setelah kita jalankan agent dan berhasil mengirimkan sejumlah data dari file log yang dihasilkan oleh snort. Maka kita akan lakukan verifikasi apakah data tersebut

sudah masuk ke *hadoop*, untuk membuktikannya kita akan langsung query di *hive*.



Gambar 4.4 query dari *snort*

Dari gambar diatas data dari log *snort* sudah di dalam *hive* table dan sudah siap untuk dilakukan tahap selanjutnya yaitu proses cleansing atau pembersihan data.

4.1.2. Testing Rest API ETL Service

Percobaan selanjutnya adalah uji coba dari performa *data ingestion* yaitu performa dari Rest API dan *ingest data* ke *HDFS*. Dalam hal ini menggunakan metode *pubsub* atau *publisher/subscriber*. Arsitektur dari Rest API adalah *request* dan *response* sedangkan dari *hadoop* khususnya *hive* bekerja dengan metode *batch*. Sehingga data yang diterima oleh Rest API tidak langsung dimasukkan ke dalam *hive* tetapi ditangani oleh sebuah *process* atau *thread* lain dan menggunakan *pub/sub* sebagai media pertukaran pesan atau *message broker*.

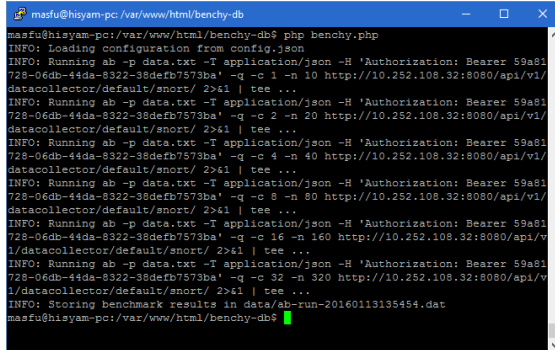
Untuk data testing yang digunakan dalam pengujian kali ini adalah berupa data *snort log* yang sebelumnya telah dirubah menjadi data *json*. Berikut contoh data *snort log* yang telah dirubah menjadi bentuk *json*.

```
[{
  "timestamp": "01/07-14:35:34.300218 ",
  "sig_generator": "1",
  "sig_id": "384",
  "sig_rev": "5",
  "msg": "ICMP PING",
  "proto": "ICMP",
  "src": "46.234.125.89",
  "srcport": "",
  "dst": "112.140.160.15",
  "dstport": "",
  "ethsrc": "4C:5E:0C:6A:22:DB",
  "ethdst": "00:0C:29:67:4E:0E",
  "ethlen": "0x76",
  "tcpflags": "",
  "tcpseq": "",
  "tcpack": "",
  "tcpplen": "",
  "tcpwindow": "",
  "ttl": "52",
  "tos": "40",
  "id": "43743",
  "dgmlen": "104",
  "iplen": "106496",
  "icmptype": "8",
  "icmpcode": "0",
  "icmpid": "23392",
  "icmpseq": "0"
}]
```

Setiap data json tersebut akan disimpan langsung di table hive. Dengan menggunakan fitur dari redis yaitu publisher/subscriber maka ketika data diterima oleh *REST API* data tersebut tidak langsung dimasukkan kedalam hive table karena proses untuk memasukkan tersebut membutuhkan

waktu karena bersifat *batch processing* sehingga disimpan kedalam database redis untuk selanjutnya diberikan kepada *subscriber* untuk kemudian dimasukkan kedalam hive table.

