TUGAS AKHIR – KI141502

KINERJA KLASTER HADOOP PADA RASPBERRY PI

Wahyu Kukuh Herlambang
NRP 5112100070

Dosen Pembimbing
Ir. F.X. Arunanto, M.Sc.
Hudan Studiawan, S.Kom., M.Kom.

JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya 2016

[Halaman ini sengaja dikosongkan]

FINAL PROJECT – KI141502

HADOOP CLUSTER PERFORMANCE ON RASPBERRY PI

Wahyu Kukuh Herlambang
NRP 5112100070

Advisor
Ir. F.X. Arunanto, M.Sc.
Hudan Studiawan, S.Kom., M.Kom.

DEPARTMENT OF INFORMATICS
Faculty of Information Technology
Institut Teknologi Sepuluh Nopember
Surabaya 2016

[Halaman ini sengaja dikosongkan]

LEMBAR PENGESAHAN

KINERJA KLASTER HADOOP PADA RASPBERRY PI

TUGAS AKHIR

Diajukan Guna Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Komputasi Berbasis Jaringan
Program Studi S-1 Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember

Oleh :
Wahyu Kukuh Herlambang
NRP: 5112100070

Disetujui oleh Dosen Pembimbing Tugas Akhir :

Ir. F.X. ARUNANTO, M.Sc. .....
NIP: 195701011983031004 (pembimbing 1)

HUDAN STUDIAWAN, S.Kom, M.Kom. .....
NIP: 198705112012121003 (pembimbing 2)

SURABAYA
JUNI 2016

[Halaman ini sengaja dikosongkan]

Kinerja Klaster Hadoop pada Raspberry Pi

Nama Mahasiswa : Wahyu Kukuh Herlambang
NRP : 5112100070
Jurusan : Teknik Informatika FTIf-ITS
Dosen Pembimbing 1 : Ir. F.X. Arunanto, M.Sc.
Dosen Pembimbing 2 : Hudan Studiawan, S.Kom., M.Kom.

ABSTRAK

Besarnya biaya yang diperlukan untuk membuat klaster Hadoop dengan perangkat keras merupakan salah satu permasalahan utama dalam mengimplementasikannya secara nyata. Penggunaan perangkat murah pada klaster Hadoop juga dapat mengakibatkan tidak optimalnya bahkan tidak berkerjanya fitur-fitur pada Hadoop sehingga diperlukannya penyesuaian pada Hadoop atau perangkat itu sendiri.

Untuk mengoptimalkan kerja Hadoop maka pada tugas akhir kali ini akan digunakan beberapa metode penyesuaian, diantara lain: optimasi blok data, kompresi output mapper, penyesuaian memori, optimasi ukuran file, penggunaan ulang JVM, overclock perangkat, dan manajemen bandwith pada klaster Hadoop.

Hasil dari implementasi metode ini adalah dapat dijalankan dan diperhitungkannya kinerja dari klaster Hadoop sebelum dan setelah penyesuaian pada perangkat murah.

***Kata kunci**: Hadoop, Raspberry Pi, Perhitungan Kinerja*

[Halaman ini sengaja dikosongkan]

HADOOP CLUSTER PERFORMANCE ON RASPBERRY PI

Student Name : Wahyu Kukuh Herlambang
Student ID : 5112100070
Major : Teknik Informatika FTIf-ITS
Advisor 1 : Ir. F.X. Arunanto, M.Sc.
Advisor 2 : Hudan Studiawan, S.Kom., M.Kom

ABSTRACT

The amount of costs required to make a Hadoop cluster using hardware is one of the main practical problems in implementing it. The use of lowcost devices on Hadoop clusters can also lead to less optimized and even failures on Hadoop's features so that some adjustments are needed.

In order to optimize the work of Hadoop, this final assignment will use several methods of adjustment such as: the optimization of data blocks, mapper output compression, memory adjustment, optimization of file size, the reuse of JVM, device overlock, and bandwidth management on Hadoop cluster.

The results from the implementation of this method is the practicable and measurable performances of Hadoop clusters before and after adjustment on lowcost devices.

Keywords**: **Hadoop, Raspberry Pi, Performance Measurements

[Halaman ini sengaja dikosongkan]

KATA PENGANTAR

بِسۡمِ ٱللَّهِ ٱلرَّحۡمَٰنِ ٱلرَّحِيمِ

Alhamdulillahirabbil'alamin, segala puji bagi Allah SWT, yang telah melimpahkan rahmat dan hidayah-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul **"KINERJA KLASTER HADOOP PADA RASPBERRY PI".**

Pengerjaan Tugas Akhir ini merupakan salah satu dari tugas yang saya dapatkan selama kuliah untuk mendapatkan ilmu dan pengalaman yang berharga selama saya menimba ilmu di Fakultas FTIF Jurusan Teknik Informatika ITS ini.

Selesainya Tugas Akhir ini tidak lepas dari bantuan dan dukungan dari berbagai pihak. Sehingga pada kesempatan ini penulis mengucapkan syukur dan terima kasihnya pada:

1. Allah SWT dan Nabi Muhammad SAW.
2. Ibu, Ibu, Ibu, Ayah dan Adik yang selalu memberikan do'a, dukungan, serta motivasi, sehingga penulis selalu termotivasi untuk menyelesaikan Tugas Akhir.
3. Bapak Ir. F.X. Arunanto, M.Sc. dan Bapak Hudan Studiawan, S.Kom., M.Kom. selaku dosen pembimbing yang selalu membantu, menyemangati dan memotivasi penulis dengan ilmu-ilmu dalam penngerjaan Tugas Akhir.
4. Bapak, Ibu dosen Jurusan Teknik Informatika ITS yang telah banyak memberikan ilmu dan bimbingan yang tak ternilai harganya bagi penulis.
5. Teman – Teman Kost Bunda yang telah mengisi waktu-waktu penulis dengan tantangan, pengalaman, dan kesempatan selama perkuliahan.
6. Teman – teman angkatan 2012, tanpa mereka, saya tidak akan merasakan apa itu yang dinamakan "Angkatan".

Penulis menyadari bahwa Tugas Akhir ini masih memiliki banyak kekurangan. Sehingga dengan kerendahan hati, penulis

mengharapkan kritik dan saran dari pembaca untuk perbaikan ke depannya.

Surabaya, Mei 2016
Penulis

Wahyu Kukuh Herlambang

DAFTAR ISI

[Halaman ini sengaja dikosongkan]

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR KODE SUMBER

BAB I
PENDAHULUAN

Pada bab ini akan dipaparkan mengenai garis besar Tugas Akhir yang meliputi latar belakang, tujuan, rumusan dan batasan permasalahan, metodologi pembuatan Tugas Akhir, dan sistematika penulisan.

1.1. Latar Belakang

Hadoop merupakan salah satu *platform big data* yang memanfaatkan komputasi paralel. Hadoop terdiri dari 2 komponen utama, yaitu HDFS (*Hadoop Distributed File System*) dan MapReduce. HDFS berfungsi sebagai file sistem untuk penyimpanan data dimana data disimpan dalam bentuk blok data. Sedangkan MapReduce adalah *framework* pada Hadoop yang digunakan untuk memudahkan pembuatan aplikasi untuk mengolah data dalam jumlah besar (*big data*). Dalam memproses *big data*, Hadoop memanfaatkan banyak komputer untuk melakukan komputasi paralel dalam mengerjakan *job* yang diberikan.

Komputasi paralel adalah salah satu teknik untuk melakukan komputasi secara bersamaan dengan memanfaatkan kemampuan komputasi dari beberapa komputer.

Penggunaan Hadoop pada umumnya menggunakan perangkat *high end*. Oleh karena itu, harga perangkatnya menjadi mahal dan menjadi kendala tersendiri untuk melakukan uji coba. Salah satu solusi alternatif untuk mengurangi biaya pada uji coba penggunaan Hadoop dapat dilakukan dengan memanfaatkan perangkat yang jauh lebih murah yaitu Raspberry Pi.

Hadoop merupakan platform yang sebenarnya dapat berjalan di berbagai macam perangkat keras, sehingga implementasi pada Raspberry Pi dapat dilakukan. Namun, keterbatasan pada komponennya seperti memori, prosesor, dll, menyebabkan proses penyimpanan dan pengolahan data menjadi terbatas.

Hasil akhir dari Tugas Akhir ini adalah membandingkan kinerja antara Hadoop menggunakan beberapa Datanode pada Raspberry Pi.

1.2. Rumusan Permasalahan

Rumusan masalah yang diangkat pada tugas akhir ini dapat dipaparkan sebagai berikut:

1. Bagaimana mengimplementasikan Hadoop pada Raspberry Pi ?
2. Bagaimana membangun arsitektur klaster Hadoop untuk meningkatkan kinerja paralel ?

1.3. Batasan Permasalahan

Beberapa batasan masalah yang terdapat dalam tugas akhir ini dapat dipaparkan sebagai berikut:

1. Klaster Hadoop yang dibuat menggunakan Raspberry Pi 1 Model B dan Raspberry Pi 2 Model B.
2. Program MapReduce yang digunakan untuk uji coba adalah program WordCount.
3. Kluster Hadoop yang akan diimplementasikan sebanyak 1 klaster, yang terdiri dari Namenode dan Datanode
4. Optimasi difokuskan pada program yang menggunakan *framework* MapReduce.
5. File yang digunakan dalam uji coba memiliki ukuran kurang dari 20MB.

1.4. Tujuan

Tujuan dari pembuatan tugas akhir ini antara adalah Melakukan perhitungan kinerja Hadoop pada setiap node dan perbandingan kinerja sebelum dan setelah penyesuaian.

1.5. Manfaat

Manfaat dari pembuatan tugas akhir ini antara lain :

1. Memberikan solusi alternatif untuk penyimpanan dan pemrosesan data secara paralel.
2. Sebagai rekomendasi dalam membuat *micro data center*.

1.6. Metodologi

Pembuatan Tugas Akhir dilakukan menggunakan metodologi sebagai berikut

1.6.1. Penyusunan proposal Tugas Akhir

Proposal tugas akhir ini berisi tentang deskripsi pendahuluan dari tugas akhir yang akan dibuat. Pendahuluan pada proposal tugas akhir ini terdiri dari latar belakang diajukannya usulan tugas akhir, rumusan masalah yang diangkat, batasan masalah untuk tugas akhir, tujuan dari pembuatan tugas akhir, dan manfaat hasil dari pembuatan tugas akhir. Selain itu dijelaskan pula tinjauan pustaka yang digunakan sebagai referensi pendukung implementasi tugas akhir. Pada proposal ini juga terdapat perencanaan jadwal pengerjaan tugas akhir.

1.6.2. Studi literatur

Pada studi literatur ini, akan dipelajari sejumlah referensi yang diperlukan dalam implementasi sistem, yaitu mengenai Raspberry Pi, MapReduce, HDFS, dan bahasa pemrograman Java.

1.6.3. Analisis dan Perancangan Sistem

Tahap ini meliputi perancangan sitem berdasarkan studi literatur dan pembelajaran konsep teknologi dari perangkat lunak yang ada. Langkah-langkah yang dikerjakan juga didefinisikan pada tahap ini. Pada tahapan ini dibuat purwa-rupa sistem dengan menggunakan Virtual Box, yang digunakan sebagai rancangan dasar dari sistem yang akan dibuat. Serta dilakukan desain suatu sistem dan desain proses-proses yang ada.

1.6.4. Implementasi

Implementasi merupakan tahap membangun rancangan program yang telah dibuat. Pada tahapan ini merealisasikan apa yang terdapat pada tahapan sebelumnya.

1.6.5. Pengujian dan Evaluasi

Pada tahap ini dilakukan penyusunan laporan yang menjelaskan dasar teori dan metode yang digunakan dalam tugas akhir ini serta hasil dari implementasi.

1.6.6. Penyusunan Buku Tugas Akhir

Pada tahap ini dilakukan penyusunan laporan yang menjelaskan dasar teori dan metode yang digunakan dalam tugas akhir ini serta hasil dari implementasi aplikasi perangkat lunak yang telah dibuat.

1.7. Sistematika Penulisan

Buku Tugas Akhir ini bertujuan mendapatkan gambaran dari pengerjaan. Selain itu, diharapkan dapat berguna untuk mengetahui pentingnya penyesuaian Hadoop pada perangkat keras dan perhitungan *processing capability* dan *busy level* sebagai tanda bahwa suatu node sibuk dan perlunya dilakukan penyeimbang pembagian *task*. Secara garis besar, Tugas Akhir terdiri atas beberapa bagian seperti berikut ini:

Bab I Pendahuluan
Bab ini berisi latar belakang, tujuan, dan manfaat dari pembuatan Tugas Akhir. Selain itu, juga terdapat permasalahan, batasan masalah, metodologi yang digunakan, dan sistematika penulisan juga merupakan bagian dari bab ini.

Bab II Tinjauan Pustaka
Bab ini membahas dasar pembuatan dan beberapa teori penunjang yang berhubungan dengan pokok pembahasan yang mendasari pembuatan Tugas Akhir ini.

Bab III Analisis dan Perancangan Sistem

Bab ini membahas analisis dari sistem yang dibuat meliputi deskripsi umum sistem, arsitektur klaster Hadoop, dan data. Kemudian membahas analisis dari sistem yang dibuat meliputi analisis pemilihan perangkat keras, analisis pemilihan versi Hadoop, Raspberry Pi *overclock*, optimasi blok data Hadoop, kompresi *output* Mapper, *memory tunning*, optimasi ukuran file, JVM *reuse*, dan manajemen *transfer rate* dengan SDN.

Bab IV Implementasi

Bab ini membahas implementasi dari desain yang telah dibuat pada bab sebelumnya. Penjelasan berupa code yang digunakan untuk proses implementasi.

Bab V Pengujian dan Evaluasi

Bab ini membahas pengujian dari penyesuaian perangkat yang dibuat dengan melihat keluaran yang dihasilkan dan evaluasi untuk mengetahui kemampuan sistem.

Bab VI Penutup

Bab ini merupakan bab terakhir yang menyampaikan kesimpulan dari hasil uji coba yang dilakukan dan saran untuk pengembangan perangkat lunak ke depannya.

[Halaman ini sengaja dikosongkan]

BAB II
TINJAUAN PUSTAKA

Bab ini berisi penjelasan teori-teori yang berkaitan dengan rancangan yang diajukan pada penyesuaian Hadoop pada Raspberry Pi. Penjelasan ini bertujuan untuk memberikan gambaran secara umum dalam penyesuaian Hadoop pada Raspberry Pi.

2.1. Raspberry Pi

Raspberry Pi atau lebih sering disingkat dengan Raspi. Raspberry pi merupakan salah satu *low cost embedded machine*. Raspberry pi adalah komputer dengan *motherboard* (*Single Board Circuit*/SBC) yang memiliki ukuran sebesar kartu kredit. Raspberry Pi dapat digunakan untuk keperluan uji coba, *spreadsheet*, server kecil dan sebagainya.

Fondasi Sistem Operasi resmi milik Raspberry Pi adalah Raspbian. Bahasa yang didukung oleh Raspbian bermacam-macam, utamanya adalah Python. Bahasa lainnya juga didukung pada Raspbian seperti C, C++, Java, Perl, Ruby, dan lain sebagainya. Dengan dukungan berbagai macam bahasa maka penggunaan Hadoop pada Raspberry akan menjadi memungkinkan.

Saat ini Raspberry Pi memiliki beberapa model yaitu Raspberry Pi 3 Model B, Raspberry Pi 2 Model B, Pi Zero, dan Raspberry Pi 1 Model B+ dan A+ [1].

Raspberry Pi 1 Model A+ adalah varian murah dari Raspberry Pi. Dimana pada Raspberry Pi Model A memiliki ukuran RAM 256MB, 1 USB *port*, dan tidak memiliki Ethernet *port*. Sedangkan pada Raspberry Pi dengan Model B adalah hasil dari perubahan Raspberry Pi 1 Model A. Dimana memiliki ukuran RAM mencapai 512MB (2 kali lebih besar dari Raspberry Pi 1 Model A), 4 USB *port*, 40 GPIO pins, dan sebuah Ethernet *port*. Raspberry Pi 2 Model B adalah generasi dari Raspberry Pi 1 Model B+. Pada Raspberry Pi 2 memiliki banyak kesamaan

dengan Raspberry Pi 1 Model B+, tetapi menggunakan prosessor 900MHz quad-core ARM Cortex-A7 CPU dan memiliki RAM dengan ukuran 1GB. Raspberry Pi 2 Model B adalah model yang direkomendasikan untuk riset karena fleksibilitasnya untuk pembejaran [1].

2.2. Apache Hadoop

Hadoop [2] adalah *framework* atau *platform open source* yang digunakan untuk pengolahan terdistribusi terhadap data dalam jumlah besar. Hadoop didesain untuk memperbesar skala dari server tunggal menjadi ribuan mesin, dimana masing-masing mesin membagi kemampuan penyimpanan dan komputasi. Prinsip dari Hadoop adalah daripada bergantung pada mesin yang memiliki *avalaibility* yang tinggi, *library* pada Hadoop didesain untuk mendeteksi dan menangani kegagalan di *layer application* .

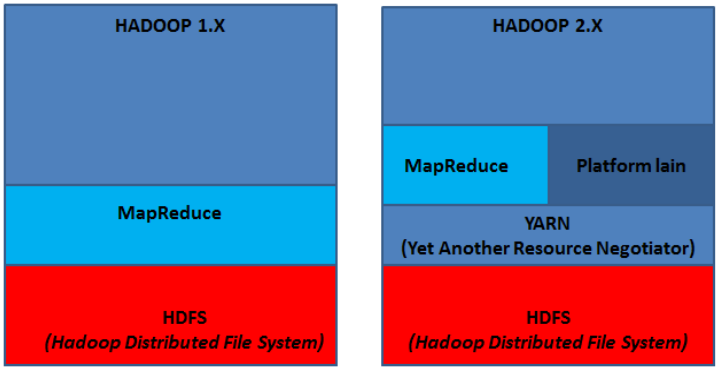
Hadoop berbeda dengan basis data, Hadoop lebih ke sebuah *software* yang secara khusus mengangani data dengan jumlah yang besar dan data yang tidak terstruktur berbeda dengan basis data tradisional yang menangani data yang bersifat terstruktur.

Data terstruktur adalah data yang memiliki batas-batas yang tetap pada suatu *record* atau file. Meskipun dalam jumlah besar, data terstruktur dapat dimasukkan, disimpan, di-*query*-kan, dan dianalisa dengan menggunakan basis data tradisional. Sedangkan data tidak terstruktur merupakan data yang datang dari berbagai macam sumber, seperti *email*, dokumen, teks, video, foto, audio, file, dan bahkan posting-posting dari media sosial. Hadoop memiliki kemampuan untuk menggabungkan, membedakan dan menganalisa berbagai macam data tanpa harus mengetahui strukturnya.

Dengan demikian alasan penggunakan Hadoop dalam pengolahan *dataset* dengan jumlah yang sangat besar (*big data*) dan data yang tak terstruktur. Dalam memproses suatu operasi atau *job* Hadoop akan membagi/memecah *job* menjadi *task* saat

memproses data, pembagian *task* dilakukan secara terdistribusi dengan memanfaatkan komputer-komputer lain yang terhubung membentuk suatu klaster. Hadoop dirancang untuk meningkatkan atau menggabungkan suatu server tunggal menjadi ribuan mesin mesin, dengan mesin-mesin lain saling membagi kemampuan komputasi dan penyimpanannya. Daripada mengandalkan suatu *hardware* untuk menyediakan kemampuan *avalaibility* (Sistem berjalan secara terus menerus dalam waktu yang lama) yang tinggi, maka penggunaan *library* Hadoop sangat baik karena dirancang untuk mendeteksi dan menangani setiap kegagalan pada lapisan *application layer*. Sehingga memberikan layanan *avalaibility* yang baik.

Dalam perkembangannya, secara garis besar Hadoop memiliki dua versi. Kedua versi ini memiliki perubahan segnifikan, sehingga berpengaruh terhadap perangkat yang akan digunakan untuk mengimplementasikan lingkungan Hadoop tersebut. Perubahan Hadoop yang paling signifikan terlihat antara Hadoop versi pertama dan Hadoop versi kedua.



Gambar 2.1 Perbandingan Arsitektur Hadoop

Perbedaan antara Hadoop versi pertama dan versi kedua dapat dilihat pada arsitekturnya. Pada arsitektur Hadoop pertama terdapat dua *layer* utama. *Layer* pertama adalah HDFS (*Hadoop*

Distributed File System), dan *layer* kedua adalah MapReduce. Sedangkan pada arsitektur Hadoop kedua terdapat tiga *layer* Utama. *Layer* pertama adalah HDFS (*Hadoop Distributed File System*), *layer* kedua adalah YARN (*Yet Another Resource Negotiator*), dan *layer* ketiga adalah MapReduce.

Komponen utama Hadoop versi pertama terdiri dari Namenode, Datanode, TaskTracker, JobTracker, dan SecondaryNamenode. Komponen utama Hadoop versi kedua terdiri dari Namenode, Datanode, ResourceManager, NodeManager, dan SecondaryNamenode. Berikut adalah penjelasan mengenai komponen-komponen tersebut:

2.2.1. Namenode

Namenode [3] merupakan salah satu komponen utama Hadoop. Berikut adalah detailnya :

- Pusat dari *file* sistem HDFS. Namenode sendiri sering disebut dengan *master node*.
- Namenode hanya menyimpan *metadata* dari atau pohon direktori (*Directory Tree*) seluruh *file* yang ada didalam *file* sistem yang sebenarnya disimpan pada Datanode.
- Namenode juga dapat melacak dimana data disimpan pada klaster lain.
- Namenode juga mengatur penyimpanan data dengan mengatur ukuran dan *metadata* dari blok data yang akan disimpan. Sehingga Namenode mengetahui list dari blok dimana dari informasi ini Namenode dapat membangun kembali data dari blok yang disimpan.
- Namenode sangat penting bagi HDFS karena jika Namenode mati, maka HDFS/klaster Hadoop tidak akan dapat diakses.
- Namenode sendiri tidak bertugas untuk melakukan penyimpanan data.
- Pemilihan perangkat keras untuk Namenode, biasanya dipilih dari perangkat keras yang memiliki jumlah RAM yang besar.

2.2.2. Datanode

Datanode [3] adalah *node* pekerja dari klaster Hadoop. Berikut adalah detailnya:

- Datanode bertugas dalam menyimpan blok data dari Hadoop.
- Datanode juga dikenal sebagai *slave node*.
- Data yang disimpan pada Datanode bersifat nyata
- Ketika Datanode mati, maka hal itu tidak akan memberi efek ke *avalaibility* data. Karena Namenode mengatur replikasi data ketika data disimpan.
- Datanode biasanya diatur agar memiliki alokasi *harddisk* yang besar.

