Implementasi *Hadoop*: Studi Kasus Pengolahan Data Peminjaman Perpustakaan Universitas Kristen Petra

Kenny Basuki[1], Henry Novianus Palit[2], Lily Puspa Dewi[3]
Program Studi Teknik Informatika Fakultas Teknologi Industri Universitas Kristen Petra
Jl. Siwalankerto 121 – 131 Surabaya 60236
Telp. (031) – 2983455, Fax. (031) – 8417658
E-Mail: kennybasuki@gmail.com[1], hnpalit@petra.ac.id[2], lily@petra.ac.id[3]

ABSTRAK

Masih minimnya penggunaan dan pengetahuan mengenai metode pengolahan data selain *SQL* merupakan ide dasar dari implementasi *hadoop* ini. Dengan adanya perkembangan teknologi yang semakin pesat membuat pertumbuhan dan pengolahan data semakin besar, untuk itu diperlukanlah suatu metode dalam pengolahan data yang berukuran besar atau biasa disebut *big data*. Dalam implementasi *hadoop* ini juga dilakukan perbandingan performa dengan *SQL* untuk membuktikan spesialisasi dari kemampuan *hadoop* yang ditujukan untuk pengolahan *big data*. Selain itu juga dilakukan pengujian *hadoop* dengan menggunakan jumlah *node* yang bervariasi, menggunakan *combiner*, dan menggunakan ukuran *block* yang berbeda.

Implementasi ini dilakukan dengan menggunakan lima *query* atau permasalahan yang digunakan dalam mengolah data perpustakaan. Kelima permasalahan tersebut adalah mencari jenis audio video, jenis koleksi, judul, lokasi, dan jurusan terbanyak terkait peminjaman buku perpustakaan.

Berdasarkan hasil pengujian implementasi *hadoop mapreduce* didapat beberapa kesimpulan. Performa *hadoop* dari segi waktu jauh lebih baik daripada *SQL* saat mengolah data yang berukuran besar, tetapi tidak dengan data yang berukuran kecil. Semakin banyak komputer yang digunakan dalam implementasi *hadoop* maka semakin cepat pula waktu eksekusi aplikasi *mapreduce*. Penggunaan *combiner* dapat mempercepat waktu eksekusi aplikasi. Penggunaan *block* secara penuh akan memberikan waktu eksekusi program yang jauh lebih baik daripada *block* yang tidak penuh. Dalam implementasi *hadoop* untuk mengolah data peminjaman perpustakaan ini, *block* berukuran 128 MB memberikan waktu eksekusi yang lebih rendah dibanding *block* berukuran 28 MB dan 512 MB yang dicoba dalam implementasi ini.

Kata Kunci: *Hadoop, Big Data, Mapreduce*

ABSTRACT

The lack of knowledge regarding the utilization of data processing methods other than SQL is the general idea of this Hadoop implementation. The advancement of technology generates growing amount of data and demands a new method to process the big data. The performance of this hadoop implementation was also compared with that of SQL to prove hadoop's novelty in processing big data. Moreover different hadoop's implementations – such as various number of nodes, use of a combiner, and use of different block sizes – were evaluated.

Hadoop was implemented for five queries (or problems) in processing the library circulation data. Those five problems are finding the numbers of borrowing transactions categorized by the audio-video types, collection types, titles, locations, and users' departments.

Some conclusions can be drawn based on the hadoop mapreduce implementation. Hadoop's performance tops SQL's when large data are being processed. The more the number of computer nodes, the faster the mapreduce application is to complete its execution. Use of a combiner can speed up the application's execution. The arrangement with full data blocks can give better execution time than that with non-full data blocks does. In this hadoop implementation, the execution time using the block size of 128 MB is smaller than that of 28 MB and 512 MB.

Keywords: *Hadoop, Big Data, Mapreduce*

1. PENDAHULUAN

Seiring dengan perkembangan zaman, *big data* merupakan suatu hal yang sedang menjadi *trend* dalam dunia teknologi informasi. Sebenarnya tidak ada definisi yang pasti mengenai apakah itu *Big data. Big data* merujuk pada kumpulan atau koleksi data yang sangat besar yang dimiliki suatu lembaga atau perusahaan. Sedangkan menurut *Statistical Analysis System* (SAS), *Big data* adalah suatu kondisi yang populer yang digunakan untuk mendefinisikan perkembangan eksponensial serta ketersediaan dari data terstruktur maupun tidak. (Big Data). *Big data* tersebut selanjutnya akan diproses atau diolah oleh perusahaan menjadi suatu informasi yang berguna bagi perusahaan tersebut.

Sejalan dengan hal tersebut, maka pengolahan terhadap *big data* merupakan suatu hal yang kritis pula. Pengolahan terhadap *big data* bukan merupakan suatu hal yang mudah. Pengolahan *big data* tidak dapat disamakan dengan pengolahan data dengan ukuran yang relatif kecil. *Single computer* akan terhambat kinerjanya atau juga tidak akan dapat mengolah data jika ukurannya melebihi kapasitas *memory* pada komputer tersebut. Oleh karena itu diperlukanlah suatu *tool* atau kerangka kerja yang dapat membantu proses pengolahan terhadap *big data*, salah satunya yaitu *Apache Hadoop*. *Apache Hadoop* sendiri merupakan kerangka kerja yang dapat diimplementasikan pada *single computer* ataupun *multiple computer* dalam suatu jaringan tertentu.

Mengetahui pentingnya *issue* mengenai pengolahan *big data* sekarang ini, maka eksplorasi terhadap *Apache Hadoop* dirasa cukup penting. Eksplorasi ini adalah sebagai media implementasi atau praktik langsung agar lebih memahami cara kerja serta fitur-fitur yang ditawarkan *Apache Hadoop* sehingga dapat

meningkatkan kinerja komputer dalam mengolah *big data* bila dibanding dengan pemrosesan dengan *single computer*. Terlebih lagi, eksplorasi ini diharapkan dapat mengubah paradigma dalam pengolahan *big data* yang terlalu terpusat pada *single computer* dan SQL (*Structured Query Language*) menuju pada penggunaan komputer terdistribusi. Karena eksplorasi metode ini memerlukan suatu sumber data yang besar, maka dari itu digunakanlah data peminjaman buku perpustakaan Universitas Kristen Petra sebagai sumbernya. Diharapkan dengan pengolahan terhadap data tersebut dapat menghasilkan informasi-informasi baru terkait peminjaman buku perpustakaan, sehingga dapat ditindaklanjuti oleh pengurus perpustakaan.