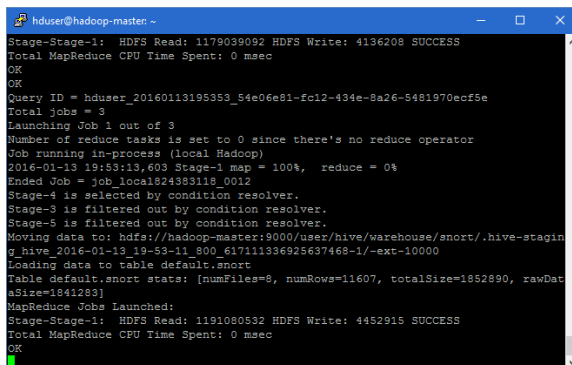
Untuk pengujian *REST API* digunakan sebuah perangkat testing yaitu apache bench. Apache bench merupakan aplikasi kecil yang melakukan *stress test* dengan mengirim sejumlah besar *request* ke sebuah *API* secara *paralel*.



```
masfu@hisyam-pc: /var/www/html/benchy-db$ php benchy.php
INFO: Loading configuration from config.json
INFO: Running ab -p data.txt -T application/json -H 'Authorization: Bearer 59a61728-06db-44da-8322-38defb7573ba' -q -c 1 -n 10 http://10.252.108.32:8080/api/v1/datacollector/default/snort/ 2>&1 | tee ...
INFO: Running ab -p data.txt -T application/json -H 'Authorization: Bearer 59a61728-06db-44da-8322-38defb7573ba' -q -c 2 -n 20 http://10.252.108.32:8080/api/v1/datacollector/default/snort/ 2>&1 | tee ...
INFO: Running ab -p data.txt -T application/json -H 'Authorization: Bearer 59a61728-06db-44da-8322-38defb7573ba' -q -c 4 -n 40 http://10.252.108.32:8080/api/v1/datacollector/default/snort/ 2>&1 | tee ...
INFO: Running ab -p data.txt -T application/json -H 'Authorization: Bearer 59a61728-06db-44da-8322-38defb7573ba' -q -c 8 -n 80 http://10.252.108.32:8080/api/v1/datacollector/default/snort/ 2>&1 | tee ...
INFO: Running ab -p data.txt -T application/json -H 'Authorization: Bearer 59a61728-06db-44da-8322-38defb7573ba' -q -c 16 -n 160 http://10.252.108.32:8080/api/v1/datacollector/default/snort/ 2>&1 | tee ...
INFO: Running ab -p data.txt -T application/json -H 'Authorization: Bearer 59a61728-06db-44da-8322-38defb7573ba' -q -c 32 -n 320 http://10.252.108.32:8080/api/v1/datacollector/default/snort/ 2>&1 | tee ...
INFO: Storing benchmark results in data/ab-run-20160113135454.dat
masfu@hisyam-pc: /var/www/html/benchy-db$
```

Gambar 4.5 testing dari apache bench

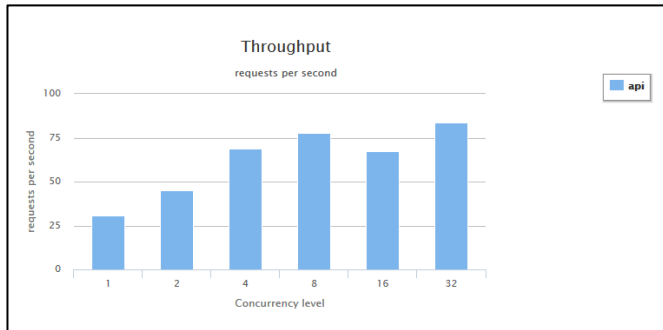
Pada gambar .. pengujian dilakukan sebanyak enam kali dengan jumlah request yang berbeda-beda setiap pengujian dan juga jumlah request secara paralel juga berbeda.



```
hduser@hadoop-master: ~
Stage-Stage-1: HDFS Read: 1179039092 HDFS Write: 4136208 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
OK
OK
Query ID = hduser_20160113195353_54e06e81-fc12-434e-8a26-5481970ecf5e
Total jobs = 3
Launching Job 1 out of 3
Number of reduce tasks is set to 0 since there's no reduce operator
Job running in-process (local Hadoop)
2016-01-13 19:53:13,603 Stage-1 map = 100%, reduce = 0%
Ended Job = job local824383118_0012
Stage-4 is selected by condition resolver.
Stage-3 is filtered out by condition resolver.
Stage-5 is filtered out by condition resolver.
Moving data to: hdfs://hadoop-master:9000/user/hive/warehouse/snort/.hive-staging_hive-2016-01-13_19-53-11_900_617111336925637468-1/-ext-10000
Loading data to table default.snort
Table default.snort stats: [numFiles=8, numRows=11607, totalSize=1852890, rawDataSize=1841283]
MapReduce Jobs Launched:
Stage-Stage-1: HDFS Read: 1191080532 HDFS Write: 4452915 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
OK
```