2.2.3. JobTracker

JobTracker [3] adalah komponen pusat pengaturan yang menangani masalah proses MapReduce. JobTracker hanya ada pada Hadoop versi pertama. Berikut adalah detailnya:

- JobTracker menampilkan seluruh sumber daya proses pada klaster Hadoop.
- JobTracker hanya dapat dijalankan di Namenode
- Menangani proses *scheduling job*
- Melacak status dari proses MapReduce
- Koordinator seluruh aplikasi MapReduce

2.2.4. TaskTracker

TaskTracker [3] adalah komponen Hadoop yang bertindak sebagai *slave node* dalam menjalankan proses MapReduce. TaskTracker hanya ada pada Hadoop versi pertama. Berikut adalah Detailnya:

- TaskTracker berjalan pada setiap Datanode. Yang berarti setiap Datanode mampu menyimpan *file* dan memroses *file* yang diminta.
- TaskTracker menerima *request* untuk melakukan *task* yang diberikan.

2.2.5. SecondaryNamenode

SecondaryNamenode [3] bukanlah pengganti Namenode ketika terjadi kesalahan pada Namenode atau penyimpanan *backup image* Namenode. SecondaryNamenode hanya menyimpan *log* dalam jumlah besar pada Namenode.

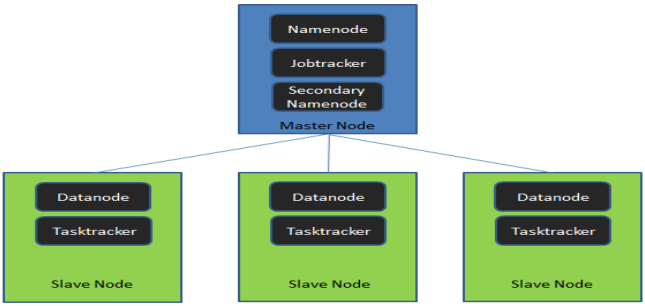
2.2.6. ResourceManager

ResourceManager [3] merupakan *framework* untuk melakukan komputasi data yang terdapat pada Hadoop kedua. ResourceManager memiliki kewenangan yang besar dalam mengatur proses MapReduce. Seperti mengatur jalannya *scheduling job*. ResourceManager akan berada satu paket bersama dengan Namenode.

2.2.7. NodeManager

NodeManager [3] merupakan *framework* untuk melakukan komputasi data yang terdapat pada Hadoop kedua. Berfungsi untuk melakukan pengawasan terhadap penggunaan CPU, memori, *disk*, dan jaringan. NodeManager akan berada pada setiap Datanode.

Pada klaster Hadoop versi pertama dikenal dengan istilah-istilah Namenode, Datanode, Tasktracker, SecondaryNamenode dan Jobtracker. Dalam implementasinya pada klaster Hadoop umumnya Namenode menjadi satu bagian dengan Jobtracker dan SecondaryNamenode. Bagian ini sering disebut *master node*. Sedangkan Datanode umumnya menjadi satu dengan Tasktracker dan sering disebut *slave node*. *Node* yang dihubungkan oleh 1 *switch* atau *router* disebut dengan 1 rak. Sedangkan rak-rak yang terhubung dan memiliki 1 Namenode disebut dengan klaster. Gambarannya dapat dilihat pada Gambar 2.2.

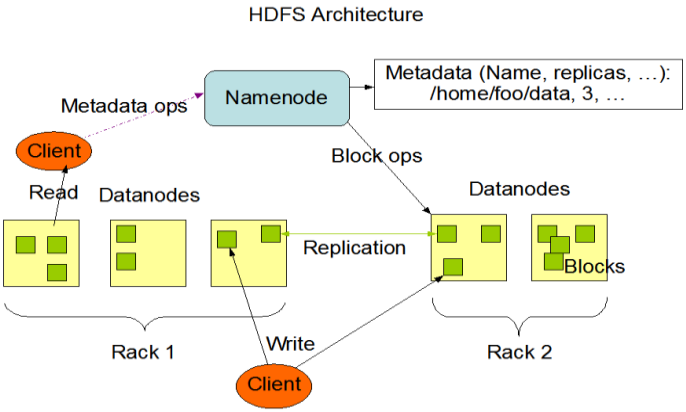


Gambar 2.2 Komponen klaster Hadoop

2.3. *HDFS (*Hadoop Distributed File System*)*

HDFS (*Hadoop Distributed File System*) [4] merupakan bagian dari Hadoop yang didesain untuk menyimpan data dengan ukuran yang besar dengan pola *streaming data access* dan mengatur proses distribusi data. Pada HDFS, *file* sistem tidak langsung menyimpan data dalam satu *harddisk* (HDD) atau media penyimpanan lainnya, tetapi data akan dipecah-pecah dan disimpan tersebar dalam klaster yang terdiri dari beberapa komputer.

Pada HDFS terdapat beberapa jenis kategori *node*, diantaranya adalah Namenode, Datanode, dan SecondaryNamenode. Namenode adalah *node* utama pada HDFS dengan tugas penempatan data di klaster dan menerima *job* untuk pengolahan dari MapReduce. Datanode adalah *node* tempat data ditempatkan. Satu blok Datanode terdiri dari 64 MB (standar Hadoop). SecondaryNamenode adalah *node* yang bertugas untuk menyimpan informasi penyimpanan data dan pengolahan data yang ada di Namenode.



Gambar 2.3 Arsitektur HDFS [2]

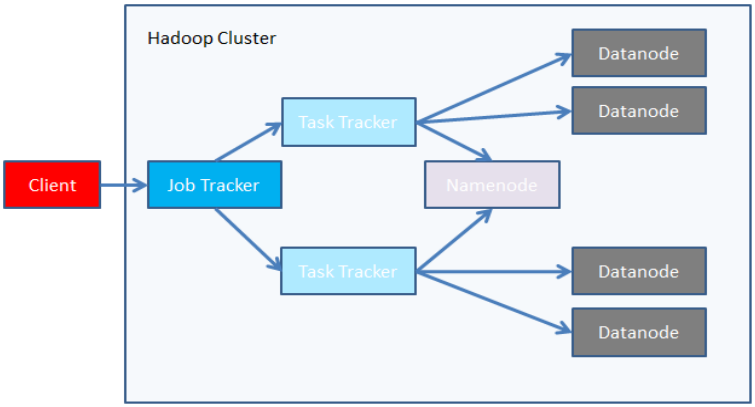
Perbedaan antara Hadoop dan SQL terletak pada penyimpanan datanya. Hadoop menggunakan pendekatan *schema on read* yang berarti dalam pertukaran datanya karena tidak memperhatikan tipe data atau struktur data tersebut. Berbeda dengan SQL yang menggunakan metode *schema on write* yang berarti jika dilakukan pertukaran data, maka basis data harus memiliki struktur yang sama. Jika tidak maka akan terjadi penolakan data. Pada SQL basis data dibentuk dengan relasi antar tabel, sedangkan pada Hadoop data akan dikompres dan disimpan dalam *format* teks dan data tersebut akan diduplikasi pada beberapa Datanode. Duplikasi dilakukan agar data dapat diakses meskipun terdapat Datanode yang mati.

2.4. MapReduce

MapReduce [5] merupakan sebuah framework pada Hadoop yang mengatur tentang *processing* data dalam jumlah besar dan model distribusi komputasi paralel yang berbasiskan Java. MapReduce job biasanya memecah proses masukan data ke dalam potongan-potongan kecil yang akan diproses oleh *map tasks* dalam proses paralel. Kemudia MapReduce mengurutkan

keluaran dari proses Map, yang merupakan input dari *task reduce*. Biasanya masukan dan hasil dari MapReduce akan disimpan pada HDFS.

Program MapReduce dikirimkan dari *client*. Kemudian Jobtracker akan melakukan inisialisasi *job* untuk mengerjakannya. Jobtracker kemudian akan melakukan *split task* lalu mengirimkan datanya ke Tasktracker. Di Tasktracker akan dibuat JVM (*Java Virtual Machine*) untuk mengeksekusi program MapReduce. Setelah proses Map Selesai maka data akan digabungkan pada proses Reduce dan Tasktracker akan mengirimkan kembali hasilnya ke Jobtracker.



Gambar 2.4 Proses job disubmit dari client

Pada MapReduce terdapat dua proses utama yaitu Map dan Reduce. Map bertugas untuk mengambil data dan mengubahnya kedalam set data yang lain, dimana terjadi pemecahan data. Sedangkan Reduce merupakan proses setelah Map dimana mendapatkan *input* dari *output* yang dihasilkan Map.

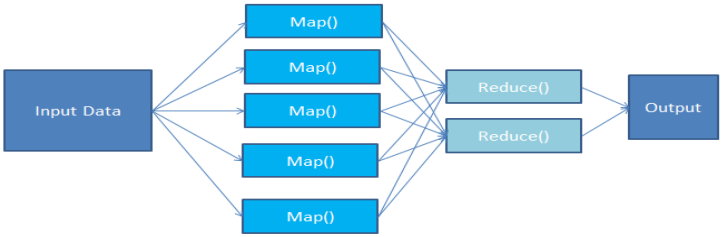
MapReduce diproses oleh dua kategori *node*, yaitu: JobTracker dan TaskTracker. Dimana JobTracker hanya dimiliki satu per klaster. Memiliki tugas untuk menerima perintah dari klien, *scheduling* dan monitor perintah dari MapReduce pada TaskTracker. *Node* TaskTracker dapat pada klaster dengan

jumlah yang banyak. Tasktracker ini memiliki tugas untuk mengeksekusi operasi dari MapReduce. Tasktracker akan mengakses Datanode untuk mendapatkan blok data yang diinginkan lalu megirimkan hasilnya menuju Namenode.

Untuk algoritma MapReduce dapat diibaratkan seperti mengirimkan komputer ke tempat dimana data itu berada. Pada algoritma MapReduce terdapat 3 tahap saat terjadi proses eksekusi program, yaitu tahap map, shuffle, dan Reduce.

- Tahap Map
  Proses Map adalah proses saat input data. Pada umumnya input data disimpan dalam bentuk *file* atau direktori dan disimpan di dalam HDFS.
- Tahap Reduce
  Pada tahap ini terjadi gabungan antara tahap Shuffle dan Reduce. Tugas dari Reduce adalah memroses data yang datang dari mapper. Setelah memroses, mkaa reducer akan menghasilkan *output* dan menyimpannya di dalam HDFS.

Selama proses MapReduce Job dieksekusi, Hadoop mengirimkan *task* Map dan Reduce menuju Datanode. *Framework* akan menangani seluruh detail tentang *passing* data, verifikasi *task* ketika selesai, dan menyalin data yang ada pada klaster. Sebagian besar, komputasi yang dilakukan ditempat data itu berada akan mengurangi lalu lintas jaringan. Setelah menyelesaikan *task* yang diberikan, klaster akan mengumpulkan dan membentuk data dalam bentuk yang sesuai, dan mengirimnya kembali ke Namenode.



Gambar 2.5 Proses MapReduce

Skema pembagian *task* pada Hadoop sederhana dan intuitif. Dimana ketika sebuah slot *task* tersedia, maka Hadoop akan mengirimkan task untuk dikerjakan pada Datanode tersebut. Sehingga Hadoop akan mencegah sebuah Datanode dari proses *idle*. Untuk *task* yang pertama kali dikerjakan, pemilihan berdasarkan Datanode mana yang pertama kali berhasil dideteksi Namenode ketika Hadoop dijalankan.

2.5. Rumen

Apache Rumen [6] adalah *tools* dari Apache Hadoop berbasis bahasa Java yang digunakan untuk mengekstrak dan menganalisis informasi dari *job* Hadoop. Rumen terdiri dari dua komponen:

- *TraceBuilder*
  Menerjemahkan catatan *job* dari Hadoop kedalam *format* JSON.
- *Folder*
  Tools yang digunakan untuk memanipulasi masukan *trace*. *trace* didapatkan berdasarkan dari *TraceBuilder* yang menjumlahkan proses *job*.

[Halaman ini sengaja dikosongkan]

BAB III
ANALISIS DAN PERANCANGAN SISTEM

Pada bab ini akan dijelaskan mengenai analisis dan perancangan dari arsitektur Hadoop dan dasar-dasar dalam penyesuaian program MapReduce pada Raspberry Pi. Adapun hal-hal yang dibahas dalam bab ini adalah deskripsi umum sistem, arsitektur klaster Hadoop, analisis pemilihan perangkat keras untuk desain arsitektur lingkungan Hadoop, optimasi blok data Hadoop, kompresi keluaran Map, *memory tuning*, optimasi ukuran *file*, dan JVM *reuse*.

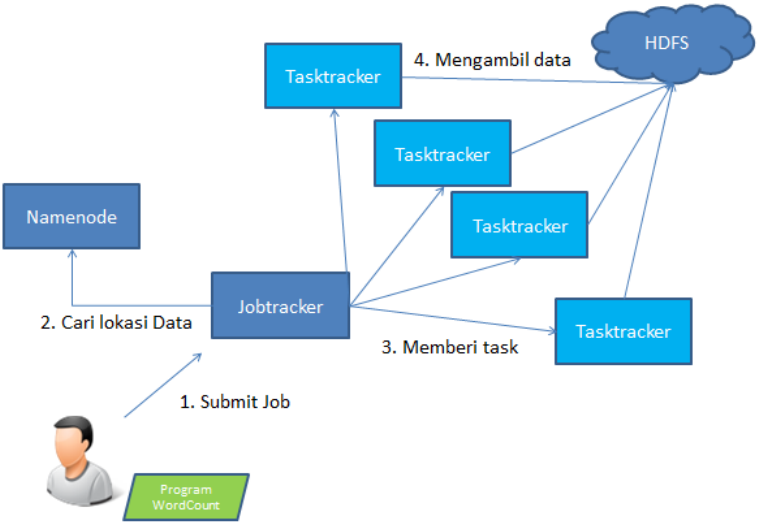
3.1. Analisis Perangkat Lunak

Pada subbab ini dijelaskan mengenai deskripsi umum sistem, arsitektur klaster Hadoop yang akan dibangun, data uji dan alur pengujian yang digunakan.

3.1.1. Deskripsi Umum Sistem

Pada Tugas Akhir ini, akan dibangun beberapa percobaan klaster Hadoop dengan menggunakan Raspberry Pi. Raspberry Pi yang digunakan dengan ketentuan bahwa Namenode Hadoop menggunakan Raspberry Pi 2 Model B dan Datanode Hadoop menggunakan Raspberry Pi 1 Model B. Sehingga dapat dikatakan Raspberry Pi 2 Model B akan menjadi *master* dan Raspberry Pi 1 Model B akan menjadi *slave*.

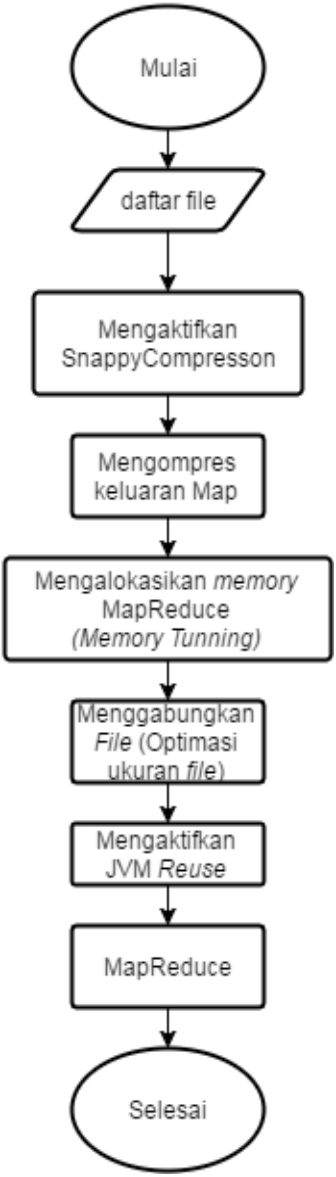
Pada klaster Hadoop yang dibangun, Hadoop mampu menjalankan sebuah *job* yang dikirimkan dari sisi *master* atau sisi *slave*. Program yang digunakan untuk uji coba dalam optimasi adalah program WordCount atau program untuk menghitung jumlah kata pada suatu *file* dengan menggunakan *framework* MapReduce.



Gambar 3.1 Gambaran umum alur sistem

Optimasi akan dilakukan pada sisi Raspberry Pi dan program MapReduce. Optimasi pada Raspberry Pi dilakukan dikarenakan Raspberry Pi merupakan perangkat *low cost embedded* sedangkan Hadoop adalah *platform big data* yang biasanya diaplikasikan pada perangkat keras yang mutakhir. Sehingga diperlukannya optimasi dalam menjalankannya pada Raspberry Pi.

Untuk uji coba penyesuaian program MapReduce digunakan program WordCount untuk mengujinya. Proses penyesuaian program MapReduce sebenarnya dibagi menjadi tahap *memory tunning*, tahap kompresi keluaran Map, optimasi ukuran *file* dan JVM *reuse*. Seluruh tahap ini dapat digabungkan menjadi tahap Konfigurasi PraProsessing. Diagram alir untuk desain umum WordCount dapat ditunjukkan pada Gambar 3.2.



Gambar 3.2 Diagram alir program WordCount

Pada Gambar 3.2 merupakan diagram alir dari program WordCount dengan mengimplementasikan fitur-fitur penyesuaiannya. Dimana ketika input dilakukan proses konfigurasi pra prosesessing yang terdiri dari optimasi blok data, kompresi keluaran Map, *memory tunning*, optimasi ukuran *file*, dan JVM *reuse*. Setelah melakukan penyesuaian, maka program MapReduce akan dieksekusi sesuai konfigurasi tersebut.

3.1.2. Arsitektur Klaster Hadoop

Klaster Hadoop yang akan dibangun menggunakan Datanode dan Namenode dari Raspberry Pi. Dengan *switch* biasa dan dari Raspberry Pi untuk mengimplementasikan konsep SDN.

Kemudian program MapReduce diuji dengan dijalankan pada klaster tersebut. Sehingga didapatkan waktu pemrosesannya dan kemudian waktu per *task* diekstrak menggunakan *tools* dari Hadoop, yaitu Rumen.

3.1.3. Data

Pada sub bab ini akan dijelaskan mengenai data yang digunakan sebagai masukan perangkat lunak untuk selanjutnya diolah dan dilakukan pengujian sehingga menghasilkan data keluaran yang diharapkan.

1) Data Masukan

Data masukan adalah data yang digunakan sebagai masukan dari sistem. Data yang akan digunakan adalah data berupa teks, dokumen, xml, atau semua dapat yang data yang dapat diubah menjadi teks. Besar data yang diuji adalah dari ukuran 100KB, 1MB, 10MB, dan 20MB. Ilustrasi data masukan dapat dilihat pada Gambar 3.3

```
-rw-r--r--   4 hduser supergroup     135380 2016-06-05 08:30 /user/hduser/input/9289.male.23.Marketing.Taurus (48th copy).xml
-rw-r--r--   4 hduser supergroup     135380 2016-06-05 08:30 /user/hduser/input/9289.male.23.Marketing.Taurus (49th copy).xml
-rw-r--r--   4 hduser supergroup     135380 2016-06-05 08:30 /user/hduser/input/9289.male.23.Marketing.Taurus (4th copy).xml
-rw-r--r--   4 hduser supergroup     135380 2016-06-05 08:30 /user/hduser/input/9289.male.23.Marketing.Taurus (50th copy).xml
-rw-r--r--   4 hduser supergroup     135380 2016-06-05 08:30 /user/hduser/input/9289.male.23.Marketing.Taurus (51st copy).xml
-rw-r--r--   4 hduser supergroup     135380 2016-06-05 08:30 /user/hduser/input/9289.male.23.Marketing.Taurus (52nd copy).xml
-rw-r--r--   4 hduser supergroup     135380 2016-06-05 08:30 /user/hduser/input/9289.male.23.Marketing.Taurus (53rd copy).xml
-rw-r--r--   4 hduser supergroup     135380 2016-06-05 08:30 /user/hduser/input/9289.male.23.Marketing.Taurus (54th copy).xml
-rw-r--r--   4 hduser supergroup     135380 2016-06-05 08:30 /user/hduser/input/9289.male.23.Marketing.Taurus (55th copy).xml
-rw-r--r--   4 hduser supergroup     135380 2016-06-05 08:30 /user/hduser/input/9289.male.23.Marketing.Taurus (56th copy).xml
-rw-r--r--   4 hduser supergroup     135380 2016-06-05 08:30 /user/hduser/input/9289.male.23.Marketing.Taurus (57th copy).xml
-rw-r--r--   4 hduser supergroup     135380 2016-06-05 08:30 /user/hduser/input/9289.male.23.Marketing.Taurus (58th copy).xml
-rw-r--r--   4 hduser supergroup     135380 2016-06-05 08:30 /user/hduser/input/9289.male.23.Marketing.Taurus (59th copy).xml
-rw-r--r--   4 hduser supergroup     135380 2016-06-05 08:30 /user/hduser/input/9289.male.23.Marketing.Taurus (5th copy).xml
-rw-r--r--   4 hduser supergroup     135380 2016-06-05 08:30 /user/hduser/input/9289.male.23.Marketing.Taurus (60th copy).xml
-rw-r--r--   4 hduser supergroup     135380 2016-06-05 08:30 /user/hduser/input/9289.male.23.Marketing.Taurus (61st copy).xml
-rw-r--r--   4 hduser supergroup     135380 2016-06-05 08:30 /user/hduser/input/9289.male.23.Marketing.Taurus (62nd copy).xml
-rw-r--r--   4 hduser supergroup     135380 2016-06-05 08:30 /user/hduser/input/9289.male.23.Marketing.Taurus (63rd copy).xml
-rw-r--r--   4 hduser supergroup     135380 2016-06-05 08:30 /user/hduser/input/9289.male.23.Marketing.Taurus (64th copy).xml
-rw-r--r--   4 hduser supergroup     135380 2016-06-05 08:30 /user/hduser/input/9289.male.23.Marketing.Taurus (65th copy).xml
-rw-r--r--   4 hduser supergroup     135380 2016-06-05 08:30 /user/hduser/input/9289.male.23.Marketing.Taurus (66th copy).xml
-rw-r--r--   4 hduser supergroup     135380 2016-06-05 08:30 /user/hduser/input/9289.male.23.Marketing.Taurus (67th copy).xml
-rw-r--r--   4 hduser supergroup     135380 2016-06-05 08:30 /user/hduser/input/9289.male.23.Marketing.Taurus (68th copy).xml
-rw-r--r--   4 hduser supergroup     135380 2016-06-05 08:30 /user/hduser/input/9289.male.23.Marketing.Taurus (69th copy).xml
-rw-r--r--   4 hduser supergroup     135380 2016-06-05 08:30 /user/hduser/input/9289.male.23.Marketing.Taurus (6th copy).xml
-rw-r--r--   4 hduser supergroup     135380 2016-06-05 08:30 /user/hduser/input/9289.male.23.Marketing.Taurus (70th copy).xml
-rw-r--r--   4 hduser supergroup     135380 2016-06-05 08:30 /user/hduser/input/9289.male.23.Marketing.Taurus (71st copy).xml
-rw-r--r--   4 hduser supergroup     135380 2016-06-05 08:30 /user/hduser/input/9289.male.23.Marketing.Taurus (72nd copy).xml
-rw-r--r--   4 hduser supergroup     135380 2016-06-05 08:30 /user/hduser/input/9289.male.23.Marketing.Taurus (73rd copy).xml
-rw-r--r--   4 hduser supergroup     135380 2016-06-05 08:30 /user/hduser/input/9289.male.23.Marketing.Taurus (74th copy).xml
-rw-r--r--   4 hduser supergroup     135380 2016-06-05 08:30 /user/hduser/input/9289.male.23.Marketing.Taurus (75th copy).xml
-rw-r--r--   4 hduser supergroup     135380 2016-06-05 08:30 /user/hduser/input/9289.male.23.Marketing.Taurus (76th copy).xml
-rw-r--r--   4 hduser supergroup     135380 2016-06-05 08:30 /user/hduser/input/9289.male.23.Marketing.Taurus (77th copy).xml
-rw-r--r--   4 hduser supergroup     135380 2016-06-05 08:30 /user/hduser/input/9289.male.23.Marketing.Taurus (78th copy).xml
-rw-r--r--   4 hduser supergroup     135380 2016-06-05 08:30 /user/hduser/input/9289.male.23.Marketing.Taurus (7th copy).xml
-rw-r--r--   4 hduser supergroup     135380 2016-06-05 08:30 /user/hduser/input/9289.male.23.Marketing.Taurus (8th copy).xml
-rw-r--r--   4 hduser supergroup     135380 2016-06-05 08:30 /user/hduser/input/9289.male.23.Marketing.Taurus (9th copy).xml
-rw-r--r--   4 hduser supergroup     135380 2016-06-05 08:30 /user/hduser/input/9289.male.23.Marketing.Taurus (another copy).xml
-rw-r--r--   4 hduser supergroup     135380 2016-06-05 08:30 /user/hduser/input/9289.male.23.Marketing.Taurus (copy).xml
-rw-r--r--   4 hduser supergroup     135380 2016-06-05 08:30 /user/hduser/input/9289.male.23.Marketing.Taurus.xml
-rw-r--r--   4 hduser supergroup     134187 2016-06-05 08:30 /user/hduser/input/9470.male.25.Communications-Media.Aries (10th copy).xml
```

Gambar 3.3 Masukkan Data

2) Data Keluaran

Data keluaran adalah data hasil perhitungan jumlah kata pada program WordCount dengan mengoptimalkan proses MapReduce dengan mengoptimalkan perangkat keras yang digunakan, Hadoop yang digunakan, mengoptimalkan blok data, mengoptimalkan memori, mengompres blok data saat menjalankan program, mengoptimalkan *file-file* saat diproses (*file* kecil). Dari proses tersebut diharapkan akan mengoptimalkan proses dari MapReduce. Ilustrasi data keluaran dapat dilihat pada Gambar 3.4.

