BAB 2. LANDASAN TEORI

Pada bab ini dijelaskan mengenai semua teori yang digunakan di dalam Analisis Korelasi pada Data *Yahoo! Properties* dan *Instant Messaging* dengan menggunakan *Hadoop*. Berikut adalah hal-hal yang dijelaskan pada bab ini antara lain pengertian mengenai *Big Data*, Metode Korelasi, *Distributed Systems*, Data *Preprocessing*, *Apache Hadoop*, *Capacity scheduler*, dan data yang digunakan yaitu Data *Yahoo! Property* dan *Instant Messaging*. Berikut adalah penjelasan mengenai hal-hal tersebut.

2.1. Big Data

Konsep "*Big Data*" pertama kali dicetuskan oleh Roger Magoulas di dalam media *O'Riley* pada tahun 2005. *Big Data* menjelaskan bahwa data yang ada begitu besar dan banyak sehingga manajemen data tradisional tidak dapat digunakan lagi. Pandangan IBM terhadap *Big Data* dapat dibagi menjadi empat aspek, yaitu:

1. *Volume*

   Kuantitas data yang dimiliki oleh perusahaan. Data ini nantinya digunakan untuk menghasilkan pengetahuan yang penting.

2. *Velocity*

   Berapa lama waktu yang dibutuhkan untuk memproses *Big Data* tersebut. Beberapa aktivitas yang ada itu sangat penting dan membutuhkan *response* yang cepat. Untuk itu proses yang dilakukan harus di efisienkan.

3. *Variety*

   Meliputi tipe data apa saja yang ada di dalam *Big Data*. Data yang digunakan dapat terstruktur atau tidak terstruktur.

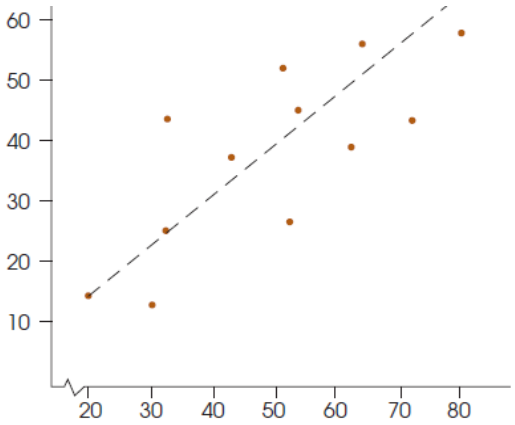
4. *Veracity*

   Seberapa percaya pemimpin perusahaan mengambil keputusan menggunakan hasil olahan data yang ada. Jadi, mencari korelasi yang tepat sangat penting bagi masa depan perusahaan.

2.2. Korelasi (Gravetter & Wallnau, 2013)

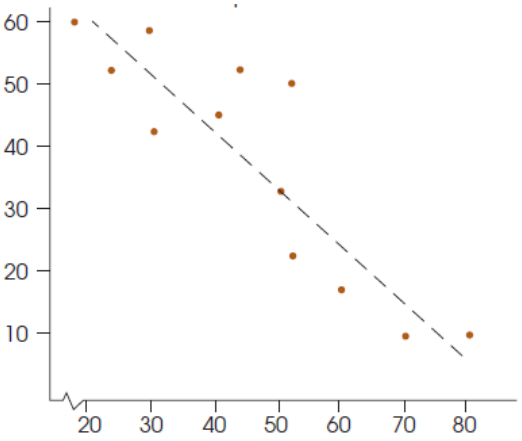
2.2.1. Pengertian nilai R

Nilai R adalah koefisien korelasi, yaitu derajat hubungan antar 2 variabel dan arah dari relasi tersebut, apakah relasi tersebut positif ataukah negatif. Koefisien korelasi dapat dilambangkan sebagai r. Interval dari r adalah $-1 < r < 1$. Semakin dekat angka r dengan angka 1 ataupun -1 maka dikatakan hubungan korelasi yang ada sangat dekat. Apabila angka mendekati 1 maka hubungan yang ada semakin positif, sedangkan semakin mendekati -1 maka hubungan korelasi yang ada semakin negatif. Sebaliknya apabila angka mendekati 0 maka hubungan semakin lemah dan bahkan tidak memiliki korelasi. Contoh hubungan korelasi yang positif dapat dilihat pada Gambar 2.1.



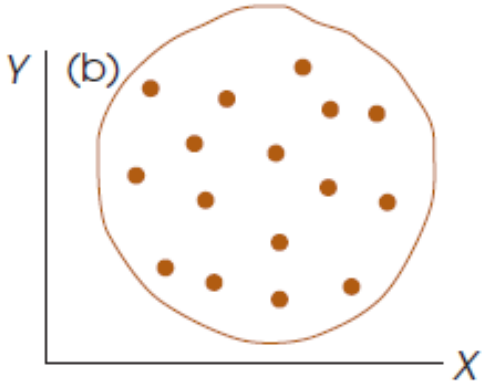
Gambar 2.1 Korelasi positif (Gravetter & Wallnau, 2013)

Korelasi positif ditandai dengan adanya garis *trend line* yang bisa ditarik ke arah kanan-atas. Untuk contoh korelasi yang negatif dapat dilihat pada Gambar 2.2.



Gambar 2.2 Korelasi negatif (Gravetter & Wallnau, 2013)

Korelasi negatif ditandai dengan adanya garis *trend line* yang bisa ditarik ke arah kanan-bawah. Nilai korelasi selain positif dan negatif juga dapat menghasilkan nilai 0 yang menandakan bahwa tidak ada hubungan antara variabel x dan y. Contoh nilai korelasi 0 dapat dilihat pada Gambar 2.3.



Gambar 2.3 Tidak ada korelasi (Gravetter & Wallnau, 2013)

Tidak adanya korelasi pada Gambar 2.3 dikarenakan tidak dapat ditarik garis *trend line* yang mendekati mayoritas data.

2.2.2. Metode Korelasi

2.2.2.1. Metode Pearson

Untuk melihat kedekatan hubungan korelasi antar data dapat dilihat melalui koefisien korelasi seperti yang dijelaskan pada 2.2.1. Terdapat beberapa macam metode untuk menghitung koefisien korelasi ini. Metode yang akan dibahas di dalam skripsi ini adalah metode *Pearson*, *Spearman*, dan *Point-Biserial*.