Gambar 4.6 proses map reduce hive

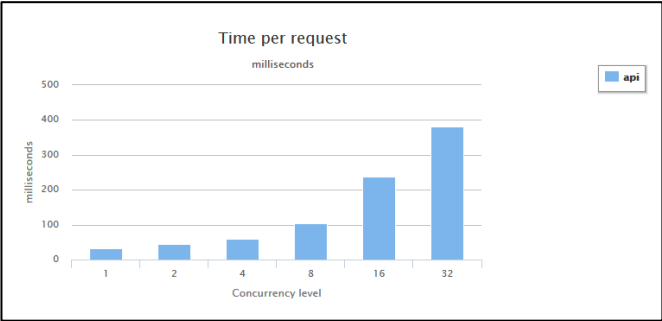
Sedangkan pada gambar .. adalah proses map reduce ketika suatu *hive* menjalankan suatu query . Karena pada praktiknya jumlah sensor atau agent lebih dari satu maka performas *REST API* juga sangat diperhatikan. Seperti jumlah *request per second* dan *time per request*. Berikut hasil benchmark dari performa *REST API*:



Gambar 4.7 request per second

Dari gambar diatas bahwa dalam satu detik bisa memproses hingga 25 request tiap detik sedangkan untuk melakukan insert data ke hive yang bekerja dengan metode *batch processing* yang membutuhkan waktu. Untuk mengatasi hal tersebut maka peran dari *publisher/subscriber* sangatlah penting untuk memproses *request* tersebut secara *background*.

Sedangkan untuk melihat waktu yang dibutuhkan untuk memproses sebuah *request* atau *time per request* ditunjukkan dengan gambar berikut:



Gambar 4.8 time per request

4.1.3. Testing Query geoip

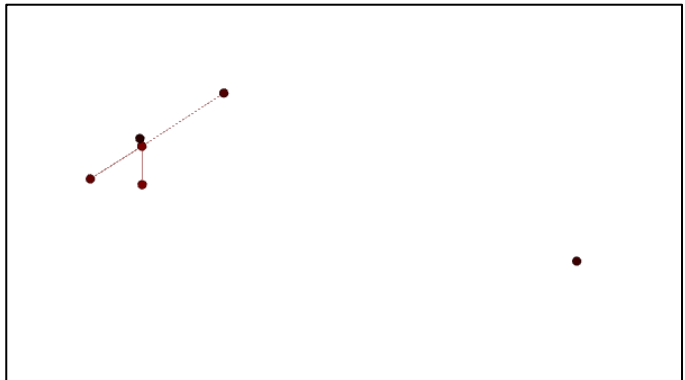
Dalam percobaan untuk digunakan dua cara yaitu menggunakan teknik join dengan database geoip dan yang kedua adalah menggunakan *HIVE UDTF* atau (*hive user defined table function*). Pada table dbip_lookup terdapat data ip sebanyak 6.896.087 baris data. Berikut table perbandingan pencarian *geoip* dengan *query join* dan *udf*.

Table 3 perbandingan query join dan UDTF

number of rows	UDTF	Query Join
1	0.086	134.229
10	0.086	452.485
100	0.08	3561.838
1000	0.099	n/a

4.1.4. Analisis geolocation menggunakan K-Means

Sedangkan pada percobaan kedua yaitu melakukan analisis terhadap *data log file* dari *snort* yang dikumpulkan dari sensor yang dipasang. Untuk metode analisisnya menggunakan *clustering k-means* untuk melihat *density* serangan berdasarkan *geoip* dari *attacker*. Penggunaan k-means ini bertujuan untuk mengurangi ukuran dari data set yang besar (4) dan juga tidak termasuk data grouping yang memiliki label sebelumnya



Gambar 4.9 k-means clustering geolocation jumlah $k=6$

Dari hasil percobaan pada gambar 6 dengan menjalankan k-means clustering dengan nilai k sebanyak 6. Maka dari dataset yang berisi ip dari attacker sehingga dapat di-cluster menjadi 6 cluster.