Oleh karena itu, skripsi ini dibuat untuk memahami instalasi, cara penggunaan serta fitur yang ditawarkan *Apache Hadoop* lebih lanjut lagi, sehingga nantinya akan lebih banyak lagi pengembangan serta implementasi *Apache Hadoop* di Indonesia. Lebih lanjut lagi kedepannya, diharapkan melalui skripsi ini dapat membantu pertumbuhan penggunaan serta pengolahan *big data* di Indonesia yang masih jarang sekarang ini, terutama dalam lingkup Universitas Kristen Petra.

2. DASAR TEORI

2.1 *Big Data*

Big data merupakan konsep data yang memiliki dimensi 3Vs volume, variasi, dan kecepatan data. Volume merujuk pada banyaknya data, variasi merujuk pada jumlah dari tipe data yang digunakan dan kecepatan merujuk pada kecepatan dari pemrosesan data. Berdasarkan model 3Vs, tantangan yang dihadapi untuk mengelola *big data* adalah dari ketiga dimensi tersebut, bukan hanya mengenai jumlah atau volume data saja. Doug Laney analis dari *Gartner* memperkenalkan konsep 3Vs pada saat publikasi riset MetaGroup yang bertajuk *3D Data Management: Controling data volume, variety, and velocity* [5].

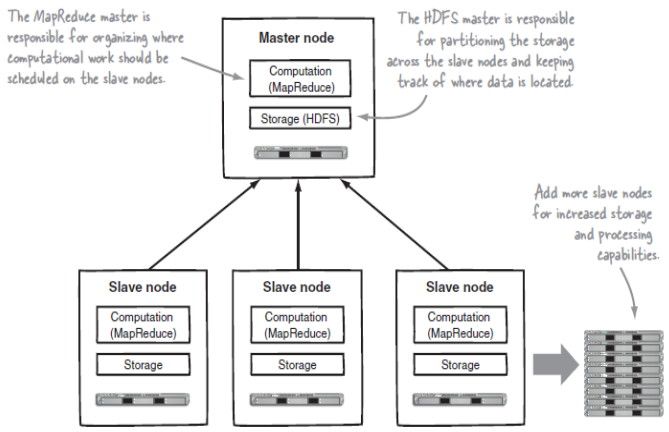
2.2 *Distributed Systems* (DS)

DS adalah adalah suatu sistem dimana komponen-komponennya terletak pada suatu jaringan komputer, berkomunikasi dan mengkoordinasikan tindakan mereka dengan hanya mengirim pesan. Definisi tersebut mengikuti karakteristik utama dari DS yaitu komponen yang digunakan bersamaan, kurangnya waktu global, dan kegagalan komponen yang bersifat independen [1].

2.3 *Apache Hadoop*

Hadoop merupakan sebuah *platform* yang menyediakan penyimpanan terdistribusi dan kemampuan komputasi. *Hadoop* pertama kali disusun untuk memecahkan masalah skalabilitas yang ada pada Nutch (sebuah *open source* dan *search engine*). Sebelumnya *Google* telah mempublikasikan tulisan mengenai *distributed file system*, *Google File System* (GFS), dan *MapReduce*, yang merupakan sebuah kerangka kerja komputasi untuk pemrosesan paralel. Keberhasilan dari implementasi konsep tersebut pada Nutch menghasilkan dua buah proyek yang terpisah, proyek kedua tersebut adalah menjadi *Hadoop*.

Secara lebih tepat, seperti terlihat pada Gambar 1, *Hadoop* merupakan *master-slave architecture* yang terdiri dari *Hadoop Distributed File System* (HDFS) untuk media penyimpanan serta *MapReduce* untuk kemampuan komputasi. Sifat intrinsik dari *Hadoop* adalah partisi data dan komputasi paralel dari sumber data yang besar. Media penyimpanan tersebut dan skala kemampuan komputasi dengan penambahan *hosts* untuk *Hadoop cluster*, serta dapat mencapai ukuran volume hingga *petabytes* pada *clusters* dengan ribuan *hosts* [2].



Gambar 1. Arsitektur *Hadoop*

2.4 *Hadoop Distributed File System* (HDFS)

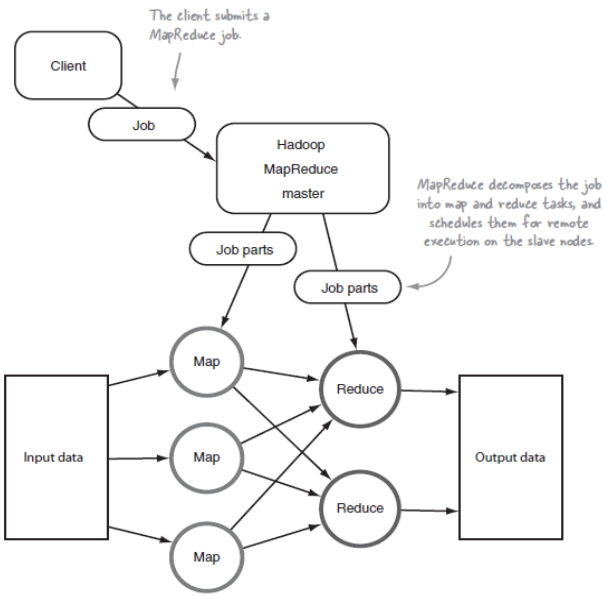
HDFS adalah komponen penyimpanan *Hadoop*. HDFS adalah *file system* terdistribusi yang dimodelkan setelah tulisan mengenai *Google File System* (GFS). HDFS dioptimasi untuk *throughput* yang tinggi dan berkerja paling baik ketika membaca serta menulis file besar (*gigabytes* atau lebih besar). Pengaruh HDFS ini adalah untuk mendukung ukuran blok yang besar (untuk *file system*) dan optimasi data lokal untuk mengurangi *input/output* jaringan [2].

