Institut
Teknologi
Sepuluh Nopember

TUGAS AKHIR - TE 141599

PENGELOMPOKAN *BIG DOCUMENT* MENGGUNAKAN METODE *K-MEANS* PADA KOMPUTASI TERDISTRIBUSI

Ilham Laeur Hikmat
NRP 2211100109

Dosen Pembimbing
Mochamad Hariadi, ST., M.Sc., Ph.D.
Christyowidiasmoro, ST., MT.

JURUSAN TEKNIK ELEKTRO
Fakultas Teknologi Industri
Institut Teknologi Sepuluh Nopember
Surabaya 2016

ITS
Institut
Teknologi
Sepuluh Nopember

FINAL PROJECT - TE 141599

BIG DOCUMENT CLUSTERING WITH K-MEANS METHOD ON DISTRIBUTED COMPUTING

Ilham Laenur Hikmat
NRP 2211100109

Advisor
Mochamad Hariadi, ST., M.Sc., Ph.D.
Christyowidiasmoro, ST., MT.

Departement of Electrical Enginering
Faculty of Industrial Technology
Sepuluh Nopember Institute of Technology
Surabaya 2016

PENGELOMPOKAN *BIG DOCUMENT* MENGGUNAKAN METODE *K-MEANS* PADA KOMPUTASI TERDISTRIBUSI

TUGAS AKHIR

Diajukan untuk Memenuhi Sebagian Persyaratan
Untuk Memperoleh Gelar Sarjana Teknik
Pada
Bidang Studi Teknik Komputer dan Telematika
Jurusan Teknik Elektro
Institut Teknologi Sepuluh Nopember

Menyetujui:

Dosen Pembimbing I	Dosen Pembimbing II
Mochamad Hariadi, ST., M.Sc., PhD.	Christyowidiasmoro, ST., M.Sc.
NIP. 196912091997031001	NIP. 198301272009121004

SURABAYA
JANUARI, 2016

ABSTRAK

Nama Mahasiswa	:	Ilham Laenur Hikmat
Judul Tugas Akhir	:	Pengelompokan *Big Document* Menggunakan Metode *K-means* pada Komputasi Terdistribusi
Pembimbing	:	1. Mochamad Hariadi, ST., M.Sc., Ph.D. 2. Christyowidiasmoro, ST., MT.

Artikel dan konten berita adalah bentuk dokumen yang laju pertambahannya pesat berkat kemudahan penggunaan dan akses internet. Pengelompokan data (*data clustering*) adalah metode pembelajaran mesin tanpa pengawasan, yang dapat membagi kumpulan data kedalam sub kelompok berdasarkan kemiripan karakteristik data. Dalam penelitian ini akan diimplementasikan pengelompokan data berupa konten berita dengan jumlah besar menggunakan metode *K-Means* pada sistem komputasi terdistribusi. Sistem yang didesain terdiri dari dua subsistem, praposes data yang berfungsi untuk mecari fitur dari teks berita dan subsistem pengelompokan data yang berfungsi untuk membagi kelompok data. Fungsi-fungsi dalam sistem dibuat dengan model program *MapReduce* dan dijalankan pada *cluster* komputer berbasis *Hadoop*. Dari pengujian yang dilakukan diperoleh akurasi hasil pengelompokan lebih dari 83% dengan data yang kategorinya sudah ditentukan. Waktu proses sistem juga mendapat peningkatan 20% dengan memperbanyak jumlah *slave node* pada sistem sejumlah 25%-50%.

Kata Kunci : *big data*, *distributed computation*, *hadoop*, *k-means clustering*, *text mining*

ABSTRACT

Name : Ilham Laenur Hikmat
Title : Big Document Clustering with K-means Method on Distributed Computing
Advisors : 1. Mochamad Hariadi, ST., M.Sc., Ph.D.
2. Christyowidiasmoro, ST., MT.

Articles and news is a document that has rapid growth rate thanks to the ease of use of internet. Data clustering is an unsupervised machine learning method, which can split a set of data into subgroups based on similar characteristics of the data. In this research will be implemented data clustering for a large number of news using K-Means method on a distributed computing system. Designed systems has two subsystems, data preprocess that is used to look for the features of the news and data clustering system that serves to divide the data into groups. The functions in the system created with MapReduce programming model and run on Hadoop based computer cluster. From the tests, the accuracy of the results obtained by this system is over 83% with the data collected from news website with specified category. Faster processing time about 20% can be achieved with increasing the number of slave node 25%-50%.

Keywords : big data, distributed computation, hadoop, k-means clustering, text mining

KATA PENGANTAR

Puji dan syukur kehadirat Allah SWT atas segala limpahan berkah, rahmat, serta hidayah-Nya, penulis dapat menyelesaikan penelitian ini dengan judul **Pengelompokan *Big Document* Menggunakan Metode *K-Means* pada Komputasi Terdistribusi**.

Penelitian ini disusun dalam rangka pemenuhan bidang riset di Jurusan Teknik Elektro ITS, Bidang Studi Teknik Komputer dan Telematika, serta digunakan sebagai persyaratan menyelesaikan pendidikan S1. Penelitian ini dapat terselesaikan tidak lepas dari bantuan berbagai pihak. Oleh karena itu, penulis mengucapkan terima kasih kepada:

1. Keluarga, Ibu, Bapak dan Saudara tercinta yang telah memberikan dorongan spiritual dan material dalam penyelesaian buku penelitian ini.
2. Bapak Dr. Ardyono Priyadi, ST., M.Eng. selaku Ketua Jurusan Teknik Elektro, Fakultas Teknologi Industri, Institut Teknologi Sepuluh Nopember.
3. Secara khusus penulis mengucapkan terima kasih yang sebesar-besarnya kepada Bapak Mochamad Hariadi, ST., M.Sc., Ph.D. dan Bapak Christyowidiasmoro, ST., MT. atas bimbingan selama mengerjakan penelitian.
4. Bapak-ibu dosen pengajar Bidang Studi Teknik Komputer dan Telematika, atas pengajaran, bimbingan, serta perhatian yang diberikan kepada penulis selama ini.
5. Seluruh teman-teman *B201-crew* Laboratorium Bidang Studi Teknik Komputer dan Telematika.

Kesempurnaan hanya milik Allah SWT, untuk itu penulis memohon segenap kritik dan saran yang membangun. Semoga penelitian ini dapat memberikan manfaat bagi kita semua. Amin.

Surabaya, Januari 2016

Penulis

DAFTAR ISI

TABLE OF CONTENT

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR KODE

BAB 1
PENDAHULUAN

Penelitian ini dilatar belakangi oleh berbagai kondisi yang menjadi acuan. Selain itu juga terdapat beberapa permasalahan yang akan dijawab sebagai luaran dari penelitian ini.

1.1 Latar belakang

Akses internet yang semakin mudah tidak hanya berdampak pada jumlah data yang terus bertambah. Kemudahan internet juga mempengaruhi laju pertambahan data dan ragam jenis data. Fenomena ini termasuk ke dalam data besar (*Big Data*). Konten web adalah salah satu informasi yang dapat diperoleh dengan internet dan termasuk dalam kategori data besar. Konten web seperti artikel berita, ulasan produk dan opini politik adalah sebagian dari jenis konten yang bersifat subjektif dan mengandung sentimen penulis terhadap topik tulisannya[1][2]. Nilai sentimen termasuk sebagai informasi implisit yang dapat diperoleh menggunakan metode penggalian teks. Dengan menganalisa seluruh data konten yang terdapat pada internet dapat diperoleh sentimen global terhadap suatu topik yang bisa digunakan sebagai bahan pertimbangan suatu keputusan.

Penggalian teks (*text mining*) adalah proses pengolahan informasi teks yang tidak berstruktur untuk memperoleh informasi implisit dari teks tersebut. Pengelompokan data (*data clustering*) adalah metode pembelajaran mesin tanpa pengawasan (*unsupervised machine learning*) yang membagi sekumpulan data ke dalam sub kelompok berdasarkan kemiripan karakteristik dari setiap data[3]. Dalam pengelompokan data teks, tingkat kesamaan teks digunakan sebagai acuan untuk menentukan keanggotaan dari kelompok[4]. Data teks dalam kelompok yang sama memiliki tingkat kemiripan yang lebih tinggi dibandingkan dengan data teks pada kelompok yang berbeda. Pengelompokan data berfungsi untuk memperoleh pemahaman baru dari sekumpulan data berdasarkan kelompok-kelompoknya. Kelompok-kelompok data tersebut dapat juga digunakan sebagai tahap awal untuk proses penggalian data selanjutnya.

Metode penggalian teks yang umum memiliki keterbatasan terhadap besar data yang dapat diolah. Keterbatasan ini bersifat linier terhadap kemampuan mesin komputasinya. *Hadoop* sebagai *framework* komputasi dan penyimpanan data terdistribusi adalah salah satu solusi dalam mengatasi pengolahan data besar[5]. Kemampuan komputasi pada sistem terdistribusi berbanding lurus terhadap jumlah mesin yang terpasang pada sistem[6]. Dalam tugas akhir ini akan diterapkan metode pengelompokan data berbasis *K-means* pada model program komputer terdistribusi. Metode dengan model program ini diharapkan dapat melakukan pengelompokan data pada kelompok data dengan jumlah yang besar.

1.2 Permasalahan

Jumlah konten web yang terus bertambah menjadikan metode pengelompokan dokumen yang umum tidak lagi efektif. Hal ini dikarenakan jumlah data yang dapat diproses dengan metode tersebut terbatas pada kemampuan komputasi mesinnya. Peningkatan kemampuan komputasi mesin dapat dilakukan dengan mengganti komponen komputer dengan komponen yang memiliki spesifikasi lebih tinggi. Tetapi hal ini memerlukan biaya tinggi dan spesifikasi maksimal komponen masih terbatas pada teknologi yang ada.

1.3 Tujuan

Penelitian ini dilakukan dengan tujuan untuk menganalisa kinerja dari sistem komputasi terdistribusi yang digunakan untuk memproses pengelompokan *big document* berbasis *K-means*. Dari hasil penelitian ini, pengelompokan *big document* diharapkan tidak lagi memerlukan satu mesin komputasi dengan spesifikasi yang tinggi, tetapi beberapa mesin komputasi biasa yang tersambung dalam satu sistem komputasi terdistribusi.

1.4 Batasan masalah

Batasan masalah dalam Tugas Akhir ini adalah sistem komputasi terdistribusi yang digunakan merupakan sistem skala kecil dengan sembilan *node* pada layanan *cloud IaaS* (*Infrastructure as a Service*).

1.5 Sistematika Penulisan

Laporan penelitian Tugas akhir ini tersusun dalam sistematika dan terstruktur sehingga mudah dipahami dan dipelajari oleh pembaca maupun seseorang yang ingin melanjutkan penelitian ini. Alur sistematika penulisan laporan penelitian ini yaitu :

1. BAB I Pendahuluan
   Bab ini berisi uraian tentang latar belakang permasalahan, penegasan dan alasan pemilihan judul, sistematika laporan dan tujuan penelitian.
2. BAB II Dasar Teori
   Pada bab ini berisi tentang uraian secara sistematis teori-teori yang berhubungan dengan permasalahan yang dibahas pada penelitian ini. Teori-teori ini digunakan sebagai dasar dalam penelitian, yaitu informasi terkait analisa teks, algoritma K-Means, dan teori-teori penunjang lainya.
3. BAB III Perancangan Sistem dan Impementasi
   Bab ini berisi tentang penjelasan-penjelasan terkait sistem yang akan dibuat. Guna mendukung itu digunakanlah blok diagram atau *work flow* agar sistem yang akan dibuat dapat terlihat dan mudah dibaca untuk implentasi pada pelaksanaan tugas akhir.
4. BAB IV Pengujian dan Analisa
   Bab ini menjelaskan tentang pengujian yang dilakukan terhadap sistem dalam penelitian ini. Hasil pengujian juga dianalisa dalam bab ini guna mengevaluasi kinerja sistem.
5. BAB V Penutup
   Bab ini merupakan penutup yang berisi kesimpulan yang diambil dari penelitian dan pengujian yang telah dilakukan. Saran dan kritik yang membangun untuk pengembangkan lebih lanjut juga dituliskan pada bab ini.

1.6 Relevansi

Penelitian mengenai pengelompokan teks merupakan salah satu bidang penelitian dalam topik penggalian data. Metode yang sudah diteliti terfokus pada akurasi hasil dan efektifitas metode[3][7].

Penelitian yang terfokus pada data besar masih sedikit. Penelitian yang sudah ada dibidang tersebut memfokuskan solusinya pada manajemen pengaksesan data yang membebankan proses pada pembacaan dan penulisan data pada memori. Dari penelitian ini dihasilkan sebuah metode pengelompokan teks yang dapat bekerja dengan jumlah data yang besar dengan membagi beban kerja pada banyak mesin komputasi.

BAB 2
TINJAUAN PUSTAKA

Demi mendukung penelitian ini, dibutuhkan beberapa teori penunjang sebagai bahan acuan dan refrensi. Dengan demikian penelitian ini menjadi lebih terarah.

2.1 Analisa dan Representasi Teks

Teks adalah data dengan format yang tidak terstruktur. Beberapa data teks yang dibandingan satu sama lain bisa memiliki perbedaan jumlah kata yang dan ragam kata yang digunakan. Oleh sebab itu data teks perlu direpresentasikan kedalam bentuk yang dapat dianalisa. Fitur data dari teks dibuat berdasarkan pada kata-kata yang terdapat pada teks tersebut, sehingga fiturnya berupa vektor kata yang memiliki label dan nilai bobot. Label dari setiap anggota vektor adalah kata unik dalam teks dan nilai bobotnya ditentukan dengan *Term Frequency - Inverese Document Frequency* (*TF-IDF*)[4].

