IMPLEMENTASI PENGOLAHAN DATA MENGGUNAKAN FRAMEWORK HADOOP PADA SERVER VIRTUAL

IMPLEMENTATION OF DATA PROCESSING USING HADOOP FRAMEWORK ON VIRTUAL SERVER

MUSTIAR RAWI
2013.13.0043

PRODI MAGISTER SISTEM KOMPUTER
PASCASARJANA STMIK HANDAYANI
MAKASSAR
2017

TESIS

IMPLEMENTASI PENGOLAHAN DATA MENGGUNAKAN FRAMEWORK HADOOP PADA SERVER VIRTUAL

Disusun dan diajukan oleh:

MUSTIAR RAWI
2013.13.0043

Telah dipertahankan di depan Dewan Penguji Ujian Tesis

Pada hari **Kamis, 18 Oktober 2017**

Dan dinyatakan telah memenuhi syarat

Menyetujui,

Komisi Penasihat,

Dr. Eng. Agussalim, M.T	Dr. Abdul Latief Arda, M.Si., M.Kom

Mengetahui,

Ketua Program Studi S2 Sistem Komputer,	Direktur Program Pascasarjana, STMIK Handayani,
Dr. Ir. Zahir Zainuddin, M.Sc.	**Dr. Eng. Yuyun, S.Kom., M.T**

PERNYATAAN KEASLIAN TESIS

Yang Bertanda Tangan Di bawah Ini:

NAMA	: MUSTIAR RAWI
NPM	: 2013. 13.0043
PROGRAM STUDI	: SISTEM KOMPUTER

Menyatakan dengan sebenarnya bahwa tesis yang saya tulis ini benar merupakan hasil karya saya, bukan pengambilalihan tulisan atau pemikiran orang lain, adapun kutipan atau rujukan sebagai sumber yang saya gunakan dari penulisan orang lain, telah saya sebutkan pada daftar pustaka tesis ini. Apabila di kemudian hari terbukti dan telah memiliki kekuatan hukum yang sah dari lembaga yang berwenang bahwa tesis ini adalah hasil karya orang lain maka saya bersedia menerima sanksi sesuai peraturan yang berlaku.

Makassar, 04 Oktober 2017

Yang menyatakan,

MUSTIAR RAWI

ABSTRAK

Hadoop merupakan framework software berbasis Java dan open source yang berfungsi untuk mengolah data yang memiliki ukuran yang besar secara terdistribusi. Hadoop menggunakan sebuah framework untuk aplikasi dan programming yang disebut dengan MapReduce. Empat skenario diimplementasikan untuk menganalisa performa kecepatan MapReduce pada Hadoop. Berdasarkan hasil pengujian yang dilakukan diketahui penambahan jumlah virtual machine dari satu menjadi dua virtual machine dengan spesifikasi virtual machine yang sesuai perancangan dapat memperlambat kecepatan rata-rata MapReduce. Pada ukuran file 512 MB, 1 GB, 1.5 GB, dan 2 GB, penambahan virtual machine dapat memperlambat kecepatan rata-rata MapReduce pada masing-masing ukuran file sebesar 164.00, 504.34, 781.27, dan 1070.46 detik. Berdasarkan hasil pengukuran juga diketahui bahwa block size dan jumlah slot map pada Hadoop dapat mempengaruhi kecepatan MapReduce.

Kata Kunci: Hadoop, MapReduce, Block Size, Slot Map, Virtual Mesin.

ABSTRACT

Hadoop is a Java-based software framework and open source which is used to process the data that have a large size in a distributed manner. Hadoop uses a framework for application and programing which called MapReduce. Four scenarios are implemented to analyze the speed performance of Hadoop MapReduce. Based on the study, known that additional the number of virtual machines from one to two virtual machines with suitable specifications design can slow down the average speed of MapReduce. On file 512 MB, 1 GB, 1.5 GB, and 2 GB size, additional the number of virtual machines can slow down the average speed of each MapReduce on a file size for 164.00, 504.34, 781.27, and 1070.46 seconds. Based on the measurement result is also known that the block size and number of slot maps in Hadoop MapReduce can affect speed.

Keywords: Hadoop, MapReduce, Block Size, Slot Map, Virtual Machine

KATA PENGANTAR

Alhamdulillahi Rabbil Alaamiin, puji dan syukur penulis panjatkan kepada Allah SWT atas berkat rahmat dan ridho-Nya sehingga penulis dapat menyelesaikan tesis dengan judul **"IMPLEMENTASI PENGOLAHAN DATA MENGGUNAKAN FRAMEWORK HADOOP PADA SERVER VIRTUAL"** sebagai salah satu persyaratan dalam menyelesaikan Studi Strata Dua Program Pascasarjana Sistem Komputer, STMIK Handayani Makassar.

Penulis menyadari bahwa dalam penyusunan naskah ini tidak luput dari kesalahan dan masih terdapat banyak kekurangan serta masih jauh dari kesempurnaan, maka dari itu penulis mengharapkan saran dan kritik yang membangun dari semua pihak.

Penulis juga menyadari bahwa penyusunan naskah ini tidak lepas dari bantuan berbagai pihak, maka pada kesempatan ini sudah selayaknya penulis menyampaikan ucapan terima kasih dan penghargaan yang setinggi-tingginya kepada:

1. Orangtua saya dan saudara-saudari tercinta yang tiada hentinya memberikan saya nasihat, semangat dan doa dengan penuh kasih sayang selama saya menempuh pendidikan formal dan motivasi hidup dalam menggapai cita-cita saya
2. Bapak **Dr. Eng. Agussalim, M.T.,** selaku Ketua STMIK Handayani Makassar dan juga sebagai Penguji yang telah meluangkan waktunya untuk memberikan masukan, kritik dan saran dalam penyusunan tesis ini.
3. Ibu **Dr. Eng. Yuyun, S.Kom., M.T.** selaku Direktur Pascasarjana STMIK Handayani Makassar.
4. Bapak **Dr. Ir., Zahir Zainuddin, M.Sc.**, selaku Ketua Program Studi Pascasarjana STMIK Handayani Makassar dan juga sebagai pembimbing yang telah meluangkan waktunya untuk membimbing, mengarahan penulis dalam penyusunan tesis ini.
5. Bapak **Dr. Eng. Agussalim, M.T**, selaku Pembimbing yang telah meluangkan waktunya untuk membimbing, mengarahkan, dan memberikan dukungan kepada penulis dalam penyusunan tesis ini.

6. Bapak **Dr. Abdul Latief Arda, S.Kom., M.Si., M.Kom,** selaku penguji yang telah memberikan kritik dan saran yang sangat membangun kepada penulis dalam penyusunan tesis ini.
7. Bapak **Dr. Eng. Wardi, ST, M.Eng,** selaku penguji yang telah memberikan kritik dan saran yang sangat membangun kepada penulis dalam penyusunan tesis ini.
8. Bapak dan Ibu Dosen, Program Studi Sistem Komputer, STMIK Handayani Makassar yang telah memberikan ilmunya kepada penulis selama perkuliahan.
9. Teman-teman seangkatan Pascasarjana STMIK Handayani selama masa perkuliahan yang telah menjadi tempat berbagi canda tawa sampai akhir penyusunan tesis selalu bersama penulis. Terima kasih atas kerjasama, waktu, dan bantuan yang diberikan kepada penulis selama masa perkuliahan dan penyusunan tesis ini.
10. Semua pihak yang telah banyak berpartisipasi, baik secara langsung maupun tidak langsung yang tak sempat penulis sebutkan satu per satu. Terima kasih untuk segala bantuan dan dukungannya.

Semoga *Allah SWT*., senantiasa melimpahkan karunia-Nya dan membalas segala amal budi serta kebaikan pihak-pihak yang telah membantu penulis dalam penyusunan tesis ini.

Akhir kata, semoga tulisan ini dapat memberikan manfaat bagi pihak-pihak yang membutuhkan

Makassar, Oktober 2017

Mustiar Rawi

DAFTAR ISI

BAB I

PENDAHULUAN

A. Latar Belakang

Untuk mengantisipasi berkembangnya kompleksitas data di masa mendatang dalam rangka menjaga daya saing, diperlukan suatu solusi mutakhir untuk dapat mendukung perkembangan pertumbuhan data tanpa mengorbankan operasional dan investasi yang besar. Mengingat perkembangan industri TI yang pesat telah menjadi peran yang sangat penting untuk pertukaran informasi yang cepat, untuk itu beberapa perusahaan memutuskan untuk mengadopsi teknologi Big Data sebagai langkah awal untuk mengantisipasi perkembangan data dan mendukung kebutuhan analitik atau perkembangan variasi fitur di masa mendatang.

Big Data telah teruji dalam perusahaan-perusahaan berkaliber tinggi di berbagai negara maju untuk mendukung efisiensi dalam operasional dan mempercepat kemampuan mengolah data dengan hasil berlipat ganda. Big Data tidak memerlukan infrastruktur yang kompleks (seperti sistem legacy DBMS komersil), melainkan hanya memerlukan tipe server dan penyimpanan berjenis komoditas. Dengan demikian, daya saing organisasi diyakinkan akan terjaga disektor publik yang akan memberikan layanan terbaik.

Dari beberapa penelitian sebelumnya yang tentang pengolahan Big data, pada umumnya hanya bersifat simulasi dan bukan implementasi langsung di lapangan, karena tidak menggunakan data dan monitoring proses secara real time. Atas pertimbangan tersebut peneliti menganggap isu tersebut layak diangkat sebagai bahan penelitian, sehingga nantinya dengan penelitian ini akan dibangun sistem hadoop untuk pemrosesan Big data dan analisis proses yang sedang berjalan disistem tersebut.

Diharapkan dengan penerapan hadoop dapat meningkatkan kinerja server dalam menyelesaikan suatu proses.

B. Perumusan Masalah

Berdasar latar belakang yang dimaksudkan di atas, maka dapat di simpulkan sebuah permasalahan yaitu bagaimana mengimplementasikan hadoop dan mapreduce pada server virtual mesin.

C. Tujuan Penelitian

Tujuan dilaksanakan penelitian ini adalah:

1. Mengimplementasikan sebuah program untuk mengolah data aliran paket TCP pada sebuah jaringan yang dapat dijalankan secara terditribusi oleh Hadoop,
2. Percobaan terhadap empat skenario berdasarkan topologi yang telah dirancang,
3. Melakukan analisis pengaruh virtual node, block size dan jumlah slot map terhadap kecepatan MapReduce pada Hadoop.

D. Manfaat Penelitian

Hasil yang diharapkan dari penelitian ini adalah :

1. Dokumentasi konsep dan hasil implementasi teknologi Hadoop dan Mapreduce.
2. Memberikan solusi penghematan biaya infrastruktur jaringan komputer.

E. Batasan Masalah

Ruang lingkup dari pembahasan penelitian ini terbatas pada hal-hal berikut ini:

1. Membahas Implementasi dan performa kecepatan MapReduce pada Hadoop dalam mengolah data aliran paket TCP pada sebuah jaringan.
2. Perancangan yang dibuat adalah dengan menjalankan Hadoop secara single node dan multi node (cluster).

3. Virtual machine yang digunakan memiliki spesifikasi CPU one core, RAM 1 GB dengan sistem operasi Ubuntu 14.04 yang berjalan diatas Virtualbox 5.4.1 pada sebuah laptop yang memiliki spesifikasi CPU Core i5, RAM 4 GB dengan sistem operasi Windows 8.1.
4. Ukuran file yang digunakan untuk di proses oleh MapReduce dibatasi hanya menggunakan lima ukuran file yang berbeda yaitu 512 MB, 1 GB, 1.5 GB, dan 2 GB.
5. Block size yang digunakan dibatasi hanya menggunakan ukuran 32 MB, 64 MB, 128 MB, dan 256 MB.
6. Jumlah slot map dibatasi hanya menggunakan 2 slot map, 4 slot map, 6 slot map, dan 8 slot map.

BAB II

TINJAUAN PUSTAKA

A. Pengertian Big Data

Big data adalah istilah yang digunakan untuk mendefinisikan perkembangan data secara eksponensial dan mekanisme untuk menyimpan data terstruktur maupun tidak terstruktur. Big data didefinisikan atas dasar 3-V yaitu volume, velocity (kecepatan), dan variety (variasi):

1. Volume – Faktor yang berkontribusi terhadap peningkatan volume data mencakup data berbasis transaksi disimpan selama bertahun-tahun, aliran data terstruktur, peningkatan jumlah data server, dan lainnya. Volume data yang berlebihan adalah sumber masalah dari penyimpanan yang berdampak terhadap biaya penyimpanan. Menentukan relevansi dalam volume data yang besar dan menggunakan analisis untuk menciptakan nilai data yang relevan dengan sumber daya yang terbatas menjadi kendala utama dari penyimpanan.