Metode *Pearson* menghitung korelasi secara linear antara 2 pasang variabel. Asumsi dari metode *Pearson* adalah variabel ditentukan dengan nilai level atau ratio, data yang ada terdistribusi normal. Metode ini dapat dituliskan sebagai berikut:

$$r = \frac{SP}{\sqrt{SSx * SSy}} \qquad (2.1)$$

Nilai SP dari persamaan 2.1 dapat dilihat pada persamaan 2.2.

$$SP = \sum XY - \frac{\sum X \sum Y}{n} \quad (2.2)$$

Nilai SSx dari persamaan 2.1, dapat dilihat pada persamaan 2.3.

$$SSx = \sum X^2 - \frac{(\sum X)^2}{n} \quad (2.3)$$

Nilai SSy dari persamaan 2.1, dapat dilihat pada persamaan 2.4.

$$SSy = \sum Y^2 - \frac{(\sum Y)^2}{n} \quad (2.4)$$

Dari persamaan 2.2,2.32.42.1 dapat dijabarkan menjadi persamaan 2.5

$$r = \frac{N \sum xy - \sum (x)(y)}{\sqrt{N \sum x^2 - \sum (x^2)][N \sum y^2 - \sum (y^2)]}} \quad (2.5)$$

Keterangan:

r = Koefisien korelasi *Pearson*

n = Banyak pasang dari variabel

∑xy = Jumlah perkalian variabel 1 dan 2

∑x = *sum of* variabel 1

∑y = *sum of* variabel 2

$\sum x^2$ = *sum of* kuadrat dari variabel 1

$\sum y^2$ = *sum of* kuadrat dari variabel 2

Contoh perhitungan dari Metode *Pearson* adalah pada Gambar 2.4.

Scores		Deviations		Squared Deviations		Products
X	Y	$X - M_X$	$Y - M_Y$	$(X - M_X)^2$	$(Y - M_Y)^2$	$(X - M_X)(Y - M_Y)$
0	2	−6	−2	36	4	+12
10	6	+4	+2	16	4	+8
4	2	−2	−2	4	4	+4
8	4	+2	0	4	0	0
8	6	+2	+2	4	4	+4
				$SS_X = 64$	$SS_Y = 16$	$SP = +28$

Gambar 2.4 Contoh perhitungan *Pearson* (Gravetter & Wallnau, 2013)

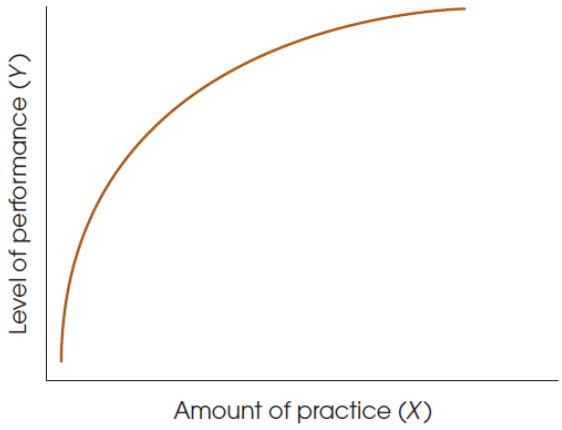
Dari nilai SS dan SP yang dihasilkan pada Gambar 2.4, maka dihasilkan:

$$r = \frac{28}{(64+16)} = 0.35$$

2.2.2.2. Metode *Spearman Rank*

Jika data yang ada tidak memenuhi asumsi dari *Pearson* maka digunakan metode *Spearman*. Metode *Spearman* adalah pengembangan dari Metode *Pearson* pada 2.2.2.1. Metode *Spearman* dapat digunakan dalam 2 kondisi, yaitu:

1. Kedua pasangan data yang digunakan adalah variabel yang dihitung dalam skala ordinal. Skala ordinal adalah nilai didapat dari *ranking* data bukan melalui nilai data.
2. Kedua pasangan data yang digunakan adalah data ratio atau interval, tetapi tidak memiliki hubungan yang linear. Seperti contoh adalah hubungan antara waktu berlatih dengan tingkat performa seseorang. Hubungan yang ada tidak dalam garis linear melainkan membentuk kurva, seperti pada Gambar 2.5.



Gambar 2.5 Kurva non linear antara waktu latihan dengan tingkat performa (Gravetter & Wallnau, 2013)

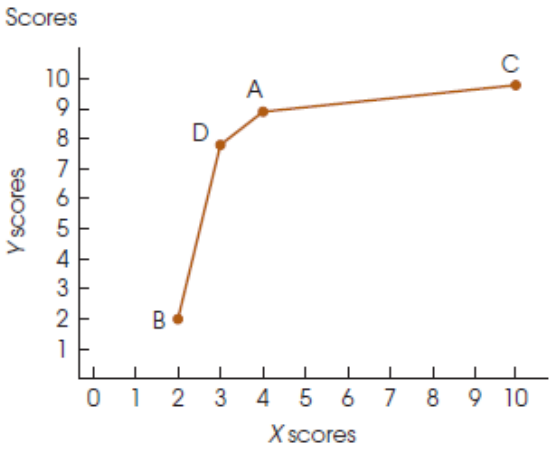
Jika nilai dari Gambar 2.5 dihitung dengan Metode *Pearson*, maka tidak akan menghasilkan nilai 1. Ini berbeda jika dihitung dengan Metode *Spearman* akan menghasilkan angka 1.

Metode *Spearman* menghitung seberapa konsisten arah korelasi dari data yang ada, independen dari bentuk grafiknya. Konsep Metode *Spearman* adalah jika dua buah data berelasi secara konsisten maka *ranking*nya akan berkorelasi

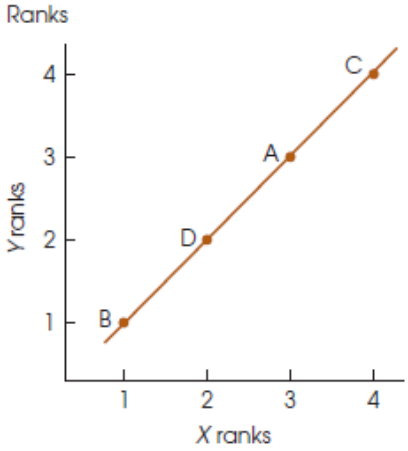
secara linear. Penjelasan konsep berikut dapat dilihat pada Gambar 2.6, Gambar 2.7, dan Gambar 2.8.

