

Hadoop

by ramakrishna

DurgaSoft

ameerpetmaterials.blogspot.in

HADOOP

1. Introduction to Big data

Big Data Characteristics

- Huge data 10000 TB's of Data → **Volume**
- Huge data slow processing speed → **Velocity**
- Different kinds of data → **Variable**
- How accurate is the data analyzed → **Veracity**

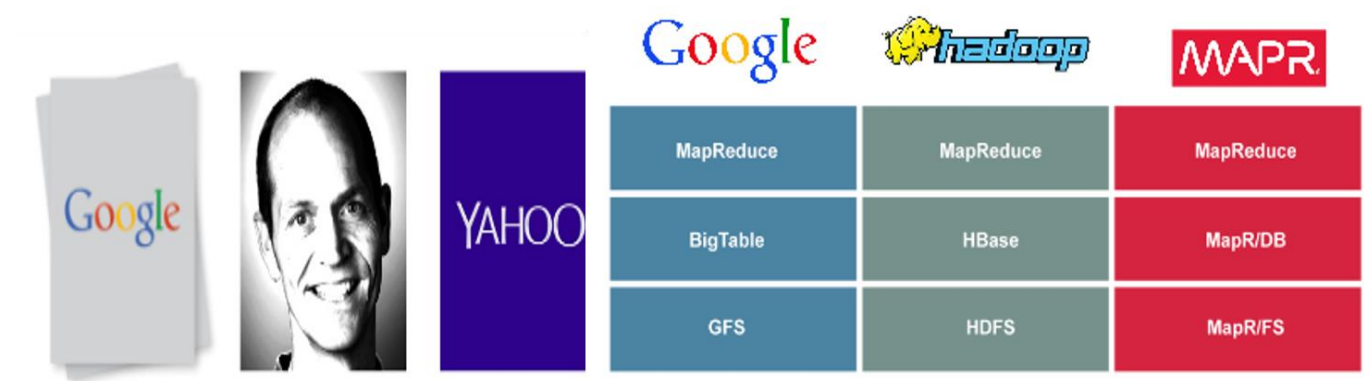
If any data having above 3 Characteristics called as **BIGDATA**

Google faced the Big Data Problem First, they released a paper on big data problem

And then they implemented solution for BigData as by taking

- no.of systems grouped together , then called it a **CLUSTER**
- Can add no.of systems additionally without effecting current configuration.
- To process data they implanted **MapReduce**

After that YAHOO implemented these things and named it as **HADOOP**



2. File System Basics

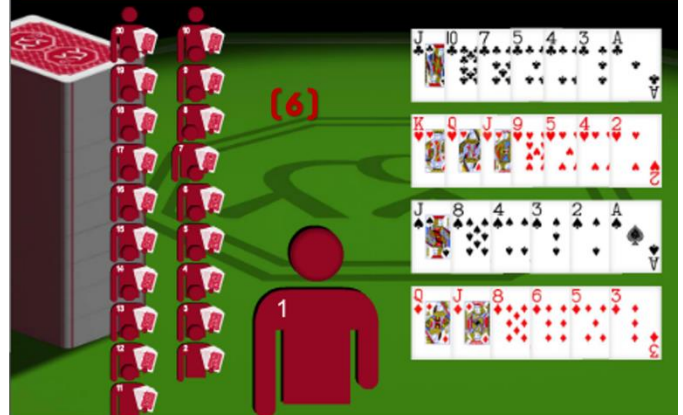
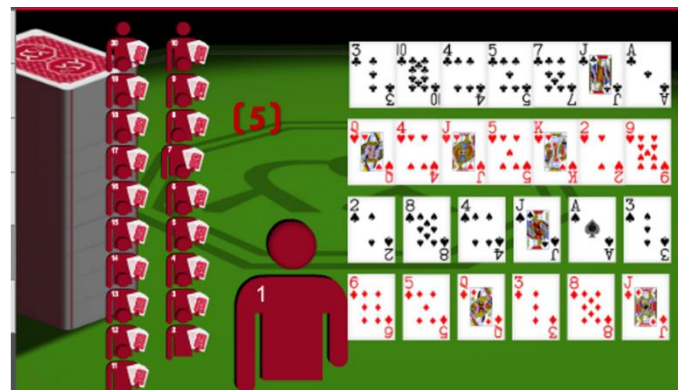
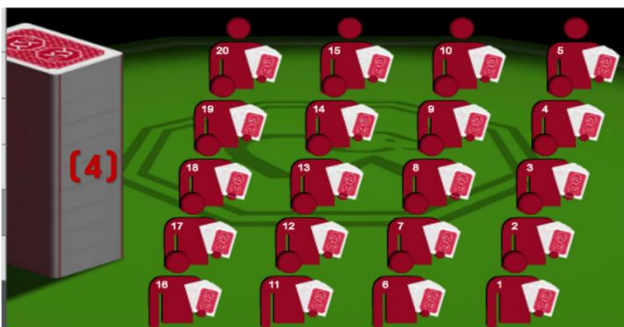
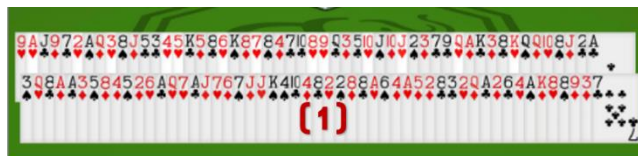