2. Velocity (kecepatan) – Diperlukan solusi untuk memecahkan faktor streaming data dengan kecepatan tinggi yang harus ditangani secara tepat waktu atau bahkan real-time. Salah satu tantangan terbesar bagi entitas yang menyimpan data berskala atau bervolume besar adalah menangani kecepatan untuk mengakses data tanpa mengurangi kapasitas sumber daya pemrosesan.
3. Variety (variasi) – Dengan berkembangnya teknologi dan kebutuhan data yang berbeda pada setiap entitas memerlukan jenis format yang berbeda untuk menyimpan informasi yang spesifik terhadap aplikasi atau spesialisasi dari entitas tersebut. Format ini pada dasarnya terdiri dari data terstruktur maupun data tidak terstruktur. Data terstruktur adalah data numerik dalam database tradisional. Data tidak terstruktur adalah berupa text, gambar, audio, video, dan data transaksi keuangan. Diperlukan solusi untuk mengelola penggabungan dan mengatur variasi data yang berbeda untuk menangani tantangan untuk penyimpanan data berskala besar.

B. Pengertian Hadoop

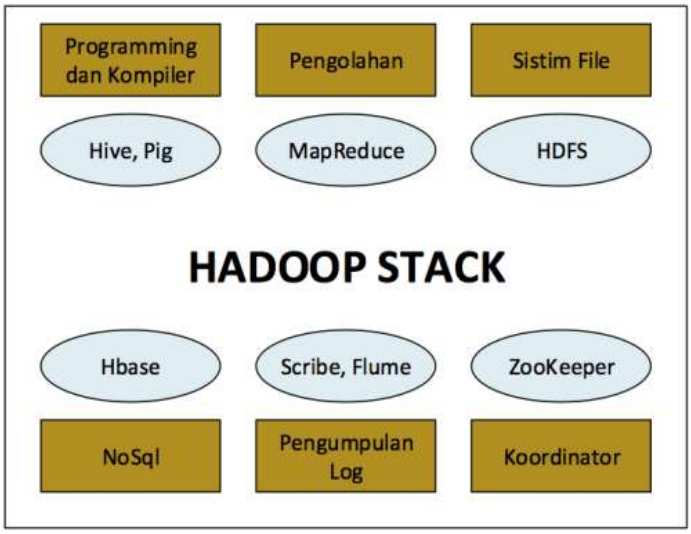
Hadoop adalah framework database yang mempunyai kemampuan untuk pendistribusian pengolahan dataset berskala besar yang tersimpan dalam sekelompok komputer yang terhubung dalam suatu jaringan atau algoritma khusus. Keunggulan Hadoop adalah fleksibilitas yang dapat menerima berbagai macam data dan tidak ada batasan penyimpanan. Framework Hadoop mempunyai dua struktur utama yang terdiri dari MapReduce dan Hadoop Distributed File System (HDFS).

C. Komponen Hadoop

1. **MapReduce** adalah software pengolahan dibalik kemampuan Hadoop dalam pendistribusian data sesuai dengan algoritma yang telah ditentukan untuk mengurangi redundansi. Konsep dasar dari teknologi MapReduce adalah dengan menggunakan pemetaan (Map) dan pengelompokan (Reduce). MapReduce mempunyai keunggulan sebagai berikut:
   - Skalabilitas yang memungkinkan untuk beroperasi untuk pemetaan dan pengelompokan secara paralel.
   - Toleransi terhadap error yang rendah dengan kemampuan untuk replikasi data dalam HDFS secara otomatis yang dapat mendeteksi kegagalan secara otomatis untuk pengulangan tanpa mengganggu kegiatan komputasi.
   - MapReduce menggunakan struktur yang efisien sehingga berdampak terhadap biaya yang lebih efisien.
2. **HDFS** adalah 14sistem file dalam Hadoop framework yang bersifat distributed, scalable, dan portable. Hadoop menyimpan data bedasarkan node, dan setiap kumpulan node adalah cluster dalam HDFS. Manfaat dari penyimpanan berformat HDFS adalah dapat menyimpan file bervolume besar (berskala gigabyte sampai dengan terabyte) dalam server yang berbeda, dan terjamin kehandalannya dengan replikasi data dalam host yang beragam.
3. **Hive** berfungsi sebagai penghubung Hadoop untuk kepentingan pengolahan log, mencarian teks, indexing dokumen, predictive modeling, hypothesis testing, dan menjalankan fungsi business intelligence. Hive dapat mendukung proses query secara ad-hoc dengan cepat dengan menjalankan high level yang dapat menyimpulkan join yang kompleks.
4. **Pig** adalah high-level platform dan programming berbasis teks untuk memberikan perintah kepada MapReduce dalam eksekusi tugas yang diperintahkan kedalam

Hadoop. Pig memberikan kemudahan dalam mengelola sistem MapReduce secara eksplisit mengendalikan aliran dan pengolahan data.

5. **Hbase** adalah 15sistem manajemen untuk menulis dan membaca data berskala besar yang terdistribusi dalam framework Hadoop. Hbase menggunakan bahasa programming Java untuk eksekusi dan berjalan sebagai komplementer HDFS.
6. **ZooKeeper** berfungsi sebagai koordinator untuk distribusi file sistem Hadoop dalam meminimalisir kemungkinan untuk gagal dalam penyimpanan atau pendistribusian. ZooKeeper menyediakan layanan untuk pengelolaan konfigurasi, message queue, notifikasi, sinkronisasi, dan penamaan registry untuk sistem yang terdistribusi.



Gambar 2.1 Arsitektur Hadoop

D. Kemampuan Analitik dalam Big Data

Tantangan dari penyimpanan data berskala besar adalah bukan dari mendapatkan data yang mempunyai volume besar namun dari segi pengolahan data berskala besar. Salah satu manfaat dengan menggunakan Big Data adalah memiliki kemampuan untuk menjalankan aplikasi analitik dalam framework yang sama tanpa mengurangi sumber daya komputasi. Big Data Analitik adalah proses penelitian dalam sumber daya data untuk mengungkap pola, mengetahui korelasi, analisa trend, dan membuka informasi lainnya yang dapat bermanfaat bagi pengguna informasi dari data yang terkumpul. Proses analisa dilakukan oleh data scientist dan ahli data lainnya yang dapat menganalisa data berskala besar diluar kemampuan solusi business intelligence dan sistem analitik konvensional.

Sistem Big Data Analitik mempunyai keunggulan untuk mengolah data secara langsung dari data yang disimpan dalam HDFS dengan menggunakan algoritma data mining, text mining, forecasting dan optimasi dari big data. Pada dasarnya, model sederhana untuk metode analitik yang dimasukkan kedalam data berskala besar melebihi kemampuan analitik itu sendiri. Dalam melakukan proses analitik dalam big data memerlukan kemampuan untuk mencegah agregasi, mengurangi pergerakan data

dan replikasi, dan optimasi pengolahan secara efisien. Analitik dalam big data mempunyai karakteristik dan kemampuan untuk pengolahan secara batch, bekerja secara independen dan paralel pada setiap data slice.

E. Roadmap Penelitian

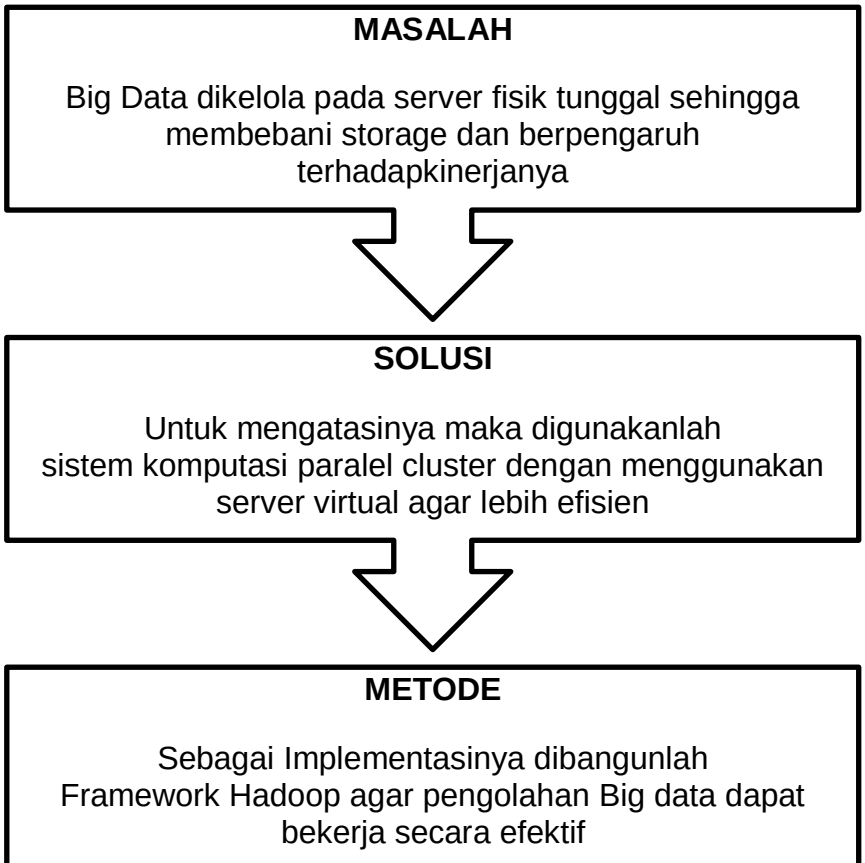
Penelitian yang paling terkait tentang Hadoop dan Big Data yang pernah dilakukan sebelumnya yaitu penelitian [3] dan [7].

Penelitian [1] membangun suatu algoritma rekursif dengan metode Learning Machine yang dikolaborasi dengan Hadoop dalam penyimpanan Big data. Hasilnya menetukan seberapa besar ukuran setiap file dan jumlah pecahannya yang dapat tersimpan di server.

Penelitian [7] menganalisis kinerja dari Hadoop Map Reduce dalam pengolahan Big data secara komputasi paralel. Hasilnya berupa penentuan seberapa besar tingkat efektifitas komputasi yang terjadi di setiap server node yang indikatornya berdasarkan parameter kecepatan setiap proses dan ukuran file yang tersimpan.

F. Kerangka Pikir Penelitian

Kerangka pikir penelitian ini adalah sebagai berikut :

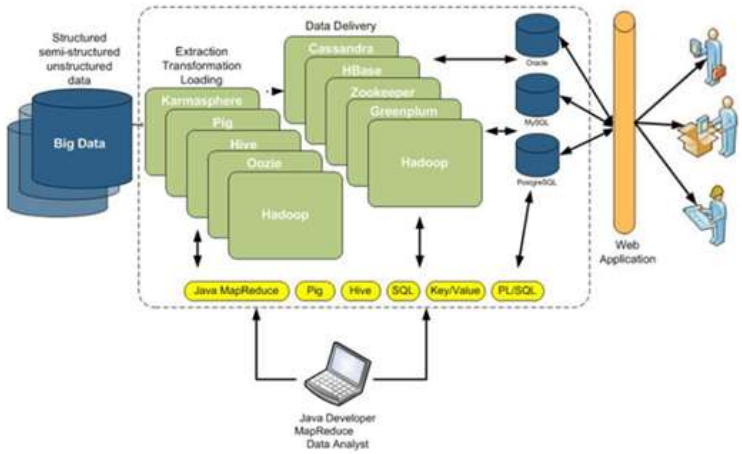


BAB III

METODE PENELITIAN

A. Rancangan Penelitian

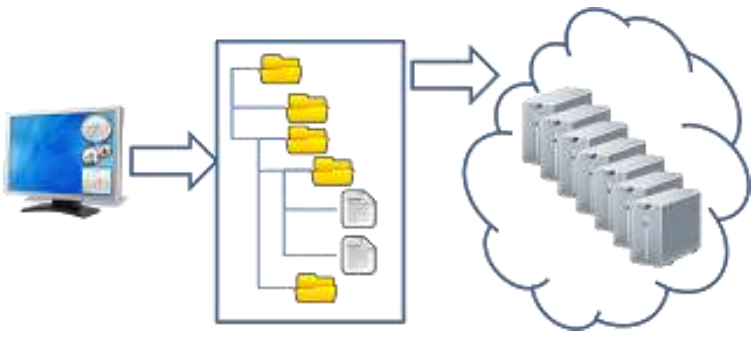
Penelitian ini berfokus untuk membangun sistem framework Hadoop untuk mengolah penyimpanan dan pembacaan Big data pada infrastruktur jaringan yang sudah berjalan. Metode penelitian yang digunakan adalah *System Development Life Cycle (SDLC)* meliputi tahap perencanaan, analisis, rancangan, penerapan dan penggunaan. Sedangkan rancangan kerja sistem secara umum dapat dilihat pada gambar berikut :



Gambar 3.1 RancanganKerja Framework Hadoop

1. Model Data dan Struktur HDFS

HDFS menyimpan suatu data dengan cara membelahnya menjadi potongan-potongan data yang berukuran 64 MB, dan potongan-potongan data ini kemudian disimpan tersebar dalam komputer-komputer yang membentuk clusternya. Potongan-potongan data tersebut dalam HDFS disebut block, dan ukurannya tidak terpaku harus 64 MB. Ukuran block dapat diatur sesuai kebutuhan dan selera.Walaupun data disimpan secara tersebar, dari kacamata client / pengguna, data tetap terlihat seperti halnya kita mengakses file pada satu komputer. File yang secara fisik disimpan tersebar dalam banyak komputer itu pun dapat diperlakukan layaknya memperlakukan file dalam satu komputer.



