

## MODUL 4 – DISTRIBUTED PROCESSING

---

### 1.1. Deskripsi Singkat

Pemrosesan Data Terdistribusi (*Distributed Data Processing*) merupakan suatu proses pemrosesan data di beberapa komputer dalam sistem terdistribusi. Proses ini memungkinkan data diproses lebih cepat dan lebih efisien karena beban kerja tersebar di banyak komputer. Data akan dibagi menjadi segmen-segmen kecil dan diproses secara paralel di beberapa node. Hasilnya kemudian digabungkan untuk menghasilkan output akhir. Apache Hadoop menawarkan keuntungan pemrosesan big data secara terdistribusi dengan adanya komponen MapReduce di dalamnya. MapReduce adalah *framework* perangkat lunak dan model pemrograman yang digunakan untuk memproses data dalam jumlah besar. Program MapReduce bekerja dalam dua fase, yaitu *Map* dan *Reduce*. Tugas *Map* berurusan dengan *splitting* dan *mapping* dari input data sedangkan tugas *Reduce* melakukan *shuffling* dan *reducing* terhadap data. Hadoop mampu menjalankan program MapReduce yang ditulis dalam bahasa Java, atau bahasa lain seperti Ruby, Python, dan C++ menggunakan Hadoop Streaming.

Pengenalan Hadoop versi 2 mengubah cara aplikasi MapReduce berjalan di sebuah cluster. Manajemen sumber daya cluster telah dipindahkan dari MapReduce ke YARN, sehingga memungkinkan aplikasi lain memanfaatkan YARN dan Hadoop. Dengan YARN, banyak aplikasi dapat dijalankan di Hadoop, semuanya berbagi manajemen sumber daya yang sama.

### 1.2. Tujuan Praktikum

Sebelum melakukan praktikum, mahasiswa diasumsikan telah memiliki pengetahuan pemrograman Java dan dasar perintah di Linux OS. Setelah praktikum pada modul 4 ini diharapkan mahasiswa mempunyai kompetensi dapat membuat program MapReduce sederhana dan mengeksekusi Job tersebut di Hadoop.

### 1.3. Material Praktikum

Pada kegiatan modul 5 diperlukan beberapa material, yaitu:

- 1) Internet Browser (direkomendasikan Google Chrome) atau MobaXterm untuk mengakses sistem Hadoop
- 2) HDP yang telah terinstal pada VirtualBox dan dapat diakses

- 3) IDE pemrograman Java (jika diperlukan)
- 4) Input data untuk MapReduce (booth.txt) dan input data penugasan (salaryinfo.txt)
- 5) File program Java (WordCount.java)

## 1.4. Kegiatan Praktikum

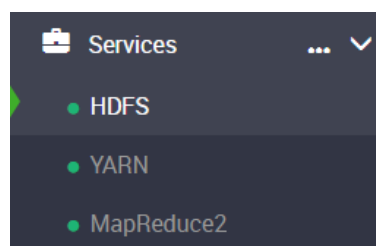
### A. Persiapan

Sebelum membuat dan menjalankan Job MapReduce, kita memastikan bahwa Hortonworks Data Platform (HDP) Sandbox telah berjalan dan dapat diakses.

1. Pada VirtualBox, lakukan start pada HDP Sandbox
2. Pastikan services berikut ini telah berstatus *started* (dapat dilihat di Ambari).

MapReduce membutuhkan services yaitu

- HDFS
- MapReduce2

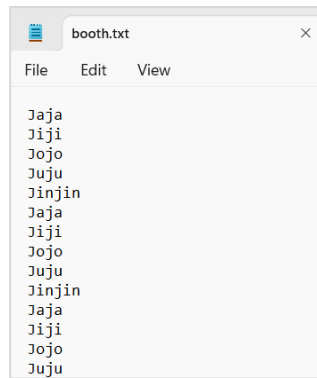


3. Untuk kegiatan praktikum ini kita akan menggunakan user **root**, kita dapat mengakses terminal SSH sistem HDP Sandbox menggunakan MobaXterm (Remote host 127.0.0.1 dan port 2222) atau di web browser <http://localhost:4200>
4. Kita buat folder di dalam folder `/root`, kita beri nama `tpdmr` untuk menyimpan program java dan file input data

```
# mkdir tpdmr
```

### B. Studi Kasus: Word Count

Pada studi kasus ini, kita akan membuat suatu aplikasi untuk menghitung hasil voting dari data sebagai berikut. Dari data tersebut kita dapat mengimplementasikan aplikasi MapReduce untuk menghitung jumlah kata.