HDFS memiliki dua jenis *node* yang beroperasi dalam arsitektur *master-slave*. Sebuah *namenode* (*master*) dan beberapa *datanodes* (*slaves*). *Namenode* mengelola *filesystem namespace*. *Namenode* menjaga *filesystem tree* dan metadata untuk semua *file* dan direktori dalam *tree*. *Datanode* merupakan bagian yang berkerja keras dalam *fileysystem*. *Datanode* menyimpan dan mendapatkan blok ketika diperintahkan oleh (aplikasi *client* dan *namenode*), dan *datanode* juga melaporkan balik kepada *namenode* secara periodik dengan daftar blok yang disimpan dalam *datanode* tersebut [7].

2.5 *Hadoop Mapreduce*

MapReduce adalah sebuah model pemrograman untuk pemrosesan data. Model ini cukup sederhana, tetapi tidak terlalu sederhana dalam mengekspresikan suatu program tertentu. Lebih penting, program *MapReduce* secara inheren bersifat paralel, sehingga menempatkan analisis data skala besar kepada siapapun dengan cukup mesin sebagai penyelesaiannya. *MapReduce* pun hadir sebagai solusi untuk mengolah data yang besar [7].

Model *MapReduce* menyederhanakan proses paralel dengan meringkas kompleksitas yang terlibat dalam berkerja dengan *distributed system* seperti komputasi paralel, pembagian kerja, serta berurusan dengan *hardware* dan *software* yang tidak dapat dihandalkan. Dengan abstraksi ini, *MapReduce* memperbolehkan *programmer* untuk fokus pada kebutuhan bisnis, daripada berurusan dengan komplikasi *distributed system*. Untuk model kerja dari algoritma *mapreduce* dapat dilihat pada Gambar 2 [2].



Gambar 2. Penyerahan *job* dari *client* ke *Mapreduce*

Operasi *map* adalah dengan membagi *input file* secara paralel menjadi beberapa bagian yang dinamakan *filesplits*. Jika sebuah *file* tunggal terlalu besar maka akan mempengaruhi waktu membaca sehingga *file* tersebut dibagi menjadi beberapa bagian. Proses pembagian *file* tidak mengerti apapun mengenai struktur logika dari *input file*, sebagai contoh *file* berbasis teks bergaris dibagi kedalam beberapa bagian dan dipecah secara *byte*. Lalu *map task* akan dibuat per *filesplit* yang ada.

Ketika sebuah *reduce task* dijalankan, proses *reduce* tersebut mengambil input yang terbagi kedalam beberapa *file* dalam semua *nodes* yang digunakan saat menjalankan *map tasks*. Setelah semua data yang tersedia secara lokal ditambahkan kedalam satu *file* pada fase penambahan. *File* tersebut kemudian digabung dan diurutkan sehingga pasangan *key value* untuk *key* tertentu akan bersebelahan. Hal ini membuat operasi *reduce* yang sebenarnya menjadi sederhana: *file* dibaca secara berurutan dan *value* dikirim ke proses *reduce* secara berulang-ulang berdasarkan *key* yang ditemui [6].

2.6 *YARN*

Untuk setiap *cluster* yang sangat besar dengan sekitar 4000 *nodes* atau lebih, sistem *MapReduce* yang ada sebelumnya memiliki problem dalam hal skalabilitas, jadi pada tahun 2010 sebuah grup di Yahoo! memulai untuk melakukan desain *MapReduce* generasi selanjurnya. Hasilnya adalah YARN, yang disingkat dari *Yet Another Resource Negotiator*.

YARN menanggulangi kekurangan skalabilitas dari *MapReduce* klasik dengan membagi tanggung jawab dari *job tracker* kedalam beberapa bagian. *Jobtracker* yang bertanggung jawab terhadap *job scheduling* (mencocokkan *task* dengan *task trackers*) dan memonitor kemajuan dari *task* (tetap melacak tugas dan melakukan *restart* pada tugas yang lambat atau gagal, serta melakukan pembukuan seperti mencatat total tugas sejauh ini) [7].

2.7 *Java*

Java, yang paling diingat adalah bahwa Java merupakan bahasa pertama yang memperbolehkan *developers* untuk membuat *cross-platform networked software* untuk diterapkan di *internet*. Sekali menggunakan Java, seseorang dapat membuang gagasan bahwa semua *software* harus dibangun dan dijalankan pada suatu *platform* yang sama, kemudian dilakukan *porting* bila program tersebut butuh untuk diterapkan pada sistem lainnya [4].

2.8 Perpustakaan Universitas Kristen Petra

Perpustakaan UK Petra mulai berdiri pada akhir tahun 1966, 5 tahun setelah berdirinya Universitas Kristen Petra yang berlokasi di Jalan Embong Kemiri no. 11 Surabaya, dengan koleksi awal berjumlah 696 eksemplar buku, serta 46 judul majalah dalam dan luar negeri. Kemudian pada tahun 1977 secara berangsur-angsur kampus Universitas Kristen Petra beserta perpustakaan yang merupakan perpustakaan pusat ini pindah ke gedung yang dirancang sebagai kampus Universitas Kristen Petra di jalan Siwalankerto 121-131 [3].

3. ANALISIS DAN DESAIN SISTEM

3.1 Perencanaan dan Garis Besar Implementasi *Hadoop*

Dalam implementasi Hadoop untuk melakukan pengolahan data peminjaman perpustakaan UK. Petra, dibutuhkan langkah-langkah perencanaan sebelum *Hadoop* dapat benar-benar diimplementasikan. Langkah-langkah tersebut dilakukan untuk mempersiapkan program pengolah data dengan *Hadoop* dapat dijalankan, dapat menghasilkan suatu keluaran serta dapat dipahami cara kerjanya. Langkah-langkah tersebut adalah sebagai berikut:

1. Melakukan instalasi dan konfigurasi *Hadoop* pada *namenode*, dan *datanode*.
2. Mengolah data dari perpustakaan menjadi bentuk yang lebih mudah untuk diolah lebih lanjut dalam program *Hadoop*.
3. Menambah ukuran data peminjaman perpustakaan menjadi berukuran cukup besar sekitar 9,6 GB (*Gigabytes*) karena dirasa ukuran data tersebut masih belum terlalu besar serta menginputkannya kedalam HDFS.
4. Membuat 5 buah program dengan permasalahan yang berbeda untuk mengolah data peminjaman perpustakaan berdasarkan paradigma *mapreduce*.