Term Frequency adalah frekuensi kemunculan suatu *term* atau istilah dalam teks. *TF* dapat juga disebut sebagai bobot lokal, karena nilainya hanya tergantung pada *term* di dalam teks itu sendiri. Teks sebagai data yang tidak terstruktur memiliki panjang data yang beragam. Untuk menjaga relevansi nilainya diperlukan normalisasi. Normalisasi tersebut dilakukan dengan membagi nilai *TF* terhadap jumlah semua *term* di dalam teks.

Document Frequency (*DF*) adalah banyaknya dokumen dimana suatu *term* muncul. *Inverse Document Frequency* (*IDF*) berfungsi untuk memperbesar bobot kata yang unik yang tidak terdapat pada dokumen lain dan memperkecil bobot kata yang umum yang hampir terdapat di semua dokumen. *IDF* merupakan bobot global dari kata, karena nilai ini dipengaruhi oleh semua dokumen yang akan diproses.

$$idf(t) = 1 + \log \frac{n}{df(t)} \tag{2.1}$$

$$tfidf(d,t) = tf(d,t) \times idf(t)) \tag{2.2}$$

Pada persamaan 2.2, $tf(d,t)$ adalah *Term Frequency* kata t pada teks d. $df(t)$ adalah banyak teks yang terdapat kata t didalamnya. n adalah jumlah total teks yang digunakan.

Representasi dari teks sangat bergantung dari kata-kata didalamnya. Kata yang dapat dijadikan fitur sebuah kalimat adalah kata yang memiliki makna ketika berdiri sendiri. Kata-kata yang tidak dapat berdiri sendiri diantaranya adalah kata-kata penunjuk tempat (di, ke, dari). Kata-kata seperti ini disebut sebagai *stop word*. *Stop word* yang digunakan dalam penelitian ini adalah *stop word* yang dibuat oleh Tala[8]. *Stop word* tersebut dibuat dengan melakukan analisa frekuensi kata dalam penelitiannya dan diambil berdasarkan frekuensi kata tertinggi. *Stop word* dapat dicari dengan mencari kata yang memiliki kemunculan terbanyak dalam keseluruhan dokumen. *Stop word* yang digunakan dalam penelitian Tala[8] dibangun berdasarkan dokumen berita *online* kompas dari Januari 2001 sampai Desember 2001. 10 kata dengan kemunculan tertinggi dari dokumen berita tersebut dapat dilihat pada tabel 2.1.

Tabel 2.1: 10 kata dengan jumlah kemunculan tertinggi pada berita *online* kompas dari Januari 2001 sampai Desember 2001[8]

Kata	Jumlah Kemunculan
yang	55971
dan	41286
itu	24768
tidak	18723
dengan	18281
dari	17632
untuk	16886
dalam	15681
ini	14707
akan	12433

Imbuhan adalah bentuk modifikasi dari satu kata untuk mendapatkan makna baru tetapi tetap merujuk pada kata dasarnya. Dalam menghitungan *TF-IDF*, setiap kata yang memilki bentuk yang berbeda akan diperlakukan sebagai fitur yang berbeda. *Stemming* adalah metode yang digunakan untuk merubah kata berimbuhan menjadi kata dasarnya. Dengan mengubah kata berimbuhan menjadi kata dasarnya, hasil vektor *TF-IDF* menjadi lebih valid [9][8]. Metode yang digunakan dalam penelitian ini adalah metode *porter stemmer*. Pada metode ini, setiap kata akan dicek imbuhannya berdasarkan morfologi penyusun kata 2.3, dan dihapus sampai tersisa kata yang dianggap non-imbuhan(*root*).

Tabel 2.2: Daftar imbuhan dan tipenya

Tipe imbuhan	**Imbuhan**
prefix	ber-, per-, ter-, di-, ke-, meng-, peng-
particle	-kah, -lah, -tah, -pun
possesive pronoun	-ku, -mu, -nya
derivational suffix	-i, -kan, -an

Daftar imbuhan dan tipenya dapat dilhat pada tabel 2.2. Pada metode ini terdapat beberapa kata yang tidak berubah menjadi kata dasar yang seharusnya, tetapi hasil tersebut masih dianggap valid karena semua imbuhan yang dikenakan pada kata tersebut akan diubah menjadi kata yang sama.

$$word = prefix1 + prefix2 + root + suffix + possesive + particle \tag{2.3}$$

2.2 Pengelompokan teks berbasis *K-Means*

K-means adalah algoritma yang umum digunakan dalam pengelompokan data[3][7]. Algoritam *K-means* dapat dijelaskan sebagai berikut:

1. Algoritam ini pertama akan menentukan titik tengah untuk

setiap kelompok dengan mengambil data secara acak dari keseluruhan data.
2. Setelah diperoleh titik tengah semua kelompok, setiap data akan dihitung kemiripannya dengan semua titik tengah kelompok. Data akan dianggap sebagai anggota kelompok yang titik tengahnya memiliki kemiripan tertinggi.
3. Titik tengah baru kemudian dihitung berdasarkan rata-rata nilai anggotanya.
4. Poin 2 dan 3 terus diulang sampai jumlah iterasi mencapai batas maksimal atau tidak lagi terjadi perubahan pada titik tengah setiap kelompok.

Algoritma ini memerlukan masukan berupa jumlah kelompok yang akan dicari, maksimal iterasi yang boleh dilakukan dan minimal nilai kemiripan perubahan nilai tengah kelompok sebagai penanda bahwa data kelompok sudah stabil.

Cosine Similarity adalah metode perhitungan yang digunakan untuk mengukur kemiripan dua teks. Metode ini mengalikan dua vektor menggunakan *dot product* dan membaginya dengan hasil kali *magnitude* dari kedua vektor. Hasil perhitungan ini adalah nilai antara nol dan satu, dimana nilai satu artinya dua teks tersebut sangat mirip.

$$cosineSimilarity(d_i, d_j) = \frac{\sum_{k=1}^{n} W_{ik} W_{jk}}{\sqrt{\sum_{k=1}^{n} w_{ik}^2} \sqrt{\sum_{k=1}^{n} w_{jk}^2}} \tag{2.4}$$

2.3 *Hadoop*

Hadoop adalah sebuah *platform* yang memiliki kemampuan untuk melakukan komputasi dan penyimpanan data terdistribusi. Komputasi terdistribusi adalah metode komputasi yang dilakukan untuk membagi sebuah proses besar menjadi banyak proses kecil yang proses tersebut kemudian dikerjakan pada mesin komputasi yang berbeda. *Hadoop* adalah sebuah ekosistem *BigData* yang didalamnya terdapat komponen-komponen yang menunjang untuk pen-

golahan data besar, diantaranya adalah *Hadoop Distributed File System* (*HDFS*) dan *MapReduce*[5].

2.3.1 *HDFS*

Hadoop Distributed File System (*HDFS*) merupakan komponen file sistem dari ekosistem *Hadoop*. *HDFS* menyimpan data dan metadata secara terpisah. Metadata disimpan dalam server khusus yang disebut sebagai *NameNode*, sedangkan datanya disimpan dalam server lain yang disebut *DataNode*. Server *NameNode* dan *DataNode* terhubung dalam satu jaringan dan saling berkomunikasi dengan protokol berbasis *TCP*. *HDFS* tidak memiliki sistem proteksi data seperti *RAID*, dalam menjaga keutuhan datanya *HDFS* mereplikasi data ke beberapa *DataNode*.

Data yang tersimpan didalam hadoop akan dibagi berdasarkan *block size*. Selain mempermudah dalam replikasi, juga membantu untuk proses komputasi terdistribusi. Secara default, *HDFS* memiliki nilai *block size* sebesar 128MB. Artinya setiap data akan dibagi menjadi beberapa bagian dengan besar maksimal setiap bagian adalah 128MB. Jika kita menyimpan data sebesar 1000MB makan data tersebut akan dibagi kedalam 8 blok, 7 blok sebesar 128MB dan 1 blok sebesar 104MB. Setiap blok akan disimpan ditempat yang secara acak ditentukan oleh HDSF *NameNode*. Komputasi distribusi adalah komputasi yang dilakukan terhadap potongan data yang tersimpan di memori lokalnya.

Pembagian data pada *HDFS* hanya berbasiskan pada ukuran data saja, sehingga konten yang terpotong ditengah data menjadi tidak valid. Hal ini diatasi saat pembacaan data nantinya. Pembacaan data saat melakukan komputasi akan mengecek posisi pembagian data. Jika pembagian berada di awal artinya data pertama adalah valid, pembacaan data dilakukan normal sampai data akhir. Jika data akhir tidak ditemukan penutup baris maka dilakukan pembacaan *remote* pada potongan data selanjutnya sampai ditemukan penutup baris atau penutup data. Jika data dilakukan pada potongan data selanjutnya, baris data pertama akan dihiraukan karena dianggap tidak valid dan dilanjutkan langsung pada baris data selanjutnya.

2.3.2 *MapReduce*

Hadoop MapReduce (*MR*) adalah sebuah model program yang dapat diimplementasikan untuk memproses data set yang besar. *MapReduce* disebut sebagai komputasi terdistribusi karena *MapReduce* akan mengirim programnya ke semua *node*, menjalankannya untuk memproses data yang tersimpan lokal di setiap *node* dan mengumpulkan hasil akhir prosesnya sebagai hasil akhir komputasi. *MapReduce* memerlukan dua *service* untuk bekerja. *JobTracker* dan *Task Tracker*. *JobTracker* adalah service yang ber-fungsi untuk memonitor status pekerjaan yang dikirmkan di setiap *slave node*, sedangkan *TaskTracker* adalah *service* yang berada pada *slave node* untuk memantau dan melaporkan status pekerjaan yang dikerjakan oleh *node* tersebut.

MapReduce terdiri dari dua fungsi, fungsi *mapper* dan *reducer*. Model program *MapReduce* menggunakan masukan berisi kumpulan pasangan *key-value* yang dipisahkan dengan karakter tab (\t), masukan tersebut diproses oleh *mapper*, dibagi ke setiap *node* dalam *cluster* dan menghasilkan nilai *key-value* lagi. Nilai tersebut kemudian dikumpulkan oleh fungsi *reducer* dan disajikan sebagai hasil akhir proses *MapReduce*. Selain fungsi utama *mapper()* dan *reducer()*, dalam setiap *Class* terdapat juga fungsi *setup()* dan fungsi *cleanup()*. Fungsi *setup()* berfungsi untuk menjalankan perintah sebelum *mapper* membaca data. Dalam *setup()* biasanya dilakukan inisialisasi variabel atau pembacaan parameter yang dikirim oleh program *client*. Fungsi *cleanup()* adalah sebaliknya, yaitu fungsi yang dijalankan ketika *mapper* atau *reducer* selesai membaca semua data yang tersedia. Salah satu jenis komputasi yang efektif menggunakan model program *MapReduce* adalah komputasi penghitung.

2.3.3 *YARN* dan *MapReduce* 2.0 (*MRv2*)

YARN adalah pengembangan dari *MapReduce* 1.0 (*MRv1*) yang membagi dua fungsi service JobTracker, penjadwalan/pemantauan dari pekerjaan dan pengelolaan resource, menjadi dua service yang berbeda. Dengan pembagian ini diperoleh pengelolaan resource yang bersifat global (*ResourceManager*) dan pengelolaan aplikasi per-pekerjaan (*ApplicationMaster*).

Setiap pekerjaan *MapReduce* (*MR*) yang dibuat oleh client akan dibuatkan satu *ApplicationMaster* (*AM*) oleh *ResourceManager* (*RM*). Kemudian *AM* akan menghitung pembagian pekerjaan untuk setiap node dan meminta *RM* untuk mengalokasikan *resource* di setiap *node*. Jika *resource* setiap *node* menyanggupi, maka *resource* setiap *node* akan dialokasikan sebagai *Container* untuk digunakan oleh *AM* melakukan pekerjaan *MR*.

BAB 3
DESAIN DAN IMPLEMENTASI SISTEM

Desain sistem dalam tugas akhir ini dibagi menjadi dua sistem besar yaitu desain sistem praproses data dan desain sistem pengelompokan data.

3.1 Desain Sistem Praproses Data

Sistem praproses data dibuat untuk mempersiapkan data untuk diproses oleh sistem pengelompokan. Pada sistem ini akan diubah data yang berbentuk teks kedalam bentuk vektor *TF-IDF*. Fungsi-fungsi dalam sistem ini digambarkan pada bagan alur 3.1. Sebelum melakukan perhitungan nilai *TF-IDF*, dilakukan penyaringan data terlebih dahulu. Penyaringan yang dilakukan adalah penyaringan non-konten, penyaringan *stop word* dan *stemming*.