1. Buat file program Java yang terdiri dari Class Map dan Reduce serta fungsi utama main() yang akan kita gunakan untuk menghitung jumlah kata (Word Count), misal kita beri nama **WordCount.java** (Class Map dan Class Reduce dapat disimpan dalam file .java terpisah)

```
//Standard Java imports
import java.io.IOException;
import java.util.Iterator;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import java.util.StringTokenizer;

//Hadoop imports
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.mapred.TextInputFormat;
import org.apache.hadoop.mapred.TextOutputFormat;

public class WordCount
{
    //The Mapper
    public static class Map extends MapReduceBase implements
Mapper<LongWritable, Text, Text, IntWritable>
    {
        private static final IntWritable accumulator = new
IntWritable(1);
        private Text word = new Text();

        public void map(LongWritable key, Text value,
OutputCollector<Text, IntWritable> collector, Reporter reporter)
throws IOException
        {
            String line = value.toString();
```

```

        StringTokenizer tokenizer = new StringTokenizer(line);
        while (tokenizer.hasMoreTokens()) {
            word.set(tokenizer.nextToken());
            collector.collect(word, accumulator);
        }
    }
}

//The Reducer
public static class Reduce extends MapReduceBase implements
Reducer<Text, IntWritable, Text, IntWritable>
{
    public void reduce(Text key, Iterator<IntWritable> values,
OutputCollector<Text, IntWritable> collector, Reporter reporter)
throws IOException
    {
        int count = 0;

        //code to aggregate the occurrence
        while(values.hasNext())
        {
            count += values.next().get();
        }
        System.out.println(key + "\t" + count);

        collector.collect(key, new IntWritable(count));
    }
}

//The java main method to execute the MapReduce job
public static void main(String[] args) throws Exception
{
    //Code to create a new Job specifying the MapReduce class
    final JobConf conf = new JobConf(WordCount.class);
    conf.setOutputKeyClass(Text.class);
    conf.setOutputValueClass(IntWritable.class);
    conf.setMapperClass(Map.class);
    //Combiner is commented out - to be used in bonus activity
    //conf.setCombinerClass(Reduce.class);
    conf.setReducerClass(Reduce.class);
    conf.setInputFormat(TextInputFormat.class);
    conf.setOutputFormat(TextOutputFormat.class);
    //File Input argument passed as a command line argument
    TextInputFormat.setInputPaths(conf, new Path(args[0]));
    //File Output argument passed as a command line argument
    TextOutputFormat.setOutputPath(conf, new Path(args[1]));
    //statement to execute the job
    JobClient.runJob(conf);
}
}

```

2. Pada Class Map kita akan membaca sebuah input data bertipe Text dan kita akan membaca setiap kata dari input data tersebut menggunakan StringTokenizer. Hasilnya disimpan dalam output collector yang merupakan pasangan <key:kata,value:jumlah>.

3. Upload file program Java yaitu **WordCount.java** ke folder `root/tpdmr` yang telah kita buat sebelumnya. Jika lebih dari satu file program, maka upload semuanya.
4. Kita dapat *view* atau *edit* file tersebut menggunakan UNIX vi editor. Masuk ke folder `root/tpdmr` dan jalankan perintah:

```
# vi WordCount.java
```

Ketik :q untuk keluar atau :wq untuk menyimpan perubahan dan keluar.