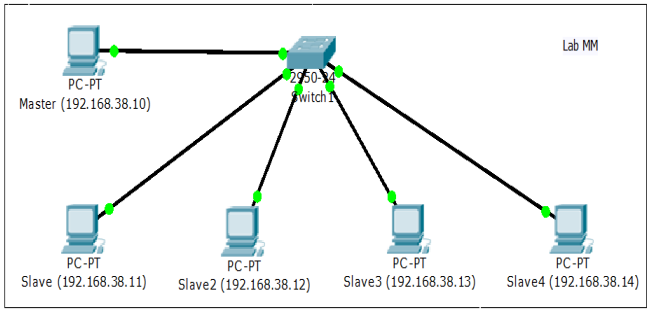
Setelah langkah-langkah tersebut dilakukan, maka program *mapreduce* dengan *Hadoop* baru dapat dijalankan sehingga memberikan suatu hasil analisa dari sekumpulan data berukuran besar. Setelah program *Hadoop* dapat berjalan dengan lancar, maka dapat dilakukan beberapa perbandingan untuk mengetahui sejauh mana performa *Hadoop* dalam mengolah data berukuran besar yang diukur berdasarkan kecepatan pemrosesan data serta mendapatkan komposisi *Hadoop* dengan performa yang baik. Perbandingan-perbandingan tersebut meliputi beberapa aspek yaitu kecepatan, *CPU* load, dan *memory usage*. Perbandingan yang akan dilakukan adalah sebagai berikut:

1. Membandingkan *query* SQL dengan program *Hadoop mapreduce*.
2. Membandingkan program *Hadoop mapreduce* dengan adanya penambahan jumlah *node*.
3. Membandingkan program *Hadoop mapreduce* tanpa dan menggunakan *combiner class*.
4. Membandingkan program *Hadoop mapreduce* dengan beberapa ukuran *block*.

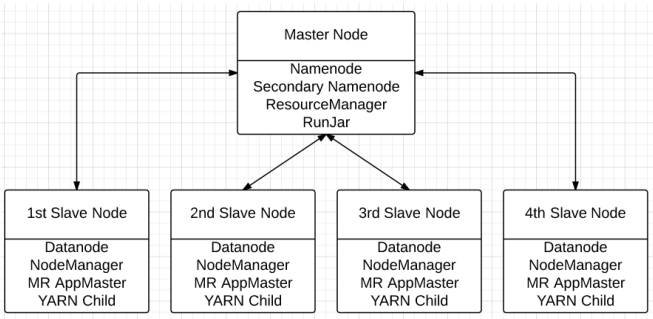
Semua perbandingan yang diatas dilakukan dalam 5 permasalahan yang berbeda, ukuran *input data* yang sama, serta menghasilkan keluaran yang mendekati sama. Sehingga dari perbandingan tersebut dapat diketahui sejauh mana performa *Hadoop* serta bagaimana dampak aspek-aspek pembanding dengan performa *Hadoop*.

3.2 Desain Arsitektur *Hadoop*

Sistem *hadoop* akan diimplementasikan pada Laboratorium Multimedia, Program Studi Teknik Informatika, Universitas Kristen Petra. Untuk memudahkan dalam memahami arsitektur sistem *hadoop* yang akan diimplementasikan maka dibuatlah desain arsitektur secara fisik dan logika. Secara fisik, komputer-komputer yang digunakan terhubung ke *switch* yang berada di laboratorium dengan kabel UTP (*Unshielded Twisted Pair*) berkecepatan maksimal 100 Mbps (*Megabit per second*). Desain fisik sistem *hadoop* dalam dilihat pada Gambar 3, sedangkan untuk desain logika serta *daemon* (*background service*) yang dijalankan dalam sistem *hadoop* dapat dilihat pada Gambar 4.



Gambar 3. Desain Jaringan Implementasi *Hadoop*



Gambar 4. Desain Logika Implementasi *Hadoop*

3.3 Pengolahan Data Perpustakaan

Sebelum implementasi *hadoop mapreduce* pada data peminjaman perpustakaan dapat dijalankan, maka perlu dilakukan pembersihan (*cleansing*) terhadap *file* data perpustakaan agar dapat lebih mudah digunakan dalam implementasi *Hadoop*. Data perpustakaan yang digunakan terdiri dari 7 *file* yaitu lib_buku,csv, lib_judul.csv, lib_jurusan.csv, lib_m_jenisav.csv, lib_m_jnskol.csv, lib_m_lokasi.csv, dan lib_sir_sejarah.csv.

Pengolahan data perpustakaan dibagi dalam dua tahap, yaitu pertama dilakukan pada semua *file* untuk menghilangkan karakter "\n" didalam konten, menghilangkan *header file*, serta menghilangkan tanda petik dan mengganti tanda koma menjadi "|" (*pipe line*). Tahap kedua adalah memisahkan *file* lib_sir_sejarah menjadi 2 *file* yaitu sejarah_no_rel.csv dan sejarah_rel.csv. Pemisahan *file* tersebut untuk memudahkan dalam pengolahan data oleh *hadoop*, karena terdapat perbedaan pola dalam *file* lib_sir_sejarah sehingga harus dipisah menjadi 2 *file* tersebut,

3.4 Pembesaran dan Penggandaan Ukuran Data

Pembesaran dan penggandaan data dilakukan karena dirasa setelah pengolahan data ukuran *file* transaksi yaitu sejarah_rel.csv dan sejarah_no_rel,csv belum terlalu besar, maka dilakukanlah upaya pembesaran. Pembesaran ini dilakukan dengan menambah isi dari *file* tersebut sampai ukuran tertentu sehingga lebih besar. Sedangkan penggandaan dilakukan dengan mengunggah kedua *file* tersebut ke HDFS hingga jumlah tertentu.