Gambar 3.2 Mengakses data pada HDFS

Sebagai distributed file system, HDFS memiliki komponen-komponen utama berupa NameNode dan DataNode. NameNode adalah sebuah komputer yang

bertindak sebagai master, sedangkan DataNode adalah komputer-komputer dalam Hadoop Cluster yang bertugas sebagai slaves atau anak buah. NameNode bertanggungjawab menyimpan informasi tentang penempatan block-block data dalam Hadoop Cluster. Ia bertanggungjawab mengorganisir dan mengontrol block-block data yang disimpan tersebar dalam komputer-komputer yang menyusun Hadoop Cluster. Sedangkan DataNode bertugas menyimpan block-block data yang dialamatkan kepadanya, dan secara berkala melaporkan kondisinya kepada NameNode. Laporan berkala DataNode kepada NameNode ini disebut Heartbeat. Berdasarkan Heartbeat ini NameNode dapat mengetahui dan menguasai kondisi cluster secara keseluruhan. Sebagai balasan atas Heartbeat dari DataNode, NameNode akan mengirimkan perintah kepada DataNode. Jadi, dalam HDFS, NameNode adalah boss yang mengatur dan mengendalikan atau me-manage Hadoop Cluster. Sedangkan, DataNode adalah pekerja atau karyawan yang bertugas menyimpan data dan melaksanakan perintah dari NameNode.

2. Prosedur Menyimpan dan Membaca Data dalam HDFS

Untuk menyimpan data ke dalam HDFS, kita harus memiliki satu komputer yang sudah terhubung dengan Hadoop Cluster. Komputer ini disebut sebagai Client. Cukup dengan mengetikkan kata-kata perintah pada Command Prompt komputer client, maka file akan ditransfer ke Hadoop cluster dan disimpan tersebar dalam komputer-komputer yang ditugaskan sebagai DataNode. Secara detailnya, pada saat kita mengeksekusi perintah penyimpanan, komputer client akan berkomunikasi dengan NameNode untuk memberitahu bahwa ada file yang hendak disimpan di HDFS dan menanyakan lokasi DataNode yang bisa diakses untuk menyimpan data tersebut. Setelah mendapat daftar nama dan alamat DataNode yang tersedia, komputer client akan secara langsung mentransfer data ke DataNode

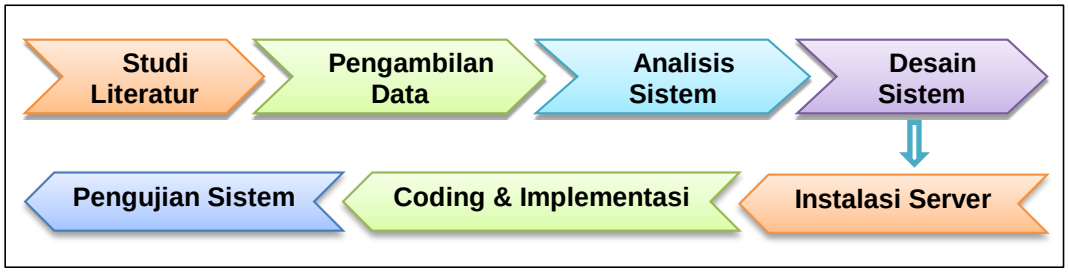
- DataNode tersebut. Data yang akan ditransfer tersebut tentunya secara otomatis sudah dibelah menjadi block-block data yang berukuran 64 MB atau sesuai setting. Block-block atau kepingan-kepingan data inilah yang disimpan oleh tiap DataNode. Kemudian, setelah mendapat kepingan data dan menyimpannya, tiap DataNode akan mengirimkan laporan kepada NameNode bahwa data sudah diterima dan disimpan normal.

Gambar 3.3 Menyimpan data ke dalam HDFS

Sedangkan untuk membaca data, prosedurnya mirip dengan prosedur menyimpan data. Cukup dengan mengeksekusi kata-kata perintah pada Command Prompt komputer client, maka data dalam HDFS akan ditampilkan pada display komputer client. Detailnya adalah sebagai berikut. Pada saat kita mengeksekusi perintah membaca data, komputer client akan berkomunikasi dengan NameNode menanyakan nama dan alamat DataNode yang harus diakses untuk mendapatkan data yang diinginkan. Setelah mendapat informasi tersebut, komputer client akan secara langsung mengakses DataNode yang bersangkutan untuk mendapatkan data yang diinginkan. Kemudian data akan ditampilkan pada display komputer client, atau sesuai perintah pengguna.

Gambar 3.4 Membaca data yang tersimpan dalam HDFS

B. Tahapan Penelitian



Gambar 3.2 Tahapan Penelitian

Secara umum tahapan penelitian akan dilakukan sebagai berikut :

1. Studi Literatur, yaitu dengan melakukan studi dari buku-buku yang berkaitan dengan masalah, juga melalui artikel dari internet.
2. Metode pengambilan data yaitu observasi dan wawancara langsung dengan pihak yang berkaitan.
3. Analisis sistem adalah tahap mencari kelemahan dari sistem, sehingga diusulkan perbaikan pada sistem yang akan dikembangkan.
4. Desain sistem merupakan tahap merancang sistem yang akan dibangun.
5. Instalasi server berdasarkan kebutuhan fungsinya masing-masing pada desain sistem.
6. Coding dan implementasi adalah tahap mengimplementasikan rancangan sistem ke dalam kode program.
7. Pengujian sistem merupakan tahap pengimplementasian terhadap sistem yang telah dibuat serta menguji apakah sistem telah sesuai dengan tujuan yang diinginkan atau masih perlu diperbaiki.

C. Perangkat Pengembangan Sistem

Berikut ini kebutuhan sistem dari perancangan hingga instalasi:

1. Perangkat keras pengembangan
   - Komputer Server Dual Core RAM 2 GB.

- Laptop (Client)
- Switch / Access Point
- Modem
- Kabel UTP

2. Perangkat Lunak Pengembangan
   - OS Linux Ubuntu 14.04 (Trusty)
   - Apache Hadoop 2.6.0
   - Virtualbox 5.4.1
   - Java NetBeans 7.0
   - Putty
   - Cacti
   - Browser
3. Kebutuhan non perangkat keras dan lunak
   - Bandwith
   - IP Publik
   - DNS

D. Waktu dan Tempat Pelaksanaan Penelitian

Lokasi dan tempat penelitian dilakukan di Lab Komputer Kampus Politeknik Negeri Ujung Pandang, kota Makassar dan dilaksanakan selama kurang lebih 4 (empat) bulan sejak persetujuan proposal penelitian ini diterima.

E. Metodologi Penelitian

Metode yang digunakan dalam membantu penulisan tesis ini yaitu:

1. Studi literatur, yaitu pencarian jurnal-jurnal ilmiah, pencarian buku dan sumber referensi lainnya. Kemudian mempelajari dan memahami konsep Hadoop baik secara single node maupun multi node dan konsep MapReduce yang berjalan pada Hadoop.
2. Konsultasi kepada pembimbing mengenai perancangan dan skenario yang akan dilakukan.
3. Melakukan percobaan dari perancangan dan skenario yang telah dibuat.
4. Melakukan pengamatan, pengambilan data, analisis dan penarikan kesimpulan dari percobaan yang dilakukan.

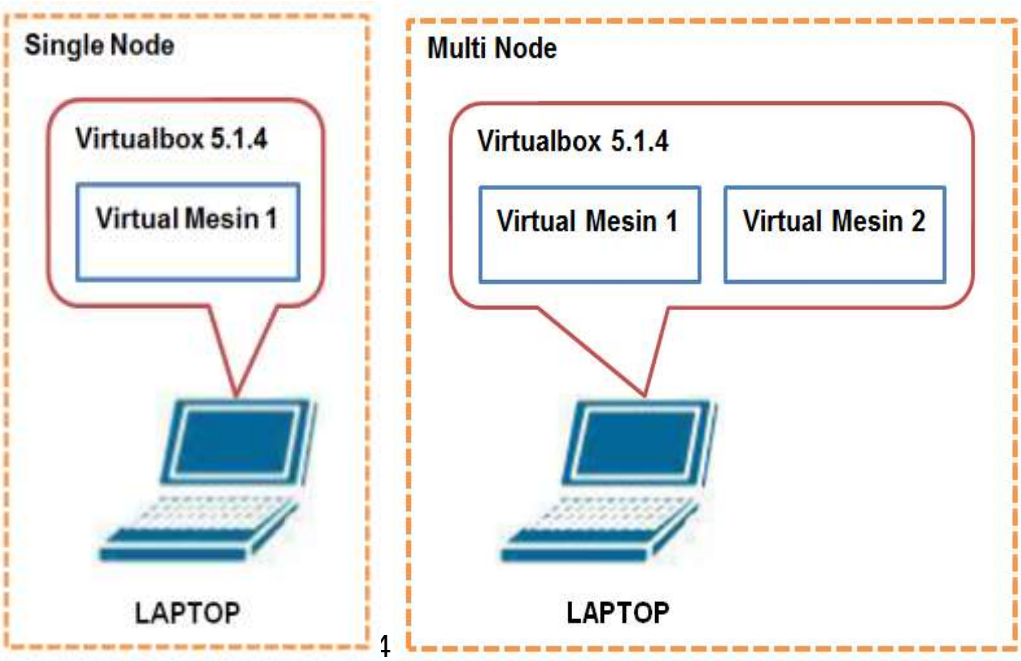
BAB IV

HASIL DAN PEMBAHASAN

A. Hasil Penelitian

1. Desain Topologi

Desain sistem yang dirancang dalam penelitian ini menggunakan sebuah software yang mendukung laptop untuk melakukan virtualisasi yaitu Virtualbox 5.1.4. Virtualbox 5.1.4 diinstall pada sebuah laptop yang menjalankan sebuah sistem operasi Ubuntu 14.04 dan kemudian laptop tersebut menjalankan satu virtual machine untuk single node dan dua virtual node untuk multi node. Topologi sistem dapat dilihat pada Gambar 4.1.



(a) single node (b) multi node

Spesifikasi laptop (host) dan masing-masing virtual machine yang dijalankan baik secara single node maupun multi node (cluster) dapat dilihat pada Tabel 4.1.

Tabel 4.1 Spesifikasi host dan virtual machine

Hardware	***Host***	***VM 1***	***VM 1***
CPU	Intel Core i5, Clock Speed 2.2 GHz	One Core	One Core
RAM	4.00 GB	1.00 GB	1.00 GB
Sistem Operasi	Windows 8.1	Ubuntu 14.04	Ubuntu 14.04

Untuk keperluan pengambilan data dalam menguji performa kecepatan sistem yang dirancang, maka dibuatlah skenario pengujian menggunakan program yang menggunakan konsep MapReduce yang bernama TCP Packet Flow Analysis yang

berfungsi untuk menganalisa data aliran paket TCP dan protokol yang terdapat pada sebuah jaringan.

TCP Packet Flow Analysis adalah sebuah program yang menggunakan konsep MapReduce dalam menganalisa aliran paket yang terdapat pada sebuah jaringan menggunakan Hadoop. File yang diproses oleh program ini adalah file dengan format pcap yang telah dikonversi ke dalam bentuk plaintext. Dalam penelitian ini, untuk mengkonversi file dengan format pcap ke dalam bentuk plaintext digunakan tool yang bernama tshark.