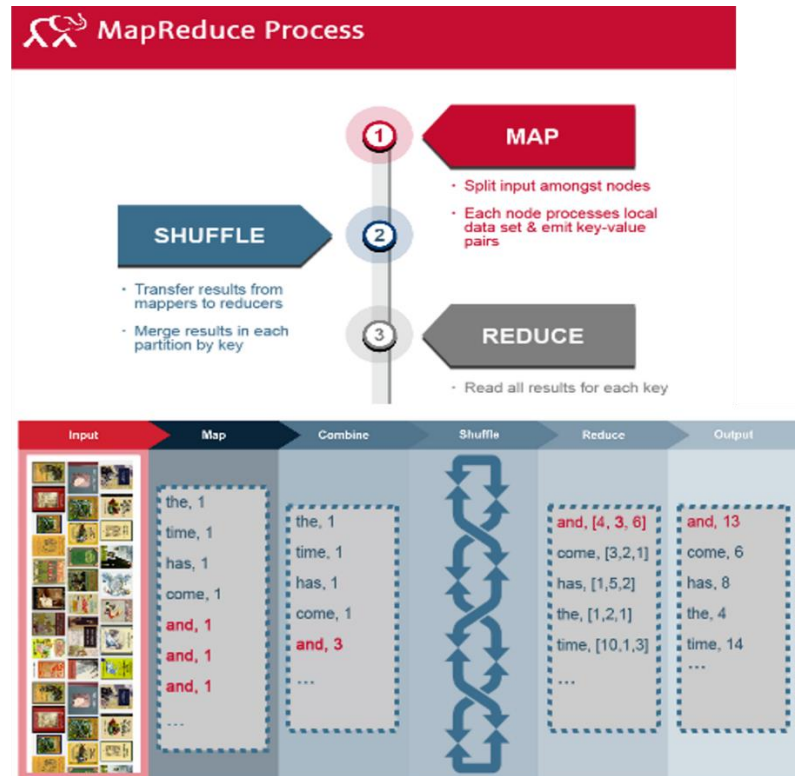
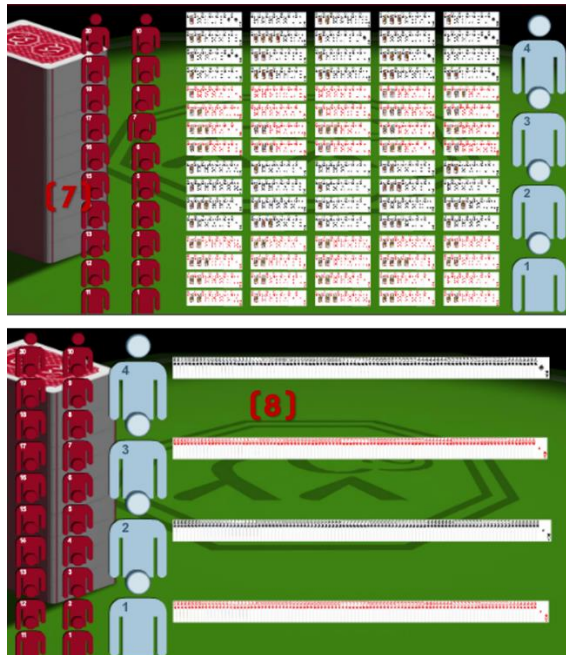
- HFS → Hard Disc File System
- CDFS → CD File System
- Space in the file system is divided into **Data Blocks**
- The metadata about these blocks are stored into **iNode**

Similarly for the distributed file system no. of systems are group together & call it as **NFS**