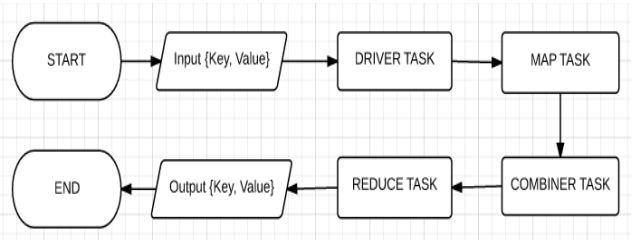
3.5 Pembuatan Aplikasi

Pembuatan aplikasi dilakukan dengan menggunakan IDE (*Integrated Development Environment*) Eclipse dan menggunakan bahasa pemrograman Java. *Hadoop* telah menyediakan *library* yang digunakan untuk mengembangkan aplikasi *mapreduce*. Lakukan *export library* JAR dari *folder hadoop* dari lokasi common/*.jar, common/lib/*.jar, hdfs/*.jar, mapreduce/*.jar, tools/lib/*.jar, dan yarn/*.jar.

Dalam implementasi *hadoop* untuk mengolah data perpustakaan terdapat lima aplikasi yang akan dibuat, yaitu :

1. Aplikasi untuk mencari jenis *audio video* yang paling sering dipinjam.
2. Aplikasi untuk mencari jenis koleksi yang paling sering dipinjam.
3. Aplikasi untuk mencari judul yang paling sering melakukan peminjaman.
4. Aplikasi untuk mencari jurusan dari koleksi yang paling sering dipinjam.
5. Aplikasi untuk mencari lokasi yang paling sering dipinjam.

Secara umum proses kerja aplikasi *hadoop* dapat dilihat pada Gambar 5.



Gambar 5. *Flowchart* Umum *Mapreduce*

4. IMPLEMENTASI SISTEM

4.1 Instalasi dan Konfigurasi *Hadoop*

Instalasi *Hadoop* dilakukan diatas sistem operasi Ubuntu. Sebelum instalasi dan konfigurasi *hadoop* dilakukan pastikan pada semua komputer telah terpasang modul SSH ,JVM (*Java Virtual Machine*), dan *rsnyc*. Modul-modul tersebut akan digunakan oleh *Hadoop* untuk menjalankan apliakasi *mapreduce*. Selanjutnya, buat SSH *public key* dan sebarkan antar komputer yang digunakan.

Setelah semua dilakukan *Apache Hadoop* dapat diunduh pada website *Apache Hadoop*, selanjutnya ektrak *file Hadoop* dalam bentuk .tar.gz dan tempatkan pada direktori /usr/local di Ubuntu. Setelah semua hal tersebut dilakukan barulah dapat melakukan konfigurasi pada *Hadoop*. Secara garis besar terdapat lima *file* yang perlu dikonfigurasi yaitu sebagai beriktu :

1. hadoop-env.sh untuk export lokasi JVM.
2. core-site.xml untuk mendefinisikan lokasi dan port dari master.
3. yarn-site.xml untuk melakukan konfigurasi terkait *node manager* dan *resource manager*.
4. hdfs-site.xml untuk mendefinisikan jumlah replikasi, lokasi *namenode* dan *datanode*.
5. mapred-site.xml untuk mendefinisikan nama *framework*, dan lokasi penyimpanan *job history*.

Setelah konfigurasi selesai, direktori *hadoop* dapat dikirimkan pada setiap komputer (dalam hal ini *slave*). Kemudian, service *hadoop* dapat dimulai dengan perintah start-all.sh pada *master*. Perintah tersebut mengaktifkan *service* berupa *resource manager*, *node manager*, *namenode*, *datanode*, dan *secondary namenode*.

4.2 Implementasi *Driver Class*

Driver class merupakan salah satu dari bagian utama dalam algoritma *mapreduce*. *Driver class* merupakan pusat yang mengendalikan suatu *job* berupa *mapper*, *combiner*, dan *reducer* agar berjalan dengan baik. *Driver* dapat memiliki bagian-bagian yang bersifat opsional seperti konfigurasi *cache file*, *sort key comparator*, dan *partition class*. Adapun fungsi-fungsi umum *driver* untuk mengatur lokasi *input file*, lokasi *output file*, jumlah *reduce task* yang akan dijalankan dan sebagainya.

4.3 Implementasi *Mapper Class*

Mapper class merupakan bagian dari algoritma *mapreduce* yang bertujuan dalam memetakan data sehingga nantinya dikirimkan ke *combiner* ataupun *reducer*. Dalam implementasi pengolahan data perpustakaan ini terdapat enam jenis *mapper* yang digunakan yaitu *sejarah relation mapper*, *sejarah no relation mapper*, *counting join mapper*, *data join mapper*, *counting mapper*, dan *sorting mapper*.

4.4 Implementasi *Combiner Class*

Combiner class merupakan bagian dari algoritma *mapreduce* yang bertujuan dalam membantu meringankan beban *reducer* dalam berkerja. *Combiner* berkerja dengan mengolah hasil keluaran dari *mapper*, sehingga nantinya *reducer* akan menerima lebih sedikit pasangan *key*, *value* untuk diolah. Dalam implementasi pengolahan data perpustakaan ini terdapat tiga jenis *combiner* yang digunakan yaitu *sejarah relation combiner*, *two input combiner*, dan *general combiner*.

4.5 Implementasi *Reducer Class*

Reducer class merupakan bagian dari algoritma *mapreduce* yang bertujuan untuk melakukan reduksi terhadap pasangan *key*, *value* yang diterima, sehingga akan menghasilkan satu atau beberapa *value* untuk setiap *key* yang diterima. Dalam implementasi pengolahan data perpustakaan ini terdapat lima jenis *reducer* yang digunakan *sejarah relation reducer*, *two input reducer*, *general reducer*, *join reducer*, dan *output reducer*.