5. *Compile* Java file dengan perintah sebagai berikut (pastikan *working directory* ada di folder dimana file program Java berada).

```
# javac -classpath /usr/hdp/3.0.1.0-187/hadoop/hadoop-common-3.1.1.3.0.1.0-187.jar:/usr/hdp/3.0.1.0-187/hadoop-mapreduce/hadoop-mapreduce-client-core-3.1.1.3.0.1.0-187.jar:/usr/hdp/3.0.1.0-187/hadoop-mapreduce/hadoop-mapreduce-client-common-3.1.1.3.0.1.0-187.jar WordCount.java
```

Sesuaikan PATH untuk file `hadoop-common-*.jar`, `hadoop-mapreduce-client-core-*.jar`, dan `hadoop-mapreduce-client-common-*.jar`. (di Hadoop versi sebelumnya, hanya dibutuhkan satu file `Hadoop-core-*.jar` untuk meng-*compile*)

Setelah berhasil dijalankan, jika kita lihat dalam folder tersebut, akan terbentuk file Class Java.

```
# ll
```

```
[root@sandbox-hdp tpdmr]# javac -classpath /usr/hdp/3.0.1.0-187/hadoop/hadoop-common-3.1.1.3.0.1.0-187.jar:/usr/hdp/3.0.1.0-187/hadoop-mapreduce/hadoop-mapreduce-client-core-3.1.1.3.0.1.0-187.jar:/usr/hdp/3.0.1.0-187/hadoop-mapreduce/hadoop-mapreduce-client-common-3.1.1.3.0.1.0-187.jar WordCount.java
[root@sandbox-hdp tpdmr]# ll
total 20
-rw-r--r-- 1 root root 1417 Feb 15 01:47 WordCount.class
-rw-r--r-- 1 root root 3274 Feb 15 01:42 wordcount.jar
-rw-r--r-- 1 root root 3270 Feb 15 01:39 WordCount.java
-rw-r--r-- 1 root root 2396 Feb 15 01:47 WordCount$Map.class
-rw-r--r-- 1 root root 2007 Feb 15 01:47 WordCount$Reduce.class
```

6. Kemudian kita buat JAR file kita beri nama **wordcount.jar** dari file Class tersebut

```
# jar -cvf wordcount.jar *.class
```

```
[root@sandbox-hdp tpdmr]# jar -cvf wordcount.jar *.class
added manifest
adding: WordCount.class(in = 1417) (out= 691)(deflated 51%)
adding: WordCount$Map.class(in = 2396) (out= 1036)(deflated 56%)
adding: WordCount$Reduce.class(in = 2007) (out= 817)(deflated 59%)
```

Dari perintah tersebut akan terbentuk **wordcount.jar** yang akan kita gunakan sebagai program aplikasi MapReduce.

```

/root/tpdml/
└─ Name
   ├── ..
   ├── WordCount.java
   ├── wordcount.jar
   ├── WordCount.class
   ├── WordCount$Reduce.class
   └── WordCount$Map.class

```

- Setelah program selesai disiapkan. Kita buat folder HDFS baru di dalam folder root untuk menyimpan seluruh input data untuk Job MapReduce.

```
# hdfs dfs -mkdir input/
```

- Upload input data yang akan kita gunakan (booth.txt) ke sistem local dan transfer menjadi HDFS.

```
# hdfs dfs -copyFromLocal booth.txt input/
```

- Dalam folder dimana JAR file berada, jalankan Job MapReduce menggunakan perintah:

```
# hadoop jar wordcount.jar WordCount input/booth.txt output
```

Dimana `wordcount.jar` merupakan JAR yang akan kita jalankan, `WordCount` merupakan nama Class utama, `input/booth.txt` merupakan input data, dan `output` merupakan folder output.