Person	*X*	*Y*	*X*–Rank	*Y*–Rank
A	4	9	3	3
B	2	2	1	1
C	10	10	4	4
D	3	8	2	2

Gambar 2.6 Contoh data yang digunakan (Gravetter & Wallnau, 2013)



Gambar 2.7 Grafik nilai dari nilai X dan Y (Gravetter & Wallnau, 2013)



Gambar 2.8 Grafik *ranking* dari nilai X dan Y (Gravetter & Wallnau, 2013)

Pada Gambar 2.7 dapat dilihat bahwa grafik tidak menunjukkan adanya relasi linear antara 2 data tersebut. Jika dihitung dengan metode *Pearson* maka nilai yang dihasilkan bukanlah 1. Sedangkan pada Gambar 2.8, ketika nilai tersebut diubah menjadi *ranking* maka grafik menunjukkan nilai linear antara kedua data. Inilah konsep yang digunakan oleh Metode *Pearson*.

Rumus dari metode *Spearman* adalah sama dengan *Pearson* pada persamaan 2.5, hanya saja nilai x dan y yang dimasukkan ke dalam pesamaan adalah hasil *ranking* data. Cara melakukan *ranking* dapat dilihat pada Gambar 2.6. Untuk perhitungan *ranking* data yang memiliki nilai yang sama dapat dilihat pada Gambar 2.9.

Scores	Rank Position	Final Rank	
3	1	1.5	Mean of 1 and 2
3	2	1.5	
5	3	3	
6	4	5	Mean of 4, 5, and 6
6	5	5	
6	6	5	
12	7	7	

Gambar 2.9 Cara merangking untuk nilai data yang sama

2.2.2.3. Metode *Point-Biserial*

Metode *Point-Biserial* digunakan untuk menghitung korelasi antara 2 variabel yang satu adalah variabel dengan nilai numerik, yang kedua adalah variabel *dichtonomous* atau binomial. Variabel *dichtonomous* adalah variabel yang hanya memiliki 2 nilai. Seperti contoh adalah pria dan wanita. Untuk menghitung korelasi dengan metode *Point-Biserial* maka perlu dilakukan konversi dari kategori ke nilai 0 atau 1. Misal pria adalah 0 dan wanita adalah 1. Setelah dilakukan konversi maka dilakukan perhitungan dengan rumus *Pearson* pada persamaan 2.5, dimana nilai X dan Y adalah data yang telah dikonversikan. Contoh pengaplikasian Metode *Point-Biserial* dapat dilihat pada Gambar 2.10.

Data for the Independent-Measures *t* test. Two separate samples, each with $n = 10$ scores.

Average High School Grade			
Watched Seasame Street		Did Not Watch Sesame Street	
86	99	90	79
87	97	89	83
91	94	82	86
97	89	83	81
98	92	85	92
$n = 10$		$n = 10$	
$M = 93$		$M = 85$	
$SS = 200$		$SS = 160$	

Data for the Point-Biserial Correlation. Two scores, *X* and *Y* for each of the $n = 20$ participants.

Participant	Grade *X*	Group *Y*
A	86	1
B	87	1
C	91	1
D	97	1
E	98	1
F	99	1
G	97	1
H	94	1
I	89	1
J	92	1
K	90	0
L	89	0
M	82	0
N	83	0
O	85	0
P	79	0
Q	83	0
R	86	0
S	81	0
T	92	0

Gambar 2.10 Contoh konversi nilai data untuk perhitungan *Spearman*

Pada Gambar 2.10, untuk anak yang menonton *sesame street* dimasukkan ke dalam *group* dengan nilai 1 dan untuk yang tidak menonton *sesame street* dimasukkan ke dalam *group* dengan nilai 0. Nilai X yang digunakan adalah nilai sekolah dan Y yang digunakan adalah nilai *group*. Setelah itu dilakukan perhitungan dengan persamaan 2.5.

2.2.3. Tes Signifikansi

Tes signifikansi pada koefisien korelasi digunakan untuk testing hipotesa, dimana H0 adalah tidak ada korelasi atau $r = 0$ dan H1 adalah ada korelasi atau $r \neq 0$. Untuk perhitungan signifikansi maka dicari nilai *P-Value*. Cara Perhitungan *P-Value* dimulai dari mencari nilai t. Rumus untuk mencari nilai t dapat dilihat pada persamaan 2.6

$$t = \left| \frac{r\sqrt{n-2}}{\sqrt{1-r^2}} \right| \quad (2.6)$$

Penjelasan:

r = koefisien korelasi

n = jumlah data

Setelah didapatkan nilai t maka dicari nilai *P-Value* dengan cara mencari nlai distribusi t dengan df = n-2. Kemudian *P-Value* didapatkan dengan persamaan 2.7.

$$P = 2 * (1 - Cumulative\ Distribution(t)) \quad (2.7)$$

Persamaan di atas menggunakan *two-tailed test*. Pengecekan nilai signifikansi berdasarkan level signifikansi. Terdapat 3 level signifikansi yaitu 0.001, 0.01, dan 0.05. Jika nilai signifikansi adalah 0.04 artinya adalah 4% kemungkinan melakukan type I error. Jadi 4% kemungkinan kesalahan ternyata H0 ditolak. Semakin kecil nilai signifikansi maka semakin kecil kesalahan ini terjadi.