4.6 Implementasi *Sort Key Comparator Class*

Sort key comparator class merupakan bagian dari implementasi program yang digunakan untuk mengembalikan nilai dari *key* yang satu dengan lainnya. Hasil dari implementasi ini akan menghasilkan *record* dari *mapper* ke *reducer* yang terurut secara *descending*.

4.7 Implementasi *Composite Class*

Composite class merupakan sebuah *class* yang berisi variabel-variabel dan fungsi yang digunakan untuk membantu proses *mapping*. Maksud dari membantu proses *mapping* yang dimaksud adalah obyek dari *composite class* ini akan menjadi *key* dan *value* dari proses *mapper* ke *reducer*. *Class* ini dibutuhkan karena pada dasarnya *key* dan *value* merupakan obyek yang memiliki nilai tunggal saja. Sedangkan untuk melakukan proses *sejarah relation map* dibutuhkan komposisi berupa nomor induk peminjam, nomor induk buku, dan tanggal pinjam sebagai *key* untuk dibandingkan dan direduksi dengan *record* lainnya.

5. PENGUJIAN SISTEM

Pengujian dilakukan dengan mengambil nilai puncak pada parameter berupa persentase penggunaan *CPU*, penggunaan *memory*, serta waktu eksekusi saat aplikasi *mapreduce* ataupun *SQL* berjalan. Setelah mendapatkan nilai ketiga parameter tersebut, langkah selanjutnya adalah membandingkan nilai-nilai tersebut sehingga dapat mengetahui aplikasi mana yang lebih cepat serta konsumsi aplikasi terhadap *resource* komputer.

5.1 Perbandingan *Query SQL* dengan *Mapreduce*

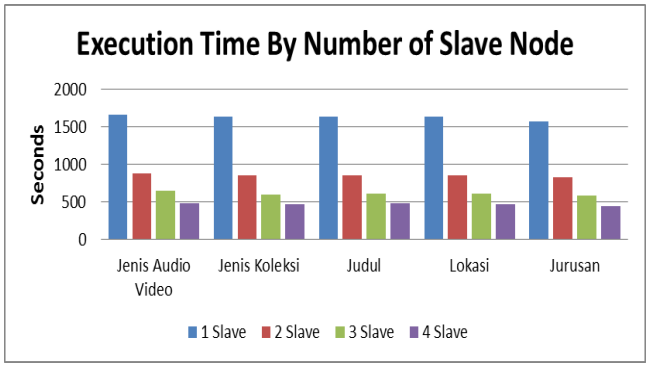
Pengujian dengan membandingkan *query SQL* dengan *mapreduce* terbagi menjadi dua jenis, yaitu saat data transaksi belum digandakan dan dengan data transaksi yang telah digandakan hingga 425 kali. Untuk hasil sebelum penggandaan menunjukkan bahwa *SQL* lebih baik daripada *hadoop mapreduce* dalam mengolah data yang belum digandakan (berukuran kecil). *SQL* memberikan waktu eksekusi yang lebih rendah serta konsumsi persentase konsumsi *CPU* dan *memory* yang cukup rendah dibanding *hadoop*. Melalui pengujian tersebut disimpulkan rata-rata konsumsi *CPU SQL* kurang lebih 304,44% lebih rendah dibanding *mapreduce*, rata-rata konsumsi *memory SQL* kurang lebih 1745,17 MB (*Megabytes*) lebih rendah dibanding *mapreduce*, dan rata-rata waktu eksekusi *SQL* kurang lebih 78,804 detik lebih rendah dibanding *mapreduce*.

Untuk hasil setelah penggandaan menunjukkan bahwa dengan menggunakan *SQL* mendapatkan nilai yang baik pada dua dari tiga kriteria yang diujikan. Dari segi *CPU usage* dan *memory usage*, *SQL* memiliki *CPU usage peak* dan *memory usage peak* yang lebih rendah dibanding *Hadoop mapreduce*. Untuk *CPU usage peak SQL* terpaut antara 100,166% sampai 129,266% dibanding *hadoop* yang terpaut antara 402,167% sampai 403,167%. Sedangkan untuk *memory usage peak SQL* terpaut antara 218,61 MB sampai 312,801 MB dibanding *hadoop* yang terpaut antara 3147,955 MB sampai 3320,708 MB. Sedangkan mengenai waktu eksekusi dimana pada titik ini, *Hadoop mapreduce* terlihat sangat unggul jauh dari *SQL*. Dalam hal waktu eksekusi *SQL* yang hasilnya dapat lebih besar jauh daripada *hadoop* karena beberapa faktor antara jumlah *records*, kompleksnya *query* yang termasuk didalamnya banyaknya *join* yang terjadi dan adanya *aggregate function* dalam *query* tersebut.

Jadi semakin banyak *records* dan terdapat beberapa fungsi *join* dapat memperlambat waktu eksekusi *SQL*.

5.2 Perbandingan Jumlah *Node* pada Aplikasi *Mapreduce*

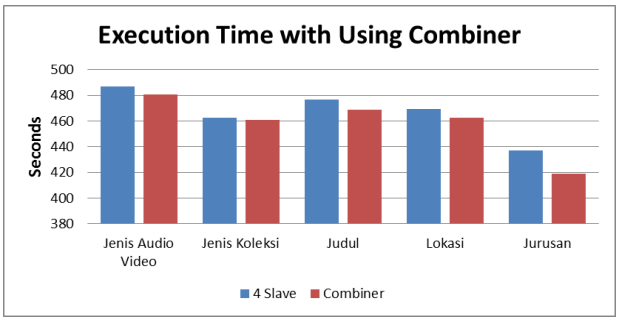
Pada pengujian aplikasi *hadoop mapreduce* dengan menggunakan jumlah *slave* bervariasi untuk konsumsi *CPU* dan *memory* tidak ada perbedaan yang signifikan. Dalam hal konsumsi *CPU master node* menggunakan kurang lebih 180% dan *slave node* menggunakan kurang lebih 400%. Sedangkan untuk *memory*, *master node* menggunakan kurang lebih 1 GB (*Gigabytes)* dan *slave node* menggunakan kurang lebih 2 GB. Sedangkan untuk waktu eksekusi, pada aspek ini sangat terlihat jelas dari bertambahnya jumlah *slave* atau pekerja, maka waktu eksekusi semakin cepat, tetapi semakin banyak pertambahan *slave* yang dilakukan memberi semakin sedikit pengurangan waktu yang didapat dari sebelumnya. Perbandingan waktu eksekusi pada aplikasi *mapreduce* dapat dilihat pada Gambar 6.