Command line yang digunakan untuk mengkonversi file dari format pcap menjadi bentuk plaintext dapat dilihat pada Gambar 4.1.

```
tshark -T fields -n -r inputdata.pcap -e frame.time -e tcp.len -e ip.src -e tcp.srcport
-e ip.dst -e tcp.dstport -e col.Protocol > outputdata.txt
```

Gambar 4.2 Command line tshark untuk mengkonversi file dari format pcap ke dalam bentuk plaintext

Command line pada Gambar 4.2 berfungsi untuk menyeleksi bagian timestamp, length (size in bytes), source IP address, source TCP port, destination IP address, destination TCP port, dan protocol pada file dengan format pcap yang kemudian disimpan ke dalam bentuk plaintext. Contoh hasil konversi dari command line yang terdapat pada Gambar 4.1 dapat dilihat pada Gambar 4.3.

```
Aug  5, 2007 04:49:45.113970000     196.139.120.174     71.126.222.64       ICMP
Aug  5, 2007 04:49:45.113976000     194.231.76.215      71.126.222.64       ICMP
Aug  5, 2007 04:49:45.113981000     192.132.66.87       71.126.222.64       ICMP
Aug  5, 2007 04:49:45.113987000 0   192.87.52.192   63781   71.126.222.64   25601   TCP
Aug  5, 2007 04:49:45.113995000     39.89.39.121        71.126.222.64       ICMP
Aug  5, 2007 04:49:45.114001000     203.149.28.189      71.126.222.64       ICMP
Aug  5, 2007 04:49:45.114005000     203.149.28.189      71.126.222.64       ICMP
Aug  5, 2007 04:49:45.114009000     203.149.28.189      71.126.222.64       ICMP
Aug  5, 2007 04:49:45.114013000 0   202.223.248.93  41821   71.126.222.64   37528   TCP
Aug  5, 2007 04:49:45.114018000     192.177.214.246     71.126.222.64       ICMP
Aug  5, 2007 04:49:45.114022000     198.254.6.110       71.126.222.64       ICMP
Aug  5, 2007 04:49:45.114027000 0   98.31.185.143   20960   71.126.222.64   44420   TCP
Aug  5, 2007 04:49:45.114032000     194.9.200.169       71.126.222.64       ICMP
Aug  5, 2007 04:49:45.114038000     250.242.57.128      71.126.222.64       ICMP
```

Gambar 4.3 Contoh hasil konversi kedalam bentuk plaintext

TCP Packet Flow Analysis ini cocok dijalankan pada Hadoop karena program ini menggunakan konsep MapReduce sehingga dapat dijalankan pada satu komputer maupun beberapa komputer (cluster). Pseudocode dari TCP Packet Flow Analysis dapat dilihat pada Gambar 4.4.

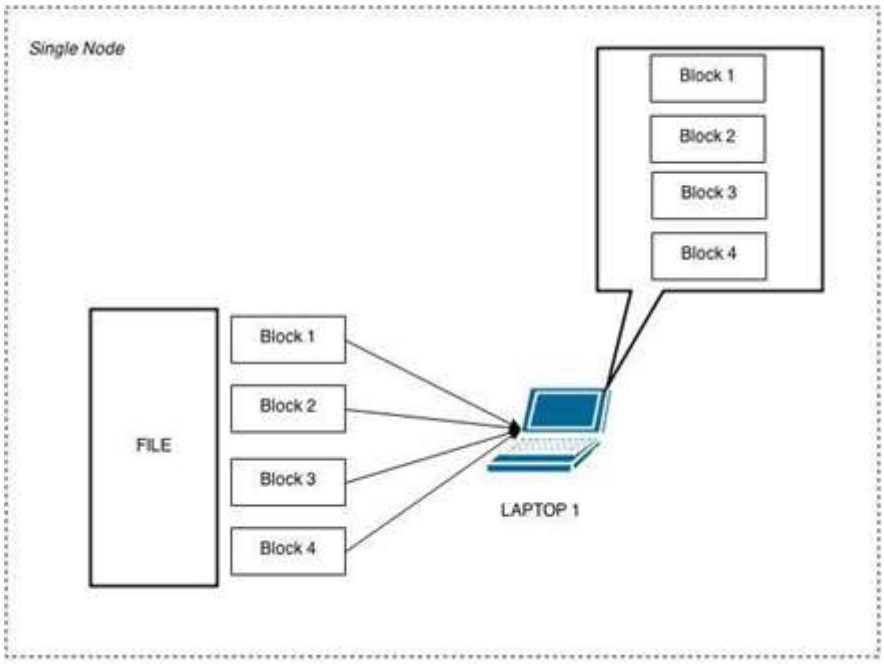
```
Map:
reads the input line by line
split a string into separate "hasilPecahInput" if:
hasilPecahInput[2] > 1
   create a token for hasilPecahInput[2] and hasilPecahInput[4]
   while: token available from the hasilPecahInput[2] and hasilPecahInput[4]
         set keyword: Koneksi "hasilPecahInput[2]" dan "hasilPecahInput[4]" Protokol
"hasilPecahInput[6]" else: set keyword : Koneksi protokol "hasilPecahInput[6]"
set lengthValue from value of hasilPecahInput[1]
create a pair <keyword,"Packet", one>
create a pair <keyword,"Length", lengthValue>

Reduce:
split pair into separate "splitKey" with comma is
the separator set splitKey[0] to keyOut
if: spiltKey[1] = 'Packet'
   increment sumPacket
   set sumPacket as outputValue
   display ("PacketTotal", keyOut,
outputValue) else : increment sumLength
      set PacketLength as outputValue
      display ("PacketLength", keyOut, outputValue")
```

Gambar 4.4 Pseudocode TCP Packet Flow Analysis

2. Skenario Implementasi dan Pengujian Sistem

a. Skenario Pertama

Skenario pertama bertujuan untuk menganalisis pengaruh block size terhadap kecepatan MapReduce pada Hadoop yang dijalankan secara single node. Percobaan akan dilakukan menggunakan ukuran file yang bervariasi yaitu 512 MB, 1 GB, 1.5 GB, dan 2 GB dengan menggunakan ukuran block size yang bervariasi untuk setiap ukuran file yaitu 32 MB, 64 MB, 128 MB, dan 256 MB. Gambaran pembagian block size pada skenario ini dapat dilihat pada Gambar 4.5.



Gambar 4.5 Pembagian block size pada Hadoop yang dijalankan secara single node

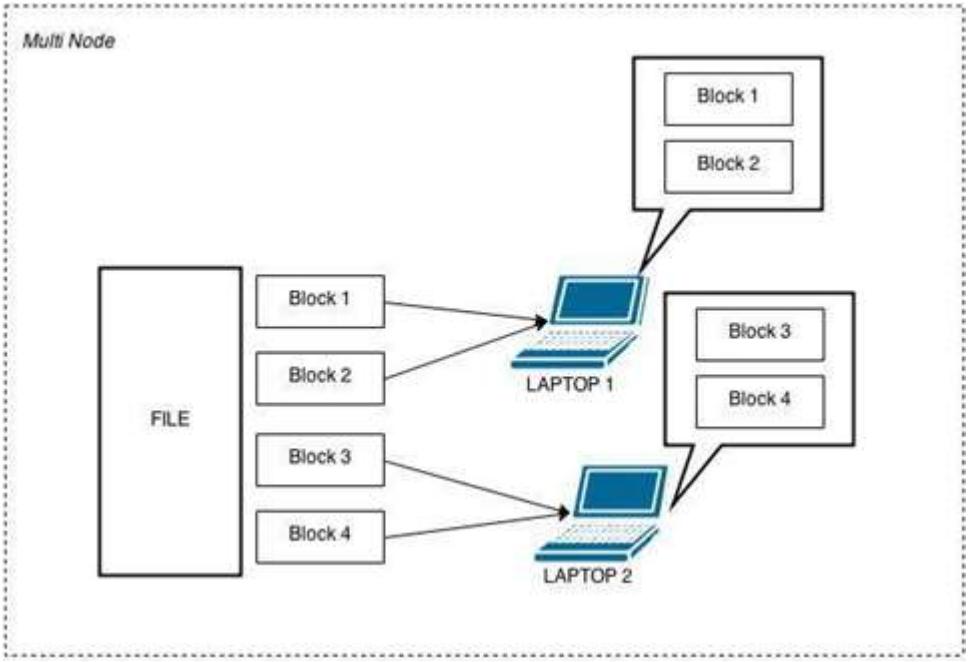
Gambar 4.5 menunjukan gambaran Hadoop dalam membagi sebuah file yang memiliki ukuran besar menjadi beberapa block. Blocks tersebut merupakan potongan-potongan file yang memiliki ukuran kecil yang berasal dari file yang memiliki ukuran besar. Pada skenario ini parameter Hadoop yang digunakan dapat dilihat pada Tabel 4.2.

Tabel 4.2 Parameter Hadoop skenario pertama

Parameter	***Value***
Blok Size	32 MB, 64 MB, 128 MB dan 256 MB
Slot Map	2

b. Skenario Kedua

Skenario kedua bertujuan untuk menganalisis pengaruh block size terhadap kecepatan MapReduce pada Hadoop yang dijalankan secara multi node. Ppercobaan akan dilakukan menggunakan ukuran file yang bervariasi yaitu 512 MB, 1 GB, 1.5 GB, dan 2 GB dengan menggunakan ukuran block size yang bervariasi untuk setiap ukuran file yaitu 32 MB, 64 MB, 128 MB, dan 256 MB. Gambaran pembagian block size pada skenario ini dapat dilihat pada Gambar 4.6.



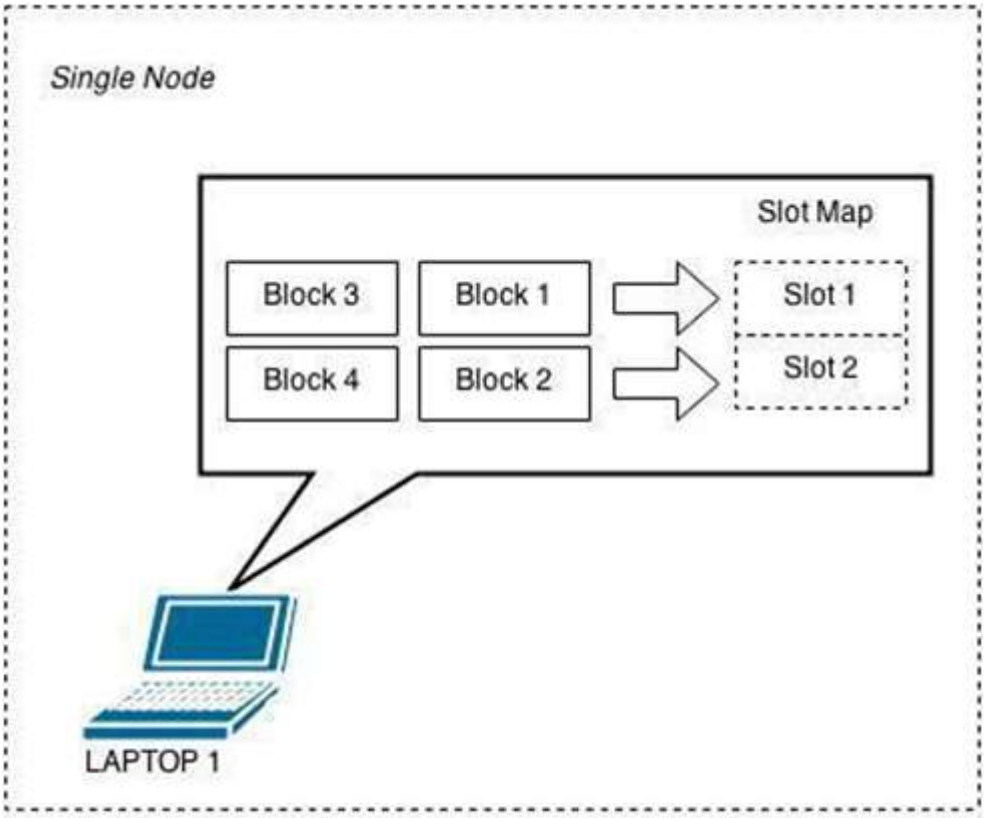
Gambar 4.6 Pembagian block size pada Hadoop yang dijalankan secara multi node

Gambar 4.6 menunjukan gambaran Hadoop dalam membagi sebuah file yang memiliki ukuran besar menjadi beberapa beberapa block. Pada skenario kedua, potongan blocks tersebut akan dibagikan secara merata pada setiap node yang ada pada sebuah cluster sedangkan parameter Hadoop yang digunakan adalah parameter yang sama dengan Tabel 4.2.

c. Skenario Ketiga

Skenario ketiga bertujuan untuk menganalisis pengaruh jumlah slot map terhadap kecepatan MapReduce pada Hadoop secara single node. Percobaan akan dilakukan

menggunakan parameter jumlah slot map yang bervariasi pada setiap percobaan yaitu 2 slot map, 4 slot map, 6 slot map, dan 8 slot map. Contoh gambaran slot map pada Hadoop dapat dilihat pada Gambar 4.7.



Gambar 4.7 Gambaran slot map pada Hadoop

Gambar 4.7 menunjukan slot map yang akan memproses setiap block yang ada. Pada skenario ketiga, percobaan akan dilakukan menggunakan ukuran file yang bervariasi yaitu 512 MB, 1 GB, 1.5 GB, dan 2 GB, sedangkan parameter Hadoop yang digunakan dapat dilihat pada Tabel 4.3.