Gambar 3.4 Keluaran Data

3.2. Analisis

Subbab ini membahas analisis bagaimana penyesuaian pada Hadoop agar dapat berjalan pada Raspberry Pi. Hal ini meliputi analisis pemilihan perangkat keras, analisis pemilihan versi Hadoop, Raspberry Pi *overclock*, optimasi blok data, kompresi keluaran Map, *memory tunning*, optimasi ukuran *file*, JVM *reuse*, dan manajemen *transfer rate* dengan SDN (*Software Defined Networking*).

3.2.1. Analisis Pemilihan Perangkat Keras

Hadoop merupakan salah satu *platform big data* dimana dalam implementasinya menggunakan perangkat *high end*. Dikarenakan dalam pembangunan arsitektur menggunakan Raspberry Pi maka untuk meningkatkan performanya diperlukan pemilihan jenis Raspberry Pi untuk menentukan peran Namenode dan Datanode-nya.

1) Menentukan Namenode

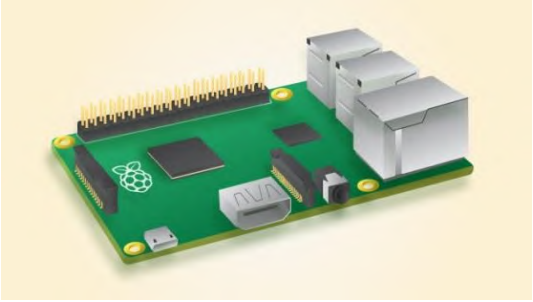
Namenode adalah pusat dari sistem HDFS. Dimana pada Namenode dicatat *directory tree* pada data yang disimpan pada sistem *file*. Pada Namenode tidak terjadi penyimpanan data.

Namenode juga mendefinisikan Jobtracker. Dimana JobTracker adalah sebuah fitur Hadoop yang mengurus *task* untuk *node* dalam klaster.

Namenode memiliki peran penting dikarenakan mengurus keseluruhan node dalam klaster. Sehingga, memerlukan spesifikasi yang lebih baik dari datanode

Pemilihan Raspberry Pi 2 Model B ke atas disarankan untuk memainkan peran sebagai Datanode. Dikarenakan memiliki spesifikasi :

- 900 MHz quad-core ARM Cortex-A7 CPU (ARMv7)
- 1 GB RAM



Gambar 3.5 Raspberry Pi 2 Model B

2) Menentukan Datanode

Datanode berfungsi sebagai penyimpan data pada HDFS. Secara fungsional Datanode akan melakukan replikasi data jika terdapat lebih dari satu *node*.

Pada Datanode terdapat juga TaskTracker. Tasktracker adalah *node* Hadoop dimana berfungsi sebagai penerima *task* (berupa operasi Map, Reduce dan Shuffle) yang diberikan oleh JobTracker dari Namenode.

Pemilihan Raspberry Pi 1 Model B ke atas cocok untuk memainkan peran sebagai Datanode. Berikut spesifikasi yang dari Raspberry Pi 1 Model B:

- 700 MHz ARM1176JZF-S Core CPU (ARMv6)
- 512 MB RAM

Gambar 3.6 Raspberry Pi 1 Model B

3.2.2. Analisis Pemilihan Versi Hadoop

Pada dasarnya Hadoop adalah *platform* yang digunakan untuk menangani data dalam jumlah yang besar. Sehingga untuk mengimplementasikannya pada Raspberry Pi dibutuhkan Hadoop yang cocok dengan spesifikasi perangkatnya.

Dalam perkembangannya, secara garis besar Hadoop memiliki dua versi. Kedua versi ini memiliki perubahan segnifikan, sehingga berpengaruh terhadap perangkat yang akan digunakan untuk mengimplementasikan lingkungan Hadoop tersebut. Perubahan Hadoop yang paling signifikan terlihat antara Hadoop versi pertama dan Hadoop versi kedua [2]. Perbedaan antara keduanya adalah sebagai berikut.

1) Hadoop versi pertama

- Hadoop hanya mendukung model proses MapReduce. Namun, tidak mendukung tools yang bersifat non-MapReduce.
- MapReduce melakukan operasi dan manajemen *resource* pada klaster

- Untuk jumlah *node* Hadoop pertama memiliki batas 4000 *node* per klaster.
- Namenode tunggal mengatur seluruh *namespace* pada klaster tersebut.
- Didukung oleh *prosessor* minimum ARMv6
- API MapReduce cocok dengan Hadoop pertama. Program yang dibuat pada Hadoop pertama maka hanya dapat dijalankan pada Hadoop pertama.
- Hadoop pertama memiliki batasan untuk *platform* pada memproses data, *streaming,* dan operasi yang bersifat *real-time*.

Hadoop versi 2

- Hadoop versi kedua membuat MapReduce dapat bekerja baik pada komputasi terdistribusi seperti Spark, Hama, Giraph, *Massage Passing Interface*(MPI) dan Hbase.
- YARN (*Yet Another Resource Negotiator*) adalah pengatur sumber daya pada klaster (*cluster resource management*), *scheduling/monitoring* pada Hadoop.
- Hadoop kedua telah meningkatkan jumlah node yang dapat ditangani, yaitu 1000 per klaster
- *Multiple* Namenode dapat mengatur *multiple namespace* dengan berbagi sumber daya.
- API MapReduce membutuhkan *file* tambahan untuk program yang ditulis pada Hadoop versi pertama, untuk dijalankan pada Hadoop versi kedua.
- Hadoop pertama memiliki batasan untuk *platform* pada memroses data, *streaming,* dan operasi yang bersifat *real-time*.
- Didukung oleh *prosessor* minimum ARMv7

Dari kesimpulan di atas, Hadoop kedua memiliki banyak keunggulan, Namun membutuhkan dukungan dari *prosessor* minimum ARMv7 yang menyebabkan Raspberry Pi 1 versi B yang hanya memiliki ARMv6 tidak dapat mengimplementasikannya. Sehingga Hadoop versi pertama yang memungkinkan untuk implementasinya.

Pemilihan Hadoop versi pertama untuk implementasi pada Raspberry mengakibatkan tidak dapat dipakainya Java versi 1.7 ke atas, dikarenakan Hadoop versi pertama hanya dapat mengeksekusi program yang ditulis dan di-*compile* dengan menggunakan Java versi 1.6 kebawah.

3.2.3. Rapsberry Pi *Overclock*

Overclock pada perangkat keras adalah meningkatkan kinerja perangkat keras untuk berjalan pada kecepatan lebih tinggi dari ketentuan perangkat keras tersebut. Hal ini dilakukan dengan meningkatkan kecepatan *clock* pada CPU. Pada dasarnya *overclock* bersifat memaksa perangkat keras dalam meningkatkan kinerjanya, pemaksaan ini membuat perangkat menjadi tidak stabil sehingga dapat merusak *mainboard, prosessor,* RAM, atau VGA pada perangkat keras tersebut.

3.2.4. Optimasi Blok Data Hadoop

Secara *default* blok data Hadoop telah dikonfigurasi oleh Hadoop dengan ukuran 64MB. Yang berarti setiap 64MB, data baru di pecah menjadi blok baru setelah ukurannya mencapai 64MB. Dengan demikian maka *node* Hadoop akan memproses data dengan ukuran 64MB per *task*.

Semisal terdapat data dengan ukuran 100MB, maka Hadoop akan memecah data menjadi dua bagian, yaitu 64MB dan 36MB.

Jika dibandingkan dengan blok data sistem Linux yang memiliki ukuran 4KB, blok data Hadoop tergolong sangat besar dengan ukuran yang mencapai 64MB. Hal ini disebabkan Hadoop didesain untuk menyimpan data mencapai ukuran petabyte. Namun, ukuran besar tersebut memiliki kelemahan.

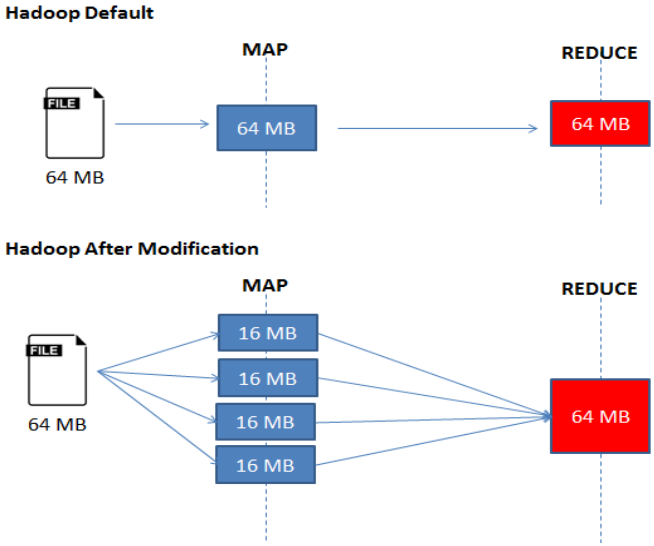
Pertama, setiap blok data disimpan dalam HDFS memiliki *metadata* dan butuh dilacak oleh server pusat (Namenode) sehingga aplikasi apapun yang membutuhkan data itu dapat memiliki akses menuju *file* tersebut dimanapun data tersebut disimpan. Hal ini memberi kelemahan jika blok data yang dicari hanya dalam ukuran

kilobyte, karena server akan kesulitan dalam mencari data disebabkan terbanjirinya metadata server yang perlu dilacak.

Kedua, HDFS telah didesain memiliki *througput* yang besar sehingga dapat melakukan proses paralel pada *file* yang besar secepat mungkin. Hal ini memberikan beberapa pertimbangan dalam menentukanukuran blok Hadoop diantaranya sebagai berikut.

- Semakin besar blok data maka akan memaksimalkan proses MapReduce pada suatu *node*. Namun hal ini dapat mengakibatkan terlalu lamanya sistem menunggu pada satu proses yang terlalu besar sehingga menyia-nyiakan proses paralel yang telah disediakan Hadoop.
- Semakin kecil blok data Hadoop maka akan memanfaatkan proses paralel dari Hadoop. Namun hal ini dapat mengakibatkan terciptanya proses Map terlalu banyak yang mengakibatkan tidak dimanfaatkannya secara maksimal kemampuan individu dari Datanode Hadoop.

Gambar 3.7 adalah simulasi yang ditunjukkan untuk optimasi dari blok data Hadoop.



Gambar 3.7 Simulasi optimasi blok data

3.2.5. Kompresi Keluaran Map

Bottleneck adalah proses dimana aliran data dibatasi karena faktor dari spesifikasi perangkat keras yang dimiliki oleh komputer tersebut atau *resource* pada jaringan komputer tersebut. Pada Hadoop, seluruh *bottleneck* terdiri dari 4 jenis yaitu penggunaan CPU, RAM, jaringan, atau I/O.

Kompresi data adalah salah teknik untuk mengurangi dampak yang ditimbulkan dari *bottleneck*, Pada Hadoop terdapat 3 jenis kompresi data [8].

1) Kompresi masukan *raw*
   Kompresi yang dilakukan pada saat MapReduce melakukan proses masukan Map.
2) Kompresi antara keluaran dari proses Map dan masukan Reduce
   Kompresi dilakukan pada hasil keluaran proses Map yang merupakan proses input dari Reduce
3) Kompresi pada keluaran Reduce
   Kompresi data yang dilakukan pada hasil dari proses Reduce yang merupakan hasil akhir.

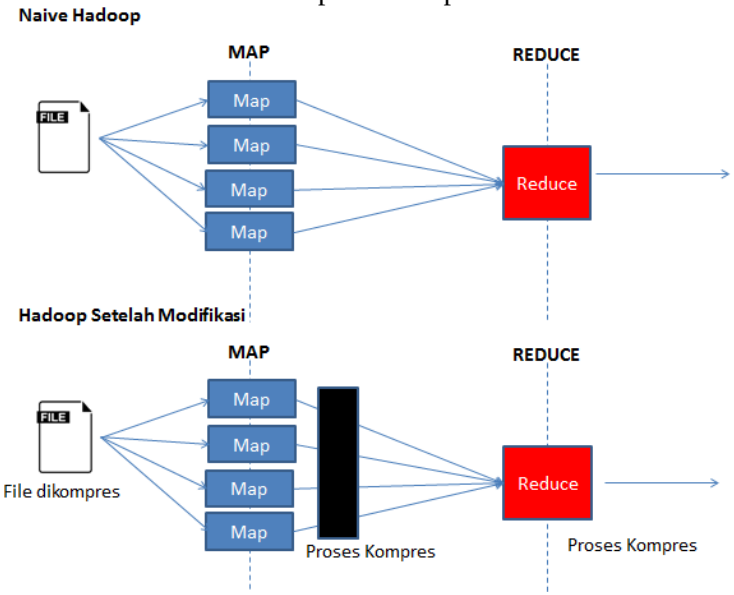
Umumnya *bandwidth* yang rendah merupakan penyebab utama dari masalah *bottleneck*. Pada Hadoop hal ini dapat diakibatkan oleh *Heartbeat* dan *transfer* data saat terjadi proses MapReduce.

Heartbeat merupakan fungsi pada Hadoop dimana Namenode akan mengirim paket (*heartbeat*) setiap 10 menit sekali menuju seluruh *node* dan Datanode tersebut akan memberi respon menuju Namenode. Dikarenakan pengiriman paket data menggunakan router yang sebenarnya adalah Raspbery Pi. Tentu perlu dilakukan pencocokan ukuran *bandwidth* yang dialokasikan dari router tersebut. Jika ukuran *bandwidth* terlalu kecil maka dapat menyebabkan *bottleneck* yang disebabkan *broadcast strorm* (Kejadian dimana jaringan dibanjiri oleh paket) pada *router* yang menghubungkan Namenode.

Transfer data MapReduce juga dapat menyebabkan *Bottleneck* pada Hadoop dikarenakan MapReduce memiliki kemampuan tranfer data mencapai 100Mbit [9].

Sehingga untuk mencegah *bottlenect*, dapat dilakukan kompresi masukan pada proses Reduce. Kompresi dilakukan pada data-data yang merupakan hasil dari Map dan yang akan digunakan sebagai input Reducer.

Dengan melakukan kompresi data-data yang akan diolah pada saat proses MapReduce maka diharapkan data-data yang dikirimkan pada proses Map dan Reduce akan mampu mengurangi dampak dari *bottleneck.*Ilustrasi dapat dilihat pada Gambar 3.8.



Gambar 3.8 Simulasi Kompresi data

3.2.6. Memory Tunning

Konsep dasar dari *memory tuning* adalah bagaimana suatu proses dapat dikerjakan dengan memori semaksimal mungkin tanpa

melakukan proses *swapping* (proses dimana dipindahkannya suatu proses dari *memory* utama ke *memory* sekunder).

Proses MapReduce adalah proses Java dengan masing-masing proses memiliki nilai maksimum *heap memory*. Untuk Hadoop konfigurasi ukuran *heap* dapat dilakukan melalui variabel mapred.map.child.java.opts. Hal ini diperlukan karena jika proses melawati batas *heap* maka Java akan terjadi *memory* [10].
Karena proses MapReduce dilakukan pada Datanode, maka diperlukan kembali penyesuaian pada Datanode. Dikarenakan Datanode adalah Raspberry Pi 1 model B yang memiliki *memory* sebesar 512MB.. Untuk Datanode ukuran dari *heap* dapat dikonfigurasi sebesar 256MB yang merupakan ukuran minimum yang diperlukan untuk java dan maksimum 512MB yang merupakan ukuran *memory* maksimum pada Raspberry Pi 1 model B.

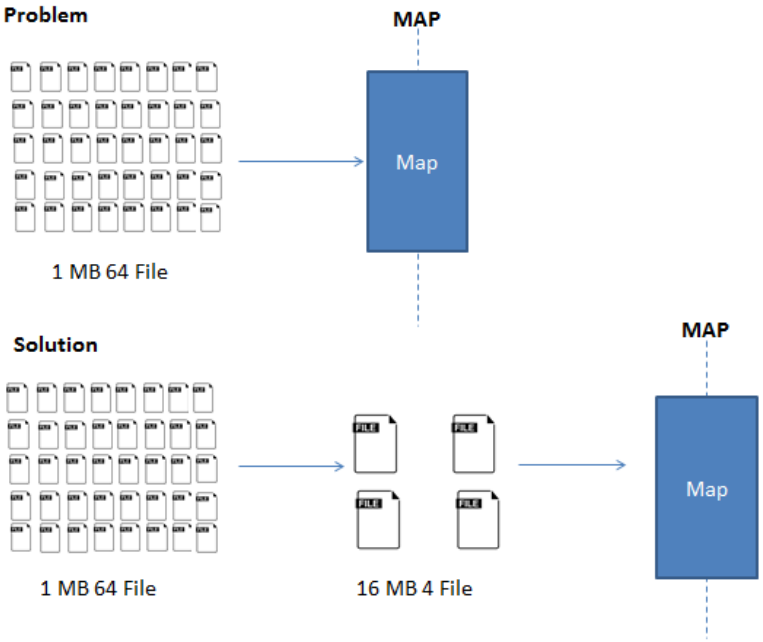
3.2.7. Optimasi Ukuran *File*

File kecil adalah suatu masalah tersendiri pada Hadoop. Pada Hadoop *file* yang tergolong ukuran kecil adalah *file* yang ukurannya kurang dari ukuran dari blok HDFS yang dialokasikan (*default* 64MB). Jika menyimpan *file* kecil dalam jumlah banyak maka akan terjadi masalah ketika HDFS mencoba mengakses *file* tersebut untuk diolah pada saat program MapReduce.

Proses map *task* biasanya dilakukan bersamaan dengan memproses blok input. Jika *file* sangat kecil dan berjumlah banyak , maka setiap proses map *task* akan memproses masukan yang sangat kecil. Maka, dapat dibayangkan jika 1GB *file* yang dipecah menjadi 16 *file* berukuran 64MB blok, dibandingkan dengan 10.000 *file* dengan ukuran 100KB, dimana 10.000 *file* tersebut masing-masing akan menggunakan satu Map proses tersendiri. Maka *file* dengan jumlah 10.000 *file* akan jauh lebih lama dibanding dengan 16 *file* berukuran 64MB.

Salah satu cara untuk mengatasi *file-file* kecil yang akan digunakan untuk proses MapReduce adalah dengan mengoptimalkan program WordCount yang yang akan dijalankan dengan cara melakukan penggabungan pada *file-file* yang akan diolah pada

MapReduce. Penggabungan *file-file* yang akan diolah oleh MapReduce perlu memperhitungkan ukuran blok dari HDFS dan ukuran dari *heap* Java. Sehingga untuk Raspberry Pi ukuran *file* yang akan digabungkan adalah 16MB, 1MB, dan 100KB per *file* atau sama dengan ukuran dari *split* blok data dari ukuran blok data Hadoop. Untuk gambarannya dapat dilihat pada Gambar 3.0.

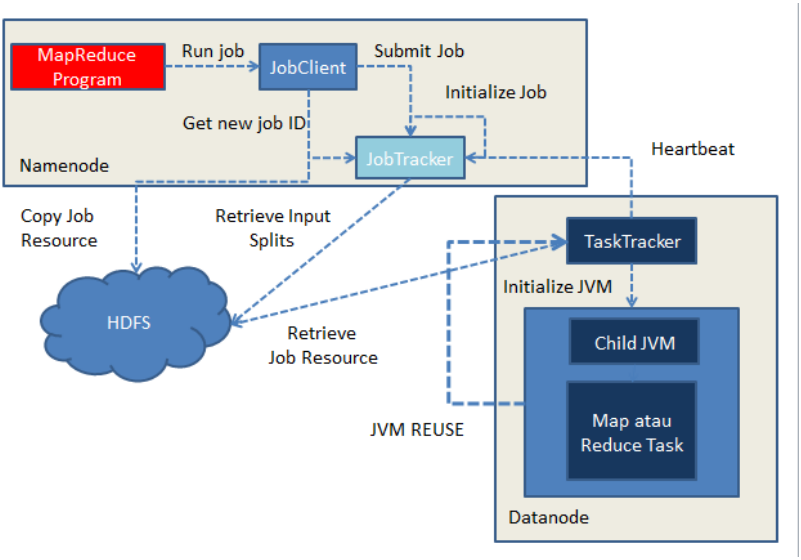


Gambar 3.9 Optimasi Ukuran *File*

3.2.8. JVM *Reuse*

JVM (Java Virtual Machine) dari *thread* yang terbentuk dari program MapReduce dapat digunakan kembali atau mengeksekusi lebih dari satu *task* yang didapatkan oleh suatu *node*. Dengan mengubah tingkah laku dari JVM maka akan membuat JVM dapat mengerjakan beberapa *task* per JVM.