Gambar 6. Grafik Waktu Eksekusi *Mapreduce* Berdasarkan Jumlah *Slave Node*

5.3 Pengujian Aplikasi *Mapreduce* dengan *Combiner*

Pada pengujian aplikasi *hadoop mapreduce* dengan menggunakan *combiner*, untuk konsumsi *CPU* dan *memory* tidak ada perbedaan yang signifikan. Dalam hal konsumsi *CPU master node* menggunakan kurang lebih 180% dan *slave node* menggunakan kurang lebih 400%. Sedangkan untuk *memory*, *master node* menggunakan kurang lebih 1 GB (*Gigabytes)* dan *slave node* menggunakan kurang lebih 2 GB. Untuk waktu eksekusi, pada aspek ini sangat terlihat jelas dari penggunaan *combiner*, waktu esksekusi semakin cepat, tetapi hanya lebih cepat sedikit. Pertambahan kecepatan tersebut diduga karena *combiner* membantu mengurangi beban kerja *reducer*. Hal tersebut berarti bahwa beban kerja *reducer* cukup berat, sehingga dengan adanya bantuan *combiner* dapat mengurangi waktu eksekusi *mapreduce*. Melalui pengujian tersebut disimpulkan rata-rata waktu eksekusi dengan menggunakan *combiner* kurang lebih 8,206 detik lebih rendah dibanding tidak mengunakan *combiner*. Perbandingan waktu eksekusi pada aplikasi *mapreduce* dengan atau tanpa *combiner* dapat dilihat pada Gambar 7.



Gambar 7. Grafik Waktu Eksekusi *Mapreduce* dengan Penggunaan *Combiner*

5.4 Pengujian Aplikasi *Mapreduce* dengan *Block* Penuh

Secara *default* ukuran *block* pada HDFS adalah 128 *Megabytes* (MB), dan setiap satu *file* yang diunggah ke HDFS menempati satu buah *block*. *File* yang dimaksud adalah yang *file* sumber yang digunakan untuk menjalankan aplikasi *mapreduce* yaitu *file* sejarah_no_rel.csv dan sejarah_rel.csv. Kedua *file* tersebut memiliki ukuran yang jauh dari *block* HDFS.

Untuk sejarah_no_rel.csv berukuran sekitar 13,33 MB dan sejarah_rel.csv berukuran sekitar 8,9 MB. Maka dari itu, dalam pengujian ini dilakukan pengunggahan ke HDFS dengan ukuran *file* yang hampir seukuran HDFS, sehingga seluruh sumber yang sebelumnya menghabiskan banyak *block* menjadi menghabiskan lebih sedikit *block*. Tujuan dari pemadatan *file* tersebut sebenarnya adalah untuk menguji aplikasi *mapreduce* dengan lebih sedikit *map task* dengan isi yang sama. Lebih sedikit *map task* karena *map task* dijalankan pada setiap *block*.

Disamping terdapat pemadatan *file* dalam pengujian terhadap *block* ini dilakukan tiga jenis pengujian dengan ukuran *block* yang berbeda, yaitu *block* dengan ukuran 28 MB, 128 MB, dan 512 MB. Pemilihan ketiga ukuran *block* tersebut didasarkan keperluan untuk mengetahui dampak ukuran *block* terhadap ketiga parameter yang diujikan, maka dari itu dipilih satu ukuran yang lebih kecil dan satu ukuran yang lebih besar dari ukuran *default block* (128 MB). Ukuran 128 MB dipilih karena merupakan ukuran *default* dari HDFS *block*, sedangkan ukuran 28 MB dipilih karena lebih kecil dari 128 MB, dan untuk membuktikan bahwa dalam HDFS dapat diimplementasikan ukuran *block* yang bukan kelipatan pangkat dua seperti ukuran data komputer pada umumnya. Hal ini dapat terjadi karena HDFS *block* merupakan *logical file system* sehingga dapat lebih fleksibel daripada *file system* pada umumnya. Sedangkan ukuran *block* 512 MB dipilih karena lebih besar daripada 128 MB sebagai ukuran *default*. Pada *block* yang tidak penuh pada implementasi sebelumnya data transaksi yaitu dalam hal ini *file* sejarah_rel.csv dan sejarah_no_rel.csv yang diproses hanya menempati sekitar 7,2% sampai 10,8% dari ukuran *block* yang tersedia, dimana *default* dari ukuran *block* tersebut adalah 128 MB.

5.4.1 Pengujian dengan Block 128 MB

Pengujian *block* penuh yang pertama dilakukan adalah dengan menggunakan ukuran *block default* yaitu 128 MB. Untuk melakukan hal tersebut maka perlu melakukan *concate* atau penggabungan *file* menjadi satu, yaitu untuk setiap 9 *file* sejarah_no_rel.csv menjadi 1 *file* dan untuk setiap 14 *file* sejarah_rel.csv menjadi 1 *file*, sehingga masing-masing *file*

tersebut berukuran mendekati 128 MB. Dalam pengujian ini *file* transaksi yang diproses menempati sekitar 93,7% sampai 96,1% dari ukuran *block* yang tersedia.

5.4.2 Pengujian dengan Block 512 MB

Dalam pengujian ini *file* transaksi yang diproses menempati sekitar 96,7% sampai 96,1% dari ukuran *block* yang tersedia. Untuk itu pada subbab ini dilakukan pengujian aplikasi dengan *file* yang di-*upload* dengan ukuran *block* sebesar 512 MB.