2.2.4. Standard Error dari Koefisien Korelasi

Standard error menunjukkan seberapa presisi nilai dari koefisien korelasi. Semakin kecil nilai *standard error* maka semakin presisi. Persaman untuk menghitung *standard error* dapat dilihat pada persaman 2.8

$$se_r = \sqrt{\frac{1-r^2}{n-2}} \quad (2.8)$$

Penjelasan:

r = koefisien korelasi

n = jumlah data

2.3. *Classification and Regression Tree (CART)*

Classification Tree adalah sebuah *decision tree* yang dibuat melalui perhitungan gini Index dari sebuah data. *Classification tree* dengan algoritma CART dapat dijelaskan sebagai berikut:

1. Algoritma dimulai dari *node* root
2. Dilakukan perhitungan Gini untuk root
3. Untuk setiap kemungkinan *split* data hitunglah Gini Indexnya kemudian cari Gini Index yang paling minimum/yang memberikan delta gini maksimum.
4. Pecah *node* berdasarkan kriteria yang didapatkan dari langkah 3. Jika *node* anak telah mencapai Gini Index 0 atau tidak ada kemungkinan lagi untuk melakukan perpecahan maka proses berhenti. Jika tidak, maka dilakukan kembali langkah nomor 2.

Gini Index adalah proporsi ketidakrataan data. Rumus gini index untuk satu *node* dapat dilihat pada persamaan 2.9.

$$gini\ (D) = 1 - \sum_{j=1}^{n} p_j^2 \qquad (2.9)$$

Setelah dilakukan perhitungan Gini Index sebuah *node* maka dilakukan perhitungan Gini Index untuk hasil perpecahan, dengan menggunakan persamaan 2.10

$$gini_A(D) = \frac{|D_1|}{|D|} gini(D_1) + \frac{|D_2|}{|D|} gini(D_2) \qquad (2.10)$$

Setelah ditemukan nilai Gini Index dari persamaan 2.9 dan nilai Gini Index Split dari persamaan 2.10, maka dilakukan pencarian delta gini yang didapatkan dari persamaan 2.11

$$\Delta gini(A) = gini(D) - gini_A(D) \qquad (2.11)$$

2.4. Distributed Systems

Menurut Kshemkalyani & Singhal (2011), *Distributed System* pada sistem komputer adalah kumpulan proses yang terotomatisasi, berkomunikasi di atas sebauh jaringan yang memiliki fitur tidak ada proses *clock* secara fisik, tidak ada *shared memory*, prosesor yang digunakan *loosely coupled*, otomatis dan heterogen. Motivasi untuk menggunakan *distributed system* adalah untuk :

1. Komputasi terdistribusi yang inheren
2. Berbagi *resource*

3. Akses ke data dan *resource* yang jauh
4. Meningkatkan reliability dari sistem
5. Meningkatkan performa
6. Skalibilitas
7. *Modularity* (Kemudahan menambah prosesor tanpa mempengaruhi sistem

2.5. Data Preprocessing

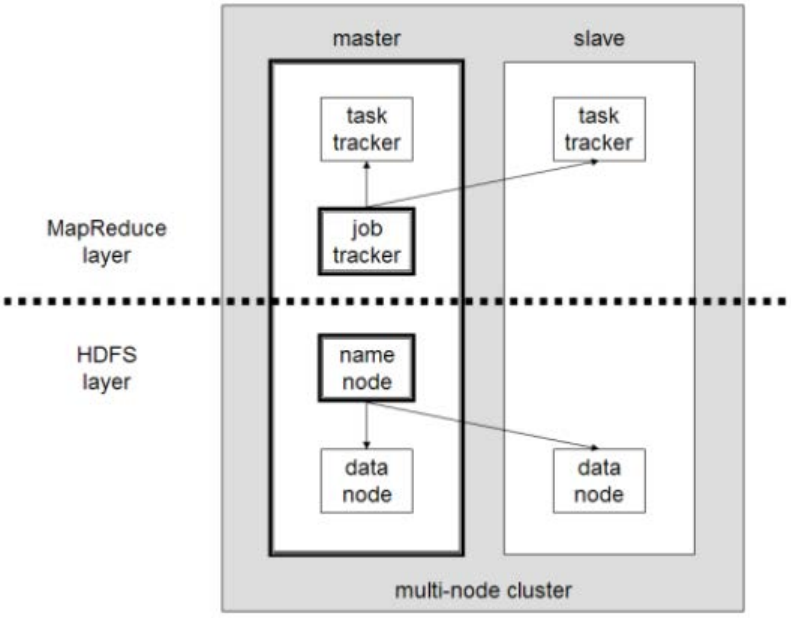
Data *raw* yang didapatkan di dunia nyata biasanya belum bersih. Masalah yang ada di dalam data:

- Tidak lengkap : Ada atribut yang tidak memiliki nilai
- Memiliki *noise* : memiliki *error* atau data yang asing dari data yang lain
- Tidak konsisten : Memiliki perbedaan pengkodean di beberapa data

Masalah – masalah ini dapat menimbulkan hasil yang tidak konsisten pada analisa data. Oleh karena itu perlu dilakukan *preprocessing* data. Salah satu tugas dari *preprocessing data* adalah *data cleaning*. Tugas dari *data cleaning* adalah mengisikan nilai – nilai yang kosong, mengurangi *noise* pada data, menghilangkan data yang asing, dan menyelesaikan ketidak konsistenan pada pengkodean.

2.6. Apache Hadoop

Hadoop dibuat oleh Doug Cutting, pembuat *Apache Lucene*. *Hadoop* adalah sebuah *framework* yang memungkinkan *Distributed processing* diberbagai *cluster* yang terpisah. *Hadoop* bekerja dalam *environment Java*. Arsitektur dari *Hadoop* dapat dilihat pada Gambar 2.11



Gambar 2.11 Arsitektur *Hadoop* Ward (1997, p.5)

Hadoop terdiri dari 2 bagian, yaitu HDFS (*Hadoop Distributed File System*) *File System* dan *MapReduce*. HDFS terdiri dari *name node* dan *data node*, sedangkan *MapReduce* terdiri dari *job tracker* dan *task tracker*. Pada saat ini, *Hadoop* adalah *System* tercepat untuk mengolah data dalam *Terabyte*. Menurut Maitreya & Jhab (2015), *Hadoop* dapat dijalankan dengan 3 macam cara, yaitu :