Pada penyaringan non-konten akan dihilangkan kata-kata yang tidak berhubungan dengan konten teks. Kata-kata tersebut berasal dari menu navigasi atau struktur *HTML* lainnya yang diperoleh saat pengambilan data. Kata-kata tersebut bersifat mengganggu yang dapat mengurangi keakurasian bobot vektor *TF-IDF* sebagai representasi dari teks. Penyaringan selanjutnya adalah penyaringan dari *stop word*. *Stop word* dalam teks akan menambah vektor yang tidak diperlukan pada vektor *TF-IDF*. Bobot kata ini akan mengganggu dan menurunkan akurasi hasil perhitungan tingkat kemiripan antar teks nantinya. Penyaringan yang dilakukan selanjutnya adalah *stemming*. *Stemming* akan menjadikan kata-kata dengan kata dasar yang sama sebagai kata yang saling berkaitan. Penyaringan-penyaringan ini dilakukan untuk meningkatkan keakurasian bobot nilai vektor *TF-IDF* sebagai representasi suatu teks. Nilai vektor *TF-IDF* sangat mempengaruhi akurasi dalam perhitungan tingkat kemiripan antar teks.

3.2 Desain Sistem Pengelompokan Data

Sistem pengelompokan data yang digunakan adalah pengelompokan data berbasis K-means. Pengelompokan data dengan metode

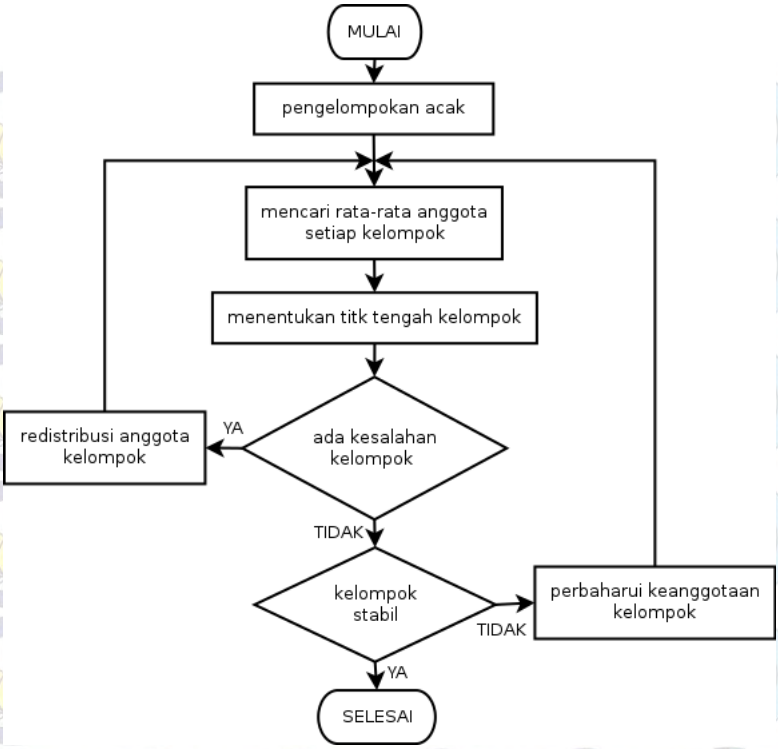
Gambar 3.1: Flowchart praproses data

ini memerlukan banyak iterasi pada data. Untuk data yang besar, iterasi terhadap seluruh data memerlukan usaha yang besar, sehingga diperlukan penyesuaian dari metode aslinya.

Tahapan pertama yang perlu dilakukan oleh sistem adalah secara acak membagi data kedalam jumlah kelompok yang sudah ditentukan. Pengelompokan secara acak ini dimaksudkan untuk mencari nilai tengah awal yang kemudian akan dievaluasi oleh sistem.

Untuk memperoleh nilai tengah kelompok, dilakukan rata-rata semua nilai vektor *TFIDF* anggotanya. Dengan data yang besar akan diperoleh kata unik yang besar juga. Banyak ragam kata ini menyebabkan semakin besar pula ukuran vektor dari nilai tengahnya. Hal ini akan mempengaruhi lama proses pengukuran kemiripan data dan mengurangi kesensitifan hasilnya. Untuk mengatasi hal tersebut, nilai tengah kelompok akan menggunakan nilai vektor anggota kelompok yang memiliki kemiripan tertinggi dengan rata-rata vektor kelompok tersebut.

Beberapa kemungkinan kesalahan metode yang mungkin terjadi adalah titik tengah yang memiliki nilai kemiripan tinggi dengan



Gambar 3.2: Flowchart pengelompokan data

titik tengah kelompok lain. Hal ini akan menjadikan kedua kelompok tersebut memiliki anggota dengan kriteria yang serupa. Untuk mengatasi hal ini, kemiripan setiap titik tengah kelompok akan dihitung, dan jika melebihi batas kemiripan maksimal yang sudah ditentukan, kedua kelompok tersebut akan dijadikan satu kelompok, kemudian kelompok data baru akan dibuat dengan mencari nilai anggota setiap kelompok yang kemiripannya kurang dari batas kemiripan minimal yang sudah ditentukan terhadap titik tengah kelompoknya. Kesalahan ini bisa disebabkan oleh pembagian kelompok awal yang dilakukan secara acak yang menempatkan banyak data dengan kemiripan tinggi di kelompok yang berbeda.

Kesalahan lain yang mungkin terjadi adalah terciptanya kelompok yang tidak memiliki anggota. Situasi ini dapat terjadi ketika kemiripan semua anggotanya terhadap titik tengah kelompok lain lebih tinggi dibandingkan kemiripan dengan titik tengah kelompoknya sendiri. Untuk menghadapi situasi ini, akan dicari kelompok dengan jumlah anggota terbesar yang kemudian dibagi dua berdasarkan tingkat kemiripan terhadap titik tengah kelompoknya untuk dijadikan kelompok data baru.

Setelah kelompok-kelompok terbentuk dan tidak terdapat masalah. Semua data teks kemudian mengevaluasi dirinya dengan mengukur tingkat kemiripan terhadap titik tengah semua kelompok. Data tersebut akan berpindah ke kelompok dengan kemiripan titik tengah tertinggi atau akan tetap jika kemiripan titik tengah kelompok sebelumnya lebih tinggi dibandingkan kelompok lain.

Proses akan terus berulang sampai dicapai kondisi untuk menghentikan perulangan. Kondisi tersebut dicapai ketika keanggotaan kelompok sudah stabil, yang ditandakan dengan tidak ada lagi perubahan titik tengah semua kelompok. Kondisi lain adalah lama proses yang sudah mencapai iterasi maksimal yang sudah ditentukan.

3.3 Alur Implementasi Sistem

Alur kerja dalam pengerjaan tugas akhir ini terbagi mejadi empat tahapan proses. Tahap tersebut meliputi pengambilan data, pemasangan dan konfigurasi *Hadoop* pada komputer *cluster*, implementasi praproses data dan implementasi pengelompokan data. Implementasi praproses data dan pengelompokan data dilakukan menggunakan model program *MapReduce*.

Tabel 3.1: Jumlah dan ukuran data yang akan dikelompokan

Kategori	Jumlah berita	Besar data (MB)
Wisata	36937	285
Sepak bola	28100	283
Kuliner	31299	277
Otomotif	49104	451

3.3.1 Pengambilan Data

Data yang digunakan pada penelitian ini adalah data hasil *crawling* pada sebuah situs berita nasional detik.com. *Crawling* dilakukan secara terpisah terhadap berita berdasarkan kategorinya. Kategori berita yang digunakan pada penelitian ini adalah wisata, kuliner, otomotif dan sepak bola. Pengambilan data berdasarkan kategori ini dilakukan untuk mempermudah proses validasi pada pengujian. Hasil dari *crawling* adalah satu file perkategori dimana setiap barisnya mengandung *url* dan konten berita yang dipisahkan dengan karakter tab (\t). Konten berita yang disimpan adalah seluruh *string* hasil *HTTP Request* pada url yang diberikan, kemudian dihapus seluruh penanda *HTML*, *JavaScript* dan *CSS*. Total jumlah dan ukuran data dapat dilihat pada tabel 3.1. Dari tabel tersebut terlihat bahwa setiap kategori memiliki jumlah berita yang berbeda dengan data berkategori otomotif sebagai kategori dengan data terbanyak.

Tabel 3.2: Konfigurasi *node* yang digunakan pada komputasi awan

Alamat *IP*	***Virtual Core* (2.4Ghz)**	***RAM* (GB)**	**Memori (GB)**	***Hostname***
192.168.100.131	1	2	20	loka1
192.168.100.136	1	2	20	loka2
192.168.100.137	1	2	20	loka3
192.168.100.138	1	2	20	loka4
192.168.100.139	1	2	20	loka5
192.168.100.140	1	2	20	loka6
192.168.100.141	1	2	20	loka7
192.168.100.142	2	4	40	loka8
192.168.100.143	2	4	40	loka9

3.3.2 Pemasangan dan Konfigurasi Hadoop pada Komputer Cluster

Pada penelitian ini digunakan distribusi *Hadoop Cloudera*, yaitu sebuah paket *Hadoop* yang didalamnya terdapat komponen-komponen *Hadoop* yang sudah dikonfigurasi dan diuji kecocokan antar komponennya.

Tabel 3.3: Konfigurasi peran *node*

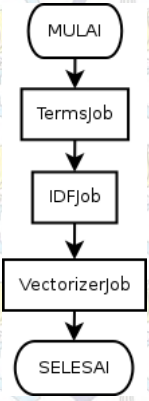
Hostname	**Peran**	**Fungsi**
loka1	Cloudera Manager Server	Monitor dan konfigurasi cluster
loka2-7	DataNode	Resource penyimpanan HDFS
	NodeManager	Resource komputasi untuk YARN
loka8	Secondary NameNode	Backup dari NameNode
	JobHistory	Penyimpanan record untuk MapReduce Job
loka9	NameNode	Server HDFS
	HueServer	Web interface untuk monitori file HDFS dan MapReduce Job
	ResourceManager	Server YARN

Hadoop akan dipasang pada sebuah komputer *cluster* dengan jumlah *node* sembilan. Setiap *node* pada *cluster* dibuat didalam sebuah infrastruktrur komputasi awan dengan konfigurasi yang terdapat pada tabel 3.2.

Tujuh dari sembilan *node* yang digunakan memiliki spesifikasi yang sama. Enam *node* ini akan digunakan sebagai penyedia *resource* untuk penyimpanan dan komputasi terdistribusi. Dua *node* lainnya memiliki spesifikasi yang lebih tinggi akan digunakan sebagai master-master dari layanan yang akan digunankan. Secara lengkap peran setiap *node* dijelaskan pada tabel 3.3.

3.3.3 Implementasi Sistem Praproses Data

Dalam sistem praproses data, semua fungsi didalamnya diimplementasikan ke dalam model program *MapReduce*. Fungsi-fungsi dalam sistem praproses saling terkait. Oleh karena itu fungsi-fungsi kemudian dibagi berdasarkan area prosesnya, apakah hanya perkata, perteks atau semua data. *Job* dalam *MapReduce* adalah proses satu iterasi semua data untuk menghasilkan satu jenis keluaran. Fungsi-fungsi dalam sistem kemudian dibagi berdasarkan jenis keluaran dan area prosesnya menjadi *Job* yang berbeda yang digambarkan pada gambar 3.3.



Gambar 3.3: Flowchart praproses data dalam model program MapReduce

1. *TermsJob*

   TermsJob adalah *Job* dengan fungsi proses perkata untuk menghasilkan *Term Frequency*. *TermsJob* merupakan fungsi dengan area proses perteks yang data masukannya berasal dari data mentah yang akan dikelompokan. Fungsi-fungsi penyaringan dilakukan juga di *Job* ini sebelum proses penghitungan kata. Semua proses dalam *Job* ini dilakukan pada fungsi *mapper*, yaitu fungsi yang dijalankan pada setiap *node*. Fungsi *reducer* pada *Job* ini hanya meneruskan keluaran dari mapper kedalam file.

```
Inisialisasi HashMap HM
Inisialisasi counter
Array Stop_word = daftar stop word

Index_awal = index awal konten dari teks
Index_akhir = index akhir konten dari teks
teks = teks.substring(Index_awal, Index_akhir)

Untuk setiap kata dalam teks :
  Jika ada Stop_word[kata] :
    lanjutkan ke kata selanjutnya

  kata = Stem(kata)

  Jika tidak terdapat HM.get(kata) :
    HM.put(kata,1)
  Jika tidak :
    HM.put(kata, HM.get(kata) + 1

  counter = counter + 1

Untuk setiap kata dalam HM:
  HM.put(kata, MH.get(kata)/counter)

write {url:HM.toString()}
```

Kode 3.1: Algoritma mapper pada *TermsJob*

```
Masukan:
http://food.detik.com/read
    /2015/12/03/064207/3086494/297/berani-coba-ramen-
    seberat-3-kg-dari-resto-ramen-ini  berani coba
    ramen seberat 3 kg dari resto ramen ini

    import url http   detik net id food css detik frame
     css m 0704        import url http   detik net id
    food css detikfood style css m 0704        import
    url http   detik net id food css ... ramen
    merupakan makanan populer di jepang  walau bukan
    varian baru  ramen di gerai ini disebut-sebut
    sebagai ramen paling konyol  apa sebabnya nama
    gerai ini ... karir        kotak pos        info
    iklan        disclaimer follow us

Penyaringan non-konten:
```

```
5  http://food.detik.com/read
       /2015/12/03/064207/3086494/297/berani-coba-ramen-
       seberat-3-kg-dari-resto-ramen-ini  ramen merupakan
       makanan populer di jepang  walau bukan varian baru
        ramen di gerai ini disebut-sebut sebagai ramen
       paling konyol  apa sebabnya nama gerai ini ...
6
7  Penyaringan stop word:
8  http://food.detik.com/read
       /2015/12/03/064207/3086494/297/berani-coba-ramen-
       seberat-3-kg-dari-resto-ramen-ini  ramen makanan
       populer jepang varian ramen gerai ramen konyol nama
        gerai ...
9
10 Penyaringan imbuhan:
11 http://food.detik.com/read
       /2015/12/03/064207/3086494/297/berani-coba-ramen-
       seberat-3-kg-dari-resto-ramen-ini  ramen makan
       populer jepang varian ramen gerai ramen konyol nama
        gerai ...
12
13 Keluaran:
14 http://food.detik.com/read
       /2015/12/03/064207/3086494/297/berani-coba-ramen-
       seberat-3-kg-dari-resto-ramen-ini  ramen:=0.12
       _makan:=0.01_populer:=0.0034_jepang:=0.021_varian
       :=0.0042_gerai:=0.065_konyol:=0.31_nama:=0.0052 ...
```

Kode 3.2: Format data masukan dan keluaran *TermsJob*

Proses penyaringan diurutkan berdasarkan area prosesnya, yaitu penyaringan non-konten, angka dan simbol, *stop word* kemudian penyaringan imbuhan. Penyaringan konten dilakukan dengan pencocokan teks dengan sebelumnya mengamati data untuk mendapatkan pola non-konten dari teks. Konten teks biasanya terletak ditengah teks, pada bagian awal teks biasanya terdiri dari kata-kata yang berasal dari fungsi navigasi dan *header* dari halaman web. Sedangkan bagian akhir teks adalah kata-kata yang berasal dari bagian *footer* halaman web. Penyaringan *stop word* dilakukan ketika setiap kata sudah dipisahkan dan akan dihitung. Kata akan dicek apakah terdaftar dalam daftar kata *stop word*. Kata akan ditolak jika kata terdaftar dalam daftar *stop word* dan kata selanjutnya akan diproses. Jika kata tersebut tidak termasuk dalam

stop word, kata kemudian dicek komposisi imbuhannya. Jika imbuhan ditemukan dalam kata, imbuhan tersebut kemudian dihilangkan.