Mengaktifkan fitur JVM *reuse* dapat mereduksi proses *overhead* yang terjadi pada proses inisialisasi dan mematikan JVM. Normalnya fitur ini hanya dapat dirasakan pada kasus dimana diprosesnya *task* dalam jumlah besar yang memiliki waktu proses yang lama. JVM *Reuse* hanya dapat digunakan pada Hadoop versi pertama [8].



Gambar 3.10 Gambar skenario JVM Reuse

Gambar 3.10 menjelaskan kapan penggunaan JVM *reuse*. Ketika program MapReduce dijalankan maka akan membentuk *job.* Kemudian fungsi dari JobClient akan memberi ID pada *job* tersebut dan akan memasukkan *job* tersebut kedalam JobTracker. *Job* yang masuk kedalam JobTracker akan masuk kedalam *queue*. Dimana ketika sudah gilirannya akan dipecah menjadi *task* dan dikirimkan ke TaskTracker. Tasktracker akan membentuk JVM *child* untuk mengerjakan *task* yang diberikan oleh Jobtracker. Pada umumnya setelah *task* selesai JVM akan dihancurkan (*destroy*) dan ketika *task*

baru masuk maka Tasktracker akan menginisialisasi kembali JVM yang baru.

Ide dari penggunaan JVM *reuse* adalah memanfaatkan JVM yang telah terinisialisasi untuk dipakai kembali jika *task* baru muncul di Datanode tersebut.

3.2.9. Manajemen *Transfer Rate* dengan SDN (*Software Defined Networking*)

Pengaturan manajemen *transfer rate* perlu dilakukan dikarenakan saat program MapReduce berjalan maka lalu lintas jaringan akan menjadi lebih padat. Padatnya lalu lintas jaringan pada klaster Hadoop dikarenakan program MapReduce membutuhkan *transfer rate* yang besar yaitu 100Mbit [10].

Pengaturan manajemen *transfer rate* ini dilakukan untuk mengatur prioritas pada lalu lintas jaringan. Sehingga saat program MapReduce berjalan program-program lain yang sekiranya ikut mengonsumsi *bandwidth* akan menjadi prioritas kedua. Hal ini dapat dilakukan dengan menggunakan konsep SDN (*Software Defined Network*).

SDN merupakan suatu konsep pada jaringan komputer dimana pengontrol paket data pada jaringan dipisahkan pada jaringan tersebut. Dimana dengan menggunakan SDN maka pengaturan prioritas *transfer rate* dapat dilakukan.

[Halaman ini sengaja dikosongkan]

BAB IV
IMPLEMENTASI

Pada bab ini akan dijelaskan tentang implementasi dari perancangan dan analisis pada bab sebelumnya yang dikembangkan untuk Tugas Akhir. Implementasi merupakan bentuk realisasi dari perancangan dan analisis dari optimasi MapReduce yang telah di jelaskan pada bab sebelumnya. Adapun hal-hal yang akan dibahas pada bab ini adalah lingkungan implementasi arsitektur Hadoop pada Raspberry Pi, *pseudocode* dari perangkat lunak, dan *screenshot* hasil implementasi.

4.1. Implementasi Arsitektur Hadoop pada Raspberry Pi

Pada subbab ini akan dijelaskan tentang implementasi arsitektur Hadoop, meliputi implementasi pemilihan perangkat keras, implementasi pemilihan versi Hadoop, instalasi Raspbian dan konfigurasi jaringan, dan instalasi dan konfigurasi Hadoop.

4.1.1. Implementasi Pemilihan Perangkat Keras

Subbab ini membahas implementasi tahap pemilihan perangkat keras sebagai *node* untuk menjalankan Hadoop. Pada dasarnya Hadoop terdiri dari *master* dan *slave node*. *Master node* adalah *node* yang bertugas menjalankan Namenode, Jobtracker, dan SecondaryNamenode. *Slave node* adalah *node* yang bertugas untuk menjalankan Datanode dan TaskTracker.

Untuk Implementasi *master node* menggunakan Raspberry Pi 2 model B dan implementasi *slave* node menggunakan Raspberry Pi 1 model B yang sesuai dengan analisis.

4.1.2. Implementasi Pemilihan Versi Hadoop

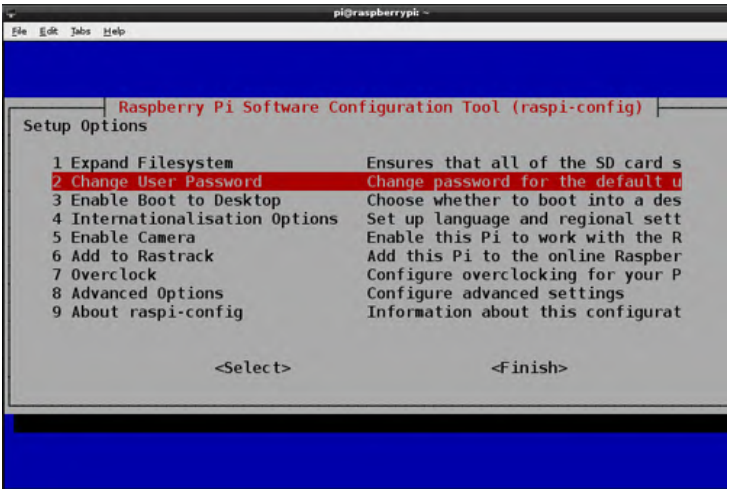
Subbab ini membahas implementasi tahap pemilihan versi Hadoop untuk dapat berjalan pada Raspberry Pi. Sesuai dengan analisis bahwa Hadoop versi kedua tidak dapat berjalan pada Raspberry Pi 1 model B dikarenakan masalah *prosessor*. Sehingga Hadoop versi 1.2.1 dipilih untuk dijalankan pada Raspberry Pi.

4.1.3. Instalasi Raspbian dan Konfigurasi Jaringan

Raspbian adalah Sistem Operasi resmi yang khusus digunakan sebagai sistem operasi pada Raspberry Pi. Raspbian dapat dengan mudah didapatkan dengan men-*download* pada situs resminya. Cara instalasi akan dijelaskan dibawah sebagai berikut :

1. *Download* Raspbian
2. Ekstrak *image* Raspbian pada Micro SD yang digunakan sebagai *storage device* pada Raspberry Pi.
3. Untuk *login* pada Raspberry Pi dapat menggunakan *username* : pi dan *password* : Raspberry
4. Konfigurasi SSH dan *hostname*
   Untuk mengaktifkan SSH dan mengatur *hostname* pada Raspberry dapat dilakukan dengan konsol dari terminal Raspbian. Dengan perintah :

```
pi@raspberrypi ~ $ sudo raspi-config
```



Gambar 4.1 raspi-config

Pada menu ketujuh "*Advanced*" terdapat konfigurasi dalam mengaktifkan atau menonaktifkan mode pada SSH, karena

Hadoop berkomunikasi dengan menggunakan SSH maka pastikan SSH aktif. Untuk *hostname* dapat dikonfigurasi pada menu ini. Konfigurasi untuk melakukan *overclock* juga dapat dilakukan dengan menggunakan perintah di atas. *overclock* dapat diubah konfigurasinya dari level *Medium* ke *High*.

5. Konfigurasi alamat IP

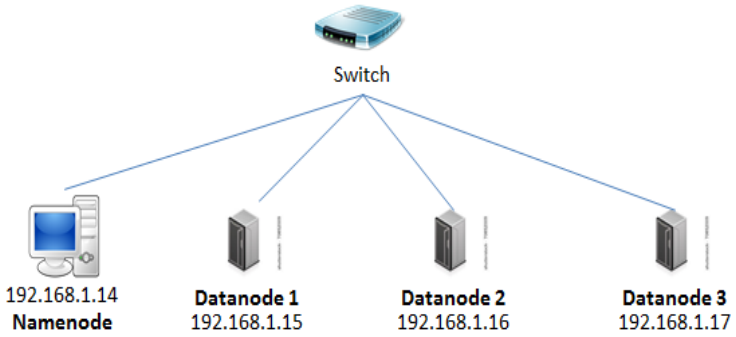
Konfigurasi alamat IP dapat dilakukan pada masing-masing perangkat Raspberry. Dengan perintah pada konsol :

```
pi@raspberrypi ~ $  sudo nano /etc/network/interfaces
```

Untuk konfigurasi alamat IP yang diperlukan adalah sebagai berikut:

- Namenode dengan ip 192.168.1.14
- Datanode 1 dengan ip 192.168.1.15
- Datanode 2 dengan ip 192.168.1.16
- Datanode 3 dengan ip 192.168.117

Maka arsitektur klaster yang akan terbentuk seperti pada Gambar 4.2.



Gambar 4.2 Arsitektur Jaringan

Lakukan *restart* pada perangkat yang diubah alamat IP. Lakukan uji ping atau SSH untuk memastikan bahwa alamat IP sudah benar.

4.1.4. Instalasi dan Konfigurasi Hadoop

Untuk persiapan instalasi Hadoop dapat didapatkan pada situs resmi Hadoop Apache. Pada situs resminya Hadoop terdapat berbagai macam versi. Versi terbaru Hadoop adalah versi 2.6.2. Namun, implementasi pada Raspberry Pi 1 menggunakan Hadoop versi 1.2.1. Langkah intallasi dan konfigurasi adalah sebagai berikut.

1. Install Java

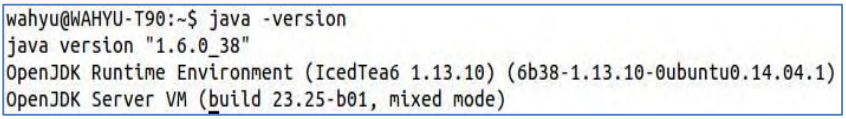
Hadoop versi pertama hanya mendukung Java versi 6. Maka untuk instalasi Java 6 dapat dilakukan dengan perintah sebagai berikut:

```
pi@raspberrypi ~ $ sudo apt-get install openjdk-6-jdk
```

Jika telah selesai dapat dicek dengan perintah sebagai berikut:

```
pi@raspberrypi ~ $ java -version
```

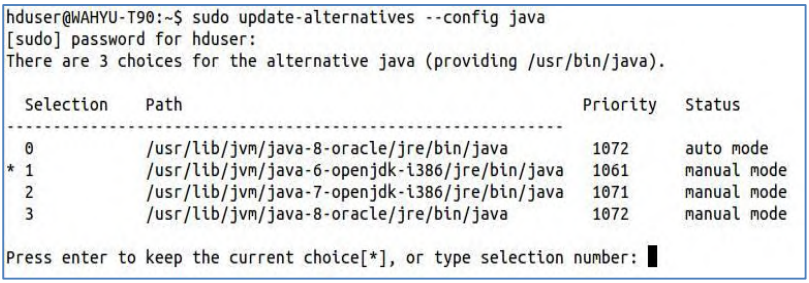
Maka akan muncul tampilan seperti Gambar 4.3.



Gambar 4.3 Versi Java

Jika Java 6 sudah terinstall namun tidak berubah menjadi Java 6. Maka ada kemungkinan terdapat dua atau lebih versi Java didalam komputer tersebut, maka untuk menukar versi Java dapat dilakukan dengan perintah sebagai berikut:

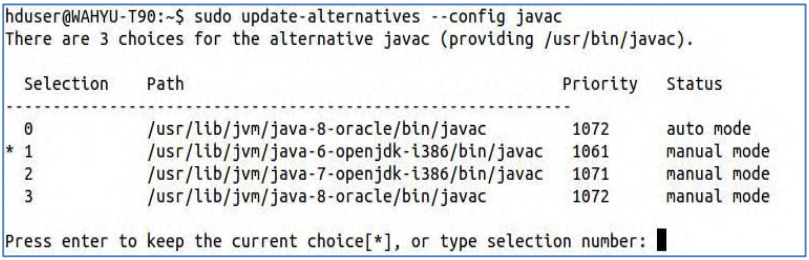
```
sudo  update-alternatives --config java
```



Gambar 4.4 Memilih Versi Java

Jika sukses mengganti Java maka akan muncul tampilan seperti gambar di atas. Perintah untuk mengubah versi Javac adalah sebagai berikut:

```
sudo  update-alternatives --config javac
```



Gambar 4.5 Memilih Versi Javac

Jika sukses mengganti Java maka akan muncul tampilan seperti gambar di atas.

Karena Java versi 6 sudah tertinggal maka ada kemungkinan perintah di atas tak dapat dilakukan. Maka untuk mendapatkan Java versi 6 perlu men-*download* secara manual pada situs resmi Oracle Dan mengistallnya secara *manual*.

2. Membuat Akun Pengguna untuk Hadoop

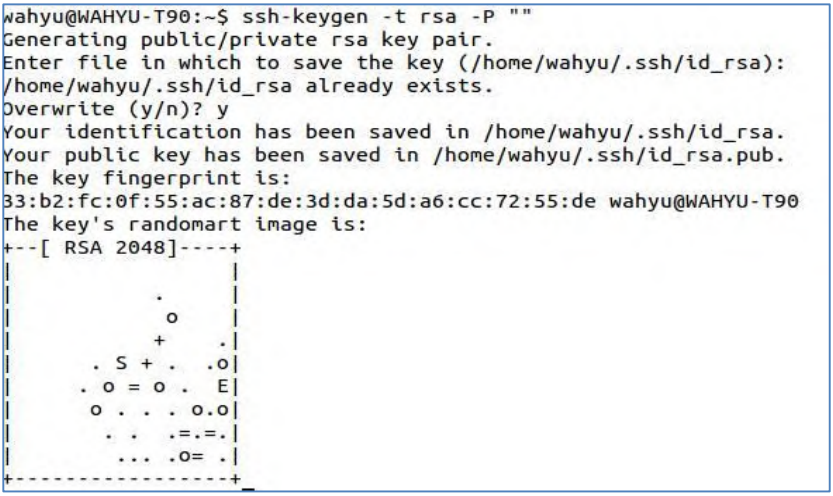
Akun pengguna khusus untuk akses Hadoop juga perlu dibuat sebab berguna untuk mempermudah akses ke setiap node yang telah terbentuk. Pengguna dapat dibuat dengan perintah sebagai berikut.

```
pi@raspberrypi ~ $ sudo addgroup hadoop
pi@raspberrypi ~ $ sudo adduser –ingroup hadoop hduser
pi@raspberrypi ~ $ sudo adduser hduser
```

3. Konfigurasi SSH

Agar Namenode Hadoop pada berkomunikasi dengan Datanode maka diperlukan SSH (*Secure Shell*) dalam proses komunikasinya.

```
pi@raspberrypi ~ $ su hduser
hduser@raspberrypi ~ $ ssh-keygen –t rsa –P  “”
```



Gambar 4.6 ***Generate SSH Key***

Copy kunci RSA ke *file authorized_keys.* Dengan perintah:

```
hduser@raspberrypi ~ $ cat ~/.ssh/id_rsa.pub >>
~/.ssh/authorized_keys
```

Dengan perintah ini, maka sistem akan membuat kunci RSA dengan *password* yang kosong. Jadi Hadoop tidak akan menanyakan *passpharse* saat melakukan *prompting* ke node-nodenya. SSH dapat dicoba dengan perintah.

```
hduser@raspberrypi ~ $ ssh 127.0.0.1
```

4. Nonaktifkan Ipv6

Hadoop untuk saat ini tidak mendukung jaringan dengan Ipv6. Hadoop hanya diuji dan dikembangkan dengan menggunakan Ipv4, sehingga dalam implementasinya Hadoop hanya bekerja pada Ipv4. Sehingga untuk menginstall Hadoop perlu menonaktifkan Ipv6 terlebih dahulu. Menonaktifkannya dengan mengubah *file sysctl.conf.*

```
hduser@raspberrypi ~ $ sudo nano /etc/sysctl.conf
```

Tambahkan kode berikut untuk menonaktifkan Ipv6

```
net.ipv6.conf.all.disable_ipv6 = 1
net.ipv6.conf.default.disable_ipv6 = 1
net.ipv6.conf.lo.disable_ipv6 = 1
```

Restart mesin untuk melihat perubahan yang terjadi. Untuk melihat apakah Ipv6 telah dinonaktifkan atau tidak dapat menggunakan perintah:

```
hduser@raspberrypi ~ $ cat
```

Jika nilai kembali yang dikeluarkan adalah 0 berarti Ipv6 telah dinonaktifkan, jika 1 maka sebaliknya.

5. Download dan Install Hadoop

Hadoop dapat di-*download* pada situs resminya. Hadoop yang digunakan dalam implementasi adalah Hadoop versi 1.2.1. Dengan perintah sebagai berikut.

```
hduser@raspberrypi~$ sudo tar vwzf hadoop-1.2.1.tar.gz
hduser@raspberrypi~$ sudo mv hadoop-1.2.1.tar.gz
/usr/local
hduser@raspberrypi~$  cd /usr/local/
hduser@raspberrypi /usr/local/~$ sudo mv hadoop-1.2.1
hadoop
hduser@raspberrypi /usr/local/~$ sudo chown –R
hduser:hadoop hadoop
```

6. Konfigurasi Hadoop

Pengaturan Konfigurasi yang utama adalah konfigurasi yang akan dipakai oleh Namenode. Untuk konfigurasi Datanode dapat memakai konfigurasi hasil konfigurasi Namenode. *File-file* yang perlu dikonfigurasi adalah sebagai berikut.

- *hadoop-env.sh*

File hadoop-env.sh berisi tentang konfigurasi lingkungan Hadoop saat menjalankan JVM. Konfigurasinya meliputi letak Java yang akan digunakan Hadoop, ukuran *heap* dan sebagainya. Konfigurasi hadoop-env.sh dapat dilakukan dengan perintah sebagai berikut.

```
hduser@raspberrypi~$sudo nano $HADOOP_HOME/conf/hadoop-
env.sh
```

- *core-site.xml*

File core-site.xml berisi berbagai macam informasi inti dari Hadoop, seperti nomor *port* yang digunakan Hadoop, alokasi memori untuk sistem, batas memori penyimpanan data, dan ukuran ukuran *buffer* untuk proses *read/write*. *File* core-site.xml dapat diakses dengan perintah sebagai berikut.

```
sudo nano $HADOOP_HOME/conf/core-site.xml
```

Tabel 4.1 adalah konfigurasi dan nilai yang dipakai pada lingkungan klaster Hadoop secara umum.

Tabel 4.1 Konfigurasi core-site.xml

Kelas	Fungsi	Nilai
hadoop.tmp.dir	Mendefinisikan letak dari direktori tempat	/fs/hadoop/tmp

	penyimpanan dari HDFS dan penyimpanan data *file* sistem dari MapReduce.	
fs.default.name	Mendefinisikan nama *hostname* dari Namenode dan *port* yang digunakannya	hdfs://master:54310

- *mapred-site.xml*

File mapred-site.xml berisi tentang konfigurasi pada MapReduce yang akan dijalankan. Melakukan konfigurasi MapReduce di *file* ini akan memprioritaskan konfigurasi ini saat menjalankan program MapReduce. *File mapred-site.xml* dapat diakses dengan perintah sebagai berikut.

```
hduser@raspberrypi~$nano $HADOOP_HOME/conf/mapred-site.xml
```

Tabel 4.2 adalah konfigurasi dan nilai yang dipakai pada lingkungan klaster Hadoop secara umum.

Tabel 4.2 Konfigurasi mapred-site.xml

Kelas	Fungsi	Nilai
mapred.job.tracker	Mendefinisikan nama JobTracker dan alokasi port untuk JobTracker	master:54311

- *hdfs-site.xml*

File hdfs-site,xml berisi tentang informasi-informasi mengenai HDFS (*Hadoop Distributed File System*), seperti jumlah replikasi data, path Namenode, dan *path* Datanode pada sistem yang bersifat lokal. *File hdfs-site.xml* dapat diakses dengan perintah sebagai berikut.

```
hduser@raspberrypi~$nano $HADOOP_HOME/conf/hdfs-site.xml
```

Tabel 4.3 adalah konfigurasi dan nilai yang dipakai pada lingkungan klaster Hadoop secara umum.

Tabel 4.3 Konfigurasi hdfs-site.xml

Kelas	Fungsi	Nilai
dfs.replication	Mereplikasi *file* yang disimpan ke dalam HDFS (nilai bawaan adalah 3)	3
dfs.block.size	Mengatur nilai blok data dari HDFS	16777216

Nilai dari *dfs.replication* adalah satu menyebabkan data yang disimpan kedalam Hadoop hanya memiliki satu data. Hal, ini bertujuan untuk memaksa program MapReduce untuk membagikan *task* ke Datanode.

Untuk *dfs.block size* diatur 16777216 byte atau 16MB dari yang seharusnya 64MB. Memperkecil blok data dari HDFS adalah bagian penyesuaian dari blok data hadoop karena mempertimbangkan memori Raspberry yang relatif rendah, ukuran *file* yang kecil dan total *bandwidth* yang dialokasikan oleh *router*.

Berbeda dengan pengaturan lain, setiap perubahan hdfs-site.xml maka *format* ulang Namenode dan menghapus semua direktori temporer (*hadoop.tmp.dir*) harus dilakukan. Dikarenakan HDFS berkaitan dengan blok data, jadi perubahan yang terjadi tidak akan berpengaruh pada data yang telah disimpan. Perintah untuk melakukan *format* ulang Hadoop sebagai berikut.

- *master*

File master berisi tentang *hostname* yang menjadi *master* atau Namenode. Fungsi dari *file* ini adalah agar Datanode mengetahui *master* atau Namenode. *File master* dapat diakses dengan perintah sebagai berikut.

```
hduser@raspberrypi~$sudo nano $HADOOP_HOME/conf/master
```

- *slaves*

File slaves berisi tentang *hostname* dari node-node yang akan dijadikan *slaves* atau Datanode. Fungsi dari *file* ini adalah agar Namenode mengetahui nama-nama Datanodenya. *File slaves* dapat diakses dengan perintah sebagai berikut.

```
hduser@raspberrypi~$nano $HADOOP_HOME/conf/slaves
```

7. Menjalankan Klaster Hadoop

Untuk memudahkan menjalankan klaster Hadoop maka perlu dibuatkan *script* untuk mengaksesnya. Untuk mengaksesnya dapat menggunakan perintah sebagai berikut.

```
hduser@raspberrypi~$sudo nano .bashrc
```

Tambahkan kode berikut yang berisi tentang lokasi folder Hadoop dan lokasi *file* binary Hadoop.

```
export HADOOP_INSTALL=/usr/local/hadoop
export PATH=$PATH:HADOOP_INSTALL/bin
```

Dengan mengedit *bashrc* maka Hadoop siap dijalankan melalui konsol terminal.