Perhatikan log dari MapReduce:

```

[root@sandbox-hdp tpdml]# hadoop jar wordcount.jar WordCount input/booth.txt output
23/02/15 02:09:22 INFO client.RMProxy: Connecting to ResourceManager at sandbox-hdp.hortonworks.com/172.18.0.2:8050
23/02/15 02:09:23 INFO client.AHSProxy: Connecting to Application History server at sandbox-hdp.hortonworks.com/172.18.0.2:10200
23/02/15 02:09:23 INFO client.RMProxy: Connecting to ResourceManager at sandbox-hdp.hortonworks.com/172.18.0.2:8050
23/02/15 02:09:23 INFO client.AHSProxy: Connecting to Application History server at sandbox-hdp.hortonworks.com/172.18.0.2:10200
23/02/15 02:09:23 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.
23/02/15 02:09:23 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /user/root/.staging/job_1676414688803_0002
23/02/15 02:09:23 INFO mapred.FileInputFormat: Total input files to process : 1
23/02/15 02:09:23 INFO mapreduce.JobSubmitter: number of splits:2
23/02/15 02:09:23 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1676414688803_0002
23/02/15 02:09:23 INFO mapreduce.JobSubmitter: Executing with tokens: []
23/02/15 02:09:24 INFO conf.Configuration: found resource resource-types.xml at file:/etc/hadoop/3.0.1.0-187/0/resource-types.xml
23/02/15 02:09:24 INFO impl.YarnClientImpl: Submitted application application_1676414688803_0002
23/02/15 02:09:24 INFO mapreduce.Job: The url to track the job: http://sandbox-hdp.hortonworks.com:8088/proxy/application_1676414688803_0002/
23/02/15 02:09:24 INFO mapreduce.Job: Running job: job_1676414688803_0002
23/02/15 02:09:28 INFO mapreduce.Job: Job job_1676414688803_0002 running in uber mode : false
23/02/15 02:09:28 INFO mapreduce.Job: map 0% reduce 0%
23/02/15 02:09:39 INFO mapreduce.Job: map 100% reduce 0%
23/02/15 02:09:44 INFO mapreduce.Job: map 100% reduce 100%
23/02/15 02:09:46 INFO mapreduce.Job: Job job_1676414688803_0002 completed successfully
23/02/15 02:09:46 INFO mapreduce.Job: Counters: 53
File System Counters
  FILE: Number of bytes read=6
  FILE: Number of bytes written=704213
  FILE: Number of read operations=0
  FILE: Number of large read operations=0
  FILE: Number of write operations=0
  HDFS: Number of bytes read=792
  HDFS: Number of bytes written=0
  HDFS: Number of read operations=11

```

Reducer tidak akan memproses sebelum Mapper selesai (100%).

```

23/02/15 02:09:46 INFO mapreduce.Job: Counters: 53
File System Counters
  FILE: Number of bytes read=6
  FILE: Number of bytes written=704213
  FILE: Number of read operations=0
  FILE: Number of large read operations=0
  FILE: Number of write operations=0
  HDFS: Number of bytes read=792
  HDFS: Number of bytes written=0
  HDFS: Number of read operations=11
  HDFS: Number of large read operations=0
  HDFS: Number of write operations=2
Job Counters
  Launched map tasks=2
  Launched reduce tasks=1
  Data-local map tasks=2
  Total time spent by all maps in occupied slots (ms)=62824
  Total time spent by all reduces in occupied slots (ms)=12232
  Total time spent by all map tasks (ms)=15706
  Total time spent by all reduce tasks (ms)=3058
  Total vcore-milliseconds taken by all map tasks=15706
  Total vcore-milliseconds taken by all reduce tasks=3058
  Total megabyte-milliseconds taken by all map tasks=16082944
  Total megabyte-milliseconds taken by all reduce tasks=3131392
Map-Reduce Framework
  Map input records=59
  Map output records=0
  Map output bytes=0
  Map output materialized bytes=12
  Input split bytes=234
  Combine input records=0
  Combine output records=0
  Reduce input groups=0
  Reduce shuffle bytes=12
  Reduce input records=0
  Reduce output records=0
  Spilled Records=0
  Shuffled Maps=2
  Failed Shuffles=0
  Merged Map outputs=2

```

Jumlah data <key,value> untuk setiap tahapan

10. Ouput dari Job MapReduce dapat dilihat di folder output.

Name >		Size >
_SUCCESS		0.1 kB
part-00000		0.1 kB

Kita jalankan perintah berikut untuk membaca file HDFS part-00000.

```
# hdfs dfs -cat output/part-00000
```

```

[root@sandbox-hdp tpdmr]# hdfs dfs -cat output/part-00000
Jaja      12
Jiji      11
Jinjin    10
Jojo      16
Juju      10