Proses penghitungan frekuensi kata dilakukan dengan menghitung banyak suatu kata digunakan dalam teks tersebut. Setelah semua kata terhitung, dilakukan normalisasi pada setiap kata unik dengan membagi nilai frekuensi kata unik tersebut dengan jumlah keseluruhan kata dalam teks. Semua proses pada *Job* ini dilakukan pada fungsi *mapper* sehingga fungsi *reducer* hanya meneruskan data hasil *mapper* untuk disimpan kedalam file. Bentuk-bentuk perubahan data pada *Job* ini dicontohkan pada data 3.2.

2. *IDFJob*

   IDFJob adalah *Job* selanjutnya yang dibuat untuk menghitung bobot *IDF* setiap kata. Perhitungan *IDF* termasuk dalam proses dengan area proses semua data. Setiap *Job* hanya dapat menggunkan satu jenis masukan dan satu keluaran, sehinga fungsi perhitungan *IDF* tidak dapat digabungkan dengan fungsi penghitungan *TF*. *Job* ini menggunakan masukan yang berasal dari hasil keluaran `TermsJob`.

```
1 Untuk setiap kata dalam teks :
2   write {kata:1}
```

Kode 3.3: Algoritma mapper pada *IDFJob*

```
1 Inisialisasi Array nilai_arr = setiap nilai untuk kata
      yang sama
2 Inisialisasi total_teks = total dokumen pada dataset
3 Inisialisasi total_nilai
4
5 Untuk setiap nilai pada nilai_arr:
6   total_nilai = total_nilai + nilai
7
8 idf = 1 + log(total_teks/total_nilai)
9 write {kata:idf}
```

Kode 3.4: Algoritma reducer pada *IDFJob*

```
Masukan:
http://food.detik.com/read
    /2015/12/03/064207/3086494/297/berani-coba-ramen-
    seberat-3-kg-dari-resto-ramen-ini  ramen:=0.12
    _makan:=0.01_populer:=0.0034_jepang:=0.021_varian
    :=0.0042_gerai:=0.065_konyol:=0.31_nama:=0.0052 ...
...

Keluaran mapper:
ramen 1
makan 1
makan 1
populer 1
makan 1
...

Keluaran:
ramen 120.0
makan 53.55
populer 102.56
...
```

Kode 3.5: Format data masukan dan keluaran *IDFJob*

Untuk mendapatkan nilai IDF dari suatu kata, dua hal yang harus diketahui adalah jumlah teks yang diproses dan jumlah teks yang memiliki kata tersebut didalamnya. Pada proses *mapper*, setelah data *TF* dipisahkan perkata, dan dikirim dengan reducer dengan *key:value* kata:1. Setelah semua hasil keluaran *mapper* pada kata yang sama selesai dibaca, dilakukan perhitungan *IDF* dengan rumus persamaan 2.2 dan menuliskannya pada file dengan format data seperti pada data 3.5.

3. *VectorizerJob*

VectorizerJob adalah *Job* yang berfungsi untuk mengalikan nilai *TF* dengan nilai *IDF* suatu kata. *Job* ini menggunakan masukan dari hasil *TermsJob* dan *IDFJob*. *Job* hanya bisa menggunakan satu sumber data sebagai masukannya, dilihat dari ukuran data dan fungsinya, keluaran *TermsJob* digunakan sebagai masukan *Job* ini, sedangkan keluaran *IDFJob* dibaca secara *remote* oleh program *client* dan dimasukan kedalam se-

buah variable dengan tipe data *string*. Data *IDF* ini kemudian dikirimkan sebagai data statis bersamaan dengan pengiriman *Job* ke *ResourceManager* untuk disebarkan ke *slave node*.

```
1  Inisialisasi HM
2  Array IDF = nilai idf semua kata
3
4  Untuk setiap pasangan kata dan nilai TF dalam teks :
5    IDF_el = IDF[kata]
6    HM.put(kata, TF*iDF_el)
7
8  write {url:HM.toString()}
```

Kode 3.6: Algoritma mapper pada *VectorizerJob*

```
1  Masukan:
2  http://food.detik.com/read
       /2015/12/03/064207/3086494/297/berani-coba-ramen-
       seberat-3-kg-dari-resto-ramen-ini  ramen:=0.12
       _makan:=0.01_populer:=0.0034_jepang:=0.021_varian
       :=0.0042_gerai:=0.065_konyol:=0.31_nama:=0.0052 ...
3  ...
4
5  Keluaran:
6  http://food.detik.com/read
       /2015/12/03/064207/3086494/297/berani-coba-ramen-
       seberat-3-kg-dari-resto-ramen-ini  ramen:=302.2
       _makan:=14.12_populer:=56.21_jepang:=242.65_varian
       :=4.23_gerai:=53.12_konyol:=32.78_nama:=21.67 ...
7  ...
```

Kode 3.7: Format data masukan dan keluaran *VectorizerJob*

VecrorizerJob memiliki fungsi sebagai penguat nilai *TF* yang area prosesnya perteks, sehingga beban proses *Job* ini berada pada fungsi *mapper*. Data *IDF* yang dikirimkan dibaca pada fungsi *setup()*, sehingga cukup sekali dilakukan oleh satu *node*. Data *IDF* yang sudah dibaca disimpan ke dalam *HashMap* dengan *key* berisi kata dan *value* berisi nilai *IDF*. Untuk setiap teks yang dibaca, teks akan dipecah menjadi kumpulan kata dengan nilai *TF*. Nilai *TFIDF* didaptkan dengan mengalikan nilai *TF* kata tersebut dengan nilai *IDF* yang diper-

oleh dari *HashMap*. Kata dan nilai tersebut disimpan dengan format yang sama dengan format keluaran *TermsJob* dan dikumpulkan pada satu data bertipe *string*. Setelah semua kata dalam teks dibaca, data keluaran dibentuk lagi dengan format sama seperti data masukan dan kemudian dikirim ke *reducer*. *Reducer* dalam *Job* ini hanya memiliki fungsi untuk menulis data yang diterima dari *mapper* kedalam file di *HDFS*. Format data pada *Job* ini dicontohkan pada data 3.7.

3.3.4 Implementasi Sistem Pengelompokan Data

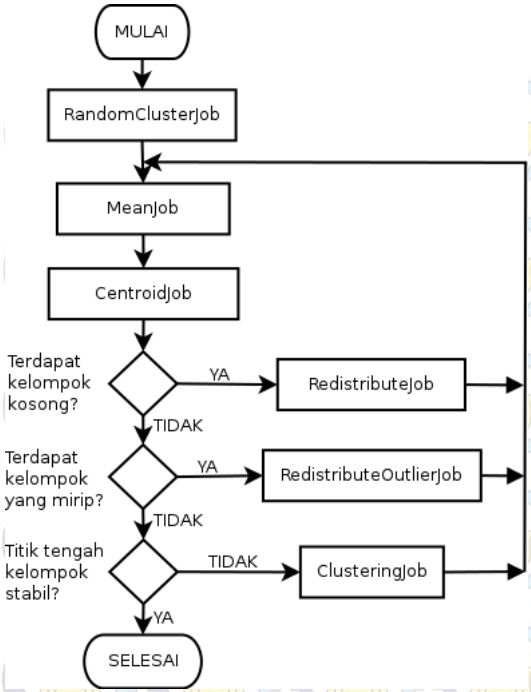
Tahap ini adalah tahapan implementasi dari pengelompokan data (*clustering*) menggunakan metode *K-means* ke dalam model program *MapReduce* yang dibagi berdasarkan fungsinya seperti pada gambar 3.4. Fungsi yang memerlukan iterasi seluruh data dilakukan dengan *MapReduce* sedangkan fungsi untuk menganalisa dan mengecek nilai tengah kelompok, yang merupakan data yang relatif kecil dilakukan dengan pembacaan *remote* dalam program *client*.

1. *RandomClusterJob*

   Dalam metode *K-means* yang pertama dilakukan adalah pengelompokan data secara acak. *RandomClusterJob* adalah *Job* dengan fungsi sederhana menyisipkan data tambahan dari data inputan. Data yang disisipkan adalah data id kelompok yang diperoleh dengan metode *randomizer()* yang akan menghasilkan nilai antara 0-1 yang kemudian dikalikan dengan jumlah kelompok data yang ingin dicari. *Job* ini memiliki area proses per teks sehingga beban kerjanya akan berada di *mapper*. Setelah *mapper* menyisipkan id kelompok, data yang langsung dikirimkan ke pada *reducer* langsung di tulis pada data keluaran sesuai dengan format data 3.9.

```
1 Inisialisasi cluster = jumlah kelompok yang akan dicari
2 randomId = randomizer() * cluster
3 write {randomId:data}
```

Kode 3.8: Algoritma mapper pada *RandomClusterJob*



Gambar 3.4: Flowchart clustering dalam model program MapReduce

```
1  Masukan:
2  http://food.detik.com/read
       /2015/12/03/064207/3086494/297/berani-coba-ramen-
       seberat-3-kg-dari-resto-ramen-ini  ramen:=302.2
       _makan:=14.12_populer:=56.21_jepang:=242.65_varian
       :=4.23_gerai:=53.12_konyol:=32.78_nama:=21.67 ...
3  ...
4
5  Keluaran:
6  1 http://food.detik.com/read
       /2015/12/03/064207/3086494/297/berani-coba-ramen-
       seberat-3-kg-dari-resto-ramen-ini  ramen:=302.2
       _makan:=14.12_populer:=56.21_jepang:=242.65_varian
       :=4.23_gerai:=53.12_konyol:=32.78_nama:=21.67 ...
7  ...
```

Kode 3.9: Format data masukan dan keluaran *VectorizerJob*

2. *MeanJob*

   Setelah semua data memiliki id kelompok, titik tengah kelompok ditentukan. Penentuan titik tengah kelompok dibagi kedalam dua *Job* karena terdapat dua fungsi yang area prosesnya perkelompok atau setara dengan semua data. *Job* pertama dilakukan perhitungan nilai rata-rata anggota kelompoknya dengan *MeanJob*. Untuk mendapatkan nilai rata-rata dari setiap kata dalam teks, pertama data masukan akan dipecah untuk didapatkan kumpulan kata dengan nilai *TFIDF*. Kemudian setiap data dimasukan kedalam variable *HashMap* global dengan nilai *key* adalah kombinasi id kelompok dengan kata dan nilai *value* adalah nilai *TFIDF* kata yang bersangkutan jika *key* belum ada dalam *HashMap* atau hasil jumlah nilai yang sudah ada dengan nilai *TFIDF* baru. Setelah proses *mapper* pada *node* selesai, fungsi *cleanup()* kemudian membaca nilai *HashMap*, dan untuk setiap data didalamnya akan dikirim ke *reducer* dengan *output-key* adalah id kelompok dan *output-value* adalah kombinasi kata dengan akumulasi jumlah *TFIDF*. Data yang kemudian diterima oleh *reducer* adalah data yang dikelompokan berdasarkan nilai *key*, yaitu data perkelompok dengan nilai *value* adalah kata dan nilai akumlasi *TFIDF*. Untuk setiap data yang masuk, dimasukan kembali kedalam satu *HashMap* untuk menyimpan jumlah keseluruhan *TFIDF* perkata. Setelah semua data dibaca, rata-rata dihitung dengan membagi total *TFIDF* perkata dengan jumlah anggota kelompok. Hasil rata-rata disimpan dalam data bertipe *string* yang dikemudian diformat seperti pada 3.12 sebelum ditulis kedalam file.

```
1  Inisialisasi Array vektorKata = semua pasangan kata dan
       TFIDF dari data
2  Inisialisasi clusterId = cluster id dari data
3
4  write {clusterId:"dataCounter_1"}
5
```

```
untuk setiap kata dalam vektor_kata :
  write {clusterId:kata_vektorKata[kata]}
```