Tabel 4.3. Parameter Hadoop skenario ketiga

Parameter	***Value***
Blok Size	64 MB
Slot Map	2, 4, 6 dan 8

B. Implementasi Sistem

1. Cara Mengukur Kecepatan MapReduce pada Hadoop

Pada penelitian ini, cara yang digunakan untuk mengukur kecepatan MapReduce pada Hadoop adalah dengan mengakses tampilan webservice yang disediakan oleh Hadoop untuk melihat JobTracker history untuk setiap jobs yang telah diselesaikan oleh Hadoop. Adapun contoh tampilannya dapat dilihat pada Gambar 4.8

Gambar 4.8 Contoh tampilan JobTracker history pada webservice Hadoop

2. Menentukan Block Size pada HDFS

Secara default Hadoop membagi file menjadi beberapa block dengan ukuran 64 MB pada setiap block. Akan tetapi blok-blok tersebut dapat diubah-ubah ukurannya dengan menggunakan dua cara, yaitu dengan cara melakukan konfigurasi pada file hdfs-site.xml atau dengan melakukan pembagian pada saat mengcopy file dari lokal sistem kedalam HDFS. Cara pertama dapat dilakukan dengan cara memasukkan konfigurasi yang ada pada Gambar 4.9 ke dalam file hdfs-site.xml.

Gambar 4.9 Konfigurasi block size pada hdfs-site.xml

Konfigurasi pada hdfs-site.xml

```
<property>
        <name>dfs.block.size</name>
        <value>67108864</value>
</property>
```

Nilai yang terdapat pada tag value menunjukan ukuran block dalam bytes pada HDFS. Besar nilai pada tag value dapat diganti sesuai dengan keinginan pada saat

sebelum menjalankan Hadoop. Cara kedua dapat dilakukan pada saat setelah menjalankan Hadoop, yaitu dengan cara memasukkan command line yang dapat dilihat pada Gambar 4.10.

```
$hadoop fs -D dfs.block.size=67108864 -put /local disk/ hadoop file system
```

Gambar 4.10 Command line untuk menentukan block size

Fungsi dari command line yang terdapat pada Gambar 4.9 adalah membagi file menjadi beberapa block sesuai dengan parameter pada dfs.block.size yang diinginkan dan mengcopy file tersebut dari local disk ke dalam HDFS. Pada percobaan ini cara yang dilakukan adalah menggunakan cara kedua dalam mengatur block size pada Hadoop.

3. Menentukan Jumlah Slot Map

Secara default Hadoop memiliki dua slot map. Akan tetapi jumlah slot map tersebut dapat divariasikan jumlahnya dengan melakukan konfigurasi pada file mapred-site.xml yang terdapat pada folder conf pada Hadoop. Konfigurasi file mapred-site.xml untuk melakukan perubahan pada jumlah slot map dapat dilihat pada Gambar 4.11.

Konfigurasi pada mapred-site.xml

```
<property>
        <name>mapred.tasktracker.map.tasks.maximum</name>
        <value>2</value>
</property>
```

Gambar 4.11 Konfigurasi jumlah slot map pada file mapred-site.xml

Nilai pada tag name menunjukan nama parameter yang dapat diganti pada Hadoop yaitu jumlah slot map. Nilai yang terdapat pada tag value menentukan jumlah slot map yang akan dijalankan oleh Hadoop.

C. Analisis dan Pembahasan

Perancangan yang telah dibuat dengan skenario dan topologi yang disarankan pada bagian sebelumnya, selanjutnya dilakukan pengambilan data dan analisis. Analisis dilakukan dengan tujuan untuk menganalisis performa kecepatan MapReduce pada Hadoop pada setiap skenario yang telah dirancang sebelumnya.

1. Pengujian Skenario Pertama

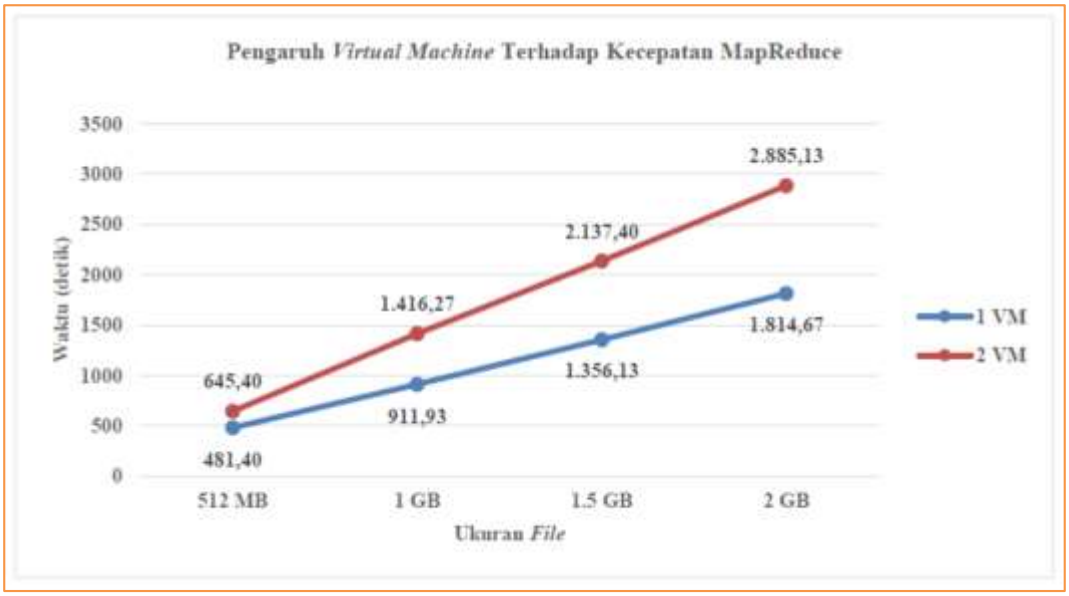
Skenario pertama bertujuan untuk mengetahui pengaruh virtual machine sebagai node terhadap performa kecepatan MapReduce pada Hadoop. Skenario pertama akan menjalankan Hadoop secara single node dan multi node menggunakan virtual machine. Ukuran file yang diproses oleh MapReduce pada skenario kedua untuk setiap percobaan adalah 512 MB, 1 GB, 1.5 GB, dan 2 GB. Percobaan pada skenario ini dilakukan sebanyak 15 kali percobaan.

a. Hasil Pengukuran

Hasil performa kecepatan rata-rata MapReduce dengan menjalankan Hadoop menggunakan satu virtual machine dan dengan menggunakan dua virtual machine sebagai node dapat dilihat pada Tabel 4.4.

Tabel 4.4 Hasil kecepatan rata-rata MapReduce menggunakan virtual node

Jumlah Virtual Machine	***Kecepatan (detik)***			
	***File* 512 MB**	***File* 1 GB**	***File* 1.5 GB**	***File* 2 GB**
1 VM	481.40	911.93	1356.13	1814.67
2 VM	645.40	1416.27	2137.40	2885.13



Gambar 4.12 Grafik pengaruh virtual node terhadap kecepatan MapReduce

Gambar 4.12 yang merupakan gambaran dari Tabel 4.4 yang memperlihatkan bahwa jumlah virtual node pada setiap percobaan yang dilakukan dengan menggunakan ukuran file yang bervariasi yaitu 512 MB, 1 GB, 1.5 GB, dan 2 GB dapat mempengaruhi kecepatan proses MapReduce pada Hadoop. Pada Gambar 4.12 terlihat bahwa penambahan jumlah virtual node dapat memperlambat proses MapReduce pada Hadoop.

b. Analisis

Pada skenario pertama penambahan jumlah virtual machine sebagai node dapat memperlambat proses MapReduce pada Hadoop. Penambahan jumlah virtual machine dari satu virtual machine menjadi dua virtual machine dengan spesifikasi virtual

machine yang sesuai perancangan dapat memperlambat kecepatan rata-rata MapReduce sebesar 164.00 detik pada ukuran file 512 MB, 504.34 detik pada ukuran file 1 GB, 781.27 detik pada ukuran file 1.5 GB, dan 1070.46 detik pada ukuran file 2 GB.

Hal ini terjadi karena setiap virtual machine mengambil resource dari machine yang sama yaitu dari laptop yang menjalankan virtual machine tersebut. Hal ini menyebabkan kecepatan MapReduce menjadi lambat ketika penambahan jumlah virtual machine sebagai node dikarenakan akan memperberat kerja dari laptop yang menjalankan virtual machine tersebut. Hal ini menyebabkan kecepatan MapReduce pada Hadoop dengan menggunakan virtual machine akan lebih lambat dibandingan dengan menggunakan physical machine.

2. Pengujian Skenario Kedua

Skenario kedua bertujuan untuk mengetahui pengaruh block size terhadap performa kecepatan MapReduce pada Hadoop yang dijalankan secara single node dengan menggunakan satu laptop sebagai node. Ukuran file yang diproses oleh MapReduce pada skenario kedua yaitu 512 MB, 1 GB, 1.5 GB, dan 2 GB. Percobaan pada skenario kedua dilakukan sebanyak 15 kali percobaan dengan ukuran block size yang bervariasi yaitu 32 MB, 64 MB, 128 MB, dan 256 MB.

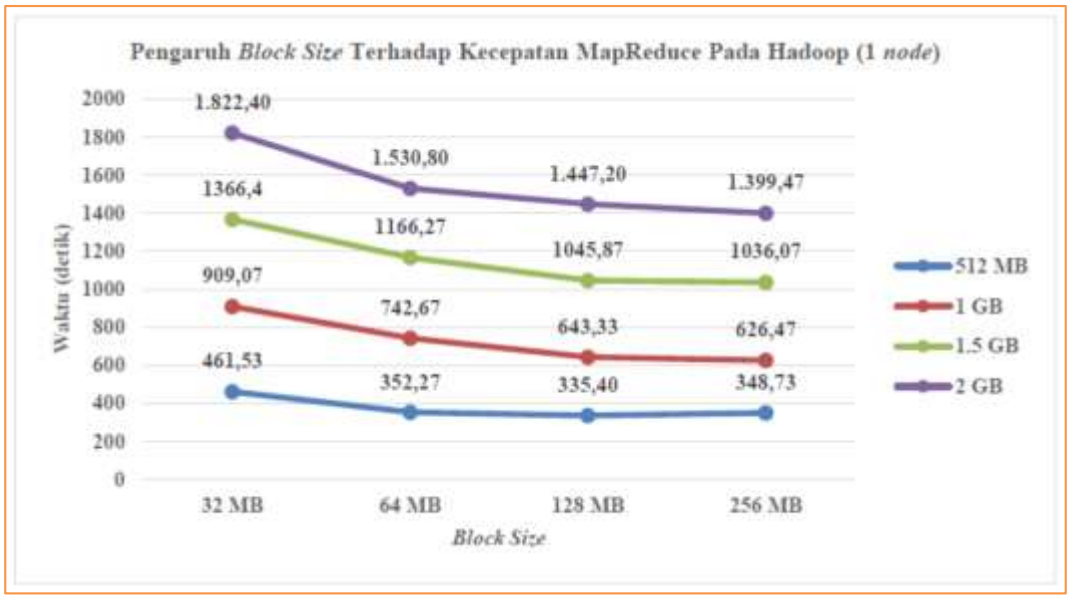
a. Hasil Pengukuran

Hasil performa kecepatan rata-rata MapReduce pada skenario kedua dengan menggunakan block size 32 MB, 64 MB, 128 MB, dan 256 MB pada ukuran file 512 MB, 1 GB, 1.5 GB, dan 2 GB dapat dilihat pada Tabel 4.5.

Tabel 4.5 Hasil kecepatan rata-rata MapReduce skenario kedua

Ukuran File	**Kecepatan (detik)**			
	***Block* 32 MB**	***Block* 64 MB**	***Block* 128 MB**	***Block* 256 MB**
512 MB	461.53	352.27	335.40	348.73
1 GB	909.07	742.67	643.33	626.47

1.5 GB	1366.40	1166.27	1045.87	1036.07
2 GB	1822.40	1530.80	1447.20	1399.47

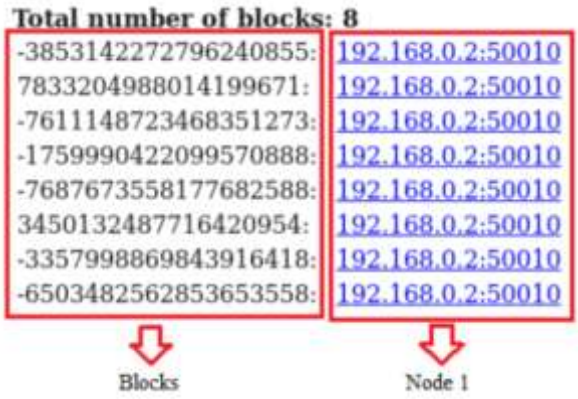


Gambar 4.13 Grafik pengaruh block size terhadap kecepatan MapReduce skenario kedua

Gambar 4.13 yang merupakan gambaran dari Tabel 4.3 yang memperlihatkan bahwa block size dapat mempengaruhi performa kecepatan MapReduce pada Hadoop untuk setiap percobaan yang dilakukan pada ukuran file yang bervariasi yaitu 512 MB, 1 GB, 1.5 GB, dan 2 GB.

b. Analisis

Pada skenario kedua, file dengan ukurn 512 MB, 1 GB, 1.5 GB, dan 2GB akan dipotong menjadi beberapa blocks. User dapat memastikan files tersebut telah terpotong menjadi beberapa blocks dengan cara mengakses webservice untuk melihat kondisi HDFS yang disediakan oleh Hadoop. Gambaran blocks tersebut tersebut dapat dilihat pada Gambar 4.14. Berdasarkan Gambar 4.14 dapat terlihat bahwa file dengan ukuran 512 MB dipotong-potong menjadi 8 blocks yang disimpan pada node 1 (laptop 1).