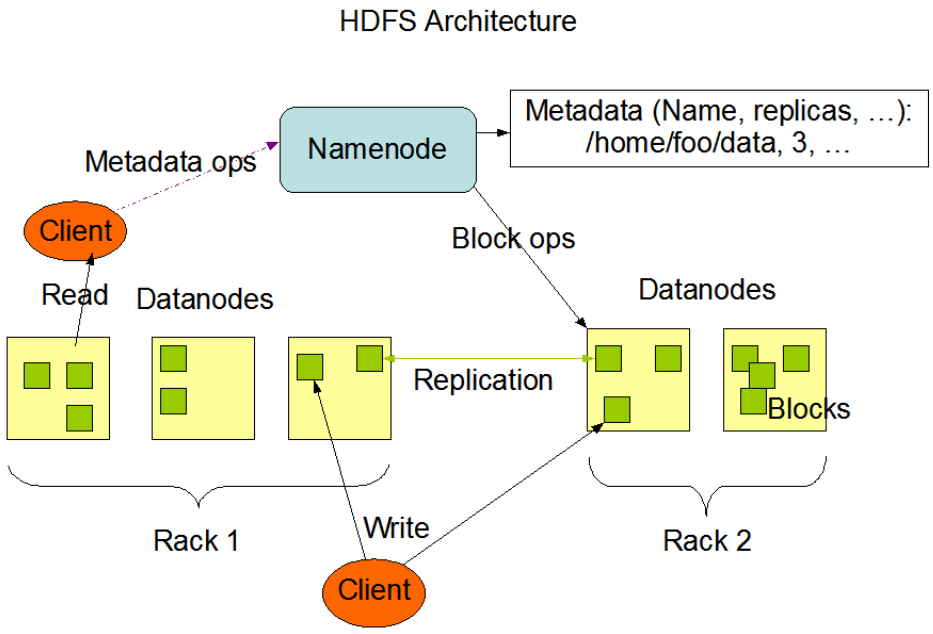
- *Standalone mode* : *default mode* yang disediakan oleh *Hadoop*. Semua dijalankan melalui sebuah proses *Java*.
- *Pseudo – Distributed mode* : *Hadoop* dikonfigurasi untuk berjalan pada sebuah mesin, akan tetapi dengan beberapa *Hadoop Daemon* yang berjalan sebagai proses *Java* yang berbeda.
- *Fully distributed* atau *cluster mode* : Di sini sebuah mesin di sebuah *cluster* yang dilabel *Namenode* dan yang lain sebagai *JobTracker*. Hanya satu *Namenode* yang diletakkan dalam sebuah *cluster*. *Namenode* ini bertugas me*manage namespace*, metadata *FileSystem*, dan kontrol akses. *SecondaryNamenode* bisa diletakkan untuk proses *handshaking* secara periodik dengan *Namenode* untuk toleransi kegagalan. Semua mesin di dalam *cluster* bekerja sebagai *DataNode* dan *TaskTracker*. *DataNode* menyimpan data sistem. Tiap *DataNode* menyimpan local storage masing – masing. *TaskTracker* yang bertugas menjalankan operasi *Map* dan *Reduce*.

Pada tugas akhir ini digunakan *fully distributed mode.*

2.7. Hadoop File System

Ketika sebuah *dataset* ukurannya telah melebihi kapasitas penyimpanan dari sebuah komputer maka diperlukan adanya partisi di komputer yang lain. *FileSystem* yang mengatur penyimpanan di jaringan mesin yang ada disebut dengan *Distributed FileSystem*. Dikarenakan *Distributed FileSystem* menggunakan beberapa komponen dari *networking*, *FileSystem* ini lebih kompleks daripada *disk FileSystem* biasanya. Salah satunya adalah kesulitan membuat *FileSystem* dapat menoleransi kegagalan *node* tanpa menderita *data loss*. Pertama kali *Distributed FileSystem* digunakan oleh GFS (*Google File System*).

Hadoop memiliki *Distributed FileSystem* yang disebut *Hadoop Distributed FileSystem* (HDFS). HDFS di model berdasarkan GFS (*Google File System*). *FileSystem* ini didesain untuk menyimpan data yang sangat besar dengan akses data yang berpola. Arsitektur dari HDFS dapat dilihat pada Gambar 2.12



Gambar 2.12 Arsitektur HDFS (*Apache*, 2016)

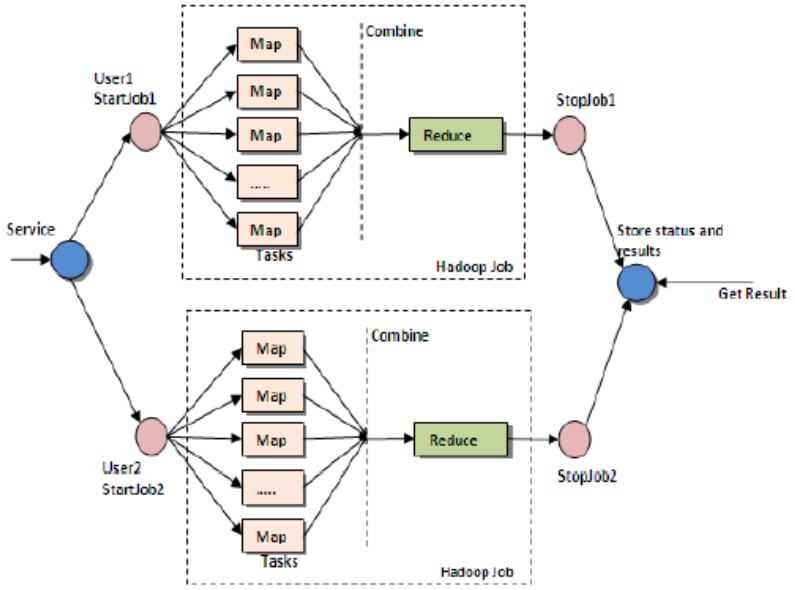
HDFS memiliki beberapa fitur utama yaitu (Turkington, 2013):

a. HDFS menyimpan datanya dalam bentuk blok yang besarnya secara umum 64 MB yang lebih besar daripada 4-32 kb yang ditemukan dalam *FileSystem* secara umum.