Kode 3.10: Algoritma mapper pada *MeanJob*

```
Inisialisasi HM
Inisialisasi Array value_arr = setiap pasangan kata dan
     TFIDF untuk cluster_id yang sama
Inisialisasi counter

Untuk setiap kata dalam value_arr :
  Jika kata adalah "dataCounter" :
    counter = counter + 1
    lanjutkan ke data selanjutnya

  Jika tidak terdapat HM.get(kata) :
    HM.put(kata, value_arr[kata])
  Jika tidak :
    HM.put(kata, HM.get(kata) + value_arr[kata])

Untuk setiap kata pada HM :
  HM.put(kata, HM.get(kata)/counter)

write {cluster_id:HM.toString()_counter}
```

Kode 3.11: Algoritma reducer pada *MeanJob*

```
Masukan:
1 http://food.detik.com/read
    /2015/12/03/064207/3086494/297/berani-coba-ramen-
    seberat-3-kg-dari-resto-ramen-ini  ramen:=302.2
    _makan:=14.12_populer:=56.21_jepang:=242.65_varian
    :=4.23_gerai:=53.12_konyol:=32.78_nama:=21.67 ...
...

Keluaran:
1 ramen:=142.52_makan:=15.12_populer:=103.51_jepang
    :=43.97_varian:=10.86_gerai:=79.44_konyol:=13.54
    _nama:=244.54 ...
...
```

Kode 3.12: Format data masukan dan keluaran *MeanJob*

3. *CentoridJob*

CentoridJob berfungsi menemukan data yang mewakili titik tengah kelompok. Data hasil keluaran *MeanJob* akan dibaca pada program *client* dan dikirim sebagai data statis. *Job* ini sama seperti *MeanJob* menggunakan data hasil *RandomClusterJob* atau *ClusterJob* sebagai data masukannya. *Mapper* pada job ini membaca data rata-rata titik tengah pada fungsi *setup()* dan disimpan kedalam *HashMap*. Pada proses pembacaan data masukan, rata-rata titik tengah yang sesuai dengan id kelompok data masukan akan diambil, yang kemudian dicari kermiripannya dengan data masukan. Hasil kemiripan kemudian disisipkan pada data masukan yang kemudian dikirim menuju *reducer* dengan *output-key* adalah id kelompok. Pada *reducer* akan dicari data yang memiliki kemiripan tertinggi. Proses ini dilakukan dengan menyediakan dua variabel lokal, data dan tingkat kemiripan. Jika tingkat kemiripan data yang baru lebih besar dari tingkat kemiripan data lokal, nilai tingkat kemiripan lokal dan datanya kemudian diperbaharui dengan data yang baru. Setelah semua pembacaan data selesai, tingkat kemiripan dan data yang tersimpan dijadikan titik tengah kelompok dan ditulis kedalam file dengan format pada 3.15 Pada data titik tengah juga disisipkan jumlah anggota kelompok, rata-rata kemiripan dan id teks yang digunakan sebagai titk tengah kelompok.

```
1  Inisialisasi Array rataRata = semua rata-rata setiap
       kelompok data
2  Inisialisasi Array vektorKata = semua pasangan kata dan
        TFIDF dari data
3  Inisialisasi clusterId = cluster id dari data
4  Inisialisasi clusterSize = banyak anggota setiap
       kelompok data
5
6  similarity = cosineSimilarity(ratarata[cluster_id],
       vektorKata)
7
8  write {clusterId:similarity_data}
```

Kode 3.13: Algoritma mapper pada *CentoridJob*

```
Inisialisasi similarityTertinggi
Inisialisasi rataRataSimilarity
Inisialisasi dataTertinggi
Inisialisasi Array valueArr = setiap pasangan
    similarity dan data untuk cluster_id yang sama

Untuk setiap pasangan similarity dan data dalam
    value_arr :
  rataRataSimilarity = rataRataSimilarity + similarity
  Jika similarity > similarityTertinggi:
    similarityTertinggi = similarity
    dataTertinggi = data

url = url dari data_tertinggi
vektorKata = vektor kata dari data_tertinggi
rataRataSimilarity = rataRataSimilarity/clusterSize

write {cluster_id:
    vektorKata_url_rataRataSimilarity_clusterSize}
```

Kode 3.14: Algoritma reducer pada *CentoridJob*

```
Masukan:
1 http://food.detik.com/read
    /2015/12/03/064207/3086494/297/berani-coba-ramen-
    seberat-3-kg-dari-resto-ramen-ini  ramen:=302.2
    _makan:=14.12_populer:=56.21_jepang:=242.65_varian
    :=4.23_gerai:=53.12_konyol:=32.78_nama:=21.67 ...
...

Keluaran:
1 ramen:=142.52_makan:=15.12_populer:=103.51_jepang
    :=43.97_varian:=10.86_gerai:=79.44_konyol:=13.54
    _nama:=244.54... http://food.detik.com/read
    /2015/12/03/064207/3086494/297/berani-coba-ramen-
    seberat-3-kg-dari-resto-ramen-ini  0.12  15223
...
```

Kode 3.15: Format data masukan dan keluaran *CentoridJob*

4. *RedistributeJob*

Pengecekan kesalahan pengelompokan dilakukan dengan dua metode, pengecekan kelompok yang kosong dan pengecekan

kelompok dengan kemiripan titik tengah yang tinggi. Setelah titik tengah kelompok didapatkan dengan *CentroidJob*, datanya dibaca secara *remote* oleh program *client* dan disimpan dalam *HashMap*. Pengecekan kelompok yang kosong dilakukan dengan pengecekan id kelompok pada *HashMap*. Jika terdapat kelompok yang tidak ada atau kelompok dengan anggota kurang dari 2 maka dilakukan *RedistributeJob* dengan parameter id kelompok yang akan didistribusi ulang dan id kelompok yang tidak memiliki anggota. Pemilihan kelompok yang didistribusikan ulang ditentukan dengan jumlah anggota kelompok terbesar. Kemudian parameter pembagiannya diambil dari nilai rata-rata kelompok dibagi dua. Pada *Job* ini hanya dilakukan pengecekan tingkat kemiripan saja. Pada *mapper* jika id kelompok yang sedang dibaca sama dengan id kelompok yang akan didistribusi dilakukan pengecekan tingkat kemiripannya. Jika tingkat kemiripan kurang dari batas kemiripan yang dijadikan paramter, id kelompok data tersebut diganti menjadi id kelompok yang mendapatkan distribusi. Jika syarat-syarat tersebut tidak terpenuhi, pada tidak dilakukan perubahan pada data dan *reducer* hanya berfungsi meneruskan keluaran dari *mapper* kedalam file *HDFS*. Format data untuk *Job* ini dicontohkan pada data 3.17.

```
1  Inisialisasi KelKosong = id kelompok yang kosong
2  Inisialisasi KelBesar = id kelompok dengan anggota
       terbesar
3  Inisialisasi targetSimilarity = batas kesamaan untuk
       pembagian kelompok
4
5  similarity = tingkat kesamaan dari data terhadap titik
       tengah kelompoknya
6  clusterId = id kelompok dari data
7
8  Jika clusterId == KelBesar :
9    Jika similarity < targetSimilarity :
10     clusterId = KelKosong
11
12 write {clusterId:data}
```

Kode 3.16: Algoritma mapper pada *RedistributeJob*

```
1  Jumlah kelompok yang dicari: 4
2
3  Masukan titik tengah kelompok:
4  0 ... http... 0.10 6000
5  1 ... http... 0.14 16000
6  3 ... http... 0.13 5000
7
8  Masukan:
9  1 http://food.detik.com/read
       /2015/12/03/064207/3086494/297/berani-coba-ramen-
       seberat-3-kg-dari-resto-ramen-ini ...  0.06
10 1 http://travel.detik.com/readfoto
       /2015/12/03/173000/3077965/1026/1/yuk-wisata-
       sejarah-di-pulau-penyengat ... 0.12
11 ...
12
13 Keluaran:
14 2 http://food.detik.com/read
       /2015/12/03/064207/3086494/297/berani-coba-ramen-
       seberat-3-kg-dari-resto-ramen-ini  ... 0.06
15 1 http://travel.detik.com/readfoto
       /2015/12/03/173000/3077965/1026/1/yuk-wisata-
       sejarah-di-pulau-penyengat ... 0.12
16 ...
```

Kode 3.17: Format data masukan dan keluaran *RedistributeJob*

5. *RedistributeOutlierJob*

   Pengecekan kesalahan selanjutnya adalah pengecekan kelompok dengan kemiripan titik tengah yang tinggi. Pengecekan ini dilakukan dengan membandingkan tingat kemiripan antar titik tengah kelompok. Kemudian tingkat kemiripan tertinggi dibandingkan dengan batas tingkat kemiripan yang diperbolehkan yang sudah ditentukan sebelumnya. Jika kemiripan tidak melewati batas, dianggap tidak ditemukan kesalanan jenis ini. Jika tingkat kemiripan melebihi batas kemiripan, makan dijalankan proses *RedistributeOutlierJob*. Pada *Job* ini akan diberikan parameter berupa batas kemiripan dan kedua id kelompok data yang kemiripannya tinggi. Id kelompok yang dijadikan paramter diurutkan berdasarkan jumlah anggota kelompoknya. Id yang jumlah anggotanya lebih besar akan menjadi id kelompok baru untuk id kelompok yang jumlahnya

lebih kecil, sedangkan id kelompok yang lebih kecil akan menjadi id kelompok baru yang terdiri dari data-data pada kelompok lain yang tingkat kemiripannya lebih kecil dari batas kemiripan. Pada proses *mapper* pada *Job* ini yang pertama dilakukan adalah pengecekan id kelompok. Jika id kelompok adalah id kelompok yang akan disatukan dengan id kelompok yang mirip, id kelompok data tersebut kemudian diganti dengan id kelompok baru. Pengecekan selanjutnya adalah tingkat kemiripan. Jika tingkat kemiripan data tersebut terhadap titi tengah kelompoknya lebih kecil dari tingkat kemiripan yang di perbolehkan, id kelompok data tersebut akan digantu menjadi id kelompok yang sedang dikosongkan. Pada *Job* ini reducer berfungsi menulis langsung data dari *mapper* menuju file. Contoh permasalahan dan format data untuk *Job* ini terdapat pada data 3.19.

```
1  Inisialisasi KelDisatukan = id kelompok tempat kelompok
       yang mirip disatukan
2  Inisialisasi KelDikosongkan = id kelompok yang akan
       dikosongkan
3  Inisialisasi BatasSimilarity = batas kesamaan untuk
       pembagian kelompok
4
5  similarity = tingkat kesamaan dari data terhadap titik
       tengah kelompoknya
6  clusterId = id kelompok dari data
7
8  Jika clusterId == KelDikosongkan :
9    clusterId = KelDisatukan
10
11 Jika similarity < BatasSimilarity :
12   clusterId = KelDikosongkan
13
14 write {clusterId:data}
```

Kode 3.18: Algoritma mapper pada *RedistributeOutlierJob*

```
1  Id kelompok yang mirip: 1 dan 3
2  Batas kemiripan: 0.02
3
4  Masukan titik tengah kelompok:
5  1 ... http... 0.14 16000
```

```
3 ... http... 0.13 5000

Masukan:
3 http://food.detik.com/read
    /2015/12/03/064207/3086494/297/berani-coba-ramen-
    seberat-3-kg-dari-resto-ramen-ini ...  0.006
3 http://travel.detik.com/readfoto
    /2015/12/03/173000/3077965/1026/1/yuk-wisata-
    sejarah-di-pulau-penyengat ... 0.12
...

Keluaran:
3 http://food.detik.com/read
    /2015/12/03/064207/3086494/297/berani-coba-ramen-
    seberat-3-kg-dari-resto-ramen-ini  ... 0.06
1 http://travel.detik.com/readfoto
    /2015/12/03/173000/3077965/1026/1/yuk-wisata-
    sejarah-di-pulau-penyengat ... 0.12
...
```

Kode 3.19: Format data masukan dan keluaran *RedistributeOutlierJob*

6. *ClusterJob*

   Evaluasi kelompok oleh anggota kelompok dilakukan jika kedua kesalahan yang disebutkan sebelumnya tidak ditemukan. Pada proses ini semua titik tengah kelompok yang sudah dibaca sebeumnya dijadikan parameter data statis untuk proses *ClusterJob*. *Job* ini berfungsi untuk mengevaluasi keanggotaan kelompok. Pada fungsi *setup()* pada proses *mapper* semua data titik tengah kelompok akan dibaca dan disimpan dalam *HashMap* global. Untuk setiap data teks yang dibaca oleh *mapper* dihitung tingkat kemiripannya dengan semua titik tengah kelompok. Id kelompok baru data tersebut adalah id kelompok dengan kemiripan titik tengah tertinggi seperti yang dicontohkan pada data 3.21. *Reducer* pada *Job* ini berfungsi untuk menulis langusng keluaran dari *mapper*.

```
Inisialisasi Array center = semua titik tengah setiap
    kelompok data

Inisialisasi Array vektorKata = semua pasangan kata dan
     TFIDF dari data
```

```
4  Inisialisasi similarityTertinggi
5  Inisialisasi clusterId
6
7  Untuk semua id kelompok pada center :
8    similarity = cosineSimilarity(center[id kelompok],
         vektorKata)
9    Jika similarity > similarityTertinggi :
10     similarityTertinggi = similarity
11     clusterId = id kelompok
12
13 write {clusterId:data_similarityTertinggi}
```

Kode 3.20: Algoritma mapper pada *ClusterJob*

```
1  Data titik tengah kelompok:
2  0 ... http://food.detik.com/... 0.2 12313
3  1 ... http://travel.detik.com/... 0.14 21044
4
5  Masukan:
6  3 http://food.detik.com/read
       /2015/12/03/064207/3086494/297/berani-coba-ramen-
       seberat-3-kg-dari-resto-ramen-ini ...  0.006
7  3 http://travel.detik.com/readfoto
       /2015/12/03/173000/3077965/1026/1/yuk-wisata-
       sejarah-di-pulau-penyengat ... 0.12
8  ...
9
10 Keluaran:
11 0 http://food.detik.com/read
       /2015/12/03/064207/3086494/297/berani-coba-ramen-
       seberat-3-kg-dari-resto-ramen-ini ...  0.006
12 1 http://travel.detik.com/readfoto
       /2015/12/03/173000/3077965/1026/1/yuk-wisata-
       sejarah-di-pulau-penyengat ... 0.12
13 ...
```

Kode 3.21: Format data masukan dan keluaran *ClusterJob*

BAB 4
PENGUJIAN DAN ANALISA

Pada bab ini dilakukan pengujian yang dibagi menjadi dua tahapan yaitu pengujian praproses data dan pengujian pengelompokan data. Pengujian akan ditujukan untuk memperlihatkan kinerja sistem yang dibuat pada bab tiga. Kinerja pengujian diukur berdasarkan waktu proses dan hasil akhir sistem. Pengujian dilakukan pada empat sistem komputasi terdistribusi yang memiliki jumlah *slave node* berbeda yaitu dua, tiga, empat dan lima *slave node*.