4.1.5. Analisa

Setelah melakukan pengujian maka dapat diambil analisa sebagai berikut :

1. Dari hasil pengujian penggunaan *rest api* sebagai *data collector* dari tiap-tiap agent yang mengirimkan data dari sensor memberikan performa yang cukup tinggi jika dibandingkan menggunakan *batch processing* secara langsung.
2. Penggunaan *join* untuk query geoip membutuhkan waktu yang lama karena adanya proses *cross product* dari kedua table yang di-join.
3. Penggunaan *clustering k-means* dengan *mahout* mampu meminimalkan dataset dari geolocation ip yang sangat besar menjadi beberapa cluster. Sehingga memudahkan visualisasi serangan daripada harus menampilkan semuanya menggunakan *query group by*.

BAB V

PENUTUP

5.1 KESIMPULAN

Setelah melakukan tahap perancangan dan pembuatan sistem yang kemudian dilanjutkan dengan tahap pengujian dan analisa maka dapat diambil kesimpulan bahwa:

1. aplikasi matagaruda yang diteliti berhasil mengatasi masalah dari ETL dari *hive* yang menggunakan *batch processing* menjadi *realtime processing*.
2. penggunaan *UDTF* berhasil meminimalkan proses query untuk *geolocation* dengan waktu dibawah satu detik daripada proses query dengan metode *join*.
3. dengan menggunakan library *apache mahout* pengolahan data dengan metode k-means terbukti sangat efektif untuk dataset yang besar karena menggunakan *map-reduce* sebagai komputasi paralel

SARAN

Dari hasil PA ini masih terdapat beberapa kekurangan dan dimungkinkan untuk pengembangan lebih lanjut. Oleh karenanya penulis merasa perlu untuk *memberi* saran-saran sebagai berikut :

1. Kemudian yaitu menambah jumlah *data source* dari *bro* dan *syslog*.
2. Membuat dokumentasi yang lengkap dari *REST API*.
3. Mencoba untuk melakukan *deep analysis* lebih lanjut

**** *Halaman ini sengaja dikosongkan*****

DAFTAR PUSTAKA

1. **Sived.** <https://sived.wordpress.com>.
<https://sived.wordpress.com>. [Online] 05 12, 2012.
<https://sived.wordpress.com/2012/05/12/mengenal-ids-dan-ips-dalam-keamanan-jaringan/>.
2. **Jason, Venner.** *Pro Hadoop*. United States of America : Apress, 2009.
3. **Noll, Michael G.** *Running Hadoop On Ubuntu Linux (Single-Node Cluster and Multi-Node Cluster)*. Germany : University of Luxembourg, 2012.
4. **Boedy, Cged.** Pengertian Apache Hadoop. *Pengertian Apache Hadoop*. [Online] 05 2012. [Cited: 04 20, 2015.]
<http://cgeduntuksemua.blogspot.com/2012/04/pengertian-apache-hadoop.html>.
5. **hive apache.** *hive. hive*. [Online] 2015.
<https://hive.apache.org/>.
6. **apache.** Apache Mahout. *Apache Mahout*. [Online] [Cited: 04 20, 2015.] <https://mahout.apache.org>.
7. Mata Garuda. *Mata Garuda*. [Online] 06 14, 2014. [Cited: 04 20, 2015.]
<http://julismail.staff.telkomuniversity.ac.id/mata-garuda/>.
8. **D. Hardt, Ed.** The OAuth 2.0 Authorization Framework. *tools.ietf.org*. [Online] july 21, 2012. [Cited: 07 27, 2015.]
<https://tools.ietf.org/html/draft-ietf-oauth-v2-31>.

9. **YOSRINAL.** *PERANCANGAN DAN IMPLEMENTASI RESOURCE SERVER*. 2014 : PROGRAM STUDI EKSTENSI S1 ILMU KOMPUTER, 2013.

10. **Riyadi, Damar.** *RANCANG BANGUN REST WEB SERVICE UNTUK PERBANDINGAN*. YOGYAKARTA : SEKOLAH TINGGI MANAJEMEN INFORMATIKA DAN KOMPUTER, 2013.

11. **json.org.** json. *json.org*. [Online] 2015. [Cited: 12 31, 2015.] <http://www.json.org/json-id.html>.

12. **wikipedia.** yaml. *wikipedia*. [Online] 2015. [Cited: 12 31, 2015.] <https://id.wikipedia.org/wiki/YAML>.

13. *MATATABI: Multi-layer Threat Analysis Platform.* **Tazaki, Hajime.** Tokyo : The University of Tokyo.

14. *Comparison between JSON and YAML for data.* **ERIKSSON, MALIN.** 2011.