b. HDFS dioptimasi untuk kecepatan transfer data. HDFS sangat efisien untuk membaca *request* data yang sangat besar, tetapi kurang bagus untuk membaca *request* data yang sedikit.
c. Sebuah *cluster* HDFS memiliki 2 tipe *node* yang bekerja dalam arsitektur *master-slave*. *Namenode* adalah *node* yang bekerja sebagai *master* dan beberapa *datanode* adalah *node* yang bekerja sebagai *worker*. *Namenode* mengatur *namespace* dari *FileSystem*. *Namenode* juga bertugas untuk memaintain *tree FileSystem* dan *metadata* dari semua *File* dan direktori dari *tree*. Semua informasi ini disimpan ke dalam 2 bentuk *File*, yaitu *namespace image* dan *edit log*. *Namenode* memiliki replika sebagai *backup* yang disebut *Secondary Namenode*. *Datanode* bekerja untuk menyimpan dan mengambil blok ketika diperintah, kemudian melakukan *report* kepada *Namenode* tentang *list* blok yang akan disimpan.
d. HDFS menggunakan replikasi data sebagai data redundan untuk mengatasi kegagalan *disk*. Data direplikasi ke *harddisk* yang ada secara fisik. Tiap blok yang menyimpan *File* disebarkan ke banyak *node* di dalam satu *cluster*. HDFS *Namenode* secara konstan melakukan *monitoring* terhadap report yang diberikan oleh *datanode* untuk memastikan bahwa kegagalan yang ada tidak menghilangkan blok sampai jumlahnya di bawah faktor replikasi.

2.8. MapReduce

MapReduce adalah sebuah model *programming* sederhana untuk memproses data. Program yang mengimplementasikan *MapReduce* dapat berjalan secara paralel sehingga dapat menyelesaikan analisis data dengan ukuran yang sangat besar. *MapReduce* terdiri dari 2 bagian yaitu fase *map* dan fase *reduce*. Kedua fase memiliki sepasang *key* dan *value* sebagai *input* dan *output* yang tipenya dipilih oleh *programmer*. *Programmer* harus menentukan juga isi fungsi *map* dan fungsi *reduce*. Arsitektur *MapReduce* pada *Hadoop* dapat dilihat pada Gambar 2.13



Gambar 2.13 Arsitektur *MapReduce* pada *Hadoop*

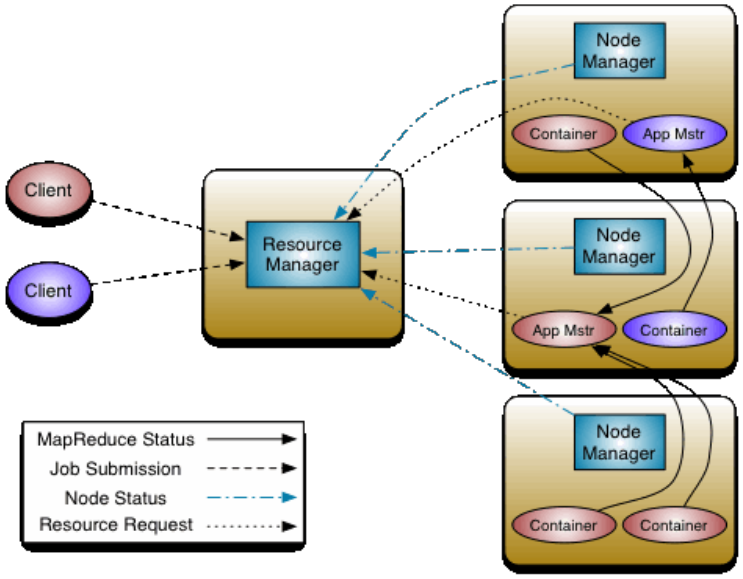
Di antara *Mapper* dan *Reducer* terdapat *combiner* yang merupakan *semi-Reducer*. *Combiner* melakukan proses data dari *output* data *map* dan kemudian diberikan kepada *Reducer*. Ketika menjalankan *MapReduce* maka *Hadoop* akan menjalankan *job*. Sebuah data dapat memiliki banyak *job* yang disebarkan ke tiap *cluster*. *Job* akan melakukan *mapping* yang disebut proses *task*. Hasil dari *Map* akan berupa *pairing key* dan *value*. Hasil dari banyak *map*ping akan digabungkan kemudian dilakukan *reduce*. Setelah *task Reduce* selesai maka *job* dari tiap *cluster* akan disatukan menghasilkan *result*.

2.9. Hadoop YARN (Yet Another Resource Negotiator)

Hadoop YARN (*Yet Another Resource Negotiator*) sering disebut sebagai *MapReduce* 2.0. Tujuan dari YARN adalah membuat *framework* di atas *Hadoop* untuk memperbolehkan *resource* dari sebuah *cluster* untuk diberikan ke beberapa aplikasi dimana *MapReduce* adalah salah satu aplikasinya.

Pada saat sebelum pengembangan dari YARN, *Job Tracker* yang ada bertanggung jawab terhadap dua tugas, yaitu me-*manage* proses dari sebuah *job MapReduce*, mengidentifikasi *resource* dari sebuah *cluster* yang tersedia sekarang dan mengalokasikan *resource* ke beberapa *stage* dari *job*. YARN membagi tugas – tugas ke dua peran berbeda yaitu *global resource manager* yang menggunakan *node manage*r tiap *host* untuk me*manage resource* yang ada dan *Application Manage*r yang berkomunikasi dengan *resource manager* untuk mendapatkan

resource yang dibutuhkan oleh *job*. Arsitektur dari YARN dapat dilihat pada Gambar 2.14



Gambar 2.14 Arsitektur dari YARN (*Apache*, 2016)