4.1 Pengujian Praproses Data

Pengujian ini dilakukan pada sistem praproses data yang meliputi pembagian beban kerja dan waktu proses.

Tabel 4.1: Jumlah dokumen dan besar data yang diproses setiap *Job*

Job	**Masukan**		**Keluaran**	
	Jumlah Dokumen	Besar (MB)	Jumlah Dokumen	Besar (MB)
TermsJob	144276	1226	139485	379
IDFJob	139485	380	379742	9
VectorizerJob	139485	380	139485	376

4.1.1 Pembagian Beban Kerja

Setiap *Job* yang dijalankan pada sistem ini akan dibagi beban kerjanya berdasarkan besar data masukan yang akan diproses. Data yang diproses oleh *TermsJob* berbeda dengan data yang akan diproses oleh *IDFJob* dan *VectorizerJob*. Perbedaan pembagian beban pekerjaan pada setiap *Job* digambarkan pada tabel 4.1. Dari tabel tersebut diketahui bahwa jumlah dokumen yang digunakan sebagai masukan untuk *Job TermsJob* adalah 144276 dokumen, dimana satu dokumen merupakan satu artikel berita. Besar memori

yang digunakan untuk menampung data ini adalah 1226 MB. Jumlah data keluaran dari *Job* ini adalah 139485 dokumen dengan besar penggunaan memori 379 MB.

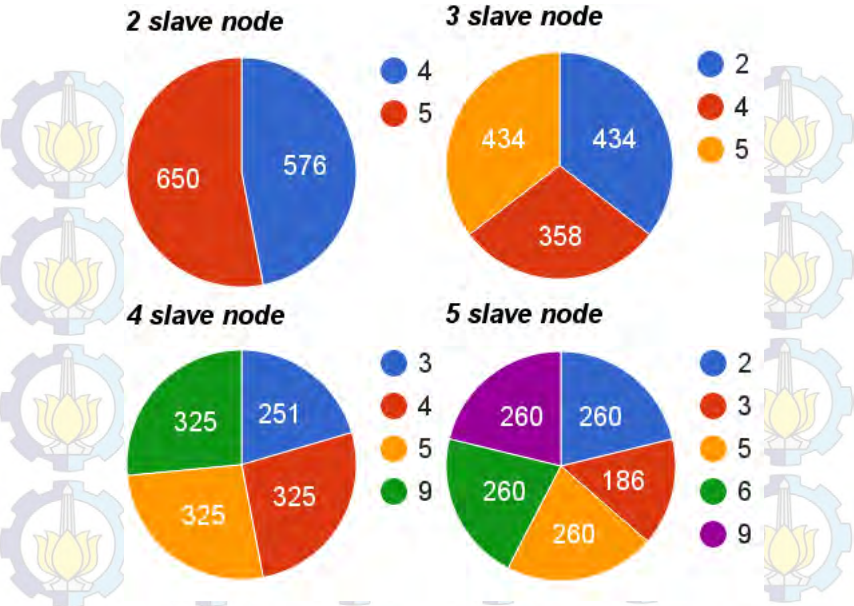
1. *TermsJob*

   Data masukan yang diolah *Job* ini memiliki ukuran yang cukup besar berdasarkan tabel 4.1. Pembagian beban kerja untuk pengolahan data ini dibagi sesuai dengan grafik 4.1. Pembagian pada grafik tersebut dibagi berdasarkan banyak data yang diproses pada setiap *slave node*. Pembagian data ini ditentukan berdasarkan besar konfigurasi blok file pada *service HDFS*. Dari semua sistem, data dibagi hampir merata pada setiap *slave node*. Akan terdapat satu *node* yang memproses data lebih sedikit, hal ini dikarenakan data tersebut merupakan sisa pembagian data berdasarkan konfigurasi blok file. Berdasarkan pada grafik 4.1, jumlah data yang diproses oleh setiap *node* akan menjadi lebih sedikit dengan semakin banyaknya julmah *slave node* yang digunakan.

   Pada tabel 4.1 terlihat bahwa jumlah data masukan dan keluaran tidak sama. Hal ini dikarenakan beberapa data yang diperoleh dari proses *crawling* ada yang tidak valid. Beberapa data tersebut tidak memiliki isi berita yang dapat disebabkan adanya galat saat mengakses *url* atau mengakses halaman *HTML* yang memerlukan authentikasi khusus. Sedangkan perbedaan besar data masukan dan keluaran disebabkan karena kebanyakan data berita yang diperoleh dari proses *crawling* berisi data non-konten.

2. *IDFJob*

   Pada tabel 4.1 diketahui bahwa besar data masukan *Job* ini sesuai dengan besar data keluaran dari *TermsJob*. Konfigurasi ukuran blok file pada data ini berbeda dengan data masukan untuk *TermsJob*. Untuk memaksimalkan pembagian beban kerja, konfigurasi pada ukuran blok file diatur berdasarkan besar data yang diproses. Dalam sistem ini terdapat dua kategori besar data, yaitu data masukan awal yang diproses oleh *TermsJob* dan data hasil vektorisasi yang kurang lebih hampir sama dengan data keluaran dari *TermsJob*. Pengaturan uku-

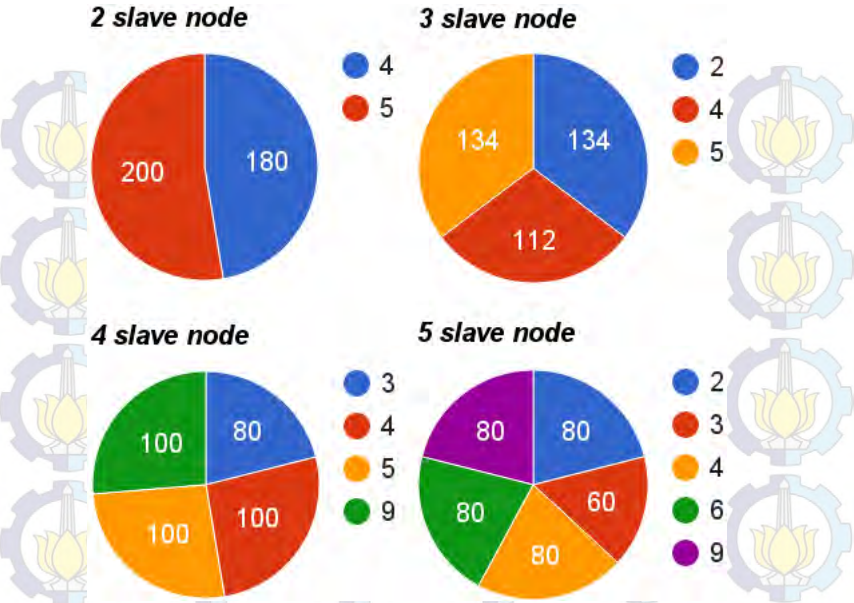


Gambar 4.1: Pembagian besar data (MB) yang diproses *TermsJob* berdasarkan jumlah *slave node*

ran blok file dapat ditentukan ketika menyalin data ke dalam *HDFS* atau hasil proses suatu *Job*. Dengan pengaturan blok file ini, pembagian beban setiap *slave node* menjadi hampir sama seperti pada grafik 4.2.

3. *VectorizerJob*
   Job ini menggunakan data masukan yang berasal dari keluaran *TermsJob*. Data itu juga yang digunakan sebagai data masukan oleh *IDFJob*. Hal ini menjadikan beban pembagian pekerjaan antara *Job* ini dengan *IDFJob* sama. Pembagian beban untuk *Job* ini pada semua sistem uji dapat dilihat pada grafik 4.3. Pada *Job* ini besar data juga dibagi hampir merata pada semua *slave node*.

   Pada tabel 4.1 terlihat bahwa data masukan tidak berbeda dengan data keluaran pada jumlah data. Hal ini dikarenakan



Gambar 4.2: Pembagian besar data (MB) yang diproses *IDFJob* berdasarkan jumlah *slave node*

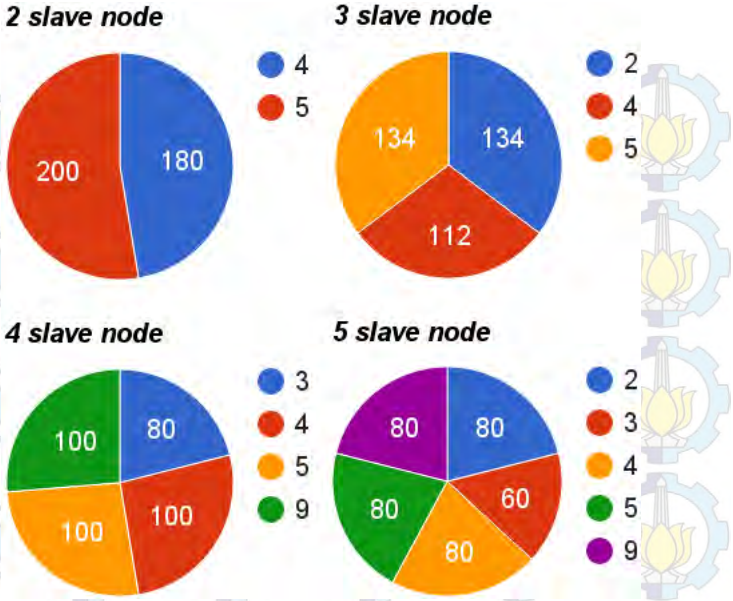
pada *Job* ini hanya merubah nilai *TF* pada data masukan menjadi nilai *TFIDF*.

4.1.2 Waktu Proses

Pengujian waktu proses setiap *Job* dilakukan untuk menganalisa kinerja dan pembagian beban kerja oleh *ApplicationMaster*. Lama proses pada pengujian ini akan dibahas terpisah setiap *Job* untuk dibandingkan antara sistem dengan jumlah *slave node* yang berbeda.

1. *TermsJob*

   Berdasarkan tabel 4.2, diketahui bahwa lama proses *Job* ini semakin cepat untuk jumlah *slave node* yang lebih banyak. *TermsJob* merupakan proses yang memerlukan waktu banyak karena selain data masukannya yang besar, pada proses ini di-



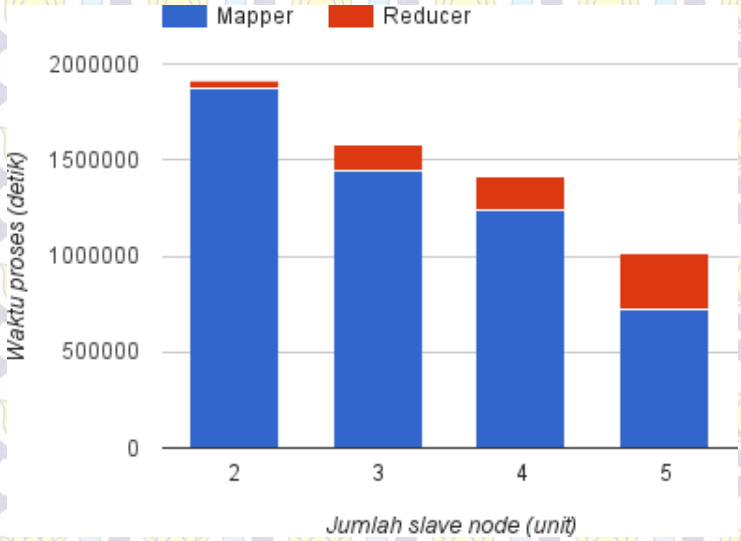
Gambar 4.3: Pembagian besar data (MB) yang diproses *VectorizerJob* berdasarkan jumlah *slave node*

Tabel 4.2: Waktu proses *TermsJob*

Jumlah *slave node* (unit)	**Rata-rata waktu prosess *mapper* (detik)**	**Rata-rata waktu proses *reducer* (detik)**
2	1878	39
3	1447	132
4	1241	178
5	723	294

lakukan banyak proses penyaringan data. Peningkatan waktu proses pada *Job* ini adalah, 18% untuk peningkatan dari sistem dua *slave node* menjadi tiga *slave node*, 10% untuk pen-

ingkatan dari sistem tiga *slave node* menjadi empat *slave node* dan 28% untuk peningkatan dari sistem empat *slave node* menjadi lima *slave node*.