3. MapReduce Example with CARD GAME



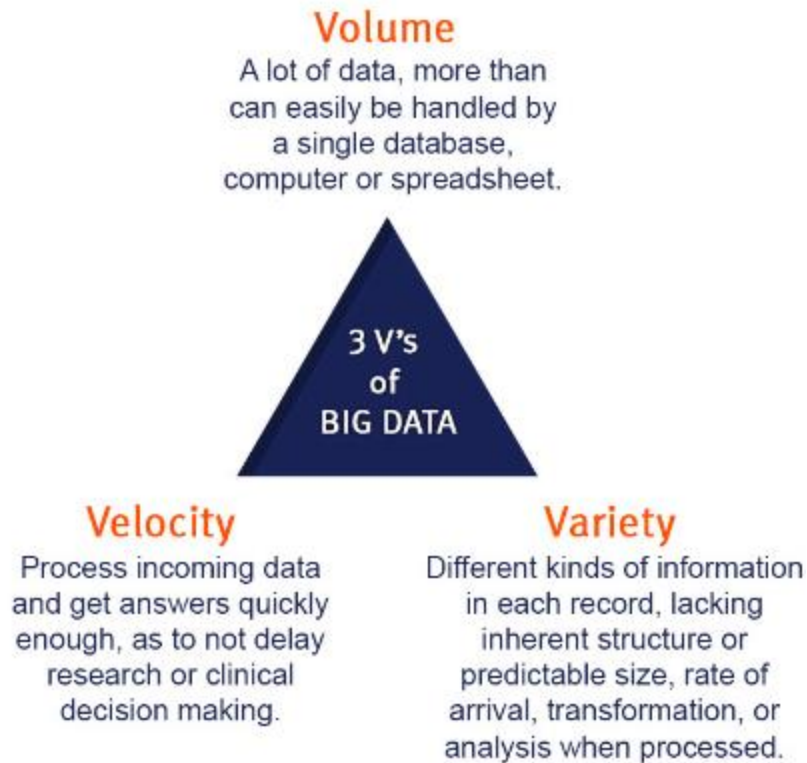


4. BigData & Hadoop Relationship

Big Data vs Hadoop

Big Data is nothing but a concept which facilitates handling large amount of data sets. Hadoop is just a single framework out of dozens of tools. Hadoop is primarily used for batch processing.

Big data Challenges



- Huge data 10000 TB's of Data → **Volume**
- Huge data slow processing speed → **Velocity**
- Different kinds of data → **Variable**

RDBMS vs DATAWARE HOUSE vs TERADATA

RDBMS

- It stores data in tables.
- Tables have rows and column.
- These tables are created using SQL.
- And data from these tables are also retrieved using SQL

DATAWARE HOUSE

- A data warehouse can be used to analyze a particular subject area. For example, "sales" can be a particular subject.
- Integrated: A data warehouse integrates data from multiple data sources
- Historical data is kept in a data warehouse. For example, one can retrieve data from 3 months, 6 months, 12 months, or even older data from a data warehouse.
- Once data is in the data warehouse, it will not change. So, historical data in a data warehouse should never be altered.

TERADATA

- Teradata brings analytical processing to all corners of your organization.
- It's proven analytical power and flexibility, scalability, and ease of use allows your entire organization to benefit from your valuable corporate information assets.

Types of Data

Structured Data - Data which has proper structure and which can be easily stored in tabular form in any relational databases like Mysql, Oracle etc is known as structured data.

Example: Employee data.

Semi-Structured Data - Data which has some structure but cannot be saved in a tabular form in relational databases is known as semi structured data.

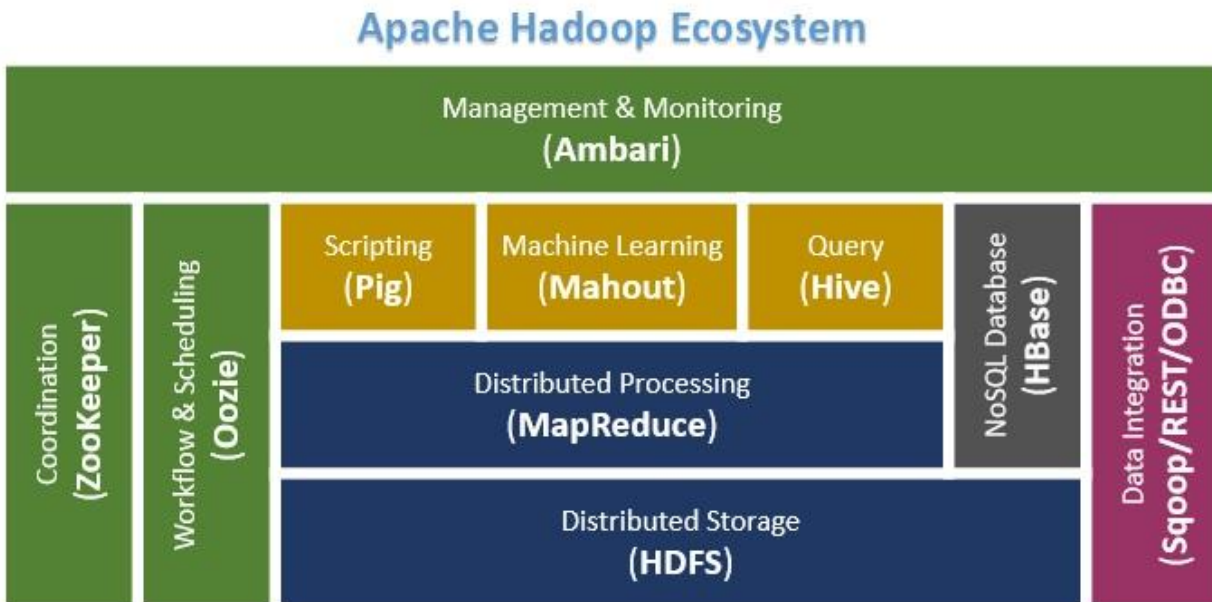
Example: XML data, email messages etc.

Unstructured Data - Data which is not having any structure and cannot be saved in tabular form of relational databases is known as unstructured data.

Example: Video files, Audio files, Text file etc.

5. Hadoop Ecosystem

Hadoop knows only Jar files. Hadoop don't know other things



Ambari: An integrated set of Hadoop administration tools for installing, monitoring, and maintaining a Hadoop cluster. Also included are tools to add or remove slave nodes.