Gambar 4.14 Gambaran potongan blocks pada file 512 MB pada satu node

Berdasarkan hasil percobaan yang dilakukan pada skenario kedua terlihat bahwa pengubahan parameter block size pada setiap ukuran file dapat mempengaruhi kecepatan proses MapReduce pada Hadoop. Pada ukuran file 1 GB, 1.5 GB, dan 2 GB terlihat bahwa penambahan block size dapat mempercepat proses MapReduce pada Hadoop. Hal ini dikarenakan dengan menambah block size akan menghasilkan jumlah block yang lebih sedikit. Jumlah block pada Hadoop menentukan jumlah task pada suatu pekerjaan (job), dimana jumlah block merupakan jumlah task yang akan dikerjakan oleh MapReduce. Hal ini dapat terlihat pada Gambar 4.15.

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	0.00%	16	14	2	0	0	0 / 0
reduce	0.00%	1	1	0	0	0	0 / 0

(a)

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	0.00%	8	6	2	0	0	0 / 0
reduce	0.00%	1	1	0	0	0	0 / 0

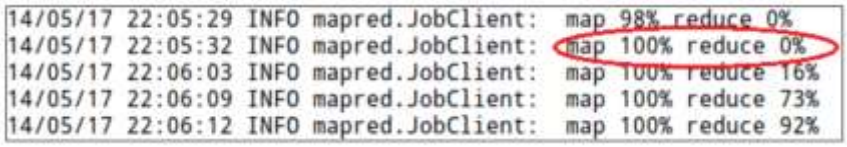
(b)

Gambar 4.15 Jumlah task pada file 1 GB dengan (a) block size 64 MB (b) block size 128 MB

Pada Gambar 4.15 (a) dan Gambar 4.15 (b) menunjukan bahwa jumlah task pada ukuran file 1 GB dengan block size 64 MB menghasilkan 16 tasks yang artinya lebih

banyak dibandingkan dengan jumlah task pada ukuran file 1 GB dengan block size 128 MB yang menghasilkan 8 tasks. Jumlah task yang lebih sedikit pada ukuran file 1 GB dengan block size 128 MB dapat memudahkan scheduler task MapReduce dalam menjadwalkan task yang diberikan sehingga dapat mengurangi kerja dari scheduler task MapReduce yang berpengaruh terhadap kecepatan MapReduce pada Hadoop. Selain itu dengan jumlah task yang semakin sedikit hal ini dapat mengurangi waktu komunikasi antara scheduler task MapReduce dengan JobTracker dan JobTracker dengan TaskTracker dalam permintaan task, sehingga hal ini menyebabkan waktu proses MapReduce pada Hadoop semakin cepat.

Pada ukuran file 512 MB dengan block size 256 MB terlihat bahwa kecepatan MapReduce pada Hadoop lebih lambat jika dibandingan dengan menggunakan block size 128 MB. Hal ini dikarenakan pada ukuran file 512 MB dengan block size 256 MB akan menghasilkan 2 tasks, yang artinya bahwa 2 tasks tersebut langsung dimapping seluruhnya secara bersamaan sampai selesai karena secara default jumlah slot map pada Hadoop adalah 2 slot map untuk setiap node. Setelah proses mapping dari seluruh tasks yang ada selesai baru dilakukan proses reducing. Hal ini dapat dilihat pada Gambar 4.16.



Gambar 4.16 Proses MapReduce pada ukuran file 512 MB dengan block size 256 MB

Proses MapReduce pada ukuran file 512 MB dengan block size 256 MB yang ditunjukkan Gambar 4.16 dapat menyebabkan keterlambatan MapReduce dalam melakukan proses reducing sehingga terjadi penurunan kecepatan MapReduce.

3. Pengujian Skenario Ketiga

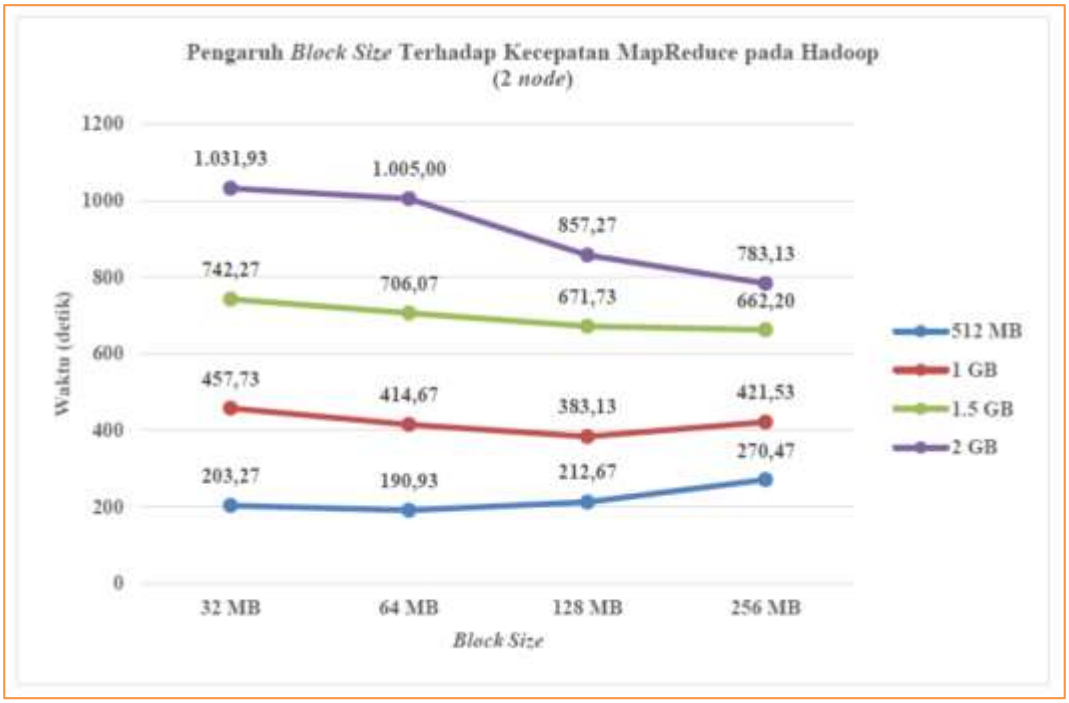
Skenario keempat bertujuan untuk mengetahui pengaruh block size terhadap performa kecepatan MapReduce pada Hadoop yang dijalankan secara multi node dengan menggunakan dua laptop sebagai node. Ukuran file yang diproses oleh MapReduce pada skenario ketiga yaitu 512 MB, 1 GB, 1.5 GB, dan 2 GB. Percobaan pada skenario ketiga dilakukan sebanyak 15 kali percobaan dengan ukuran block size yang bervariasi yaitu 32 MB, 64 MB, 128 MB, dan 256 MB.

a. Hasil Pengukuran

Hasil performa kecepatan rata-rata MapReduce pada skenario ketiga dengan menggunakan block size 32 MB, 64 MB, 128 MB, dan 256 MB dapat dilihat pada Tabel 4.6.

Tabel 4.6 Hasil kecepatan rata-rata MapReduce skenario ketiga

Ukuran File	**Kecepatan (detik)**			
	***Block* 32 MB**	***Block* 64 MB**	***Block* 128 MB**	***Block* 256 MB**
512 MB	203.27	190.93	212.67	270.47
1 GB	457.73	414.67	383.13	421.53
1.5 GB	742.27	706.07	671.73	662.20
2 GB	1031.93	1005.00	857.27	783.13

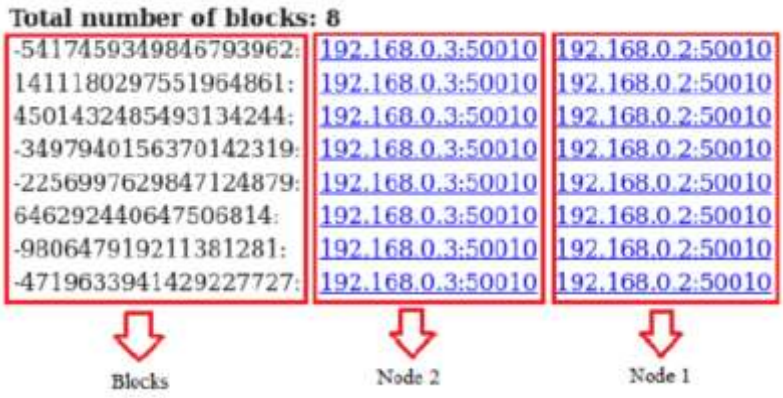


Gambar 4.17 Grafik pengaruh block size terhadap kecepatan MapReduce skenario ketiga

Gambar 4.17 yang merupakan gambaran dari Tabel 4.6 memperlihatkan bahwa block size dapat mempengaruhi kecepatan MapReduce pada setiap percobaan yang dilakukan dengan menggunakan ukuran file yang bervariasi yaitu 512 MB, 1 GB, 1.5 GB, dan 2 GB.

b. Analisis

Pada skenario keempat, blocks pada Hadoop akan disebar pada masing-masing node yang terdapat pada cluster. User dapat memastikan blocks tersebut tersebar pada setiap node dengan cara mengakses webservice untuk melihat kondisi HDFS yang disediakan oleh Hadoop. Gambaran dari penyebaran blocks tersebut tersebut dapat dilihat pada Gambar 4.18.



Gambar 4.18 Gambaran penyebaran blocks pada setiap node

Pada Gambar 4.18 terlihat bahwa setiap block tersebar pada setiap node yang terdapat pada sebuah cluster, dimana pada skenario ketiga, percobaan dilakukan dengan menggunakan dua node yaitu node 1 (laptop 1) dan node 2 (laptop 2).

Berdasarkan hasil percobaan yang dilakukan pada skenario ketiga terlihat bahwa block size dapat mempengaruhi kecepatan proses MapReduce pada Hadoop. Pada ukuran file 1.5 GB dan 2 GB terlihat bahwa penambahan block size dapat mempercepat proses MapReduce pada Hadoop. Hal ini dikarenakan dengan menambah block size akan menghasilkan jumlah block yang lebih sedikit. Jumlah block pada Hadoop menentukan jumlah task pada suatu pekerjaan (job), dimana jumlah block merupakan jumlah task yang akan dikerjakan oleh MapReduce pada Hadoop. Jumlah task yang lebih sedikit dapat memudahkan scheduler task MapReduce dalam menjadwalkan task yang diberikan sehingga dapat mengurangi kerja dari scheduler task MapReduce yang berpengaruh terhadap kecepatan MapReduce. Selain itu, dengan jumlah task yang semakin sedikit dapat mengurangi waktu komunikasi antara scheduler task MapReduce dengan JobTracker dan JobTracker dengan TaskTracker dalam permintaan task, sehingga hal ini dapat menyebabkan waktu proses MapReduce semakin cepat.

Pada ukuran file 512 MB dengan block size 128 MB dan 256 MB dan ukuran file 1 GB dengan block size 256 MB terlihat bahwa kecepatan MapReduce pada Hadoop

semakin lambat meskipun block size yang dimiliki sudah diperbesar. Hal ini dikarenakan pada ukuran file 512 MB dengan block size 128 MB dan ukuran file 1 GB dengan block size 256 MB akan menghasilkan jumlah task sebanyak 4 tasks. Pada skenario keempat, Hadoop dijalankan secara multi node (cluster) menggunakan 2 nodes dimana masing masing node memiliki 2 slot map, sehingga total dari jumlah slot map yang terdapat pada cluster adalah 4 slot map. Hal ini menyebabkan 4 tasks yang dihasilkan oleh file 512 MB dengan block size 128 MB dan file 1 GB dengan block size 256 MB akan dimapping sampai selesai kemudian baru akan dilakukan reducing, sehingga dapat memperlambat proses reducing dikarenakan proses reducing baru dapat dilakukan setelah seluruh proses mapping pada task yang ada selesai.

Pada ukuran file 512 MB dengan block size 256 MB terjadi penurunan kecepatan MapReduce pada Hadoop karena jumlah task yang dihasilkan sebanyak 2 tasks. Hal ini menyebabkan jumlah task lebih sedikit dibandingkan dengan jumlah slot map yang ada sehingga seluruh tasks yang ada akan dimapping seluruhnya secara bersamaan sampai selesai dan kemudian setelah itu baru dilakukan proses reducing. Hal ini dapat menyebabkan keterlambatan MapReduce dalam melakukan proses reducing, sehingga menyebabkan proses MapReduce menjadi lebih lama.