Gambar 4.4: Akumulasi waktu proses *mapper* dan *reducer* pada proses *TermsJob*

Berbanding terbalik dengan waktu proses fungsi *mapper*, fungsi *reducer* pada *Job* ini memerlukan waktu proses lebih lama pada peningkatan jumlah *slave node*. *Reducer* berfungsi untuk menangkap keluaran semua fungsi *mapper*, sehingga semakin banyak *slave node* yang tersedia, semakin banyak juga data yang dikirim ke *reducer*. Beban komunikasi jaringan selain digunakan untuk mengirim data dari *mapper* ke *reducer*, digunakan juga untuk menyimpan data kedalam *HDFS* yang juga menggunakan komunaksi jaringan dengan *datanode* dan *namenode*. *Reducer* juga berjalan tidak secara bersamaan, proses *reduce* mulai bekerja ketika proses *mapper* sudah melakukan proses sebanyak 80%. Akumulasi waktu proses *mapper* dan *reducer* pada grafik 4.4 menunjukan bahwa semakin banyak jumlah *slave node* yang digunakan, semakin cepat

waktu prosesnya.

Tabel 4.3: Waktu proses *IDFJob*

Jumlah *slave node* (unit)	**Rata-rata waktu prosess *mapper* (detik)**	**Rata-rata waktu proses *reducer* (detik)**
2	51	13
3	37	20
4	48	12
5	23	92

2. *IDFJob*

   Perubahan jumlah *slave node* untuk pengerjaan *Job* ini tidak terlalu berpengaruh terhadap lama prosesnya seperti pada tabel 4.3. Tidak adanya pola pada perubahan lama proses dapat disebabkan karena proses pada *IDFJob* merupakan proses yang ringan dan cepat, dan perbedaan lama prosesnya terletak pada variabel-variabel yang tidak diamati seperti kepadatan jaringan komunikasi yang mempengaruhi waktu mulai pengerjaan *mapper* yang sebenarnya.

Tabel 4.4: Waktu proses *VectorizerJob*

Jumlah *slave node* (unit)	**Rata-rata waktu prosess *mapper* (detik)**	**Rata-rata waktu proses *reducer* (detik)**
2	36	21
3	30	98
4	25	127
5	31	140

3. *VectorizerJob*

   Job ini menggunakan data masukan yang sama dengan data masukan *IDFJob* yang menjadikan karakteristik prosesnya tidak terlalu berbeda. Lama proses *mapper* dan *reducer* pada *Job* ini tidak terlalu dipengaruhi oleh peningkatan jumlah *slave node* seperti pada tabel 4.4. Hal ini disebabkan karena prosesnya yang cepat seperti pada *IDFJob* sehingga peningkatan kinerja tidak dapat diamati.

4.2 Pengujian Pengelompokan Data

Pengujian yang dilakukan pada sistem ini meliputi analisa waktu proses dan validasi hasil akhir proses pengelompokan yang akan dievaluasi secara eksternal menggunakan label kategori data yang diperoleh saat pengambilan data.

Tabel 4.5: Waktu proses dan banyak iterasi pada sistem pengelompokan

Jumlah *Slave Node* (unit)	**Jumlah Iterasi**	**Waktu Total (detik)**
2	9	24036
3	5	8838
4	5	7464
5	8	6104

4.2.1 Waktu Proses

Dalam *Job* ini dilakukan pencarian acak pada awal proses. Pencarian acak ini akan menentukan nilai awal kelompok dan berpengaruh terhadap banyak iterasi yang diperlukan untuk menstabilkan nilainya. Dari tabel 4.5 diketahui pengujian pada empat sistem berbeda memiliki jumlah iterasi yang berbeda kecuali untuk pengujian dengan tiga *slave node* dan empat *slave node*. Waktu proses total pada pengujian setiap sistem juga berbeda bahkan ketika jumlah iterasinya sama.

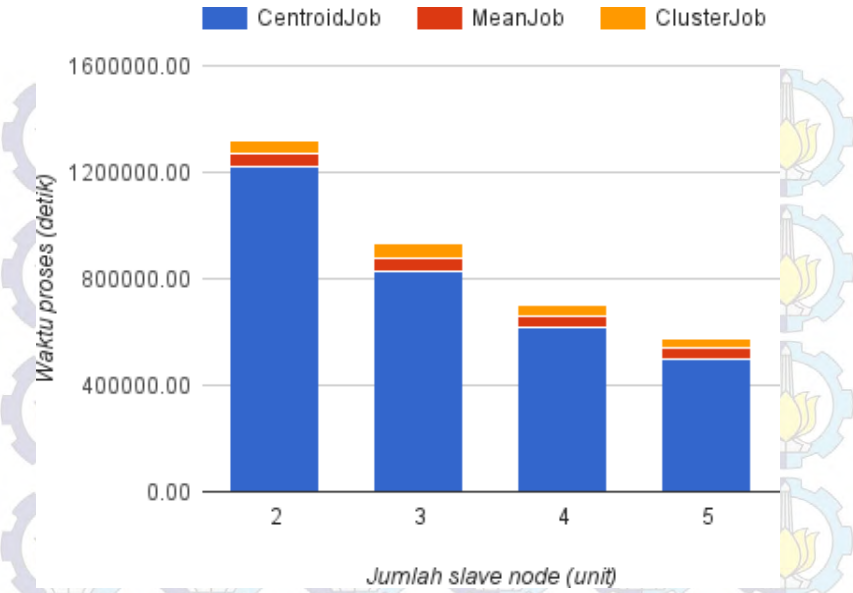
Tabel 4.6: Rata-rata lama proses*Job* pada sistem pengelompokan

Jumlah *Slave Node* (unit)	***Random Job* (detik)**	***Mean Job* (detik)**	***Centorid Job* (detik)**	***Cluster Job* (detik)**
2	60	45	1224	51
3	70	47	830	58
4	70	42	620	40
5	134	38	501	36

1. Waktu Proses

   Pada tabel 4.6 diperoleh bahwa proses *CentroidJob* memerlukan waktu proses terlama. Pada *Job* ini dilakukan proses perhitungan tingkat kemiripan antara data dengan rata-rata data anggota kelompok. Rata-rata anggota kelompok merupakan vektor kata dengan variasi kata yang beragam yang menjadikan perhitungan tingkat kesamaannya memerlukan waktu proses yang lama. Pada *ClusterJob* dilakukan juga perhitungan tingkat kemiripan, tetapi perhitungan dilakukan terhadap data dengan titik tengah setiap kelompok yang besar vektornya tidak sebesar hasil rata-rata anggota kelompok.

   Peningkatan jumlah *slave node* pada sistem ini pengaruhnya sangat terlihat pada rata-rata waktu proses *CentroidJob* yang dikarenakan proses *Job* ini merupakan proses yang berat dan memerlukan waktu proses yang lama. Dari tabel 4.6 diperoleh peningkatan waktu proses sebesar 394 detik (32%) untuk peningkatan dari sistem dua *slave node* menjadi tiga *slave node*, 210 (25%) detik untuk peningkatan dari sistem tiga *slave node* menjadi empat *slave node* dan 119 detik (19%) untuk peningkatan dari sistem empat *slave node* menjadi lima *slave node*. Pada *Job* lainnya tidak diperoleh perubahan waktu proses yang berpola. Hal ini juga disebabkan proses pada *Job* tersebut merupakan proses yang ringan yang tidak memerlukan waktu proses yang besar. Waktu proses yang digunakan pada *Job* tersebut sebagian besar terpakai untuk koordinasi



Gambar 4.5: Waktu proses rata-rata antara *MeansJob*, *CentroidJob* dan *ClusterJob* pada sistem dengan jumlah *slave node* yang berbeda

dengan *ResourceManager*, *ApplicationManager* dan pengiriman data dari *mapper* menuju *reducer* yang tidak dapat diamati dalam penelitian ini. Akumulasi waktu proses rata-rata antara *MeansJob*, *CentroidJob* dan *ClusterJob* dijadikan sebagai waktu proses rata-rata per iterasi seperti pada grafik 4.5. Grafik tersebut menunjukan peningkatan waktu proses per iterasi dari sistem yang menggunakan jumlah *slave node* lebih banyak.

2. Banyak Redistribusi

   Pada tabel 4.7 digambarkan banyak kesalahan pengelompokan yang terjadi pada pengujian setiap sistem dengan jumlah *slave node* yang berbeda. Jumlah kesalahan ini disebabkan oleh adanya kelompok yang mirip yang ditentukan oleh faktor pembagian kelompok secara acak pada tahapan awal pengelompokan. Waktu proses untuk penyelesaiannya tidak memiliki

Tabel 4.7: Banyak iterasi yang menghasilkan kesalahan kelompok dan rata-rata waktu proses penyelesaiannya

Jumlah *Slave Node* (unit)	**Banyak Iterasi**	**Rata-Rata Waktu Proses (detik)**
2	6	18
3	2	11
4	3	18
5	5	20

nilai yang berpola untuk peningkatan jumlah *slave node* sistem. Hal ini juga disebabkan oleh waktu proses penyelesaiannya yang cepat sehingga waktu proses sebagian besar dapat dipengaruhi oleh kepadatan jaringan komunikasi atau variabel lain yang tidak diamati pada penelitian ini.

Tabel 4.8: Nilai rata-rata pengukuran *Cosine Similairty* angggota kelompok terhadap titik tengahnya pada percobaan dengan dua *slave node*

Iterasi	**Kelompok 1**	**Kelompok 2**	**Kelompok 3**	**Kelompok 4**
1	0.1226	0.1223	0.1223	0.1220
2	0.1872	0.1234	0.1785	0.1884
3	0.1827	0.2336	0.1338	0.1986
4	0.1630	0.2360	0.2121	0.0956
5	0.1658	0.2357	0.2137	0.1023
6	0.1653	0.2357	0.2116	0.0983
7	0.1654	0.2355	0.2127	0.1029
8	0.1626	0.2179	0.1884	0.1684
9	0.1658	0.2200	0.1742	0.1683

3. Analisa Kestabilan Kelompok

Tabel 4.8 merupakan tabel nilai rata-rata *cosine similairty* anggota kelompok setiap iterasi untuk sistem pengelompokan dengan dua *slave node*. *Cosine similairty* merupakan perhitungan tingkat kemiripan teks dengan hasil perhitungan bernilai antara satu dengan nol, dimana semakin tinggi nilainya semakin mirip dua teks yang dibandingkan. Kelompok dalam iterasi ini belum diketahui akan memiliki kelompok data apa, sehingga digunakan notasi kelompok 1, kelompok 2, kelompok 3 dan kelompok 4. Setiap perulangan iterasi, titik tengah kelompok akan terus berubah sampai titik tengah kelompok tidak lagi mengalami perubahan atau dapat dianggap sudah stabil. Dari tabel 4.8 pada iterasi pertama diperoleh semua nilai tingkat kemiripan yang hampir sama. Pada iterasi selanjutnya, dengan titik tengah kelompok yang baru, kelompok 1, kelompok 3 dan kelompok 4 memiliki peningkatan rata-rata kemiripan anggotanya, tetapi kelompok 2 masih tetap. Perulangan iterasi pada pengujian ini berhenti pada iterasi ke sembilan, sehingga diketahui bahwa pada iterasi ke delapan dan iterasi ke sembilan sudah tidak lagi terjadi perubahan titik tengah kelompok. Pada kedua iterasi ini didapat nilai tingkat kemiripan semua kelompok lebih besar dari nilai 0.1658.

Tabel 4.9: Nilai rata-rata pengukuran *Cosine Similairty* angggota kelompok terhadap titik tengahnya pada percobaan dengan tiga *slave node*

Iterasi	**Kelompok 1**	**Kelompok 2**	**Kelompok 3**	**Kelompok 4**
1	0.1226	0.1223	0.1222	0.1220
2	0.1874	0.1503	0.1245	0.1914
3	0.1331	0.1761	0.2340	0.2066
4	0.1881	0.1767	0.2209	0.1708
5	0.1765	0.1797	0.2225	0.1723

Tabel 4.9 adalah tabel nilai rata-rata tingkat kemiripan anggota kelompok setiap iterasi untuk sistem pengelompokan dengan tiga *slave node*. Rata-rata nilai tingkat kemiripan pada iterasi pertama menunjukan bahwa pada iterasi tersebut dokumen belum terkelompok dengan baik. Empat iterasi diperlukan pada pengujian ini untuk memperoleh rata-rata tingkat kemiripan semua kelompok jauh dari nilai rata-rata tingkat kemiripan pada iterasi pertama. Pada iterasi selanjutnya rata-rata tingkat kemiripan tidak terlalu berubah, sehingga dapat disimpulkan bahwa titik tengah semua kelompok sudah stabil.