Avro: A framework for the efficient **serialization** (a kind of transformation) of data into a compact binary format

Flume: A data flow service for the movement of large volumes of **log data** into Hadoop

HBase: A distributed columnar database that uses HDFS for its underlying storage. With HBase, you can store data in extremely **large tables** with variable column structures.

HCatalog: A service for providing a **relational view** of data stored in Hadoop, including a standard approach for tabular data

HiveQL: A distributed data warehouse for data that is stored in HDFS; also provides a **query language that's based on SQL** (HiveQL)

Hue: A Hadoop administration interface with handy **GUI tools for browsing files, issuing Hive and Pig queries, and developing Oozie workflows**

Mahout: A library of **machine learning statistical algorithms** that were implemented in MapReduce and can run natively on Hadoop

Oozie: A **workflow management tool** that can handle the scheduling and chaining together of Hadoop applications

Pig: A platform for the **analysis of very large data sets that runs on HDFS** and with an infrastructure layer consisting of a compiler that produces sequences of MapReduce programs and a language layer consisting of the query language named Pig Latin

Sqoop: A tool for efficiently **moving large amounts of data between relational databases and HDFS**

Zookeeper: A simple interface to the centralized **coordination/monitor of services** (such as naming, configuration, and synchronization) used by distributed applications

6. HDFS (Hadoop Distributed File System)

a. Definitions

Cluster	:	<i>Group of Systems connected to LAN</i>
Commodity H/W	:	<i>Cheap Hardware Things</i>
Streaming access pattern	:	<i>write once – Read any no.of time – Don't change data</i>

b. HDFS Architecture

HDFS is ***specially designed file system for storing huge datasets with cluster of commodity hardware***

We have to install HDFS file system on the top of Linux. Linux block size is 4kb. If I want to store 2kb of data in Linux, another 2kb of data will be wasted

- File system is divided into blocks
- By default block size is **64mb**
- If we store 32mb of data, remaining 32mb of data will be used for another purpose

That's why it is called as 'Specially Designed File System'

c. HDFS Services

HDFS Have 5 services

Master services

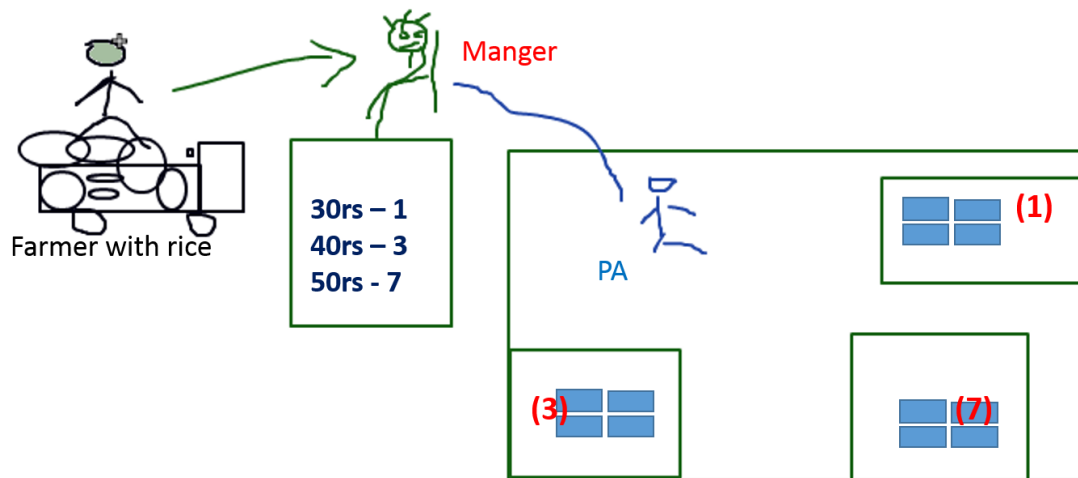
1. Name node
2. Secondary name node
3. Job Tracker