```

11. Jika akan menjalankan Job MapReduce baru, silakan hapus folder output atau mendefinisikan folder yang berbeda ketika menjalankan Job tersebut.

```
# hdfs dfs -rm -r output
```

12. Monitoring Jobs yang sedang berjalan pada cluster dapat diakses di <http://localhost:8088/cluster/apps>

hadoop

Cluster

About Nodes

Node Labels

Applications

NEW

NEW SAVING

SUBMITTED

ACCEPTED

RUNNING

FINISHED

FAILED

KILLED

Scheduler

Tools

All Applications

Logged in as: druid

Cluster Metrics

Apps Submitted

Apps Pending

Apps Running

Apps Completed

Containers Running

Memory Used

Memory Total

Memory Reserved

VCoers Used

VCoers Total

VCoers Reserved

2

0

1

1

3

3 GB

4 GB

0 B

3

4

0

Cluster Nodes Metrics

Active Nodes

Decommissioning Nodes

Decommissioned Nodes

Lost Nodes

Unhealthy Nodes

Rebooted Nodes

Shutdown Nodes

1

0

0

0

0

0

0

Scheduler Metrics

Scheduler Type

Scheduling Resource Type

Minimum Allocation

Maximum Allocation

Maximum Cluster Application Priority

Capacity Scheduler

[memory-mb (unit=Mi), vcores]

<memory 256, vCores 1>

<memory 4096, vCores 4>

0

Show 20 entries

Search

ID	User	Name	Application Type	Queue	Application Priority	StartTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU VCoers	Allocated Memory MB	Reserved CPU VCoers	Reserved Memory MB	% of Queue	% of Cluster	Progress	Tracking UI	Blacklist Nodes
application_1672196241128_0002	root	wordcount.jar	MAPREDUCE	default	0	Sun Mar 19 10:30:49 +0700 2023	N/A	RUNNING	UNDEFINED	3	3	3072	0	0	75.0	75.0	<div></div>	ApplicationMaster	0
application_1672196241128_0001	hive	HIVE- f1608e43-6305-4182-b5ec-673577ad524e	TEZ	default	0	Sun Mar 19 10:28:22 +0700 2023	Sun Mar 19 10:39:07 +0700 2023	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0	<div></div>	History	0
application_1675414658803_0004	root	wordcount.jar	MAPREDUCE	default	0	Wed Feb 15 09:36:24 +0700 2023	Wed Feb 15 09:36:45 +0700 2023	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0	<div></div>	History	0
application_1675414658803_0003	root	wordcount.jar	MAPREDUCE	default	0	Wed Feb 15 09:17:54 +0700 2023	Wed Feb 15 09:18:16 +0700 2023	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0	<div></div>	History	0

## 1.5. Penugasan

Kerjakan sesuai dengan yang dijelaskan pada bagian Kegiatan Praktikum dan kerjakan tugas praktikum sebagai berikut:

Buat sebuah program Java untuk Job MapReduce yang digunakan untuk menghitung maximum salary untuk setiap negara dari data berikut:

```
empID,country,salary
100,GM,35440
101,ZM,30205
102,PW,39788
103,EE,11679
104,IR,21978
105,IE,15959
106,KY,21986
107,IM,30461
108,KN,32668
109,US,22245
110,BB,37846
...
```

Submit program java dan screenshot hasil dari menjalankan Job tersebut pada data salaryinfo.txt

*Hints:*

- Edit file program WordCount.java (boleh ganti nama-nama Class-nya)
- Sesuaikan Class Map dengan mengambil kolom kedua sebagai key. Pada fungsi map(), split setiap string value baris. Output dari Class Map adalah key-nya ambil hasil split kedua, value-nya ambil split ketiga.
- Sesuaikan Class Reduce dengan mengubah fungsi reduce(). Gunakan fungsi Math.max(a,b) untuk mendapatkan nilai salary maksimum untuk setiap key (negara).