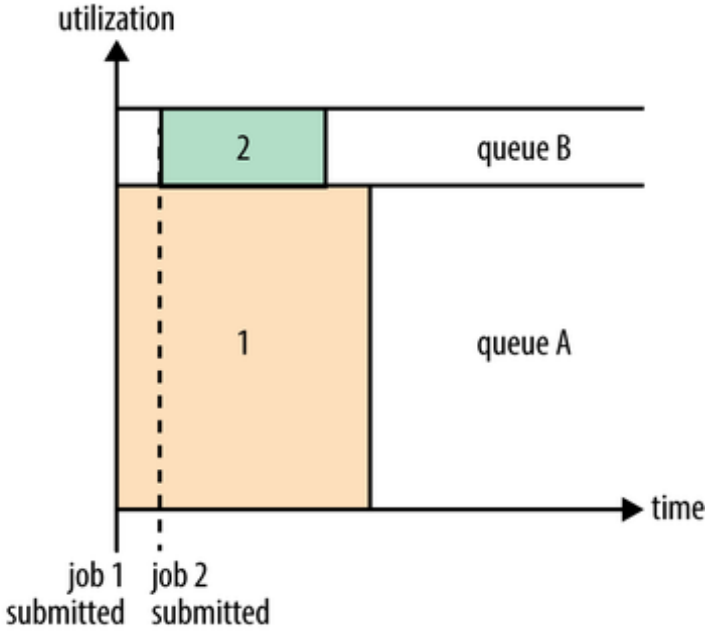
*Resource manage*r terdiri dari dua komponen utama yaitu *Scheduler* dan *ApplicationManage*r. *Scheduler* bertugas untuk mengalokasikan *resource* ke aplikasi yang sedang berjalan mengikuti antrian, kapasitas, dan lain – lain. *Scheduler* tidak ikut serta dalam melakukan *monitoring* dan *tracking* status dari aplikasi. *Scheduler* juga tidak ikut dalam memberikan garansi untuk melakukan *restart* tugas yang gagal dikarenakan kegagalan aplikasi atau perangkat keras. *Scheduler* menjalankan fungsi schedulingnya berdasarkan kebutuhan *resource* aplikasi bukan berdasarkan elemen *memory*, *cpu*, *disk*, *network*, dan lain – lain. *ApplicationManage*r memiliki fungsi untuk menerima *job* yang di*submit*, bernegosiasi dengan kontainer pertama untuk menjalankan aplikasi *ApplicationMaster* dan melakukan *restart* pada kontainer *ApplicationMaster* jika mengalami kegagalan. Tiap *ApplicationMaster* memiliki tugas untuk bernegosiasi dengan *resource container* di *scheduler* dan melakukan *tracking* dan *monitoring progress* aplikasi.

2.9.1. Hadoop *Capacity Scheduler*

Untuk *job scheduler* pada *Hadoop* sebenarnya terdapat 3 macam, yaitu FIFO *Scheduler*, *capacity scheduler*, dan *fair scheduler*. Untuk *scheduler* yang dipakai dalam skripsi ini adalah *Scheduler default* dari *Hadoop* yaitu *capacity*

scheduler. *Capacity scheduler* memungkingkan pembagian *cluster Hadoop* dalam suatu organisasi, dimana tiap bagian organisasi memerlukan kapasitas *cluster Hadoop* tertentu. Konsep dari *capacity scheduler* adalah disediakan *queue-queue* khusus. Tiap *queue* ditujukan untuk satu bagian organisasi. Besar sebuah *queue* adalah sebagian dari kapasitas total *cluster Hadoop*. *Queue* yang ada dapat dibuat hirarki, dimana *queue* tersebut dapat dibagi ke *user* atau grup dalam hirarki *queue* tersebut. Di dalam *queue*, *job* dikerjakan menurut FIFO (*First in First Out*).

Pada *capacity scheduler* sebuah *job* dalam *queue* tidak akan mengambil lebih dari kapasitas *queue*. Akan tetapi, jika terdapat lebih dari 1 *job* di dalam *queue* dan ada *resource* yang tidak digunakan, maka *resource* yang tidak digunakan tersebut dapat diberikan kepada *job-job* yang ada di dalam *queue*, meskipun nantinya jumlah *resource* yang dialokasikan ke dalam *queue* melebihi dari kapasitas maksimum *queue*. Keadaan ini dinamakan *queue elasticity*. Misal jika ada sebuah *job* yang besar berjalan maka dia akan dimasukkan ke dalam *queue* A, ketika *job* 2 masuk setelah *job* 1 telah berjalan beberapa saat, maka *job* 2 akan otomatis masuk ke dalam *queue* B. Agar lebih jelas dapat dilihat pada Gambar 2.15.



Gambar 2.15 Visualisasi *Capacity scheduler*

Kedua job akan berjalan secara paralel, sehingga *job* 2 akan selesai kemudian *job* 1 akan selesai. Kelemahan dari *Capacity scheduler* adalah ada memori yang dialokasikan untuk *queue-queue* tersebut.

2.10. Yahoo! Research

Menurut (*Yahoo!*, n.d.), *Yahoo! Research* adalah organisasi untuk *research* milik *Yahoo!*. Fokus utamanya adalah riset dalam bidang *internet search*, *machine learning*, ekonomi mikro, dan lain – lain. *Yahoo! Research* memberikan *dataset* yang dapat di*download* secara gratis untuk kepentingan riset atau edukasi. Data yang disediakan adalah data asli milik *Yahoo!*.

2.10.1. Data Yahoo! Property dan Instant Messenger

Dataset ini didapatkan dari Webscope milik *Yahoo!* Research[1] bagian G7. *Dataset* ini merupakan data dari 1 Oktober 2007 – 30 Oktober 2007. Untuk user ID yang dicatat di anonymized. *Dataset* ini terdiri dari 31 *File* yang dihost di AWS (*Amazon Web Service*). Berikut adalah isi dari *dataset* ini:

1. netuser_userdata.dat
2. netuser_im_step1_01_28.dat
3. netuser_im_step2_01_07.dat
4. netuser_im_step2_08_14.dat
5. netuser_im_step2_15_21.dat
6. netuser_im_step2_22_28.dat
7. netuser_pcnetwk_01_07.dat
8. netuser_pcnetwk_08_14.dat
9. netuser_pcnetwk_15_21.dat
10. netuser_pcnetwk_22_28.dat
11. netuser_ipctry_01_07.dat
12. netuser_ipctry_08_14.dat
13. netuser_ipctry_15_21.dat
14. netuser_ipctry_22_28.dat
15. netuser_fp_mail_search_01_07.dat
16. netuser_fp_mail_search_08_14.dat

[1] https://webscope.sandbox.yahoo.com/catalog.php?datatype=g

17. netuser_fp_mail_search_15_21.dat
18. netuser_fp_mail_search_22_28.dat
19. netuser_otherprop_01_07.dat
20. netuser_otherprop_08_14.dat
21. netuser_otherprop_15_21.dat
22. netuser_otherprop_22_28.dat
23. netuser_mobile_01_07.dat
24. netuser_mobile_08_14.dat
25. netuser_mobile_15_21.dat
26. netuser_mobile_22_28.dat
27. netuser_ygo_01_07.dat
28. netuser_ygo_08_14.dat
29. netuser_ygo_15_21.dat
30. netuser_ygo_22_28.dat
31. netuser_mobile_flicker_errata.dat
32. netuser_mw_device_catalog.txt
33. netuser_ygo_device_catalog.txt