Slave Service

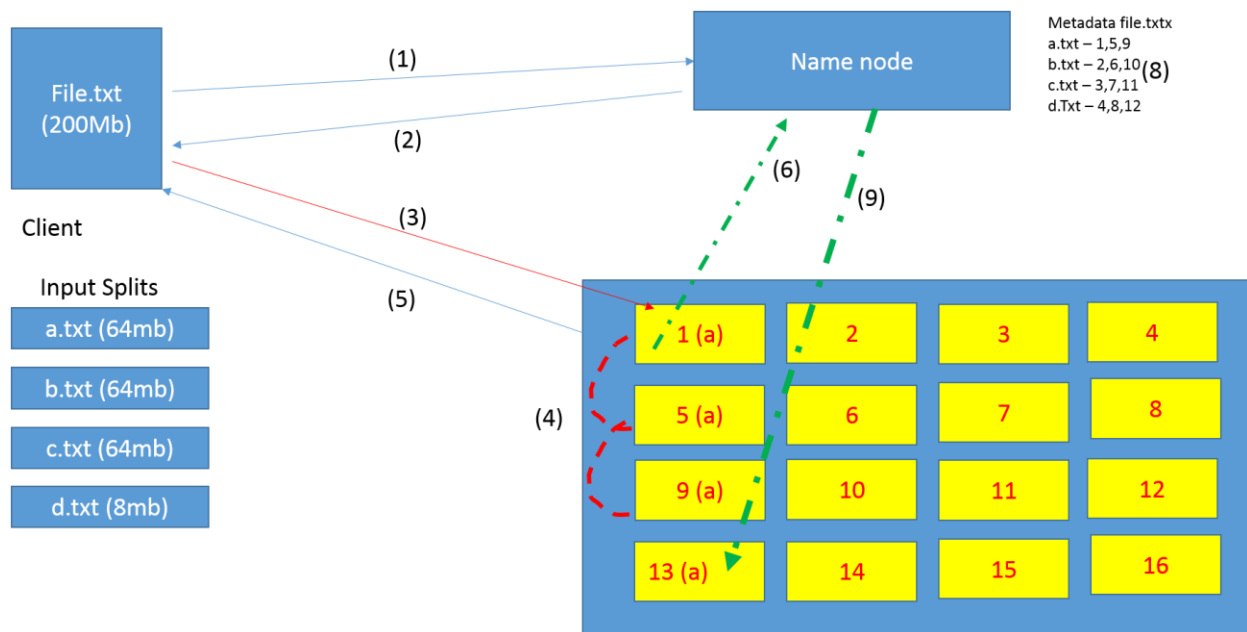
4. Data node
5. Task tracker

Before going to understand these you want to listen farmer-godown story.

One day a farmer want to store his rice packets into a godown. We went to godown and contacted manager. Manager asked what type of rice packets you have. He said 50 packs of 30rs, 50 of 40rs, 100 of 50rs. He noted down on a bill paper he called his PA to store these 30rs paks in 1st room, 40rs pacs in 3rd room, 50rs pacs in 7th room.



d. Storing/Writing data in HDFS



Storing data into HDFS

- Client want to store file.txt 200Mb into HDFS
- File.txt is divided into 64mb of sub files
- These are called as Input Splits
- Client send these to NameNode

- Namenode reads the metadata , and sends available datanodes to client
- Client send each input split to datanodes.
- A.txt file stores into datanode-1
- And HDFS also maintains the replica of data.by default replica is 3.
- So a.txt is stored in datanodes 5,9 as well
- Acknowledgment is sent to 9 to 5, 5 to 1, 1 to client.
- And the response is sent to client
- For every storage operation datanode send **BlockReport and Heartbeat** to namenode.
- Datanode sent heartbeat to namenode for every short period of time.
- If any datanode fails, namenode copies that data to another datanode

7. MapReduce

Before we did Storage,Now if u want retrieve is called process

- If you want process data we write one program(10KB)
- Processing(it is called as Map)
- client submits commands/program to JobTracker for applying job on a.txt
- JobTracker asks Namenode
- namenode replies metadata to JobTracker
- JobTracker knows on which blocks data is stored in Datanodes
- A.txt stored in 1,3,9
- Using metadata Job tracker contacts Tasktracker near to JobTracker(datanode-1)
- JobTracker assigns this task to Tasktracker
- Tasktracker apply this job on a.txt. this is called "**MAP**"
- Similarly on b.txt, c.txt, d.txt Tasktracker runs program, called **MAP**

- **A,b,c,d txt files called Input Splits**

No. of Input Splits = No. of Mappers

- If any Tasktracker is not able to do the job, it informs JobTracker
- JobTracker assigns to nearest another Tasktracker.
- for every 3secs Tasktracker send heartbeat to JobTracker
- if 10sec also if JobTracker doesn't get heartbeat, it thinks Tasktracker is dead or slow
- Tasktracker may slow if it running more jobs ex. 100 jobs
- JobTracker knows the how many jobs running by Tasktracker
- if any Tasktracker is busy , JobTracker chooses another Tasktracker
- After applying program, a.txt returns 10kb of data, b.txt 10kb, c 10kb, d 4kb.
- After getting outputs **Reducer** combines all these output data to file.

No of Reducers = No. of Output files

- **The Output file may save in local data node, or some other node.**
- The Datanode sends heartbeat to Namenode where output file is saved.
- NameNode save this metadata
- Clint contacts NameNode for Output file location

Ref: <http://www.csfundas.com/2015/09/hdfs-hadoop-distributed-file-system.html>

a. Hello world JOB Steps

1. Create a File in local file system and save file.txt
2. Loading file.txt form local file system to HDFS

\$hadoop -fs put /local/file.txt /hdfs/file
--

3. Now, we have to process. So we have to write programmers
 - i. DriverCode.java (main method)
 - ii. MapperCode.java
 - iii. ReducerCode.java

4. Compile above .java files

```
$javac -CLASSPATH $HADOOP_HOME/hadoop-core.jar *.java
```

5. Creating JAR File

```
$Jar cvf test.jar *.class
```

6. Running above test.jar on file which is their in HDFS

```
$hadoop jar test.jar DriverCode file testout
```

\$ hadoop jar input1 input2 input3 input4

input1 = WordCount.jar

input2 = Name of Class which contains main() method

input3 = Input file present in hdfs ("file.txt")

input4 = Output directory name // It must be directory. Must create a new directory

\$ hadoop jar WordCount.jar WordCount /WordCount/file.txt WordCountOP

7. Check the output present in WordCountOP directory present in HDFS.

There are three files.