Saat pertama kali menjalankan Hadoop maka Namenode perlu di-*format* terlebih dahulu. Alasannya adalah karena Namenode adalah pusat dari *file* sistem HDFS yang menyimpan *metadata* dari data yang disimpan di Datanode. Dengan melakukan proses *format* pada Namenode, maka seluruh informasi yang berhubungan dengan Datanode akan hilang dan Hadoop akan membuatkan *metadata* baru lagi sesuai konfigurasi yang diberikan.Untuk memformat Namenode dapat dilakukan dengan perintah sebagai berikut.

```
hduser@raspberrypi~$ hadoop Namenode -format
```

Pada Gambar 4.7. akan menunjukan proses *format* pada Namenode.

```
hduser@WAHYU-T90:~$ hadoop namenode -format
16/06/11 20:48:54 INFO namenode.NameNode: STARTUP_MSG:
/************************************************************
STARTUP_MSG: Starting NameNode
STARTUP_MSG:   host = WAHYU-T90/192.168.1.7
STARTUP_MSG:   args = [-format]
STARTUP_MSG:   version = 1.2.1
STARTUP_MSG:   build = https://svn.apache.org/repos/asf/hadoop/common/branches/branch-1.2 -r 1503152; compiled by 'mattf' on Mon Jul 22 1
 PDT 2013
STARTUP_MSG:   java = 1.8.0_72
************************************************************/
16/06/11 20:48:54 INFO util.GSet: Computing capacity for map BlocksMap
16/06/11 20:48:54 INFO util.GSet: VM type       = 32-bit
16/06/11 20:48:54 INFO util.GSet: 2.0% max memory = 932184064
16/06/11 20:48:54 INFO util.GSet: capacity      = 2^22 = 4194304 entries
16/06/11 20:48:54 INFO util.GSet: recommended=4194304, actual=4194304
16/06/11 20:48:55 INFO namenode.FSNamesystem: fsOwner=hduser
16/06/11 20:48:55 INFO namenode.FSNamesystem: supergroup=supergroup
16/06/11 20:48:55 INFO namenode.FSNamesystem: isPermissionEnabled=true
16/06/11 20:48:55 INFO namenode.FSNamesystem: dfs.block.invalidate.limit=100
16/06/11 20:48:55 INFO namenode.FSNamesystem: isAccessTokenEnabled=false accessKeyUpdateInterval=0 min(s), accessTokenLifetime=0 min(s)
16/06/11 20:48:55 INFO namenode.FSEditLog: dfs.namenode.edits.toleration.length = 0
16/06/11 20:48:55 INFO namenode.NameNode: Caching file names occuring more than 10 times
16/06/11 20:48:55 INFO common.Storage: Image file /fs/hadoop/tmp/dfs/name/current/fsimage of size 112 bytes saved in 0 seconds.
16/06/11 20:48:56 INFO namenode.FSEditLog: closing edit log: position=4, editlog=/fs/hadoop/tmp/dfs/name/current/edits
16/06/11 20:48:56 INFO namenode.FSEditLog: close success: truncate to 4, editlog=/fs/hadoop/tmp/dfs/name/current/edits
16/06/11 20:48:56 INFO common.Storage: Storage directory /fs/hadoop/tmp/dfs/name has been successfully formatted.
16/06/11 20:48:56 INFO namenode.NameNode: SHUTDOWN_MSG:
/************************************************************
SHUTDOWN_MSG: Shutting down NameNode at WAHYU-T90/192.168.1.7
************************************************************/
```

Gambar 4.7 Proses *format* Namenode

Untuk menjalankan Hadoop dapat menggunakan perintah konsol sebagai berikut.

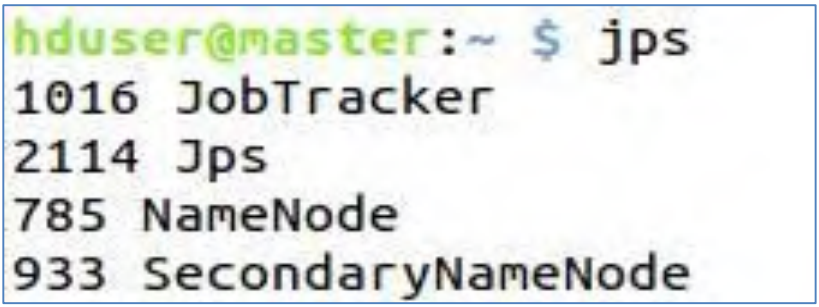
```
hduser@raspberrypi~$ start-all.sh
```

Untuk mematikan Hadoop dapat menggunakan perintah konsol sebagai berikut.

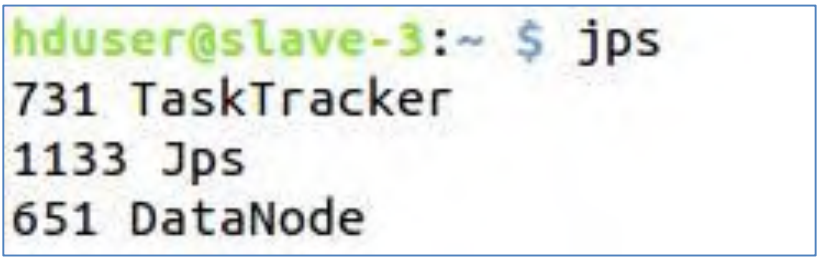
```
hduser@raspberrypi~$ stop-all.sh
```

Untuk menambah Datanode dapat dilakukan dengan mengulang langkah instalasi pada setiap mesin yang akan digunakan untuk menjalankan Hadoop. Dan tambah kan daftar dari Datanode tersebut pada *file $HADOOP_HOME/conf/slaves* sehingga Namenode dapat mengidentifikasikannya sebagai Datanode. Untuk Namenode atau Datanode yang berjalan dapat dicek dengan perintah konsol sebagai berikut.

```
hduser@raspberrypi~$ jps
```

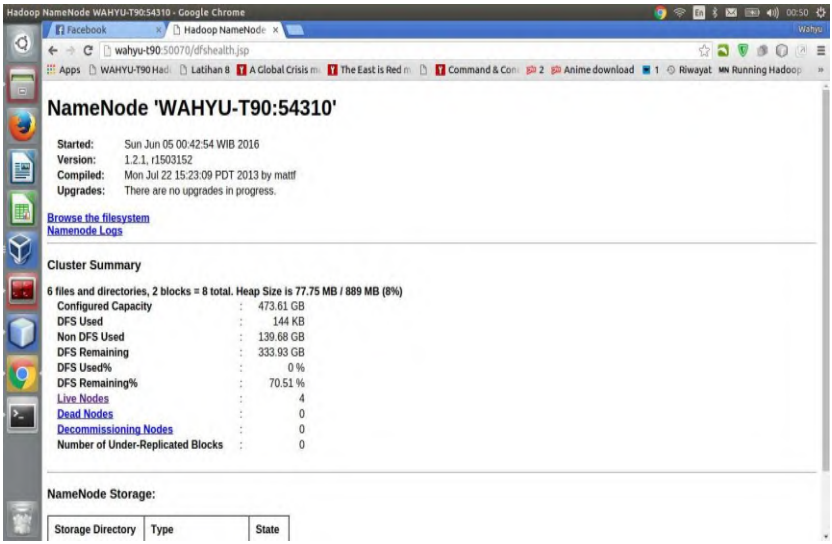


Gambar 4.8 List Proses di Namenode

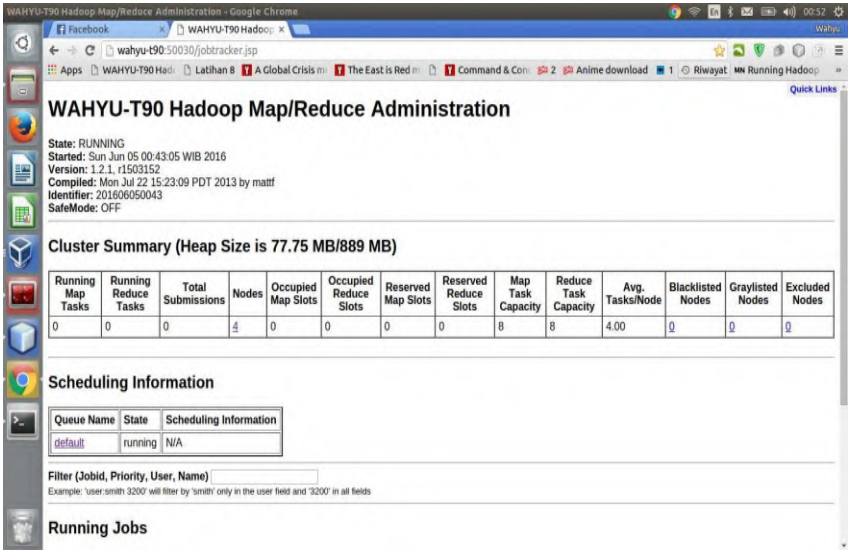


Gambar 4.9 List Proses di Datanode

Untuk *user interface* Hadoop dapat diakses pada port 50070 untuk *user interface* Namenode dan port 50030 untuk *user interface* JobTracker



Gambar 4.10 *User Interface* Namenode



Gambar 4.11 *User Interface* Datanode

4.2. Implementasi Optimasi Blok Data Hadoop

Subbab ini membahas implementasi tahap optimasi blok data Hadoop. Pada Hadoop standar, setiap data yang disimpan ke dalam HDFS (*Hadoop Distributed File System*) memiliki ukuran per blok mencapai 64MB. Diperlukan konfigurasi untuk mengatur ukuran dari blok ukuran Hadoop. Konfigurasi dapat dilakukan pada *file $HADOOP_HOME/conf/hdfs-site.xml* dengan konfigurasi sebagai berikut:

```
<property>
        <name>dfs.block.size</name>
        <value>16777216</value>
</property>
```

Kode Sumber 4.1 Implementasi Optimasi Blok Data

Konfigurasi tersebut digunakan untuk mengubah ukuran standar blok data HDFS. Ukuran dituliskan dalam satuan *byte*.

Pengurangan blok data dilakukan dikarenakan data yang disimpan dan yang akan diproses pada klaster Hadoop tidak terlalu besar. Sehingga semakin besar blok data maka akan memaksimalkan proses MapReduce pada suatu *node*.

Pada implementasi optimasi blok data Hadoop, setiap terjadi perubahan *pada $HADOOP_HOME/conf/hdfs-site.xml* maka HDFS perlu di-*format*. Sebab Namenode selalu mencatat *metadata* setelah melakukan penyimpanan. Peletakan data yang sudah diatur oleh Namenode akan gagal jika data tidak di-*format*. Hal ini dapat berakibat hilang atau tidak terdefinisinya blok data dalam Hadoop.

Ukuran blok data diatur menjadi ukuran 16MB. Penurunan blok data Hadoop dari 64MB hingga 16MB dikarenakan data yang diproses mimiliki ukuran kurang dari 20MB. Penurunan blok data menjadi ukuran 16MB juga mengurangi akses *metadata* pada ukuran blok terlalu besar dan mengurangi ukuran blok data yang perlu dipindahkan antar Datanode jika diperlukan. Percobaan dilakukan dengan mengikuti ukuran *file* yang diuji (20MB, 10MB, 1MB, dan 100KB). Namun, lebih kecil dari 20MB tidak memberi pengaruh pada uji coba sehingga ukuran yang dipakai adalah 16MB karena

biasanya blok data Hadoop dikonfigurasi menyesuaikan deret ukur biner.

4.3. Implementasi Kompresi Data

Sub bab ini membahas implementasi tahap Kompresi data Hadoop sebagai solusi untuk mencegah *bottleneck*. Pada Hadoop standar, Hadoop tidak memberi ketentuan dalam melakukan kompresi data. Hal ini disebabkan kompresi data pada proses MapReduce akan memakan memori yang cukup besar dalam prosesnya. Dan proses kompresi data ini biasanya diimplementasikan pada jaringan yang kurang baik (*bandwith* rendah). Konfigurasi dapat dilakukan pada *file $HADOOP_HOME/conf/mapred-site.xml*. Namun, dapat juga dipanggil dengan melalui program. Implementasi dapat dilihat pada Kode Sumber 4.2.

```
1  This.configuration(compressMapOutputParam,true);
2  This.configuration(compressionCodecParam,compressionC
   odecValue);
```

Kode Sumber 4.2 Implementasi Kompresi Data

Dengan kode ini maka pengkompresian data saat proses MapReduce dapat dilakukan dengan lebih dinamis tanpa melalui perubahan konfigurasi Hadoop terlebih dahulu. Pengkompresan hasil keluaran Map juga dapat dikonfigurasi melalui konfigurasi Hadoop tersediri yaitu, pada *$HADOOP_HOME/conf/mapred-site.xml*. Dengan menambahkan parameter seperti pada Kode Sumber 4.3.

```
<property>
        <name>mapreduce.map.output.compress</name>
        <value>true</value>
</property>
<property>
        <name>mapreduce.map.output.compress.codec</name>
        <value>
                mapreduce.map.output.compress.codec.snappy
        </value>
</property>
```

Kode Sumber 4.3 Implementasi Kompresi Data pada XML

Implementasi menggunakan Snappy dikarenakan Snappy jenis kompresi yang sudah merupakan bawaan Hadoop dan dapat bekerja pada perangkat Raspberry Pi.

4.4. Implementasi *Memory Tunning*

Subbab ini membahas implementasi tahap penyesuaian memori pada Hadoop untuk menyesuaikan memori pada Datanode. Konfigurasi dapat dilakukan pada *file $HADOOP_HOME/conf/mapred-site.xml* namun dapat juga dipanggil dengan melalui program. Implementasi dapat dilihat pada Kode Sumber 4.4.

```
1   This.configuration.set(memoryTunningParam,
    memoryTunningValue);
```

Kode Sumber 4.4 Implementasi *Memory Tunning*

Kode ini digunakan untuk mengatur memori minimum dan maksimum yang akan digunakan pada saat proses MapReduce. Jika diimplementasikan pada *file $HADOOP_HOME/conf/mapred-site*. Maka konfigurasinya seperti pada Kode Sumber 4.5.

```
<property>
       <name>mapred.child.java.opts</name>
       <value>-Xms1024M –Xmx2048</value>
</property>
```

Kode Sumber 4.5 Implementasi *Memory Tunning* pada XML

4.5. Implementasi Optimasi Ukuran *File*

Subbab ini membahas implementasi tahap implementasi optimasi ukuran *file* pada Hadoop. *File* kecil merupakan masalah tersendiri bagi Hadoop dikarenakan Hadoop dispesifikasi untuk mengatasi *file* dalam ukuran besar. Implementasi dapat dilihat pada Kode Sumber 4.6.

```
1   FileUtils.writeStringToFile(File, strFileCreated);
    Fs.copyFromLocalFile(
2   New Path
    (Dest_Url+”final”+Integer.toString(totalFileCreated)+
    ”.txt”);
```

```
    New Path(HDFS_Url+"final/")
    );
```

Kode Sumber 4.6 Implementasi Optimasi Ukuran *File*

Hasil dari Kode Sumber 4.6 digunakan untuk menggabungkan *file* yang akan diproses dan memasukkannya kedalam HDFS yang kemudian *file* hasil penggabungan tersebut digunakan untuk mengeksekusi program MapReduce.

4.6. Implementasi JVM *Reuse*

Subbab ini membahas implementasi tahap implementasi JVM *reuse* pada Hadoop. Penggunaan kembali JVM pada Tasktracker merupakan keuntungan tersendiri ketika Hadoop mengolah data dalam jumlah kecil dan banyak. Pada Hadoop versi pertama JVM *reuse* bukanlah standar dari Hadoop dikarenakan Tasktracker akan memboroskan memori dari Datanode jika *task* yang datang sedikit menuju Datanode tersebut. Namun, dengan mengimplementasikannya ke dalam program MapReduce yang akan dijalankan pada Raspberry Pi maka akan menjadi keuntungan tersendiri. Implementasi dapat dilihat pada Kode Sumber 4.7.

```
1   This.configuration.set(jvmTunningParam,
    jvmTunningValue);
```

Kode Sumber 4.7 Implementasi JVM *Reuse*

Dengan kode ini maka penggunaan JVM dapat dilakukan dengan lebih dinamis. Penggunaan ulang JVM juga dapat diset melalui Hadoop tersediri pada *$HADOOP_HOME/conf/mapred-site.xml*. Dengan menambahkan parameter.

```
<property>
       <name>mapred.job.reuse.jvm.num.task</name>
       <value>-1</value>
</property>
```

Kode Sumber 4.8 Implementasi JVM Reuse pada XML

BAB V
PENGUJIAN DAN EVALUASI

Bab ini membahas pengujian dan evaluasi pada sistem yang dikembangkan. Pengujian yang dilakukan adalah pengujian terhadap kinerja Hadoop pada Raspberry Pi, tingkat kesibukan (*busy level*) Datanode pada Hadoop, dan berjalannya Hadoop pada jaringan yang mengimplementasikan SDN. Hasil evaluasi menjabarkan tentang rangkuman hasil pengujian pada masing-masing pengujian.

5.1. Uji Coba Instalasi Hadoop pada Raspberry Pi

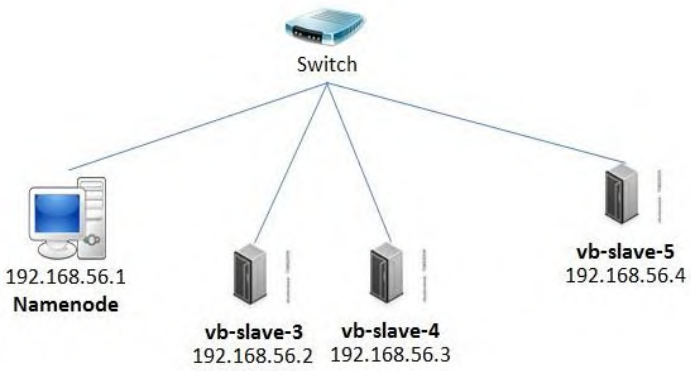
Pada subbab ini akan dibahas uji coba yang telah dilakukan pada saat proses instalasi Hadoop agar dapat berjalan pada Raspberry Pi. Berikut adalah uji coba yang telah dilakukan.

5.1.1. Instalasi Hadoop 2.6.2 pada Virtual Box

Instalasi Hadoop versi 2.6.2 pada Virtual Box bertujuan untuk melihat cara kerja dan instalasi dari Hadoop. Hasil dari uji coba ini kemudian menjadi panduan untuk diimplementasikan kemudian pada perangkat keras Raspberry Pi.
Spesifikasi dari perangkat keras yang dijalankan pada Virtual Box adalah sebagai berikut :

- Sistem Operasi Debian (32-bit)
- 512MB RAM
- AMD A8-6416 APU
- Core CPU 1
- 8GB Harddisk

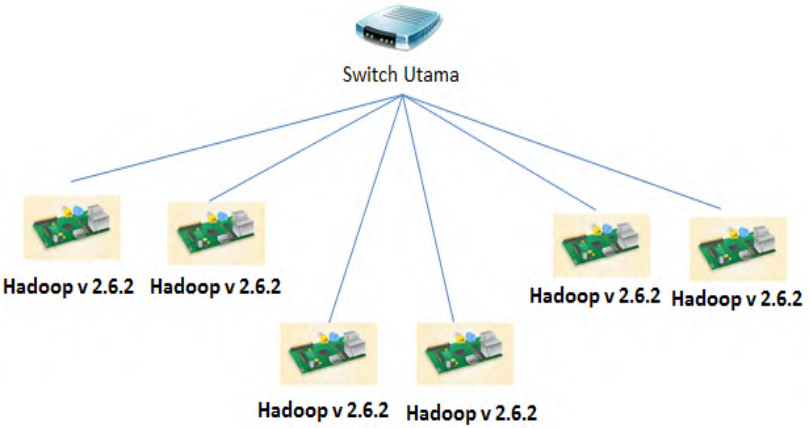
Hasil percobaan menggunakan Virtual Box berhasil dengan arsitektur klaster yang terbentuk terdapat satu Namenode dengan alamat IP 192.168.56.1 dan tiga Datanode dengan alamat IP 192.168.56.2, 192.168.56.3, dan 192.168.56.4. Ilustrasi dapat dilihat seperti pada Gambar 5.1.



Gambar 5.1 Arsitektur Klaster Hadoop pada Virtual Box

5.1.2. Instalasi Hadoop 2.6.2 pada Raspi 1

Pada sub bab ini akan dijelaskan uji coba instalasi Hadoop versi 2.6.2 pada Raspberry Pi 1 Model B sebagai bagian dari rangkaian percobaan. Arsitektur yang akan diimplementasikan terdapat pada Gambar 5.2.



Gambar 5.2 Arsitektur Klaster Hadoop menggunakan Hadoop 2.6.2

Beberapa Kegagalan dalam percobaan ini :

- Namenode tidak dapat melakukan proses menghidupkan Datanode
- port 50070 atau *port user interface* dari Namenode tidak dapat terbentuk.

Kegagalan tidak dapat dihidupkannya Datanode adalah dikarenakan Hadoop yang telah dipasang merupakan hasil dari proses *build* dari komputer dengan prosessor 64-bit. Sehingga Raspberry Pi 1 yang memiliki prosessor 32-bit tidak dapat menjalankannya.

5.1.3. Instalasi Hadoop 2.6.2 pada Raspi 2

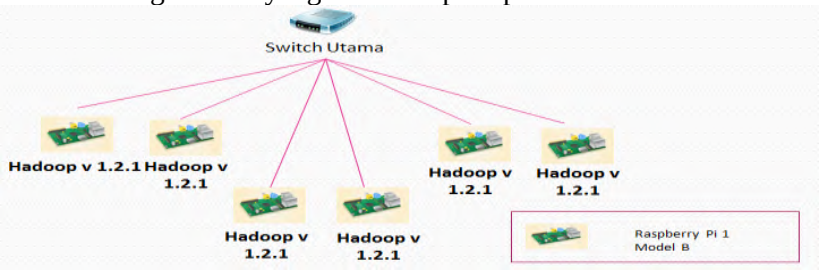
Percobaan ketiga dilakukan dengan menggunakan perangkat keras Raspberry Pi 2 Model B menggunakan Hadoop versi 2.6.2.

Pada percobaan ketiga Hadoop versi 2.6.2 dapat dijalankan pada Raspberry Pi 2. Namun kendala karena jumlah Raspberry Pi 2 yang terbatas.