Tabel 4.10: Nilai rata-rata pengukuran *Cosine Similairty* angggota kelompok terhadap titik tengahnya pada percobaan dengan empat *slave node*

Iterasi	Kelompok 1	Kelompok 2	Kelompok 3	Kelompok 4
1	0.1226	0.1222	0.1221	0.1225
2	0.1244	0.1931	0.1646	0.2109
3	0.2324	0.1300	0.1778	0.1973
4	0.2208	0.1881	0.1767	0.1708
5	0.2224	0.1766	0.1797	0.1723

Tabel 4.10 merupakan tabel nilai rata-rata kemiripan anggota kelompok setiap iterasi untuk sistem pengelompokan dengan empat *slave node*. Lima iterasi diperlukan pada pengujian dengan sistem ini untuk mendapatkan kelompok yang stabil. Dari tabel nilai rata-rata kemiripan anggota kelompok semua sistem yang diuji diperoleh kesimpulan bahwa hasil pengelompokan sudah stabil jika semua kelompok memiliki rata-rata tingkat kemiripan anggotanya lebih besar dari 0.16.

4.2.2 Hasil Proses

Pada tabel 4.11 dipaparkan *confusion matrix* sebagai validasi hasil pengelompokan dokumen untuk sistem dengan dua *slave node*.

Tabel 4.11: Tabel *confusion matrix* pengelompokan dengan dua *slave node*

	Kuliner	Wisata	Otomotif	Sepak Bola
Kuliner	26260	4733	389	148
Wisata	1078	16913	437	81
Otomotif	1816	10617	46656	775
Sepak Bola	379	1569	1182	26452

Setiap baris pada *confusion matrix* merepresentasi banyak dokumen hasil pengelompokan berdasarkan label kelompok baris, sedangkan setiap kolom merepresentasi banyak dokumen yang sebenarnya berdasarkan label kelompok kolom. Kelompok kuliner dari hasil pengelompokan dokumen dengan sistem ini terdiri dari 26260 dokumen berlabel kuliner, 4733 dokumen berlabel wisata, 389 dokumen berlabel otomotif dan 148 dokumen berlabel sepak bola. Dari kelompok ini didapat 26260 dokumen yang benar dari total 31530 dokumen pada kelompok itu. Sehingga kelompok dokumen kuliner memiliki nilai presisi 83.29%. Kelompok dokumen wisata memiliki dokumen yang benar sejumlah 16913 dari 18509 dokumen, sehingga memiliki nilai presisi 91.38%. dokumen otomotif dan sepak bola pada pengujian ini memiliki nilai presisi 77.94% dan 89.42%.

Nilai presisi terendah pada pengujian ini berada pada kelompok otomotif. Pada kelompok ini terdapat 10617 (16.70%) dokumen berlabel wisata. dokumen dengan label wisata ikut dikelompokan dalam kelompok ini karena titik tengah kelompok ini memiliki kemiripan dengan dokumen tersebut lebih tinggi daripada kemiripan dengan titik tengah kelompok wisata. Hal ini dapat disebabkan bahasan yang dibahas dalam dokumen berlabel wisata tersebut membahas tentang otomotif karena biasanya pada pembahasan topik wisata dibahas juga tentang kendaraan, kondisi jalan dan lalu lintas yang merupakan pembahasan utama pada kategori otomotif.

Akurasi dari hasil pengelompokan dokumen diperoleh dengan membandingakan semua dokumen yang benar (*true positive*) dengan total dokumen dalam himpunan. Pada pengujian sistem ini dilakukan sebanyak tiga kali dan diperoleh akurasi rata-rata 83%.

Tabel 4.12 menggambarakan *confision matrix* untuk hasil pen-

Tabel 4.12: Tabel *confusion matrix* pengelompokan dengan tiga *slave node*

	Kuliner	Wisata	Otomotif	Sepak Bola
Kuliner	26049	4501	389	143
Wisata	1332	17602	548	76
Otomotif	1768	10268	46557	772
Sepak Bola	384	1461	1171	26465

gelompokan dokumen pada sistem dengan tiga *slave node*. Pada pengujian dengan sistem ini diperoleh nilai presisi 83.81% untuk kelompok kuliner, 90% untuk kelompok wisata, 78.42% untuk kelompok otomotif dan 89.77% untuk kelompok sepak bola. Nilai presisi terendah pada pengujian ini juga berada pada kelompok otomotif, sama seperti pengujian dengan dua *slave node*. Pada kelompok ini terdapat 10268 (17.30%) dokumen berlabel wisata. Sama seperti pembahasan sebelumnya, kesalahan pengelompokan dokumen dengan kategori wisata ini dapat disebabkan bahasan yang dibahas dalam dokumen berlabel wisata tersebut didalamnya terdapat pembahasan alat transportasi, kondisi jalan dan lalu lintas yang merupakan pembahasan utama pada kategori otomotif. Pada pengujian ini juga dilakukan sebanyak tiga kali pengujian dan diperoleh rata-rata nilai akurasi sebesar 86%.

Tabel 4.13: Tabel *confusion matrix* pengelompokan dengan empat *slave node*

	Kuliner	Wisata	Otomotif	Sepak Bola
Kuliner	26260	4733	389	148
Wisata	1078	16913	438	81
Otomotif	1816	10617	46655	775
Sepak Bola	379	1569	1182	26452

Tabel *confusion matrix* 4.13 merupakan hasil pengelompokan dokumen untuk pengujian pada sistem dengan empat *slave node*. Pada pengujian sistem ini nilai presisi untuk kelompok kuliner,

wisata, otomotif dan sepak bola adalah 83.29%, 91.37%, 77.94% dan 89.42%. Pada hasil pengelompokan dokumen ini, kelompok kuliner di dalamnya terdapat dokumen berlabel wisata sejumlah 4733 (15%), merupakan presisi terendah kedua setelah otomotif. Sama seperti dokumen berlabel wisata yang dikelompokan sebagai dokumen otomotif, dokumen ini juga dikelompoak ke dalam kelompok kuliner karena tingkat kemiripan dokumen dengan titik tengah kelompok kuliner lebih tinggi dibandingkan dengan kemiripan dengan titik tengah kelompok wisata. Hal ini dapat disebabkan oleh bahasan tentang wisata yang didalamnya biasanya juga membahas tentang kuliner seperti pada wisata kuliner. Nilai rata-rata akurasi dari tiga kali pengujian pada sistem ini adalah 86%.

Tabel 4.14: Tabel *confusion matrix* pengelompokan dengan lima *slave node*

	Kuliner	Wisata	Otomotif	Sepak Bola
Kuliner	26049	4501	388	143
Wisata	1332	17602	548	76
Otomotif	1768	10268	46558	772
Sepak Bola	384	1461	1170	26465

Tabel *confusion matrix* 4.14 merupakan tabel *confusion matrix* untuk pengujian sistem dengan lima *slave node*. Nilai presisi pada pengujian ini adalah 83.81% untuk kelompok kuliner, 90.00% untuk kelompok wisata, 78.43% untuk kelompok otomotif dan 89.77% untuk kelompok sepak bola. Nilai presisi yang rendah pada kelompok kuliner dan otomotif adalah adanya kesalahan pengelompokan dokumen wisata, yaitu 4501 dokumen (14.48%) pada kelompok kuliner dan 10268 (17.30%) pada kelompok otomotif. Nilai rata-rata akurasi dari tiga kali pengujian yang dilakukan pada sistem ini adalah 84%.

Dari semua pengujian yang dilakukan pada metode ini didapat nilai akurasi rata-rata untuk setiap pengujian tidak terlalu berbeda. Hal ini menunjukan bahwa jumlah *slave node* pada sistem tidak mempengaruhi hasil dari pengelompokan dokumen. Nilai akurasi rata-rata dari semua pengujian adalah 83% dengan kesalahan pen-

gelompokan terbesar terdapat pada dokumen dengan label wisata.

BAB 5
PENUTUP

5.1 Kesimpulan

Dari hasil implementasi dan pengujian sistem praproses dan pengelompokan data yang sudah dilakukan dapat ditarik beberapa kesimpulan sebagai berikut :

1. Pada sistem komputasi terdistribusi, pembagian data untuk diproses ditentukan berdasarkan besar ukuran blok file *HDFS*. Pada *TermsJob*, data masukan dengan besar 1.2GB dibagi secara merata terhadap semua *slave node* yang terhubung dalam sistem terdistribusi. Peningkatan waktu proses ketika jumlah *slave node* ditingkatkan disebabkan oleh berkurangnya besar data yang diproses setiap *slave node*. Dampak pembagian data ini sangat berpengaruh untuk memproses data berukuran besar seperti pada *TermsJob*. Pada *Job* ini peningkatan waktu proses sebesar 20% diperoleh dengan penambahan *slave node* sejumlah 25%-50%.
2. Komputasi terdistribusi selain membagi besar data juga membagi beban proses. Dampak pembagian beban proses sangat berpengaruh untuk proses yang memerlukan waktu pengerjaan yang lama seperti pada *CentroidJob*. Waktu proses *CentroidJob* mengalami peningkatan waktu proses sebesar 23% untuk peningkatan jumlah *slave node* sebesar 25%-50%. Dengan peningkatan *slave node* tersebut, sistem pengelompokan data mengalami peningkatan waktu proses sebesar 20% per-iterasi.
3. Penggunaan data aktual sebagai titik tengah kelompok pada metode pengelompokan *K-means* berpengaruh terhadap cepat waktu perhitungan tingkat kesamaan data dengan titik tengah kelompok. Modifikasi ini juga menjadikan titik tengah kelompok lebih cepat stabil karena dapat mengevaluasi kesalahan pengelompokan, seperti adanya kelompok yang serupa. Pada penelitian ini diperoleh jumlah iterasi maksimal untuk setiap pengujian adalah sembilan iterasi dengan rata-rata akurasi

hasil pengelompokan datanya adalah 83%. Hasil ini juga menunjukan tidak adanya pengaruh jumlah *slave node* terhadap hasil pengelompokan data.

5.2 Saran

Demi pengembangan lebih lanjut mengenai tugas akhir ini, disarankan beberapa langkah lanjutan sebagai berikut :

1. Penambahan jumlah *slave node* pada sistem untuk mempercepat waktu proses.
2. Meningkatkan spesifikasi *slave node* supaya semakin banyak komponen-komponen *Hadoop* yang dapat dipasang untuk meningkatkan kemampuan sistem komputasi terdistribusi.
3. Perlu penambahan filter dan perbaikan kata pada sistem praproses untuk setiap kata tidak sempurna yang biasa disebabkan oleh kelalaian pembuat dokumen.

BIOGRAFI PENULIS

Ilham Laenur Hikmat, lahir pada 22 April 1993 di Serang, Banten. Penulis lulus dari SMP Negeri 1 Kramatwatu pada tahun 2008 kemudian melanjutkan pendidikan ke SMA Negeri 1 Kramatwatu hingga akhirnya lulus pada tahun 2011. Penulis kemudian melanjutkan pendidikan Strata satu ke Jurusan Teknik Elektro ITS Surabaya bidang studi Teknik Komputer dan Telematika. Saat di kuliah penulis aktif menjadi tim robotika ITS 2012/2015. Penulis juga aktif menjadi Asisten laboratorium B201 (Telematika) hingga saat ini dan pernah menjabat sebagai koordiator praktikum rangkaian digital pada tahun 2014. Selama masa kuliah penulis aktif dalam mengikuti ajang perlombaan seperti *ImagineCup*, aktif dalam *development group embedded system*. Penulis sangat tertarik dengan segala hal yang berhubungan dengan komputer, dan berencana mendalami cabang ilmu komputer lain selain *embedded system*.

DAFTAR PUSTAKA

[1] A. A. Hakim, A. Erwin, K. Eng, M. Galinium, W. Muliady, et al., "Automated document classification for news article in bahasa indonesia based on term frequency inverse document frequency (tf-idf) approach," in Information Technology and Electrical Engineering (ICITEE), 2014 6th International Conference on, pp. 1–4, IEEE, 2014. (Dikutip pada halaman 1).

[2] B. Kurniawan, S. Effendi, and O. S. Sitompul, "Klasifikasi konten berita dengan metode text mining," Dunia Teknologi Informasi-Jurnal Online, vol. 1, no. 1, 2012. (Dikutip pada halaman 1).

[3] Y. Safeer, A. Mustafa, and A. N. Ali, "Clustering unstructured data (flat files)-an implementation in text mining tool," arXiv preprint arXiv:1007.4324, 2010. (Dikutip pada halaman 1, 3, 7).

[4] J. Ramos, "Using tf-idf to determine word relevance in document queries," in Proceedings of the first instructional conference on machine learning, 2003. (Dikutip pada halaman 1, 5).

[5] S. G. Manikandan and S. Ravi, "Big data analysis using apache hadoop," in IT Convergence and Security (ICITCS), 2014 International Conference on, pp. 1–4, IEEE, 2014. (Dikutip pada halaman 2, 9).

[6] D. Dev and R. Patgiri, "Performance evaluation of hdfs in big data management," in High Performance Computing and Applications (ICHPCA), 2014 International Conference on, pp. 1–7, IEEE, 2014. (Dikutip pada halaman 2).

[7] I. S. Dhillon, J. Fan, and Y. Guan, "Efficient clustering of very large document collections," in Data mining for scientific and engineering applications, pp. 357–381, Springer, 2001. (Dikutip pada halaman 3, 7).

[8] F. Z. Tala, "A study of stemming effects on information retrieval in bahasa indonesia," Institute for Logic, Language and

Computation Universeit Van Amsterdam, 2003. (Dikutip pada halaman xi, 6, 7).

[9] J. Asian, H. E. Williams, and S. M. Tahaghoghi, "Stemming indonesian," in Proceedings of the Twenty-eighth Australasian conference on Computer Science-Volume 38, pp. 307–314, Australian Computer Society, Inc., 2005. (Dikutip pada halaman 7).