Berikut detail dari atribut data tersebut:

a. User Data *File*

FILE: (LINE '\n') +

LINE: USERID '\t' NONYHOO '\t' SYSID '\t' *LIST*ID '\t' PRIORGO '\t' HASREG '\t' GENDER '\t' AGEYR '\t' COUNTRY

USERID:	Integer	#Anonymized User ID
SYSID:	Integer	#Flag indicating *System* IDs
*LIST*ID:	Integer	#Code indicating seed, step1, step2)
PRIORGO:	Integer	#Most recent Go use (YYYYMMDD)
HASREG:	Integer	#Flag indicting missing reg data
GENDER:	Integer	#1 = male, 0 female, -1 N/A
AGEYR:	Integer	#Age in years as of 10/1 for users 18 yrs or older
COUNTRY:	Integer	#Code for registration country

b. IM Use *File*

FILE: (LINE '\n') +

LINE: DATE '\t' EGOID '\t' ALTERID '\t' ACTIVITY_CODE '\t' COUNT

DATE:	Integer	#October date in DD format
EGOID:	Integer	#Anonymized user ID
ALTERID:	Integer	#Anonymized user ID
ACTIVITY_CODE:	Character	#Code for action ("S", "R", "A")
COUNT:	Integer	#Number of times action occurred

c. PC *Network* Use *File*

FILE: (LINE '\n') +

LINE: DATE '\t' USERID '\t' YPV

DATE:	Integer	#October date in DD format
USERID:	Integer	#Anonymized user ID
PCPV	Integer	#Total PV on Yahoo using PC (non-mobile)

d. Country via IP Lookup *File*s

FILE: (LINE '\n') +

LINE: DATE '\t' USERID '\t' COUNTRY1 '\t' PCTPV1 '\t' COUNTRY2 '\t' PCTPV2

DATE	Integer	#October date in DD format
USERID	Integer	#Anonymized user ID
COUNTRY1	Integer	#Code for primary country
PCTPV1	Float	#PV from primary country (%)
COUNTRY2	Integer	#Code for secondary country
PCTPV2	Float	#PV from secondary country (%)

e. PC Front Page, Mail, Search Use *File*s

FILE: (LINE '\n') +

LINE: DATE '\t' USERID '\t' FPPV '\t' MAILPV '\t' SEARCHPV

DATE:	Integer	#October date in DD format

USERID:	Integer	#Anonymized user ID
FPPV	Integer	#PC PV on Front Page
MAILPV	Integer	#PC PV on Mail
SEARCHPV	Integer	#PC PV on Search

f. PC Other *Property* Use *File*

FILE: (LINE '\n') +

LINE: DATE '\t' USERID '\t' WTHRPV '\t' NEWSPV '\t' FINPV '\t' SPORTSPV '\t' FLICKRPV

DATE:	Integer	#October date in DD format
USERID:	Integer	#Anonymized user ID
WTHRPV:	Integer	#PC PV on Yahoo Weather
NEWSPV:	Integer	#PC PV on Yahoo News
FINPV:	Integer	#PC PV on Yahoo Finance
SPORTSPV:	Integer	#PC PV on Yahoo Sports
FLICKRPV:	Integer	#PC PV on Flickr

g. Mobile Web Use *File*

FILE: (LINE '\n') +

LINE: DATE '\t' USERID '\t' DEVID '\t' PCTPV '\t' MWPV '\t' FPPV '\t' MAILPV '\t' IMPV '\t' SRCHPV '\t' WTHRPV '\t' NEWSPV '\t' FINPV '\t' SPORTSPV '\t' FLICKRPV

DATE:	Integer	#October date in DD format
USERID:	Integer	#Anonymized user ID
DEVID:	Integer	#Code for primary device
PCTPV:	Integer	#PV on primary device (%)
Mobile WebPV:	Integer	#Total Mobile Web Page View (non-Y!Go)
FPPV:	Integer	#Mobile Web Front Page Page View
MAILPV:	Integer	#Mobile Web Mail Page View
IMPV:	Integer	#Wap Messenger Page View
SRCHPV:	Integer	#Mobile Web OneSearch Page View

WTHRPV:	Integer	#Mobile Web Weather Page View
NEWSPV:	Integer	#Mobile Web NewsPage View
FINPV:	Integer	#Mobile Web Finance Page View
SPORTSPV:	Integer	#Mobile Web Sports Page View
FLICKRPV:	Integer	#Mobile Web Flickr Page View

h. *Yahoo!* Go Use *File*

FILE: (LINE '\n') +

LINE: DATE '\t' USERID '\t' DEVID '\t' PCTPV '\t' GOPV '\t' FPPV '\t' MAILPV '\t' SRCHPV '\t' WTHRPV '\t' NEWSPV '\t' FINPV '\t' SPORTSPV '\t' FLICKRPV

DATE:	Integer	#October date in DD format
USERID:	Integer	#Anonymized user ID
FIRSTGO:	Integer	#Flag indicating first Y!Go use in data set
DEVID:	Integer	#Code for primary device
PCTPV:	Integer	#PV on primary device (%)
GOPV:	Integer	#Total Go PV
FPPV:	Integer	#Go Front Page PV
MAILPV:	Integer	#Go Mail PV
SRCHPV:	Integer	#Go OneSearch PV
WTHRPV:	Integer	#Go Weather PV
NEWSPV:	Integer	#Go News PV
FINPV:	Integer	#Go Finance PV
SPORTSPV:	Integer	#Go Sports PV
FLICKRPV:	Integer	#Go Flickr PV

i. Mobile Web Flickr Errata *File*

FILE: (LINE '\n') +

LINE: USERID '\t' FLICKRDELTA

USERID:	Integer	#Anonymized user ID
FLICKRDELTA:	Integer	#Number of mobile Flickr PV 10/1-10/9