5.1.4. Instalasi Hadoop 1.2.1 pada Raspi 1

Berdasarkan percobaan ketiga perbedaan antara Raspberry Pi 1 dan 2 adalah pada spesifikasi *prosessor* dan RAM. Pada pengujian instalasi keempat menggunakan Hadoop versi 1.2.1. Hal ini didasari karena versi Hadoop dibagi menjadi 2 kategori besar, yaitu Hadoop versi pertama dan Hadoop versi kedua.

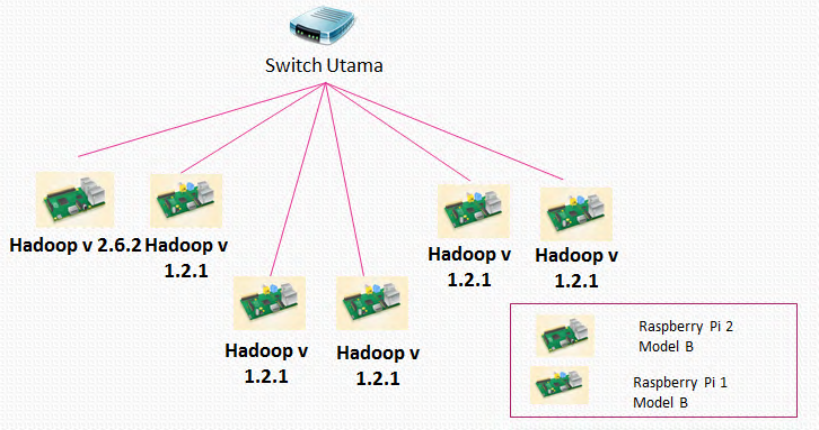
Hasil instalasi Hadoop versi 1.2.1 pada Raspberry Pi 1 berhasil. dengan node yang terbuat seperti pada Gambar 5.3.



Gambar 5.3 Arsitektur Hadoop Percobaan 4

5.1.5. Kombinasi antara Hadoop 2.6.2 dan Hadoop 1.2.1

Berdasarkan percobaan keempat maka diuji kembali penggunaan klaster Hadoop menggunakan Namenode dengan Hadoop versi 2.6.2 dan Datanode menggunakan Hadoop versi 1.2.1. Ilustrasi dapat dilihat seperti pada Gambar 5.4.



Gambar 5.4 Arsitektur jaringan kombinasi Hadoop

Hadoop 2.6.2 memiliki keunggulan dibidang ResourceManager dan pembagian *task* yang baik karena *scheduler* sudah ditangani oleh YARN. Penggunaan Hadoop versi 2.6.2 pada Namenode dimaksudkan agar dapat mengubah tipe pembagian *task* Hadoop. Sehingga pembagian *task* Hadoop dapat lebih dinamis.

Tujuan dari uji coba ini adalah agar master node memakai Hadoop versi 2.6.2 dan *slave node* memakai Hadoop versi 1.2.1.

Kesimpulan yang didapatkan dari percobaan ini adalah Namenode pada Hadoop versi 2.6.2 tidak dapat digabung dengan Datanode Hadoop versi 1.

5.2. Uji Coba Penyesuaian MapReduce

Pada subbab ini akan dijelaskan mengenai skenario uji coba yang telah dilakukan. Beberapa skenario uji coba, diantaranya yaitu:

1) Perbandingan waktu berjalan dan *busy level* program MapReduce setelah dan sebelum penyesuaian pada satu Namenode dan satu Datanode menggunakan *switch* biasa.
2) Perbandingan waktu berjalan dan *busy level* program MapReduce setelah dan sebelum penyesuaian pada satu Namenode dan tiga Datanode menggunakan *switch* biasa.
3) Kesuksesan program MapReduce setelah penyesuaian pada satu Namenode dan satu Datanode menggunakan *switch* dari Raspberry Pi yang mengimplementasikan konsep jaringan SDN.
4) Kesuksesan program MapReduce setelah penyesuaian pada satu Namenode dan satu Datanode menggunakan dua *switch* dari Raspberry Pi yang mengimplementasikan konsep jaringan SDN.

5.2.1. Data Uji Coba

Data yang digunakan untuk uji coba penyesuaian program MapReduce adalah *file* teks dengan ukuran 100KB, 1MB, 10MB, 20MB. Waktu penyelesaian *job* adalah waktu rata-rata dari 5 kali uji coba.

Waktu penyelesaian *job* kemudian dibandingkan antara program MapReduce setelah dan sebelum penyesuaian. Berikut adalah hasil uji coba program MapReduce sebelum penyesuaian pada perangkat Raspberry Pi.

Tabel 5.1 Hasil uji coba sebelum penyesuaian

No	Ukuran *File*	Waktu Proses
1	100KB	7 Menit 58 Detik
2	1MB	37 Menit 20 Detik
3	10MB	GAGAL
4	20MB	GAGAL

Pada Tabel 5.1 terlihat bahwa program MapReduce mengalami kegagalan saat mengolah data dengan ukuran lebih besar dari 10MB. Kegagalan pemrosesan terlihat setelah MapReduce

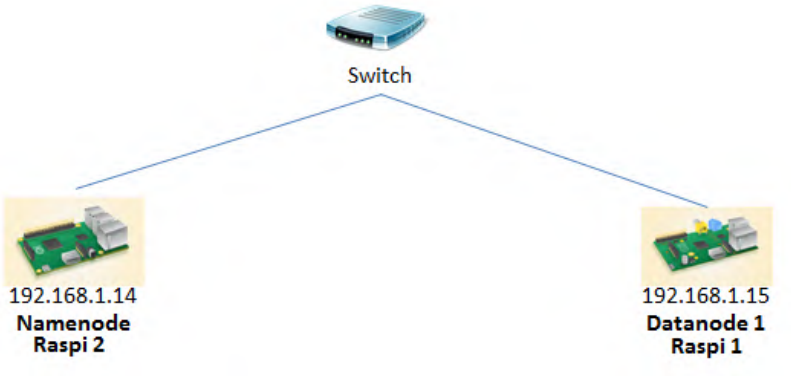
berjalan selama lebih dari 7 jam. Hal ini dikarenakan besar dan banyaknya proses Map yang diberi pada Tasktracker.

5.2.2. Skenario Uji Coba 1

Skenario uji coba 1 adalah Perhitungan waktu berjalannya program MapReduce pada *node* tunggal menggunakan *switch* biasa. Tujuan dari uji coba ini adalah melihat perbandingan waktu penyelesaian *job* dari Datanode tunggal antara program Hadoop yang telah disesuaikan dan belum. Dengan rancangan arsitektur sebagai berikut.

- Namenode
  1) Terdiri dari 1 *node*
  2) Perangkat keras : Raspberry Pi 2 Model B
  3) Alamat IP : 192.168.1.14
- Datanode
  1) Terdiri dari 1 *node*
  2) Perangkat keras : Raspberry Pi 1 Model B
  3) Alamat IP : 192.168.1.15

Secara visual arsitektur akan terlihat seperti Gambar 5.5 dan Gambar 5.6.



Gambar 5.5 Arsitektur Skenario 1

Gambar 5.6 Uji Coba 1

5.2.2.1. Skenario Uji Coba 1.1 (*Split task* 16MB)

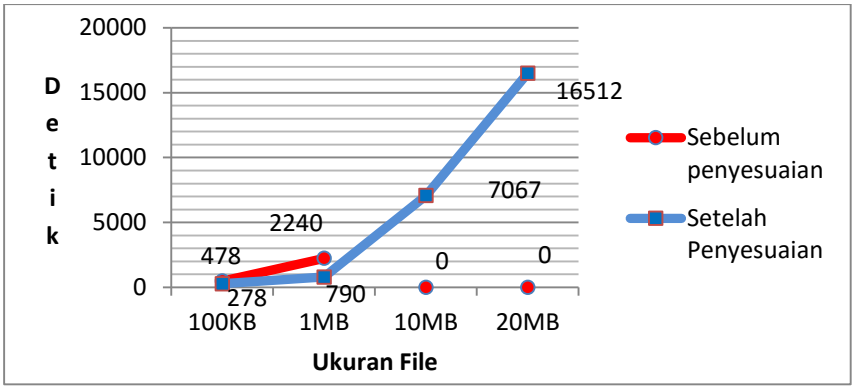
Pada uji coba ini akan digunakan ukuran *file* per *task* 16MB. Berikut Tabel 5.2 dan Tabel 5.3 adalah hasil uji coba:

Tabel 5.2 Hasil Uji Coba Skenario 1.1

No	Ukuran *File*	Waktu Proses
1	100KB	4 Menit 38 Detik
2	1MB	13 Menit 10 Detik
3	10MB	1 Jam 57 Menit 47 Detik
4	20MB	4 jam 35 menit 12 detik

Tabel 5.3 Perbandingan Total *task* Skenario 1.1

No	Ukuran *File*	Total Task	Total Task
1	100KB	10	1
2	1MB	100	1
3	10MB	1000	1
4	20MB	2000	2



Gambar 5.7 Perbandingan waktu selesai Job ukuran blok 16MB pada skenario 1.1

Pada **Gambar 5.7** adalah percobaan skenario 1 menggunakan ukuran *file* per *task* 16MB terlihat bahwa Program MapReduce hasil penyesuaian masih dapat mengolah *file* dengan ukuran 20MB dalam waktu 4 jam 35 menit 12 detik.

5.2.2.2. Skenario Uji Coba 1.2 (*Split task* 1MB)

Pada uji coba ini akan digunakan ukuran *file* per *task* 1MB. Berikut Tabel 5.4 dan Tabel 5.5 adalah hasil uji coba:

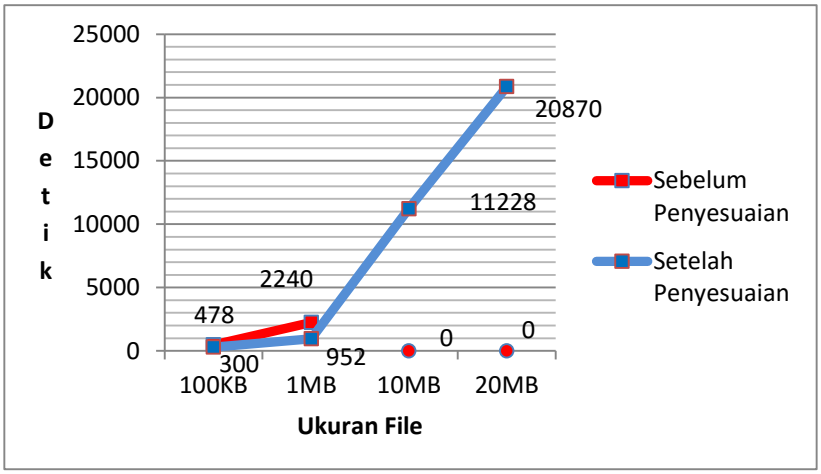
Tabel 5.4 Hasil Uji Coba Skenario 1.2

No	Ukuran *File*	Waktu Proses
1	100KB	5 Menit
2	1MB	15 Menit 52 Detik
3	10MB	3 Jam 7 Menit 8 Detik
4	20MB	5 Jam 47 Menit 50 Detik

Tabel 5.5 Perbandingan Total *task* Skenario 1.2

No	Ukuran *File*	Total Task	Total Task
1	100KB	10	1
2	1MB	100	2
3	10MB	1000	11

4	20MB	2000	21



Gambar 5.8 Perbandingan waktu selesai Job ukuran blok 1MB pada skenario 1.2

Pada Gambar 5.8 adalah percobaan skenario 1 menggunakan ukuran *file* per *task* 1MB terlihat bahwa Program MapReduce hasil penyesuaian masih dapat mengolah *file* dengan ukuran 20MB dalam waktu 5 jam 47 menit 50 detik.

5.2.2.3. Skenario Uji Coba 1.3 (*Split task* 100KB)

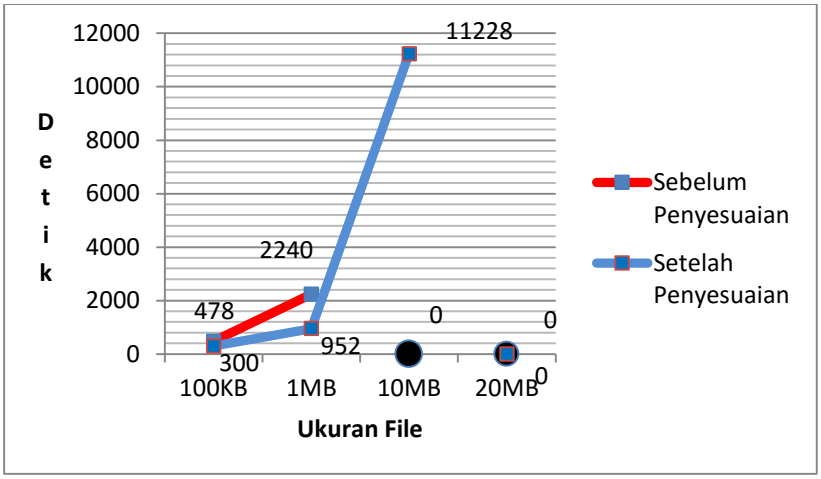
Pada uji coba ini akan digunakan ukuran *file* per *task* 100KB. Berikut adalah hasil uji coba:

Tabel 5.6 Hasil Uji Coba Skenario 1.3

No	Ukuran *File*	Waktu Proses
1	100KB	8 Menit 5 Detik
2	1MB	17 Menit 51 Detik
3	10MB	5 Jam 22 Menit 41 Detik
4	20MB	Gagal

Tabel 5.7 Perbandingan Total *task* Skenario 1.2

No	Ukuran *File*	Total Task	Total Task
1	100KB	10	1
2	1MB	100	3
3	10MB	1000	21
4	20MB	2000	42

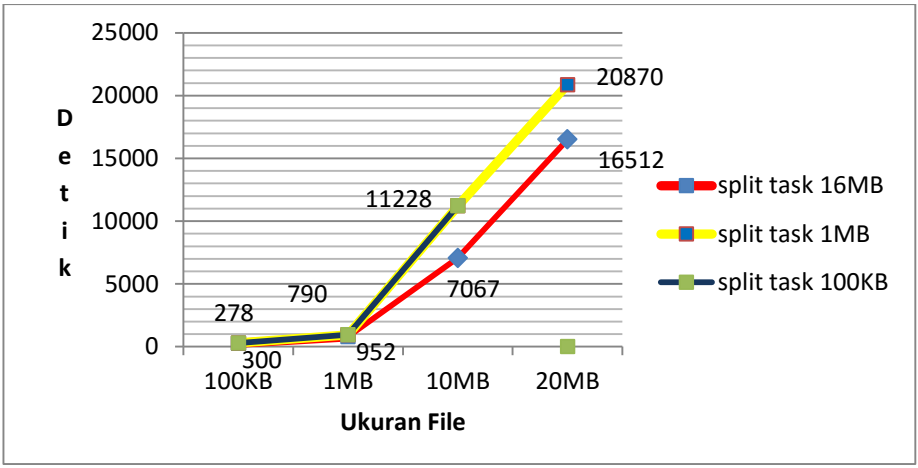


Gambar 5.9 Perbandingan waktu selesai Job ukuran blok 100KB pada skenario 1.3

Pada Gambar 5.9 adalah hasil percobaan skenario 1 menggunakan ukuran *file* per *task* 100KB terlihat bahwa Program MapReduce hasil penyesuaian masih dapat mengolah *file* dengan ukuran 10MB dalam waktu 5 Jam 22 Menit 41 Detik. Namun, mengalami kegagalan saat mengolah data 20MB.

5.2.3. Evaluasi Skenario 1

Berdasarkan uji coba skenario 1 dimana proses MapReduce hanya menggunakan 1 Datanode tunggal didapatkan hasil uji sebagai berikut:



Gambar 5.5.10 Perbandingan hasil *split task* uji coba 1

Dari Gambar 5.5.10 terlihat bahwa dengan menggunakan *split task* ukuran 16MB maka satu Datanode dapat lebih efektif dalam mengerjakan proses MapReduce. Hal ini terbukti dengan waktu proses yang lebih cepat dan kemampuan mengolah data hingga ukuran 20MB.

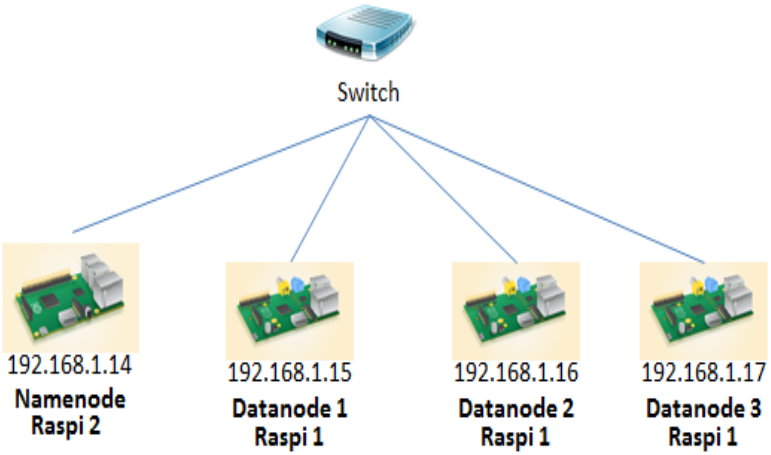
5.2.4. Skenario Uji Coba 2

Skenario uji coba 2 adalah perhitungan waktu berjalannya program MapReduce pada tiga *node* menggunakan *switch* biasa. Tujuan dari uji coba ini adalah melihat perbandingan waktu penyelesaian job dari 3 Datanode antara program Hadoop yang telah disesuaikan dan belum. Dengan rancangan arsitektur sebagai berikut.

- Namenode
  1) Terdiri dari 1 *node*
  2) Perangkat keras : Raspberry Pi 2 Model B
  3) Alamat IP : 192.168.1.14
- Datanode
  1) Terdiri dari 3 *node*
  2) Perangkat keras : Raspberry Pi 1 Model B
  3) Alamat IP Datanode 1: 192.168.1.15

Alamat IP Datanode 2: 192.168.1.16
Alamat IP Datanode 3: 192.168.1.17

Secara visual arsitektur akan terlihat seperti Gambar 5.11 dan Gambar 5.12.



Gambar 5.11 Arsitektur Skenario 2

Gambar 5.12 Uji Coba 2

5.2.4.1. Skenario Uji Coba 2.1 (*Split task* 16MB)

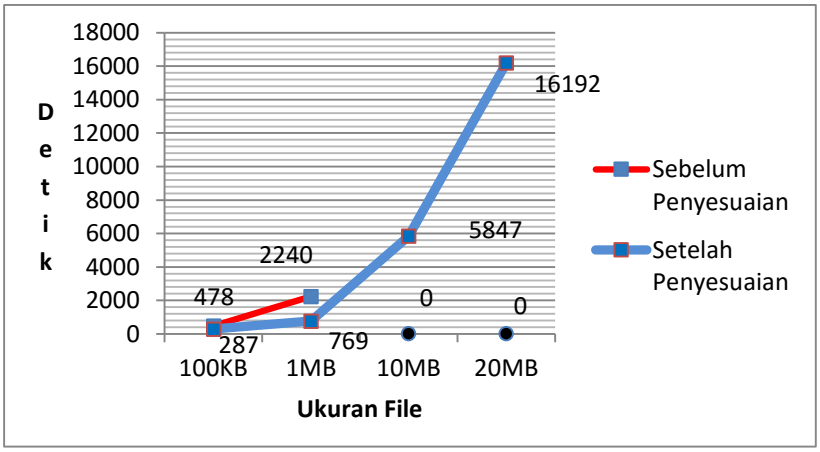
Pada uji coba ini akan digunakan ukuran *file* per *task* 16MB. Berikut Tabel 5.8 dan 5.9 adalah hasil uji coba:

Tabel 5.8 Hasil Uji Coba Skenario 2.1

No	Ukuran *File*	Waktu Proses
1	100KB	4 Menit 47 Detik
2	1MB	12 Menit 49 Detik
3	10MB	1 Jam 37 Menit 27 Detik
4	20MB	4 jam 29 menit 52 detik

Tabel 5.9 Perbandingan Total *task* Skenario 1.2

No	Ukuran *File*	Total Task	Total Task
1	100KB	10	1
2	1MB	100	1
3	10MB	1000	1
4	20MB	2000	2



Gambar 5.13 Perbandingan waktu selesai Job ukuran blok 16MB pada skenario 2.1

Pada Gambar 5.13 percobaan skenario 2 menggunakan ukuran *file* per *task* 16MB terlihat bahwa Program MapReduce hasil penyesuaian masih dapat mengolah *file* dengan ukuran 20MB dalam waktu 4 jam 29 menit 52 detik.

5.2.4.2. Skenario Uji Coba 2.2 (*Split task* 1MB)

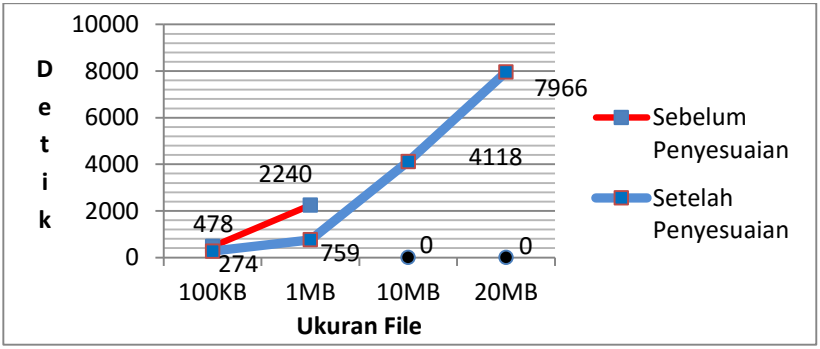
Pada uji coba ini akan digunakan ukuran *file* per *task* 1MB. Berikut Tabel 5.10 dan 5.11 adalah hasil uji coba:

Tabel 5.10 Hasil Uji Coba Skenario 2.2

No	Ukuran *File*	Waktu Proses
1	100KB	4 Menit 34 Detik
2	1MB	12 Menit 39 Detik
3	10MB	1 Jam 8 Menit 38 Detik
4	20MB	2 Jam 12 Menit 46 Detik

Tabel 5.11 Hasil Uji Coba Skenario 2.2

No	Ukuran *File*	Total Task	Total Task
1	100KB	10	1
2	1MB	100	2
3	10MB	1000	11
4	20MB	2000	21



Gambar 5.14 Perbandingan waktu selesai Job ukuran blok 1MB pada skenario 2.2

Pada Gambar 5.14 percobaan skenario 2 menggunakan ukuran *file* per *task* 1MB terlihat bahwa Program MapReduce hasil penyesuaian masih dapat mengolah *file* dengan ukuran 20MB dalam waktu 2 jam 12 menit 46 detik.