BAB V

KESIMPULAN DAN SARAN

A. Kesimpulan

Berdasarkan hasil dan pembahasan pada bab 4, dapat disimpulkan hal-hal sebagai berikut:

1. Penambahan jumlah virtual machine dari satu virtual machine menjadi dua virtual machine dengan spesifikasi virtual machine yang sesuai perancangan dapat memperlambat kecepatan rata-rata MapReduce sebesar 164.00 detik pada ukuran file 512 MB, 504.34 detik pada ukuran file 1 GB, 781.27 detik pada ukuran file 1.5 GB, dan 1070.46 detik pada ukuran file 2 GB.
2. Block size dapat mempengaruhi kecepatan MapReduce pada Hadoop, semakin besar block size maka akan mempercepat proses MapReduce pada Hadoop dengan syarat hasil pembagian ukuran file dengan block size lebih besar dari jumlah slot map (number of tasks > number of slot maps).

3. Jumlah slot map dapat mempengaruhi kecepatan MapReduce pada Hadoop, dimana performa kecepatan MapReduce lebih cepat jika jumlah slot map pada Hadoop sesuai dengan jumlah core yang dimiliki oleh machine.

4. Hadoop memiliki keunggulan dalam mengolah data yang memiliki ukuran yang besar dan jumlah yang banyak karena Hadoop dapat mengolah data tersebut secara terdistribusi, dimana performa kecepatan Hadoop dalam mengolah data masih dapat ditingkatkan dengan cara menambah physical machine sebagai node, selain itu pengaturan parameter block size dan parameter jumlah slot map yang tepat pada Hadoop juga dapat meningkatkan performa kecepatan MapReduce pada Hadoop dalam mengolah data.

DAFTAR PUSTAKA

[1] Abdurachman, Zaky. (2011). *Single Node Cluster dengan Hadoop*. Jakarta Pusat: InfoLinux Media Utama.

[2] *Apache Hadoop*. (2011). Retrieved Februari 10, 2012, from Apache Software Foundation.: http://hadoop.apache.org/

[3] Asha T, Shravanthi U.M, Nagashree N, Monika M. *Building Machine Learning Algorithms on Hadoop for Bigdata*. Department of Information Science & Engineering Bangalore Institute of Technology, Bangalore, INDIA. International Journal of Engineering and Technology Volume 3 No. 2, February, 2013.

[4] Fisher, M. (2003). JDBC(TM), *API Tutorial and Reference*. California: Sun Microsystem. Inc. Addison Weys.

[5] Ghemawat., J. D. (2004). *MapReduce: Simplified Data*. California: Google, Inc.

[6] Komputer., W. (2011). *Administrasi Jaringan dengan Linux Ubuntu 11.4*. Yogyakarta: Andi.

[7] Praveen Kumar, Dr Vijay Singh Rathore. *Efficient Capabilities of Processing of Big Data using Hadoop Map Reduce*. Research Scholar, Department of Computer Science, NIMS University, Jaipur, India[1], Professor & Director, Shri Kami College, Jaipur, Rajasthan, India[2]. International Journal of Advanced Research in Computer and Communication Engineering Vol. 3, Issue 6, June 2014.

[8] S.Tanenbaum, A. &. (1995). *Distributed Systems Principles and Paradigms*. New Jersey: Prentice Hall.

[9] Venner, Jason. (2012). *Pro Hadoop*. United States of America: Apress.

[10] White, Tom. (2009). *Hadoop: The Definitive Guide*. California: 0'Reilly Media, Inc.

[11] White, Tom. (2008). *HDFS Reability*. California: Cloudera Inc.

LAMPIRAN

Instalasi Hadoop di Ubuntu 14.04

1. Instalasi Sun Java di Ubuntu

sudo add-apt-repository ppa:webupd8team/java

sudo apt-get update

sudo apt-get install oracle-java7-installer

sudo update-java-alternatives -s java-7-oracle

Jika Java sudah terinstal, bisa dicek dengan

perintah, java -version

```
nurqalbi@nurqalbi-Aspire-4741:~/Documents/hadop$ java -version
java version "1.7.0_80"
Java(TM) SE Runtime Environment (build 1.7.0_80-b15)
Java HotSpot(TM) Client VM (build 24.80-b11, mixed mode)
nurqalbi@nurqalbi-Aspire-4741:~/Documents/hadop$
```

2. Membuat User dan Grup Hadoop

```
$ sudo addgroup hadoop
$ sudo adduser –ingroup hadoop hduser
```

```
Java HotSpot(TM) Client VM (build 24.80-b11, mixed mode)
nurqalbi@nurqalbi-Aspire-4741:~/Documents/hadop$ sudo addgroup hadoop
[sudo] password for nurqalbi:
Adding group `hadoop' (GID 1001) ...
Done.
nurqalbi@nurqalbi-Aspire-4741:~/Documents/hadop$ sudo adduser --ingroup hadoop
hduser
Adding user `hduser' ...
Adding new user `hduser' (1001) with group `hadoop' ...
Creating home directory `/home/hduser' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for hduser
Enter the new value, or press ENTER for the default
        Full Name []:
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [Y/n] Y
nurqalbi@nurqalbi-Aspire-4741:~/Documents/hadop$
```

3. Install SSH Server

SSH Server perlu diinstal jika belum ada dalam sistem. Ini diperlukan karena hadoop butuh akses ke localhost (dalam *single node cluster*) atau untuk berkomunikasi dengan titik remote (dalam *multi- node clusters*) – instalasi ini dilakukan menggunakan user yang punya hak administrator atau sudo

```
$ sudo apt-get install openssh-server
```

dan kemudian kita akan menggenerate SSH key dengan menggunakan user yg akan menjalankan ssh seperti pada contoh di bawah menggunakan user hduser

```
$ su - hduser
$   ssh-keygen -t dsa -P '' -f ~/.ssh/id_dsa
$   cat ~/.ssh/id_dsa.pub >> ~/.ssh/authorized_keys
```

```
hduser@nurqalbi-Aspire-4741:~$ ls
examples.desktop
hduser@nurqalbi-Aspire-4741:~$ ssh-keygen -t rsa -P ""
Generating public/private rsa key pair.
Enter file in which to save the key (/home/hduser/.ssh/id_rsa):
Created directory '/home/hduser/.ssh'.
Your identification has been saved in /home/hduser/.ssh/id_rsa.
Your public key has been saved in /home/hduser/.ssh/id_rsa.pub.
The key fingerprint is:
b7:fc:68:5e:8c:2e:2f:5c:c6:79:44:02:bb:29:98:e4 hduser@nurqalbi-Aspire-4741
The key's randomart image is:
+--[ RSA 2048]----+
|         ..      |
|          .. .   |
|    .   .  o     |
|   o o   o  .    |
|    E . S..o     |
|       . o=+.    |
|       . o+.o    |
|        +..+     |
|         *= .    |
+-----------------+
hduser@nurqalbi-Aspire-4741:~$
```

4. Memasang Hadoop

Unduh hadoop dari situs hadoop apache

```
$   wget http://mirrors.ibiblio.org/apache/hadoop/common/hadoop-2.7.0/hadoop-2.7.0.tar.gz -c
$   cd /home/hduser
$   tar xzf hadoop-2.7.0.tar.gz
$   mv hadoop-2.7.0 hadoop
```

5. Edit file .bashrc

Edit berkas .bashrc dengan menggunakan nano atau vim, file ini terletak di direktori home user

```
$ nano .bashrc
```

dan paste dengan isi yang berikut,

```
#   Set Hadoop-related environment variables
export HADOOP_HOME=/home/hduser/hadoop
```

```
#   Set JAVA_HOME (we will also configure JAVA_HOME directly for Hadoop later
on)
  export JAVA_HOME=/usr/lib/jvm/java-7-oracle
#   Add Hadoop bin/ directory to PATH
  export PATH=$PATH:$HADOOP_HOME/bin
  export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
  export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib"
  export HADOOP_CLASSPATH=${JAVA_HOME}/lib/tools.jar
```

setelah disimpan

$ source .bashrc

6. Update hadoop-env.sh

Update variable JAVA_HOME pada file ini.

```
$ vi /home/hduser/hadoop/etc/hadoop/hadoop-env.sh
```

Tambahkan baris berikut jika belum ada yang sesuai. Pada file ini, pada baris export JAVA_HOME=${JAVA_HOME} , namun jika hadoop nanti tidak jalan maka perlu untuk memasukkan secara manual seperti di bawah ini,

```
export JAVA_HOME=/usr/lib/jvm/java-7-oracle
```

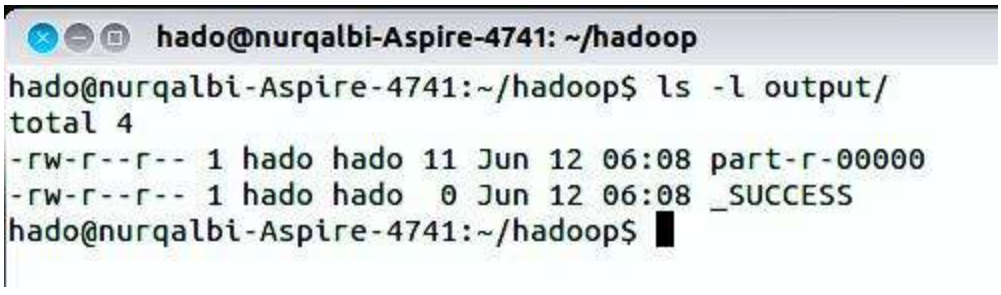
7. Operasi Standalone

Secara default, hadoop dikonfigurasi berjalan pada mode non-distributed sebagaimana sebuah proses single Java. Ini berguna untuk debugging.

Mari menjalankan perintah di bawah untuk melihat output yang diperoleh pada direktori output yang dihasilkan

$ mkdir input

```
  $ cp etc/hadoop/*.xml input
$   bin/hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.0.jar
grep input output 'dfs[a-z.]+'
$   cat output/*
```



8. Edit file core-site.xml

```
$ nano /home/hduser/hadoop/etc/hadoop/core-site.xml
```

Tambahkan konfigurasi berikut dalam tab elemen xml <configuration> .. </configuration>:

```
<configuration>
    <property>
        <name>fs.defaultFS</name>
        <value>hdfs://localhost:9000</value>
    </property>
</configuration>
```

9. mapred-site.xml dan yarn-site.xml

Seperti langkah 8 pada core-site.xml , lakukan hal yang sama
dengan ~/hadoop/etc/hadoop/mapred-site.xml

File yg tersedia biasanya mapred-site.xml.template jadi sila salin dulu jadi mapred-site.xml

```
<configuration>
    <property>
        <name>mapreduce.framework.name</name>
        <value>yarn</value>
    </property>
</configuration>
```

~/hadoop/etc/hadoop/yarn-site.xml

```
<configuration>
    <property>
        <name>yarn.nodemanager.aux-services</name>
        <value>mapreduce_shuffle</value>
    </property>
</configuration>
```

10. hdfs-site.xml

Buka hdfs-site.xml dan tambahkan konfigurasi berikut:

```
<configuration>
    <property>
        <name>dfs.replication</name>
        <value>1</value>
    </property>
</configuration>
```

11. Memformat NameNode

Jalankan perintah berikut,

```
$ /home/hduser/hadoop/bin/hdfs namenode -format
```

```
15/06/12 06:13:04 INFO util.GSet: 0.029999999329447746% max memory 966.7 MB = 297.0 KB
15/06/12 06:13:04 INFO util.GSet: capacity      = 2^16 = 65536 entries
15/06/12 06:13:05 INFO namenode.FSImage: Allocated new BlockPoolId: BP-1680344381-127.0.1.1-1434060785002
15/06/12 06:13:05 INFO common.Storage: Storage directory /tmp/hadoop-hado/dfs/name has been successfully formatted.
15/06/12 06:13:05 INFO namenode.NNStorageRetentionManager: Going to retain 1 images with txid >= 0
15/06/12 06:13:05 INFO util.ExitUtil: Exiting with status 0
15/06/12 06:13:05 INFO namenode.NameNode: SHUTDOWN_MSG:
/************************************************************
SHUTDOWN_MSG: Shutting down NameNode at nurqalbi-Aspire-4741/127.0.1.1
************************************************************/
hado@nurqalbi-Aspire-4741:~/hadoop$ sbin/start-dfs.sh
```