1) _Success //only shows successfule execution of program

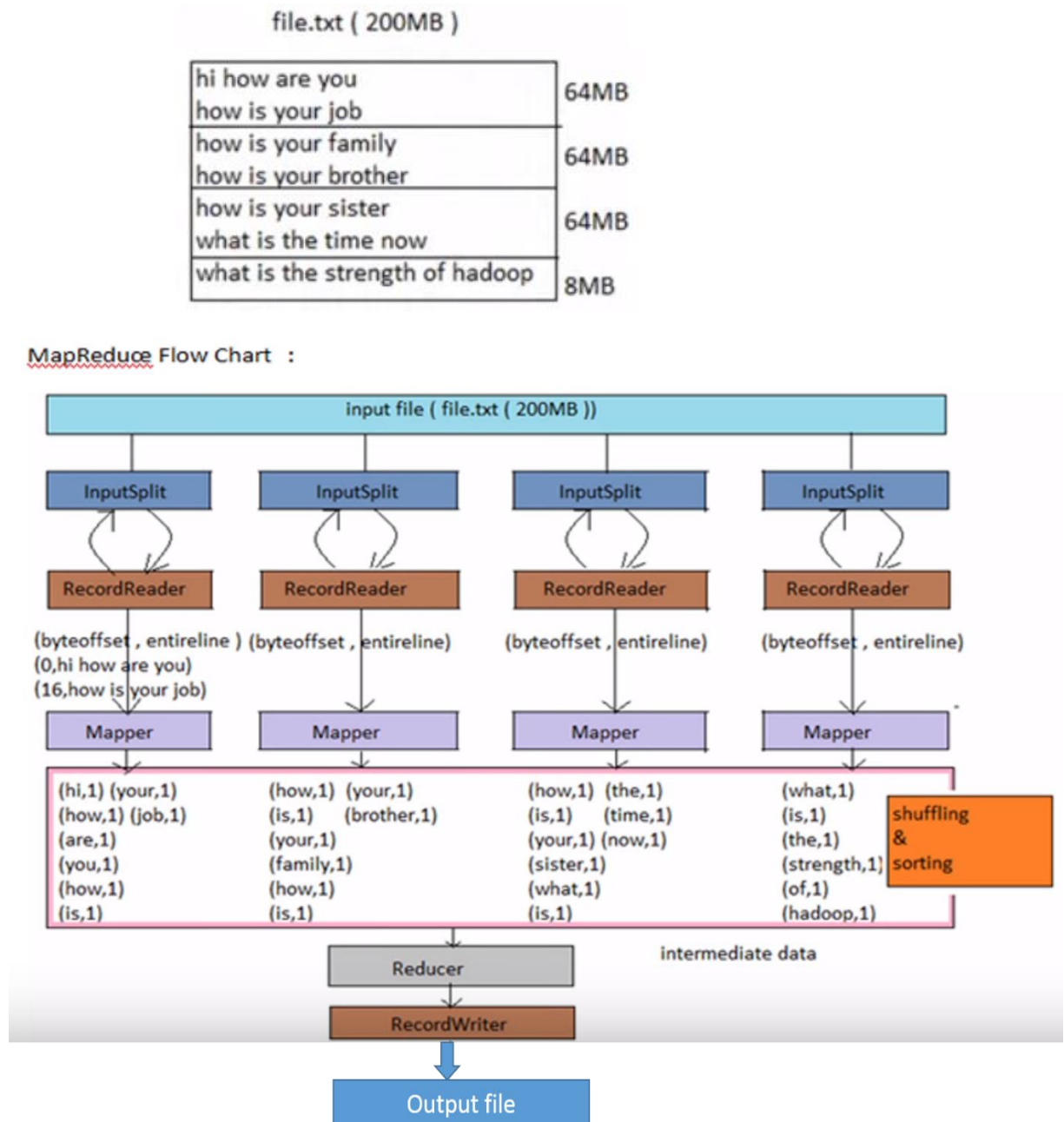
2) log //contains all log information

3) part-r-00000 //contains actual count of words.

\$ hadoop fs -ls /WordCountOP/part-r-00000

b. MapReduce flowchart

Word count problem: Counts the no of occurrences of words in the file.