5.2.4.3. Skenario Uji Coba 2.3 (*Split task* 100KB)

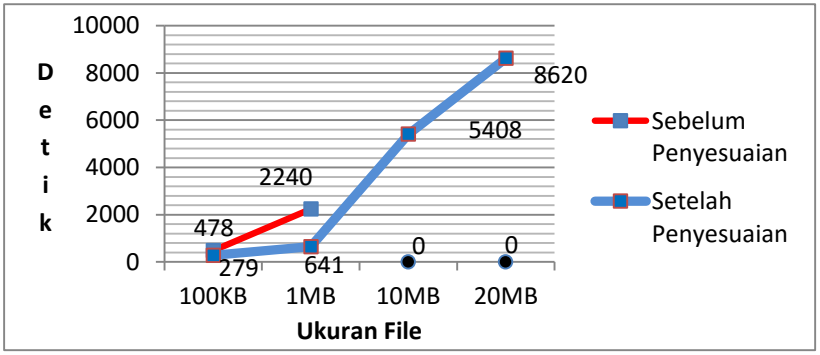
Pada uji coba ini akan digunakan ukuran *file* per *task* 16MB. Berikut adalah hasil uji coba:

Tabel 5.12 Hasil Uji Coba Skenario 2.3

No	Ukuran *File*	Waktu Proses
1	100KB	4 Menit 39 Detik
2	1MB	10 Menit 41 Detik
3	10MB	1 Jam 30 Menit 8 Detik
4	20MB	2 Jam 23 Menit 40 Detik

Tabel 5.13 Hasil Uji Coba Skenario 2.3

No	Ukuran *File*	Total Task	Total Task
1	100KB	10	1
2	1MB	100	3
3	10MB	1000	21
4	20MB	2000	42

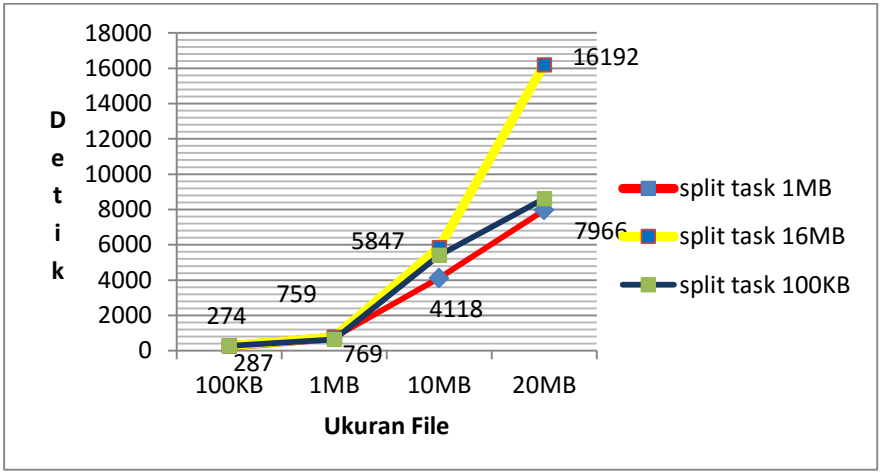


Gambar 5.15 Perbandingan waktu selesai Job ukuran blok 100KB Skenario 2.3

Pada percobaan skenario 2 menggunakan ukuran *file* per *task* 100KB terlihat bahwa Program MapReduce hasil penyesuaian masih dapat mengolah *file* dengan ukuran 20MB dalam waktu 2 Jam 23 Menit 40 Detik.

5.2.5. Evaluasi Skenario 2

Berdasarkan uji coba skenario 2 dimana proses MapReduce hanya menggunakan tiga Datanode tunggal didapatkan hasil uji sebagai berikut:



Gambar 5.16 Perbandingan hasil *split task* uji coba 2

Dari Gambar 5.14 terlihat bahwa dengan menggunakan *split task* ukuran 1MB maka tiga Datanode dapat lebih efektif dalam mengerjakan proses MapReduce. Hal ini terbukti dengan waktu proses yang lebih cepat dan kemampuan mengolah data hingga ukuran 20MB.

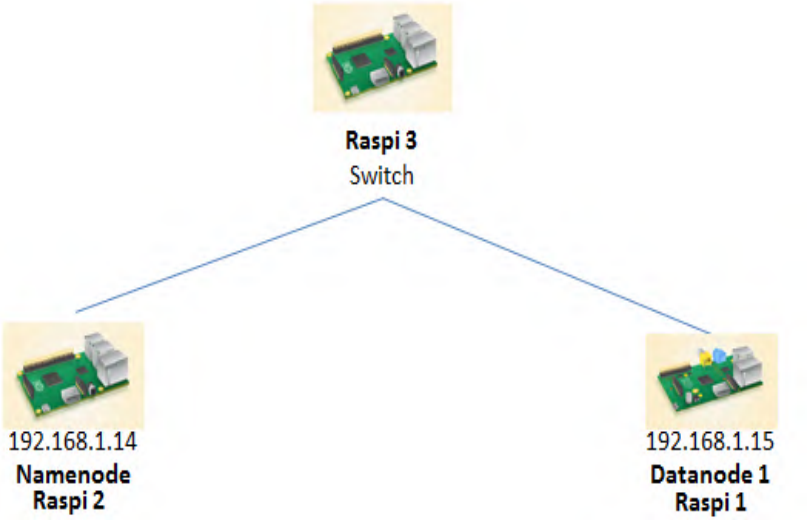
5.2.6. Skenario Uji Coba 3

Skenario uji coba 3 adalah pengujian berjalannya program MapReduce pada Datanode tunggal menggunakan *switch* dari Raspberry Pi yang mengimplementasikan konsep SDN (*Software*

Defined Networking). Tujuan dari uji coba ini adalah mengamati apakah *switch* mampu melakukan manajemen *transfer rate* saat program MapReduce dijalankan. Rancangan arsitektur uji coba sebagai berikut.

- Namenode
  1) Terdiri dari 1 *node*
  2) Perangkat keras : Raspberry Pi 2 Model B
  3) Alamat IP : 192.168.1.14
- Datanode
  1) Terdiri dari 1 *node*
  2) Perangkat keras : Raspberry Pi 1 Model B
  3) Alamat IP : 192.168.1.15

Secara visual arsitektur akan terlihat seperti Gambar 5.17 dan Gambar 5.18.



Gambar 5.17 Arsitektur Skenario 3

Gambar 5.18 Uji Coba 3

5.2.6.1. Skenario Uji Coba 3.2 (*Split task* 1MB)

Pada uji coba ini akan digunakan ukuran *file* per *task* 1MB dengan *queue setting* 500KB/s. Berikut Tabel 5.14 adalah hasil uji coba:

Tabel 5.14 Hasil Uji Coba Skenario 3.2

No	Ukuran *File*	Status Uji
1	100KB	Berhasil
2	1MB	Berhasil
3	10MB	Gagal
4	20MB	Gagal

Pada percobaan skenario 3 menggunakan ukuran *file* per *task* 1MB dengan *file* yang berhasil diproses adalah *file* dengan ukuran kurang dari 1MB.

5.2.6.2. Skenario Uji Coba 3.3 (*Split task* 100KB)

Pada uji coba ini akan digunakan ukuran *file* per *task* 100KB dengan *queue setting* 500KB/s. Berikut Tabel 5.15 adalah hasil uji coba:

Tabel 5.15 Hasil Uji Coba Skenario 3.3

No	Ukuran *File*	Status Uji
1	100KB	BERHASIL

2	1MB	BERHASIL
3	10MB	GAGAL
4	20MB	GAGAL

Pada percobaan skenario 3 menggunakan ukuran *file* per *task* 1MB dengan *file* yang berhasil diproses adalah *file* dengan ukuran kurang dari 1MB.

5.2.7. Evaluasi Skenario 3

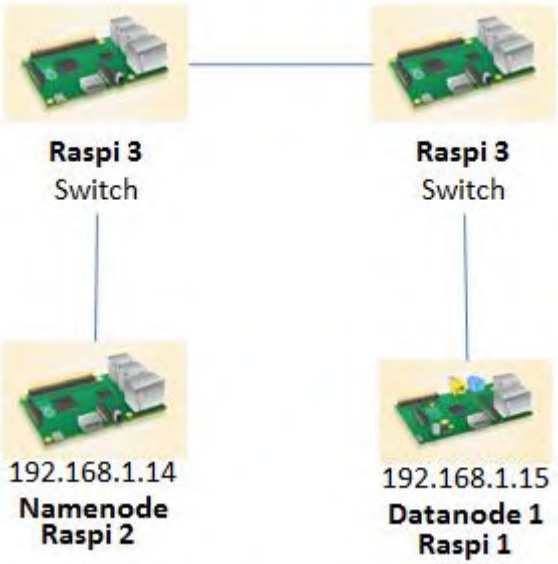
Berdasarkan uji coba skenario 3 dimana proses MapReduce hanya menggunakan Datanode tunggal dan terhubung menggunakan *switch* SDN didapatkan hasil bahwa ukuran *file* yang dapat diproses masih dibawah 1MB dengan ukuran *split task* 1MB. Hal ini dikarenakan *switch* SDN mengalami *kernel panic* saat memroses data lebih dari 1MB.

5.2.8. Skenario Uji Coba 4

Skenario uji coba 4 adalah pengujian berjalannya program MapReduce pada Datanode tunggal menggunakan *switch* dari Raspberry Pi yang mengimplementasikan konsep SDN (*Software Defined Networking*). Tujuan dari uji coba ini adalah mengamati apakah *switch* mampu melakukan manajemen *transfer rate* pada Hadoop dengan beda rak saat program MapReduce dijalankan. Rancangan arsitektur uji coba sebagai berikut:

- Namenode
  1) Terdiri dari 1 *node*
  2) Perangkat keras : Raspberry Pi 2 Model B
  3) Alamat IP : 172.16.10.10
- Datanode
  1) Terdiri dari 1 *node*
  2) Perangkat keras : Raspberry Pi 1 Model B
  3) Alamat IP : 172.16.20.10

Secara visual arsitektur akan terlihat seperti Gambar 5.19 dan Gambar 5.20.



Gambar 5.19 Arsitektur Skenario 4

Gambar 5.20 Uji Coba 4

5.2.8.1. Skenario Uji Coba 4.1 (*Split task* 1MB)

Pada uji coba ini akan digunakan ukuran *file* per *task* 1MB dengan *queue setting* 500KB/s. Berikut Table 5.16 adalah hasil uji coba:

Tabel 5.16 Hasil Uji Coba Skenario 4.2

No	Ukuran *File*	Status Uji
1	100KB	BERHASIL
2	1MB	BERHASIL
3	10MB	GAGAL
4	20MB	GAGAL

Pada percobaan skenario 4 menggunakan ukuran *file* per *task* 1MB dengan *file* yang berhasil diproses adalah *file* dengan ukuran kurang dari 1MB.

5.2.8.2. Skenario Uji Coba 4.1 (*Split task* 100KB)

Pada uji coba ini akan digunakan ukuran *file* per *task* 100KB dengan *queue setting* 500KB/s. Berikut Tabel 5.17 adalah hasil uji coba:

Tabel 5.17 Hasil Uji Coba Skenario 4.3

No	Ukuran *File*	Status Uji
1	100KB	BERHASIL
2	1MB	BERHASIL
3	10MB	GAGAL
4	20MB	GAGAL

Pada percobaan skenario 4 menggunakan ukuran *file* per *task* 100KB dengan *file* yang berhasil diproses adalah *file* dengan ukuran kurang dari 1MB.

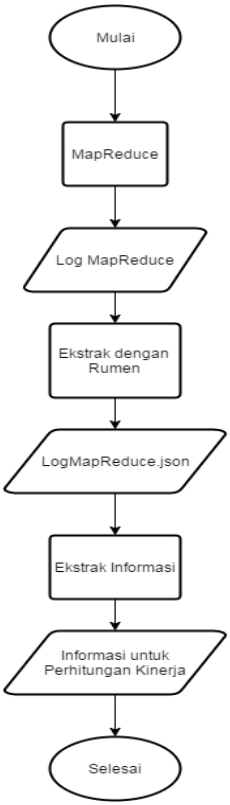
5.2.9. Evaluasi Skenario 4

Berdasarkan uji coba skenario 4 dimana proses MapReduce hanya menggunakan Datanode tunggal dan terhubung menggunakan 2 *switch* SDN didapatkan hasil bahwa ukuran *file* yang dapat diproses masih dibawah 1MB dengan ukuran *split task* 1MB. Hal ini

dikarenakan ukuran *transfer rate* antar router hanya sebesar 3Mbits/s sedangkan untuk menjalankan MapReduce dibutuhkan setidaknya *transfer rate* sebesar 100Mbits/s [9].

5.3. *Perhitungan Kinerja Setiap Node*

Perhitungan kinerja setiap *node* terdiri dari perhitungan kemampuan pemrosesan (*processing capability*) dari setiap Datanode, perhitungan kemampuan pemrosesan dari setiap rak, Perhitungan *busy level* pada setiap Datanode, dan perhitungan *busy level* pada setiap rak. Diagram alur untuk perhitungan kinerja setiap *node* seperti pada Gambar 5.1.



Gambar 5.21 Diagram Alur Perhitungan Kinerja Setiap Datanode

Perhitungan kinerja setiap Datanode didapatkan dari *log* yang dibuat oleh Hadoop setiap kali menyelesaikan program MapReduce. *Log* MapReduce ini kemudian menjadi masukan bagi Rumen. Rumen digunakan untuk mengekstrak *log* Hadoop kedalam bentuk Json yang lebih mudah untuk dibaca. Setelah *log* Hadoop berubah menjadi *file* dalam format Json langkah selanjutnya adalah melakukan proses ekstraksi untuk mendapatkan informasi-informasi yang diinginkan. Informasi-informasi yang diinginkan tersebut adalah id, waktu mulai proses, waktu selesai proses, dan tempat pemrosesan *task* tersebut. Setelah mendapatkan Informasi tersebut maka dilakukan perhitungan kemampuan pemrosesan (*prosessing capability*) dari setiap Datanode, perhitungan kemampuan pemrosesan dari setiap rak, Perhitungan *busy level* pada setiap Datanode, dan perhitungan *busy level* pada setiap rak.

Kode Sumber 5.1 adalah untuk menbaca *file* Json hasil ekstraksi dari Rumen:

```
1   String line = "";
2               br = new BufferedReader(new
3   FileReader("E:\\1MB_NONTUNNING.json"));
4               while((line=br.readLine())!=null){
5                   jsonData += line+"\n";
```

Kode Sumber 5.1 Ekstraksi Json hasil Rumen

Kode Sumber 5.2 adalah untuk memilah informasi-informasi penting untuk perhitungan kinerja setiap *node*.

```
1   if(jsonArray.get(i)instanceof JSONObject){
2   JSONObject jsObj = jsonArray.getJSONObject(i);
3   DateTimeFormatter fmt = DateTimeFormat.forPattern("d
    MMMM, yyyy kk:mm:ss:SSS");
4   DateTime startDate= new
    DateTime(Long.valueOf(jsObj.getString("startTime")));

5   DateTime finishDate = new
    DateTime(Long.valueOf(jsObj.getString("finishTime")));

6   Seconds timeToProcess =
    Seconds.secondsBetween(startDate, finishDate);
    JSONArray attemps = jsObj.getJSONArray("attempts");
```

```
7   JSONObject attemptobj = attemps.getJSONObject(0);

8   String datanode = attemptobj.getString("hostName");
9   data = new TimeTask(datanode, timeToProcess);
10  bw.write(data.toString());
```

Kode Sumber 5.2 Membaca informasi untuk perhitungan Kinerja

Kode Sumber 5.3 adalah untuk menyimpan informasi-informasi hasil ke dalam *file* agar mudah diolah untuk perhitungan kinerja setiap *node*.

```
1   File dumpFile = new File("E:\\datanode.txt");
2   dumpFile.createNewFile();
```

Kode Sumber 5.3 Menyimpan file hasil ekstraksi

5.3.1. Perhitungan Kemampuan Pemrosesan (*processing capability*) pada Datanode

Setiap Datanode memiliki *log* dalam mencatat waktu mulai dan waktu selesai sebuah *task*. Waktu rata-rata pemrosesan data dapat didapatkan berdasarkan seluruh *task* yang ada di dalam *log* [10]. Untuk perhitungan ini kemampuan pemrosesan dapat diukur dengan rumus:

$$P = \frac{\sum_{i=1}^{n} t_i}{n}$$

Dimana P adalah kemampuan pemrosesan, n adalah jumlah *task historical* yang berjalan pada Datanode. t_i adalah waktu yang digunakan untuk menyelesaikan task ke-i. Jika hanya terdapat sedikit *task* yang berjalan (jumlah **n** sedikit) dan memakan waktu yang lama (nilai **t** tinggi) maka kemampuan kerjanya yang dipakai tinggi.

Berdasarkan hasil uji coba antara Hadoop sebelum dan setelah penyesuaian. Dengan menjalankan program MapReduce sebanyak 4 kali menggunakan *file* dengan ukuran 1MB dengan data pada Tabel A.8, Tabel A.9, Tabel A.10, dan Tabel A.11. Didapatkan hasil pada Tabel 5.18.

Tabel 5.18 *processing capability* sebelum penyesuaian

No	Nama Datanode	n	$\sum_{i=1}^{n} t_i$	P
1	Slave-1	156	20286	130
2	Slave-2	182	20366	112
3	Slave-3	33	4102	124

Tabel 5.19 menggunakan data yang sama dan diuji sebanyak 4 kali dengan data pada Tabel A.12, Tabel A.13, Tabel A.14, dan Tabel A.15. Didapatkan hasil pada Tabel 5.19

Tabel 5.19 *processing capability* setelah penyesuaian

No	Nama Datanode	n	$\sum_{i=1}^{n} t_i$	P
1	Slave-1	3	615	205
2	Slave-2	2	134	67
3	Slave-3	3	1473	491

Keterangan:

P: *Processing capability*

n: Jumlah *task* pada 4 kali percobaan

$\sum_{i=1}^{n} t_i$: Waktu menyelesaikan *task*

Berdasarkan hasil uji coba secara historis terlihat bahwa jumlah *task* yang dihasilkan sebelum penyesuaian lebih banyak dari setelah penyesuaian. Hal ini mengakibatkan proses waktu penyelesaian proses sebelum penyesuaian lebih banyak dari seharusnya. Sedangkan jika dilihat dari waktu rata-rata setelah penyesuaian, Datanode pada Slave-2 dan Slave-1 memiliki ukuran tingkat *processing capability* yang sangat rendah dibandingkan dengan ukuran Slave-3. Sehingga, pada Hadoop setelah penyesuaian diperlukan sebuah pembagian kinerja Hadoop untuk mengalokasikan *task* pada Datanode untuk menyeimbangkan kinerjanya.

5.3.2. Perhitungan Tingkat Kesibukan (*Busy Level*) pada Setiap Datanode

Meskipun setiap Datanode menggunakan perangkat keras yang sama, mereka akan menyediakan performa yang berbeda. Maka dari itu informasi untuk mengetahui tingkat kesibukan dapat diestimasikan berdasarkan *log* dari Hadoop. Rumus untuk mencari tingkat kesibukan [10] adalah sebagai berikut:

$$B = \frac{\sum_{i=1}^{k} t_i}{k}$$

Dimana B adalah level kesibukan dari setiap Datanode, k adalah jumlah *task* yang berjalan pada Datanode pada *job* yang baru selesai. Berikut Tabel 5.20 merupakan perhitungan *busy level* berdasarkan *job* 1 pada Tabel A.8.

Tabel 5.20 *busy level* sebelum penyesuaian

No	Nama Datanode	k	$\sum_{i=1}^{k} t_i$	B
1	Slave-1	7	1065	152
2	Slave-2	10	1351	135

3	Slave-3	9	1097	121

Berikut Tabel 5.21 merupakan perhitungan *busy level* berdasarkan *job* 1 pada Tabel A.12.

Tabel 5.21 *busy level* setelah penyesuaian

No	**Nama Datanode**	k	$\sum_{i=1}^{k} t_i$	B
1	Slave-1	1	72	72
2	Slave-2	0	0	NaN
3	Slave-3	1	505	505

Keterangan:

B: *Busy Level*

k: Jumlah *task* pada 1 kali program MapReduce berjalan

$\sum_{i=1}^{k} t_i$: Waktu menyelesaikan *task*

Berdasarkan hasil uji coba secara per satuan *job* (1 *job* = 1 program MapReduce berjalan) terlihat bahwa jumlah *task* yang dihasilkan sebelum penyesuaian lebih banyak dari setelah penyesuaian. Mereduksi jumlah *task* tersebut membuat waktu proses dari MapReduce dapat juga lebih efisien. Sedangkan jika dilihat dari waktu rata-rata setelah penyesuaian, Datanode pada Slave-2 dan Slave-1 memiliki ukuran *busy level* yang sangat rendah dibandingkan dengan ukuran Slave-3. Sehingga, pada Hadoop setelah penyesuaian diperlukan sebuah pembagian kinerja Hadoop untuk mengalokasikan *task* pada Datanode yang tidak sibuk. Kosongnya *task* pada Slave-2 dikarenakan Hadoop tidak memecah secara merata untuk dapat dikerjakan pada masing-masing Datanode melainkan memecah proses *task* berdasarkan ukuran *file* yang dikerjakan. Dikarenakan

ukuran *file* untuk uji coba adalah 1MB dan ukuran split *task*-nya adalah 1MB, maka maksimum *task* yang dapat dihasilkan adalah dua sehingga pasti ada satu Datanode kosong yaitu Slave-2.