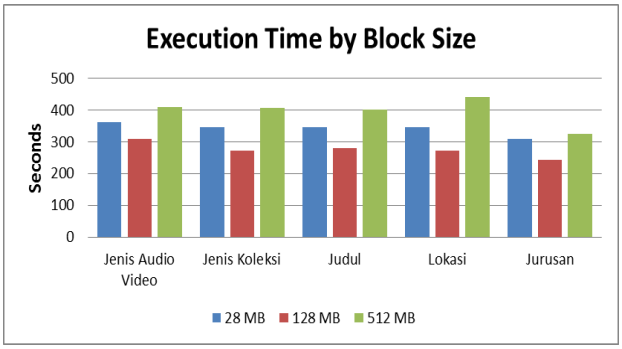
5.4.3 Pengujian dengan Block 28 MB

Dalam pengujian ini *file* transaksi yang diproses menempati sekitar 99% sampai 99,6% dari ukuran *block* yang tersedia.

Pengujian *hadoop mapreduce* dengan menggunakan ukuran *block* yang berbeda tidak mengubah konsumsi *CPU* dan *memory* secara signifikan. Sama seperti sebelumnya konsumsi *CPU* dan *memory*, dalam hal konsumsi *CPU master node* menggunakan kurang lebih 180% dan *slave node* menggunakan kurang lebih 400%. Sedangkan untuk *memory*, *master node* menggunakan kurang lebih 1 GB (*Gigabytes)* dan *slave node* menggunakan kurang lebih 2 GB. Sedangkan dalam segi waktu eksekusi dapat disimpulkan bahwa pertama, penggunaan *blockfull* pada HDFS memberikan waktu eksekusi yang lebih baik, kedua, tidak ada ukuran *block* yang pasti, performa yang baik didapat dengan melakukan eksplorasi seperti dalam implementasi pengujian ini.

Ukuran *block* yang memberikan waktu lebih baik adalah *block* 128 MB dibanding 28 MB dan 512 MB.Diduga *block* berukuran 28 MB memberikan waktu yang lebih lama dibanding 128 MB karena jumlah *block* yang harus diproses oleh suatu *node* jauh lebih besar dibandingkan yang lain sehingga menciptakan latensi bagi *slave node* dalam melakukan pemrosesan walaupuna memiliki ukuran yang lebih kecil dibandingkan 128 MB dan 512 MB. Selanjutnya, *block* berukuran 512 MB juga mendapatkan waktu yang lebih lama dibandingkan *block* berukuran 128 MB karena diduga ukuran *block* yang harus diproses jauh lebih besar daripada 128 MB, sehingga otomatis memberikan beban yang lebih besar bagi suatu *node* dalam mengolah data dalam suatu *block* tertentu. Sehingga dalam pengujian ini membuktikan tidak ada ukuran *block* yang pasti untuk mendapatkan performa yang baik, banyak faktor yang mempengaruhi performa suatu aplikasi mulai dari kompleksnya aplikasi itu sendiri, ukuran *block*, dan *resource* pada suatu komputer.

Perbandingan waktu eksekusi pada aplikasi *mapreduce* dengan atau tanpa *combiner* dapat dilihat pada Gambar 8.



Gambar 8. Grafik Waktu Eksekusi *Mapreduce* dengan Perbedaan Ukuran *Block*

Melalui pengujian tersebut disimpulkan rata-rata waktu eksekusi *mapreduce* pada HDFS dengan ukuran *block* 28 MB kurang lebih 66,395 detik lebih tinggi dibanding pada HDFS dengan ukuran *block* 128 MB, dan rata-rata waktu eksekusi *mapreduce* pada HDFS dengan ukuran *block* 128 MB kurang lebih 121,651 detik lebih rendah dibanding pada HDFS dengan ukuran *block* 512 MB.

6. KESIMPULAN

Dari proses yang dilakukan mulai dari perancangan sistem sampai pengujian aplikasi yang telah dilakukan, dapat diambil beberapa kesimpulan yaitu seperti di bawah ini.

1. Pada peengolahan data yang berukuran kecil jangan gunakan *hadoop mapreduce* karena memberikan waktu eksekusi yang cukup tinggi yaitu berkisar antara 58,583 detik hingga 101,34 detik. Waktu eksekusi tersebut disebabkan karena *hadoop* perlu melakukan inisialisasi pada setiap *job* dan *container* yang berjalan. Penggunaan *SQL* dengan *indexing* lebih baik untuk mengolah data yang releatif kecil dibandingkan *hadoop mapreduce*.
2. Pada pengolahan data yang berukuran besar baik dalam menggunakan *hadoop mapreduce* karena menghasilkan waktu eksekusi yang jauh lebih rendah daripada menggunakan *SQL* dengan ukuran data yang sama. Walaupun dengan penggunaan *hadoop* rata-rata konsumsi *CPU* dan *memory usage* meningkat.
3. Pada implementasi *hadoop mapreduce* semakin banyak jumlah *node*, menghasilkan performa waktu eksekusi yang lebih baik.
4. Pada implementasi *hadoop mapreduce* dengan penggunaan *combiner*, memberikan waktu eksekusi yang semakin rendah.
5. Pada implementasi *hadoop mapreduce* tidak ada ukuran *block* yang pasti untuk memberikan performa yang baik dalam menjalankan aplikasi. Performa yang baik didapatkan dengan melakukan percobaan terhadap jumlah data dan ukuran *block* tertentu.

7. REFERENSI

[1] Coulouris, G., Dollimore, J., Kindberg, T., & Blair, G. 2012. *Distributed Systems: Concepts and Design 5th edition*. Pearson Education.

[2] Holmes, A. 2012. *Hadoop in Practice*. New York: Manning Publications Co.

[3] Perpustakaan Universitas Kristen Petra. *Sejarah Perpustakaan*. URI = http://library.petra.ac.id/index.php?r=site/sejarah_perpustakaan

[4] Potts, A., & Friedel, J. D. 1996. *Java Programming Language Handbook*. Scottsdale: Keith Weiskamp.

[5] Rouse, M. 2013. *Big Data*. URI = http://whatis.techtarget.com/definition/3Vs

[6] Taggart, A. 2011. *How Map and Reduce operations are actually carried out*. URI = http://wiki.apache.org/hadoop/HadoopMapReduce

[7] White, T. 2012. *Hadoop: The Definitive Guide (3rd ed.)*. O'Reilly Media, Inc