- 200mb of file is divided into 4 Inputsplits(3x64, 1x8)
- each Inputsplit has one Mapper
- Mapper and reducer works on key,value pairs
- Inputsplit is in the form of text only
- Each and every line we have to convert into key,value pair.
- for this we have **RecordReader** Interface

- RecordReader reads each line at a time and converts into Key,value pair.
- lines are called as **Records**
- RecordReader analyzes the file in which format out of below 4
 - **TextInputFormat** - by default
 - **KeyValueTextInputFormat**
 - **SequenceFileInputFormat**
 - **SequenceFileAsTextInputFormat**

byteoffset, entire line)

(address,entire line)

(0, Hi how are you)

- RecordReader converts 1st line to key,value pair and reads next line
- Next line byteoffset = No. of characters+spaces, **(16, "How is your job")**
- RecordReader converts all lines to (Key,Value) pairs and passes to Mapper
- Mapper reads each and every key,value from RecordReader
- All Mapper and RecordReader are works parallel.
- Mapper generates Key,value pairs by count of words in line
- Mapper uses BoxClasses for converting the Key,value pair of RecordReader to key,values of Object data
- All these BoxClasses are implemented compareTo
- The data generated between Reader and Mapper is called Intermediate Data
- Reducer are two types

Identical Reducer (Def.) --> Only sort+ No Shuffle

Own Reducer

- This intermediate data is goto Shuffling and Sorting Phase
- Shuffling - its combine all your values associated to single key . (how, [1,1,1,1,1])
- Sorting - Sorting A to Z, BoxClasses taken care about this
- In Reducer class we write Shuffle and Sorting logic
- The Reducer output is given to RecordWriter

- It will write one key,value pair to Output file at a time
- output file is part-00000

The screenshot shows a presentation slide with a blue header bar containing the text 'Hadoop MapReduce' and 'Types and Formats'. The main title of the slide is 'InputFormat'. Below the title, there is a list of four input formats, each preceded by a blue circular bullet point. The first format is 'TextInputFormat' with a sub-bullet point stating it 'Treats each newline-terminated line of a file as a value'. The second is 'KeyValueTextInputFormat' with a sub-bullet point stating it 'Maps newline-terminated text lines of "key" SEPARATOR "value"'. The third is 'SequenceFileInputFormat' with a sub-bullet point stating it is a 'Binary file of key-value pairs with some additional metadata'. The fourth is 'SequenceFileAsTextInputFormat' with a sub-bullet point stating it is 'Same as before but, maps {k.toString(), v.toString()}'. The slide has a blue footer bar with the text 'Pete Michard / Duxcore', 'Hadoop MapReduce', and 'x1/71'.

InputFormat

- **TextInputFormat**
 - Treats each newline-terminated line of a file as a value
- **KeyValueTextInputFormat**
 - Maps newline-terminated text lines of "key" SEPARATOR "value"
- **SequenceFileInputFormat**
 - Binary file of key-value pairs with some additional metadata
- **SequenceFileAsTextInputFormat**
 - Same as before but, maps {k.toString(), v.toString()}

Pete Michard / Duxcore Hadoop MapReduce x1/71

Sorting :

Sorting is another phase on intermediate data to sort all (key, value) pairs. Because of shuffling phase , all unique keys will compare with each other and gives output in some sorting order.

Basically this sorting will be done because of Box classes as we have seen.....

IntWritable

LongWritable

FloatWritable

DoubleWritable

Text

All these above classes implemented Writable and WritableComparable interfaces so that they can compare with each other.

c. Box classes in Hadoop

Similar to wrapper classes in Java, hadoop has concept of **Box Classes**. Uses for conversion from primitive type to Box classes' type conversion and vice-versa

Wrapper Class	Primitive Type	Box Class
Integer	int	IntWritable
Long	long	LongWritable
Float	float	FloatWritable
Double	double	DoubleWritable
String	string	Text
Character	char	Text

- 1) int to IntWritable conversion "**new IntWritable(int)**" method.
IntWritable to int conversion "get()" method.
- 2) long to LongWritable conversion "new **LongWritable(long)**" method.
LongWritable to long conversion "get()" method.

- 3) float to FloatWritable conversion "new **FloatWritable(float)**" method.
FloatWritable to float conversion "get()" method.
- 4) double to DoubleWritable conversion "new **DoubleWritable(double)**" method.
DoubleWritable to double conversion "get()" method.
- 5) String to Text conversion "new Text(String)" method.
Text to String conversion "toString()" method.

What are the XML files we are using in Hadoop

Core-site.xml

All Configuration, Metada setup

Mapred-site.xml

Jobs configurations

Hdfs-site.xml

It have no.of Replication setup

e. Hello Word JOB Example Code

```
$ hadoop jar input1 input2 input3 input4
```

\$ hadoop jar	WordCount.jar	WordCount	/WordCount/file.txt	WordCountOP
	Jar	DriverClass	InputFile	OutputFile

There are three classes for writing any hadoop mapreduce program.