12. Starting Hadoop Cluster

dari hadoop/sbin

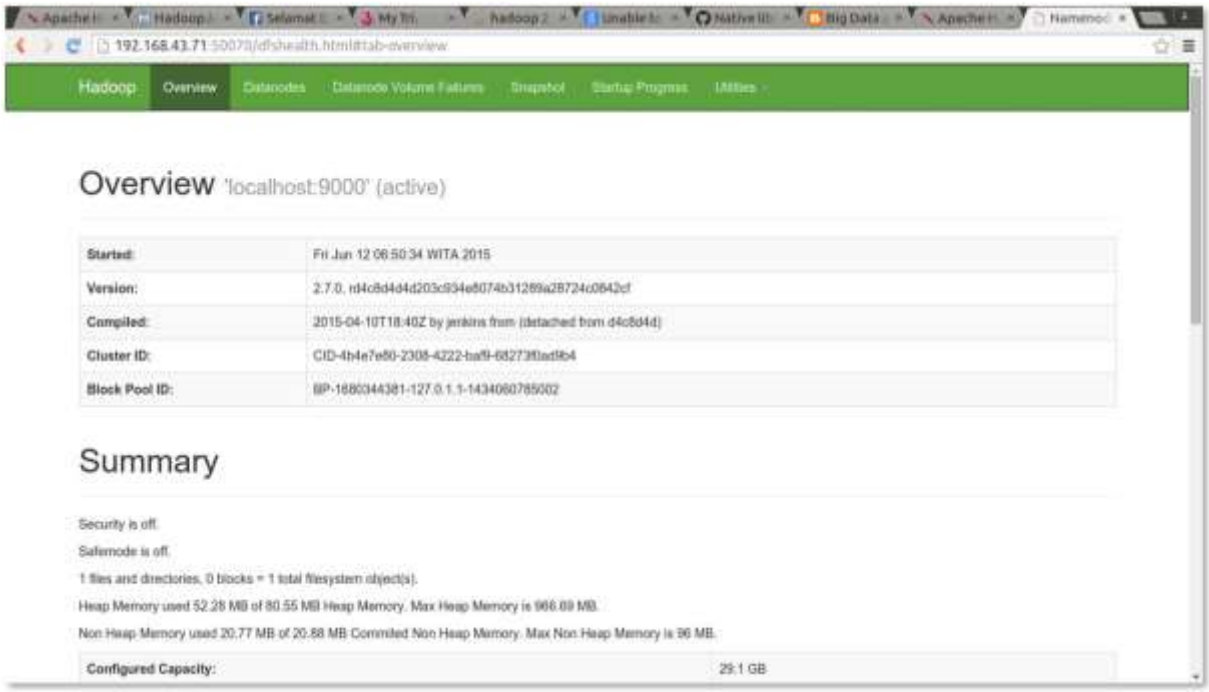
```
./start-dfs.sh
```

```
hado@nurqalbi-Aspire-4741:~/hadoop$ sbin/start-dfs.sh
15/06/12 06:50:22 WARN util.NativeCodeLoader: Unable to load native-hadoop lib
rary for your platform... using builtin-java classes where applicable
Starting namenodes on [localhost]
localhost: starting namenode, logging to /home/hado/hadoop/logs/hadoop-hado-na
menode-nurqalbi-Aspire-4741.out
localhost: starting datanode, logging to /home/hado/hadoop/logs/hadoop-hado-da
tanode-nurqalbi-Aspire-4741.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /home/hado/hadoop/logs/hadoop-
hado-secondarynamenode-nurqalbi-Aspire-4741.out
15/06/12 06:50:46 WARN util.NativeCodeLoader: Unable to load native-hadoop lib
rary for your platform... using builtin-java classes where applicable
hado@nurqalbi-Aspire-4741:~/hadoop$
```

mencoba akses ke browser,

http://localhost:50070

dalam *screenshoot* di bawah dengan mengakses ip address dari pc yang lain via wifi



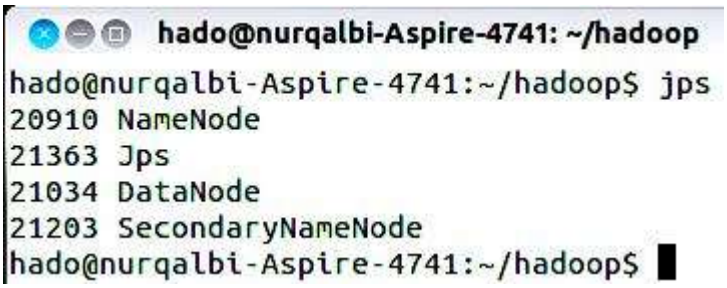
13. Menghentikan Hadoop Cluster

dari hadoop/sbin

```
./stop-dfs.sh
```

Mengecek proses yang jalan:

```
$jps
```



Pada hasil perintah di atas ada pesan eror,

WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable

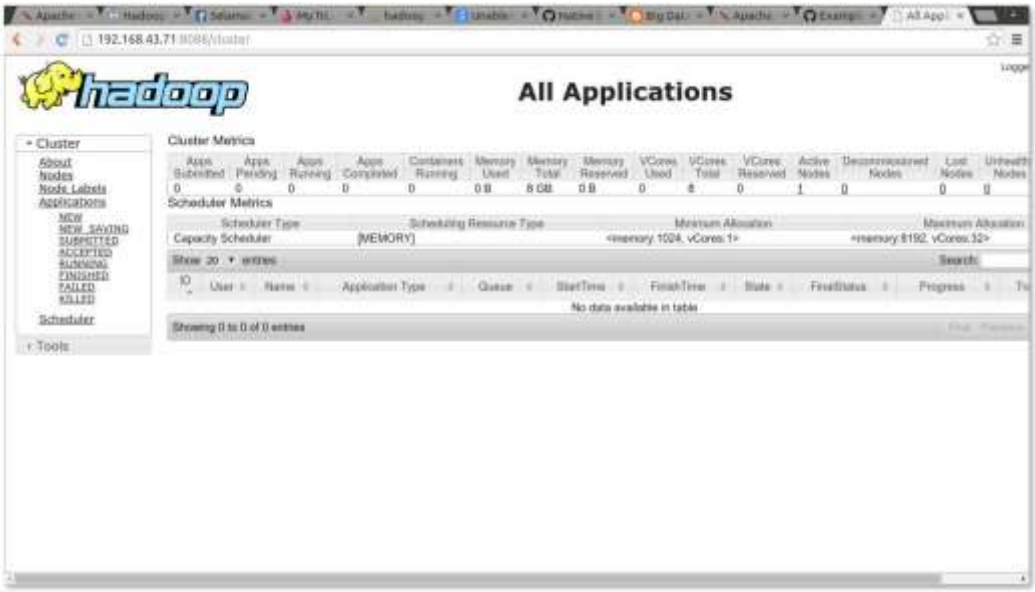
Eror ini belum belum bisa saya atasi tapi secara keseluruhan semua proses berjalan normal.

14. Menjalankan Yarn

$ sbin/star-yarn.sh

kita bisa melihat jika berjalan dengan baik maka url ini

terbuka, http://localhost:8088



15. Contoh Eksekusi dengan contoh pada distrubusi yang disertakan

Perintah di bawah ini akan menjalankan Mapreduce secara lokal, dengan nama user hado, user hduser yang diatas juga bisa. Di sini saya menginstall hadoop di 2 user.

```
$    bin/hdfs namenode -format
$    sbin/start-dfs.sh
$    sbin/start-yarn.sh
$    bin/hdfs dfs -mkdir /user
$    bin/hdfs dfs -mkdir /user/hado
$    bin/hdfs dfs -put etc/hadoop input
$    bin/hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.0.jar grep
input output 'dfs[a-z.]+'
```


```
$    bin/hdfs dfs -get output output
$    cat output/*
$    sbin/stop-yarn.sh
$    sbin/stop-dfs.sh
```

16. Eksekusi Program WordCount dari kode sumber

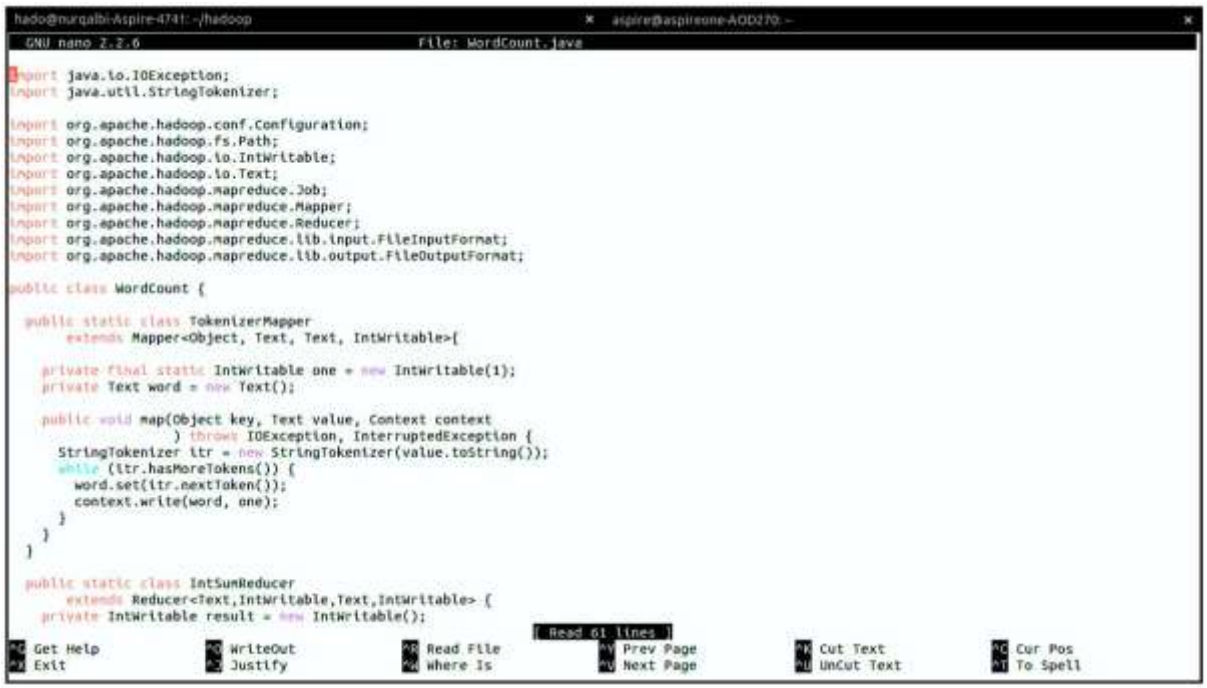
http://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html#Example:_WordCount_v1.0

```
import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text; import
org.apache.hadoop.mapreduce.Job; import
org.apache.hadoop.mapreduce.Mapper; import
org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;


public class WordCount {

  public static class TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new
    IntWritable(1); private Text word = new Text();

    public void map(Object key, Text value, Context context
)                       throws IOException, InterruptedException {
StringTokenizer itr = new StringTokenizer(value.toString()); while
(itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }

  public static class IntSumReducer
       extends Reducer<Text,IntWritable,Text,IntWritable>
    { private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable>
                       values, Context context
                       ) throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable val : values) {
        sum += val.get();
      }
      result.set(sum);
      context.write(key, result);
    }
  }

  public static void main(String[] args) throws Exception
    { Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word
    count"); job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
```

```
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

Salin dan Tempel kode di atas misalnya dengan menggunakan nano dan simpan dengan nama WordCount.java

$ nano WordCount.java



kompilasi dengan perintah berikut,

```
$   bin/hadoop com.sun.tools.javac.Main WordCount.java
$   jar cf wc.jar WordCount*.class
```

buat sebuah berkas dengan nama **test1** yang isinya,

mencoba hitung kata

dan berkas **test2** dengan isi

ini juga akan dihitung

pada tutorial ini saya simpan di direktori ~/hadoop/input/wc

17.Mari kita mulai menjalankan hadoop kembali

```
$    bin/hdfs namenode -format
$    sbin/start-dfs.sh
$    sbin/start-yarn.sh
```

Membuat direktori HDFS sebagai tempat untuk eksekusi MapReduce

```
$ bin/hdfs dfs -mkdir /user
$ bin/hdfs dfs -mkdir /user/hado
```

Kita buat sebuah direktori wc

```
$ bin/hdfs dfs -mkdir wcount
```

Salin berkas test1 dan test2 ke direktori wc di awan, dan disini saya menyalin semua direktori wc ke direktori wc di awan, berikan opsi -f jika akan menyalin ulang dan menimpa yg lama

```
$ bin/hdfs dfs -put input/wc/* wcount
```

contoh menimpa berkas test1 yang sudah ada

```
$ bin/hdfs dfs -put -f input/wc/test1 wcount
```

Melihat isi direktori yang sudah di salin

```
$ bin/hadoop fs -ls wcount
```

Jalankan aplikasi hitung kata dengan input dari direktori wc dan hasil ke direktori output,

```
$ bin/hadoop jar wc.jar WordCount wcount output
```

dan lihat berkas output

```
$ bin/hadoop fs -cat output/part-r-00000
```

```
hado@nurqalbi-Aspire-4741:~/hadoop$ bin/hadoop fs -cat /user/hado/output/part-r-00000
15/06/12 09:27:37 WARN util.NativeCodeLoader: Unable to load native-hadoop library for
le
akan    1
dihitung        1
hitung  1
ini     1
juga    1
kata    1
mencoba 1
hado@nurqalbi-Aspire-4741:~/hadoop$
```

File WordCount.java dicoba kompile kembali dengan nama HitungAngka.java dan kita bisa melihat statistiknya localhost:8088