LAMPIRAN

Tabel A.1 Hasil Uji Coba Penyesuaian MapReduce ukuran *File* 100KB

Split Task	Uji Ke-	Waktu Proses	Rata-rata
16MB	1	4 Menit 38 Detik	**4 Menit 38 Detik**
	2	4 Menit 30 Detik	
	3	4 Menit 32 Detik	
	4	4 Menit 20 Detik	
	5	4 Menit 70 Detik	
1MB	1	7 Menit 56 Detik	**7 Menit 58 Detik**
	2	8 Menit	
	3	7 Menit 58 Detik	
	4	7 Menit 54 Detik	
	5	8 Menit 2 detik	
100KB	1	8 Menit 13 Detik	**8 Menit 5 Detik**
	2	8 Menit 7 Detik	
	3	7 Menit 56 Detik	
	4	7 Menit 55 Detik	
	5	7 Menit 59 Detik	
16MB (3 Node)	1	4 Menit 45 Detik	**4 Menit 47 Detik**
	2	4 Menit 49 Detik	
	3	4 Menit 47 Detik	
	4	4 Menit 54 Detik	
	5	4 Menit 40 Detik	
1MB (3 Node)	1	4 Menit 35 Detik	**4 Menit 34 Detik**
	2	4 Menit 50 Detik	
	3	4 Menit 21 Detik	
	4	4 Menit 23 Detik	
	5	4 Menit 39 Detik	

100KB (3 Node)	1	4 Menit 42 Detik	**4 Menit 39 Detik**
	2	4 Menit 39 Detik	
	3	4 Menit 39 Detik	
	4	4 Menit 38 Detik	
	5	4 Menit 38 Detik	

Tabel A.2 Hasil Uji Coba Penyesuaian MapReduce Ukuran *File* 1MB

Split Task	Uji Ke-	Waktu Proses	Rata-rata
16MB	1	13 Menit 10 Detik	**13 Menit 10 Detik**
	2	13 Menit 20 Detik	
	3	12 Menit 52 Detik	
	4	13 Menit 5 Detik	
	5	13 Menit 25 Detik	
1MB	1	15 Menit 48 Detik	**15 Menit 52 Detik**
	2	15 Menit 54 Detik	
	3	15 Menit 50 Detik	
	4	15 Menit 58 Detik	
	5	15 Menit 52 Detik	
100KB	1	18 Menit 25 Detik	**17 Menit 51 Detik**
	2	17 Menit 32 Detik	
	3	17 Menit 44 Detik	
	4	18 Menit 23 Detik	
	5	17 Menit 12 Detik	
16MB (3 Node)	1	12 Menit 52 Detik	**12 Menit 49 Detik**
	2	12 Menit 46 Detik	
	3	12 Menit 36 Detik	
	4	12 Menit 49 Detik	
	5	12 Menit 51 Detik	

1MB (3 Node)	1	12 Menit 45 Detik	**12 Menit 39 Detik**
	2	12 Menit 48 Detik	
	3	12 Menit 34 Detik	
	4	12 Menit 30 Detik	
	5	12 Menit 31 Detik	
100KB (3 Node)	1	10 Menit 45 Detik	**10 Menit 41 Detik**
	2	10 Menit 34 Detik	
	3	10 Menit 47 Detik	
	4	10 Menit 46 Detik	
	5	10 Menit 41 Detik	

Tabel A.3 Hasil Uji Coba Penyesuaian MapReduce Ukuran *File* 10MB

Split Task	Uji Ke-	Waktu Proses	Rata-rata
16MB	1	1 Jam 54 Menit 7 Detik	**1 Jam 57 Menit 47 Detik**
	2	1 Jam 57 Menit 47 Detik	
	3	1 Jam 55 Menit 4 Detik	
	4	2 Jam 4 Menit 58 Detik	
	5	1 Jam 57 Menit 44 Detik	
1MB	1	3 Jam 4 Menit 50 Detik	**3 Jam 7 Menit 8 Detik**
	2	2 Jam 59 Menit 56 Detik	

	3	3 Jam 9 Menit 30 Detik	
	4	2 Jam 56 Menit 45 Detik	
	5	2 Jam 59 Menit 46 Detik	
100KB	1	5 Jam 21 Menit 45 Detik	**5 Jam 22 Menit 41 Detik**
	2	5 Jam 22 Menit 56 Detik	
	3	5 Jam 23 Menit 1 Detik	
	4	5 Jam 23 Menit 21 Detik	
	5	5 Jam 21 Menit 58 Detik	
16MB (3 Node)	1	1 Jam 28 Menit 25 Detik	**1 Jam 37 Menit 27 Detik**
	2	1 Jam 35 Menit 32 Detik	
	3	1 Jam 41 Menit 33 Detik	
	4	1 Jam 45 Menit 42 Detik	
	5	1 Jam 35 Menit 35 Detik	
1MB (3 Node)	1	1 Jam 7 Menit 58 Detik	**1 Jam 8 Menit 38 Detik**
	2	1 Jam 8 Menit 40 Detik	
	3	1 Jam 7 Menit 58 Detik	
	4	1 Jam 7 Menit 59 Detik	

	5	1 Jam 9 Menit 2 Detik	
100KB (3 Node)	1	1 Jam 28 Menit 38 Detik	**1 Jam 30 Menit 8 Detik**
	2	1 Jam 30 Menit 48 Detik	
	3	1 Jam 32 Menit 36 Detik	
	4	1 Jam 30 Menit 1 Detik	
	5	1 Jam 30 Menit 5 Detik	

Tabel A.4 Hasil Uji Coba Penyesuaian MapReduce ukuran *File* 20MB

Split Task	Uji Ke-	Waktu Proses	Rata-rata
16MB	1	4 jam 30 menit 20 detik	**4 jam 35 menit 12 detik**
	2	4 jam 31 menit 13 detik	
	3	4 jam 33 menit 32 detik	
	4	4 jam 37 menit 58 detik	
	5	4 jam 34 menit 45 detik	
1MB	1	5 Jam 44 Menit 5 Detik	**5 Jam 47 Menit 50 Detik**
	2	5 Jam 40 Menit 56 Detik	
	3	5 Jam 49 Menit 33 Detik	
	4	5 Jam 51 Menit 1	

		Detik	
	5	5 Jam 51 Menit 44 Detik	
100KB	1	Gagal	**Gagal**
	2	Gagal	
	3	Gagal	
	4	Gagal	
	5	Gagal	
16MB (3 Node)	1	4 jam 29 menit 40 detik	**4 jam 29 menit 52 detik**
	2	4 jam 35 menit 21 detik	
	3	4 jam 29 menit 38 detik	
	4	4 jam 25 menit 47 detik	
	5	4 jam 26 menit 25 detik	
1MB (3 Node)	1	2 Jam 10 Menit 58 Detik	**2 Jam 12 Menit 46 Detik**
	2	2 Jam 10 Menit 30 Detik	
	3	2 Jam 14 Menit 13 Detik	
	4	2 Jam 14 Menit 16 Detik	
	5	2 Jam 12 Menit 40 Detik	
100KB (3 Node)	1	2 Jam 21 Menit 44 Detik	**2 Jam 23 Menit 40 Detik**
	2	2 Jam 23 Menit 4 Detik	
	3	2 Jam 23 Menit 40 Detik	

	4	2 Jam 24 Menit 45 Detik	
	5	2 Jam 23 Menit 24 Detik	

Tabel A.5 Hasil Uji Coba Penyesuaian pada jaringan SDN ukuran *File* 1MB menggunakan 1 *switch*

Split Task	Uji Ke-	Waktu Proses
1MB	1	Gagal
	2	Berhasil
	3	Berhasil
	4	Gagal
	5	Berhasil
100KB	1	Berhasil
	2	Berhasil
	3	Berhasil
	4	Berhasil
	5	Gagal

Tabel A.6 Hasil Uji Coba Penyesuaian pada jaringan SDN ukuran *File* 100KB menggunakan 1 *switch*

Split Task	Uji Ke-	Waktu Proses
1MB	1	Berhasil
	2	Berhasil
	3	Berhasil
	4	Berhasil
	5	Gagal
100KB	1	Berhasil
	2	Berhasil
	3	Berhasil
	4	Berhasil

	5	Berhasil

Tabel A. 7 Hasil Uji Coba Penyesuaian pada jaringan SDN ukuran *File* 100KB menggunakan 2 *switch*

Split Task	Uji Ke-	Waktu Proses
1MB	1	Berhasil
	2	Gagal
	3	Gagal
	4	Gagal
	5	Berhasil
100KB	1	Gagal
	2	Berhasil
	3	Gagal
	4	Berhasil
	5	Gagal

Tabel A.8 Hasil Ekstraksi Rumen Sebelum penyesuaian Job 1

Job 1		
No	Datanode	Waktu proses *task*
1	slave-3	124
2	slave-3	123
3	slave-2	143
4	slave-3	143
5	slave-1	144
6	slave-1	142
7	slave-1	325
8	slave-3	103
9	slave-1	172
10	slave-2	124
11	slave-2	123

12	slave-1	180
13	slave-3	111
14	slave-2	110
15	slave-2	109
16	slave-3	108
17	slave-1	129
18	slave-2	251
19	slave-2	104
20	slave-2	105
21	slave-3	104
22	slave-1	129
23	slave-1	136
24	slave-3	493
25	slave-2	105
26	slave-1	298
27	slave-1	128
28	slave-2	104
29	slave-2	103
30	slave-1	133
31	slave-3	99
32	slave-2	339
33	slave-1	124
34	slave-1	125
35	slave-3	102
36	slave-2	104
37	slave-2	103
38	slave-3	101
39	slave-3	102
40	slave-2	102

41	slave-2	100
42	slave-3	100
43	slave-3	102
44	slave-1	125
45	slave-1	124
46	slave-2	107
47	slave-2	103
48	slave-3	102
49	slave-3	103
50	slave-1	123
51	slave-1	122
52	slave-2	100
53	slave-2	101
54	slave-3	104
55	slave-3	99
56	slave-1	125
57	slave-2	100
58	slave-2	101
59	slave-1	121
60	slave-3	96
61	slave-3	270
62	slave-2	104
63	slave-2	99
64	slave-3	98
65	slave-1	119
66	slave-1	120
67	slave-3	100
68	slave-2	100
69	slave-3	98

70	slave-1	123
71	slave-1	118
72	slave-3	94
73	slave-2	99
74	slave-2	99
75	slave-1	285
76	slave-1	119
77	slave-1	117
78	slave-3	92
79	slave-2	96
80	slave-2	96
81	slave-3	96
82	slave-2	209
83	slave-1	114
84	slave-2	96
85	slave-2	97
86	slave-3	190
87	slave-3	93
88	slave-3	98
89	slave-1	115
90	slave-1	114
91	slave-1	183
92	slave-3	81
93	slave-2	96
94	slave-2	95
95	slave-2	177
96	slave-1	116
97	slave-3	166
98	slave-3	207

99	slave-1	205
100	slave-2	209
101	slave-1	126

Tabel A.9 Hasil Ekstraksi Rumen Sebelum penyesuaian Job 2

Job 2		
No	**Datanode**	**Waktu prose *task***
1	slave-3	124
2	slave-3	**123**
3	slave-2	143
4	slave-2	143
5	slave-1	144
6	slave-1	142
7	slave-3	325
8	slave-3	103
9	slave-1	172
10	slave-2	124
11	slave-2	123
12	slave-1	180
13	slave-3	111
14	slave-2	110
15	slave-2	109
16	slave-3	108
17	slave-1	129
18	slave-1	251
19	slave-2	104
20	slave-2	105
21	slave-3	104

22	slave-1	129
23	slave-1	136
24	slave-2	493
25	slave-2	105
26	slave-1	298
27	slave-1	128
28	slave-2	104
29	slave-2	103
30	slave-1	133
31	slave-3	99
32	slave-2	339
33	slave-1	124
34	slave-1	125
35	slave-3	102
36	slave-2	104
37	slave-2	103
38	slave-3	101
39	slave-3	102
40	slave-2	102
41	slave-2	100
42	slave-3	100
43	slave-3	102
44	slave-1	125
45	slave-1	124
46	slave-2	107
47	slave-2	103
48	slave-3	102
49	slave-3	103
50	slave-1	123

51	slave-1	122
52	slave-2	100
53	slave-2	101
54	slave-3	104
55	slave-3	99
56	slave-1	125
57	slave-2	100
58	slave-2	101
59	slave-1	121
60	slave-3	96
61	slave-1	270
62	slave-2	104
63	slave-2	99
64	slave-3	98
65	slave-1	119
66	slave-1	120
67	slave-3	100
68	slave-2	100
69	slave-3	98
70	slave-1	123
71	slave-1	118
72	slave-3	94
73	slave-2	99
74	slave-2	99
75	slave-3	285
76	slave-1	119
77	slave-1	117
78	slave-3	92
79	slave-2	96

80	slave-2	96
81	slave-3	96
82	slave-3	209
83	slave-1	114
84	slave-2	96
85	slave-2	97
86	slave-3	190
87	slave-3	93
88	slave-3	98
89	slave-1	115
90	slave-1	114
91	slave-2	183
92	slave-3	81
93	slave-2	96
94	slave-2	95
95	slave-3	177
96	slave-1	116
97	slave-2	166
98	slave-3	207
99	slave-1	205
100	slave-2	209
101	slave-3	126

Tabel A. 10 Hasil Ekstraksi Rumen Sebelum penyesuaian Job 3

Job 3		
No	**Datanode**	**Waktu proe tak**
1	slave-1	117
2	slave-1	117

3	slave-2	131
4	slave-2	139
5	slave-1	116
6	slave-1	120
7	slave-2	119
8	slave-2	118
9	slave-1	164
10	slave-1	159
11	slave-2	105
12	slave-2	107
13	slave-2	110
14	slave-2	105
15	slave-1	129
16	slave-1	132
17	slave-2	108
18	slave-2	101
19	slave-1	134
20	slave-1	128
21	slave-2	103
22	slave-2	104
23	slave-1	125
24	slave-1	131
25	slave-2	101
26	slave-2	105
27	slave-2	101
28	slave-2	105
29	slave-1	125
30	slave-1	122
31	slave-2	102

32	slave-2	100
33	slave-1	123
34	slave-1	127
35	slave-2	100
36	slave-2	101
37	slave-1	121
38	slave-1	120
39	slave-2	102
40	slave-2	98
41	slave-1	119
42	slave-1	120
43	slave-2	100
44	slave-2	100
45	slave-1	118
46	slave-1	123
47	slave-2	100
48	slave-2	99
49	slave-2	95
50	slave-2	97
51	slave-1	116
52	slave-1	117
53	slave-2	96
54	slave-2	96
55	slave-1	121
56	slave-1	116
57	slave-2	97
58	slave-2	97
59	slave-1	118
60	slave-1	117

61	slave-2	97
62	slave-2	118
63	slave-1	115
64	slave-1	114
65	slave-2	117
66	slave-2	108
67	slave-1	119
68	slave-1	117
69	slave-2	110
70	slave-2	104
71	slave-1	115
72	slave-1	113
73	slave-2	105
74	slave-2	103
75	slave-1	114
76	slave-1	131
77	slave-2	104
78	slave-2	102
79	slave-2	99
80	slave-1	124
81	slave-2	103
82	slave-1	119
83	slave-2	105
84	slave-2	100
85	slave-1	121
86	slave-1	120
87	slave-2	101
88	slave-2	99
89	slave-1	119

90	slave-1	125
91	slave-2	96
92	slave-2	97
93	slave-1	115
94	slave-1	116
95	slave-2	96
96	slave-2	96
97	slave-1	112
98	slave-2	94
99	slave-1	104
100	slave-2	97

Tabel A. 11 Hasil Ekstraksi Rumen Sebelum penyesuaian Job 4

Job 4		
No	**Datanode**	**Waktu prose task**
1	slave-1	119
2	slave-1	123
3	slave-2	135
4	slave-2	133
5	slave-1	119
6	slave-1	119
7	slave-2	117
8	slave-2	125
9	slave-1	159
10	slave-1	157
11	slave-2	109
12	slave-2	107
13	slave-2	107

14	slave-2	106
15	slave-1	128
16	slave-1	131
17	slave-2	104
18	slave-2	102
19	slave-1	127
20	slave-1	124
21	slave-2	103
22	slave-2	103
23	slave-1	123
24	slave-1	126
25	slave-2	106
26	slave-2	108
27	slave-1	126
28	slave-1	127
29	slave-2	102
30	slave-2	101
31	slave-2	100
32	slave-2	101
33	slave-1	122
34	slave-1	124
35	slave-2	105
36	slave-2	99
37	slave-1	121
38	slave-1	123
39	slave-2	101
40	slave-2	98
41	slave-1	120
42	slave-1	119

43	slave-2	99
44	slave-2	100
45	slave-1	117
46	slave-1	118
47	slave-2	101
48	slave-2	102
49	slave-1	113
50	slave-2	96
51	slave-2	99
52	slave-1	115
53	slave-2	96
54	slave-2	95
55	slave-1	117
56	slave-1	115
57	slave-2	95
58	slave-2	95
59	slave-1	116
60	slave-1	114
61	slave-2	94
62	slave-2	121
63	slave-1	118
64	slave-1	115
65	slave-2	119
66	slave-2	110
67	slave-1	116
68	slave-1	116
69	slave-2	110
70	slave-2	107
71	slave-1	115

72	slave-1	118
73	slave-2	105
74	slave-2	104
75	slave-1	113
76	slave-1	127
77	slave-2	102
78	slave-2	104
79	slave-1	121
80	slave-2	103
81	slave-1	122
82	slave-2	102
83	slave-2	100
84	slave-1	118
85	slave-2	100
86	slave-1	122
87	slave-2	103
88	slave-1	119
89	slave-2	101
90	slave-1	118
91	slave-2	96
92	slave-2	97
93	slave-1	117
94	slave-1	119
95	slave-2	95
96	slave-2	95
97	slave-1	114
98	slave-1	118
99	slave-2	95
100	slave-2	86

Tabel A.12 Hasil Ekstraksi Rumen Setelah penyesuaian Job 1

Job 1		
No	**Datanode**	**Waktu proses task**
1	slave-3	505
2	slave-1	72

Tabel A. 13 Hasil Ekstraksi Rumen Setelah penyesuaian Job 2

Job 2		
No	**Datanode**	**Waktu proses task**
1	slave-3	489
2	slave-2	78

Tabel A. 14 Hasil Ekstraksi Rumen Setelah penyesuaian Job 3

Job 2		
No	**Datanode**	**Waktu proses task**
1	slave-2	56
2	slave-1	487

Tabel A.15 Hasil Ekstraksi Rumen Setelah penyesuaian Job 4

Job 2		
No	**Datanode**	**Waktu proses task**
1	slave-1	56
2	slave-3	479

BAB VI
KESIMPULAN DAN SARAN

Pada bab ini akan diberikan kesimpulan yang diambil selama pengerjaan Tugas Akhir serta saran-saran tentang pengembangan yang dapat dilakukan terhadap Tugas Akhir ini di masa yang akan datang.

6.1. Kesimpulan

Dari hasil selama proses perancangan, implementasi, serta pengujian dapat diambil kesimpulan sebagai berikut:

1. Hasil dari penyesuaian pada Hadoop berpengaruh signifikan pada waktu pengolahan data.
2. Berdasarkan uji coba penyesuaian MapReduce terlihat bahwa hasil terbaik pengolahan data dengan MapReduce adalah dengan menggunakan 1MB per *task* dengan beberapa Datanode pendukung.
3. Dibutuhkannya penyesuaian dalam menggunakan Hadoop pada perangkat yang murah.
4. Hadoop masih dapat berjalan pada klaster yang terdiri dari perangkat murah.
5. Pemanfaatan Raspberry Pi dalam membuat simulasi klaster Hadoop dapat dilakukan.
6. Tidak ada Raspberry Pi 1 yang dapat menjalankan program MapReduce lebih dari 8 jam.
7. Penggunaan konsep jaringan SDN pada klaster Hadoop hanya dapat menjalankan *file* dengan ukuran maksimal 1MB dan dengan menggunakan ukuran *task* 1MB.
8. Terlalu banyak terciptanya *task* dapat membebani kinerja Hadoop.
9. Ukuran *file* yang dapat diproses maksimum setelah penyesuaian adalah 80MB.
10. *Busy level* dan *processing capability* setelah penyesuaian lebih besar pada suatu Datanode. Sehingga diperlukan *workload balancing* untuk membagi *task* pada node yang *idle*.

6.2. Saran

Berikut saran-saran untuk pengembangan dan perbaikan sistem di masa yang akan datang. Diantaranya adalah sebagai berikut:

1. Diperlukannya algoritma penyatuan *file* untuk meningkatkan kinerja Hadoop.
2. Diperlukan analisis pemilihan blok data dan *split task* yang cocok untuk jenis perangkat keras untuk Hadoop.
3. Diperlukannya pembagian *task* untuk Datanode yang tidak mendapatkan kerja untuk menyeimbangkan *processing capability* dan *busy level*.

DAFTAR PUSTAKA

[1] (2016, Juni) Raspberry Pi Support. [Online]. https://www.raspberrypi.org/help/faqs/#generalDifference

[2] (2016, Feb.) Welcome to Apache Hadoop ! [Online]. http://hadoop.apache.org/

[3] (2016, Mei) Apache Top-Level Wiki. [Online]. http://wiki.apache.org/general/

[4] (2013, Mei) Welcome to Apache Hadoop. [Online]. https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

[5] (2013, Mei) Welcome To Apache Hadoop ! [Online]. https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html

[6] (2013, Mei) Rumen. [Online]. https://hadoop.apache.org/docs/r1.2.1/rumen.html

[7] Srinath Perera and Thilina Gunarathne, *Hadoop MapReduce*. Birmingham: Packt Publishing Ltd, 2013.

[8] Nick Schot, "Feasibility of Raspberry Pi 2 based Micro Data Centers in Big Data Applications," University of Twente, Enschede, 2015.

[9] Jonathan R. Owens, Jon Lentz, and Brian Femiano, *Hadoop Real-World Solutions Cookbook*. Birmingham: Packt Publishing Ltd, 2013.

[10] Xiaofei Hou, Ashwin Kumar T K, and Johnson P Thomas, "Dynamic Workload Balancing for Hadoop MapReduce," Macquarie University, Sydney, 978-1-4799-6719-3/14, 2014.

[11] White Paper, "Hadoop Performance Tunning," 2009.

[12] Ziad Benslimane, "Optimizing Hadoop Parameters Based on the Application Resource Consumption," Uppsala University, Uppsala, 2013.

[13] Shumin Guo, *Hadoop Operations and Cluster Management Cookbook*. Birmingham, UK: Packt Publishing Ltd, 2013.

[14] Shivnath Babu, "Towards Automatic Optimization of

MapReduce Programs," UK, Durham, 978-1-4503-0036-0/10/06, 2010.

BIODATA PENULIS

Wahyu Kukuh Herlambang atau biasa dipanggil Wahyu dilahirkan di Jakarta pada tanggal 29 Maret 1993. Penulis adalah anak pertama dari kedua bersaudara.

Penulis menempuh pendidikan di SD Negeri I Manokwari Papua dan SD Negeri Kompleks IKIP I Kec.Rappocini Makassar (1999-2005), SMP Negeri 6 Pekalongan dan SMP Negeri 3 Kudus (2006-2008), dan SMA Negeri I Kudus (2009-2011). Setelah lulus SMA penulis melanjutkan ke jenjang perkuliahan di Jurusan Teknik Informatika, Fakultas Teknologi Informasi, Institut Teknologi Sepuluh Nopember Surabaya. Bidang Studi yang diambil oleh penulis pada saat kuliah di Teknik Informatika ITS adalah Komputasi Berbasis Jaringan.

Selama menempuh kuliah penulis aktif sebagai anggota Himpunan Mahasiswa Teknik Computer (HMTC) ITS. Penulis juga aktif dalam kegiatan kepanitiaan Schematics sebagai staff hubungan masyarakat (Humas) Schematics 2013. Penulis pernah menjadi asisten dosen PIKTI (2015-2016). penulis juga pernah mengikuti beberapa lomba seperti Hackthon Merdeka 2.0 dan Dycode. Penulis pernah menjadi juara dari lomba Hackthon Merdeka 2.0 Kategori Penanggulan Bencana Asap.

Penulis dapat dihubungi melalui alamat *email* russians.wahyu@gmail.com.