1. **Driver class** i.e. main() method class
2. **Mapper class** // map input to different systems
3. **Reducer class** // collect output from different system

1. Driver class

- Tool interface taken care by args[] & it has run(ip,op) method
- JobConf class taken care the all configartiions

- Covering text paths to FileInputPath format
 - Setting Mapper & Reduced classes
 - Set output Key, value formats
 - Write main method call ToolRunner.run(), it will taken care 3 args of cmdline
-
- FileInputFormat.setInputPaths(job, new Path(args[0]));
//covetiing String path of input file to Path Formate
 - FileOutputFormat.setOutputPath(job, new Path(args[1]));
//covetiing String path of Output file to Path Formate

```
public class WordCount extends Configured implements Tool {

    public static void main(String[] args) throws Exception
    {

        int exitCode=ToolRunner.run(new WordCount(), args);
        System.exit(exitCode);
    }
    public int run(String[] args) throws IOException
    {

        Path inputPath=new Path(args[0]);
        Path outputPath=new Path(args[1]);
        JobConf conf=new JobConf(WordCount.class);
        conf.setJobName("word count");
        FileInputFormat.setInputPaths(conf,inputPath);
        FileOutputFormat.setOutputPath(conf,outputPath);
        conf.setMapperClass(WordMapper.class);
        conf.setReducerClass(WordReducer.class);
        conf.setMapOutputKeyClass(Text.class);
        conf.setMapOutputValueClass(IntWritable.class);
        conf.setOutputKeyClass(Text.class);
        conf.setOutputValueClass(IntWritable.class);
        JobClient.runJob(conf);
        return 0;
    }
}
```

2. Mapper class

- Mapper Interce , to make class as Mapper class
- MapReduceBase, bridge between Mapper & Reducer

- we have only one method, Map(4 args)
 - MapperInputKeyType
 - MapperInputValueType
 - MapperOutputKeyType
 - MapperIOOutputValueType
- Reporter Interface , repots any isses to Driver class

```
public class WordMapper extends MapReduceBase implements
Mapper<LongWritable,Text,Text,IntWritable>{
    public void map(LongWritable key, Text value,OutputCollector<Text,IntWritable>
output,Reporter reporter) throws IOException
    {
        String line = value.toString();
        String[] words = line.split("\\W+");
        for (String word:words) {
            output.collect(new Text(word), new IntWritable(1));
        }
    }
}
```

3. Reducer class

- Reducer Interce , to make class as Reducer class
- MapReduceBase, bridge between Reducer & Reduce
- we have only one method, Reduce(4 args)
 - ReducerInputKeyType
 - ReducerInputValueType (hi, [1,1,1,1]) it taken as iterator
 - ReducerOutputKeyType
 - ReducerIOOutputValueType
- Reporter Interface , repots any isses to Driver class

```
public class WordReducer extends MapReduceBase implements
Reducer<Text, IntWritable, Text, IntWritable> {
    public void reduce(Text key, Iterator values,
        OutputCollector<Text, IntWritable> output, Reporter reporter)
        throws IOException {
```

```

        int sum = 0;
        while(values.hasNext()) {
            sum += values.next().get();
        }
        output.collect(key, new IntWritable(sum));
    }
}

```

8. Old API vs New API

1. Driver Code (old)	(new)
1) JobConf (org.apache.hadoop.mapred) (all configuration)	1) Job (org.apache.hadoop.mapred)
2) JobClient.runJob(JobConf) (execution start - submitting job to client)	2) WaitForCompletion
3) Tool, ToolRunner class	3) Don't have such class
4) Tool, ToolRunner class	4) Don't have such class
2. Mapper Code	
1) MapReduceBase class (org.apache.hadoop.mapred)	1) Don't have such class
2) Mapper is an interface	2) Mapper is an abstract class
3) OutputCollector, Reporter methods are present	3) Context class is present

4) collect(key,value)	4) write(key,value)
Reduce Code	
1) MapReduceBase class (org.apache.hadoop.mapred)	1) Don't have such class
2) Reducer is an interface	2) Reducer is an abstract class
3) OutputCollector, Reporter methods are present	3) Context class is present
4) collect(key,value)	4) write(key,value)
5) Iterator	5) Iterable
6) Partitioner is an interface	6) Partitioner is an abstract class

Driver Code :

```

public class Driver {
    public static void main(String[] args) throws IOException,
        ClassNotFoundException, InterruptedException {
        if (args.length != 2) {
            System.out.println("IO error");
            System.exit(-1);
        }
        @SuppressWarnings("deprecation")
        Job job = new Job();
        job.setJarByClass(Driver.class);
        job.setJobName("Driver");
        job.setMapperClass(WMapper.class);
        job.setReducerClass(WReducer.class);

        FileInputFormat.setInputPaths(job, new Path(args[0]));
    }
}

```

```
FileOutputStream.setOutputPath(job, new Path(args[1]));

job.setOutputKeyClass(Text.class);
job.setOutputValueClass(Text.class);

System.exit(job.waitForCompletion(true) ? 0 : 1);
}
}
```

Mapper Code :

```
public class WMapper extends Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(LongWritable key, Text value, Context context) throws IOException,
        InterruptedException {
        String str = value.toString();
        StringTokenizer token = new StringTokenizer(str);
        while (token.hasMoreElements()) {
            word.set(token.nextToken());
            context.write(word, one);
        }
    }
}
```

Reducer Code :

```
public class WReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
    int sum=0;
    public void reduce(Text key, Iterable<IntWritable> values, Context context) throws
        IOException, InterruptedException{
        for (IntWritable val : values) {
            sum+= val.get();
        }
        context.write(key, new IntWritable(sum));
    }
}
```

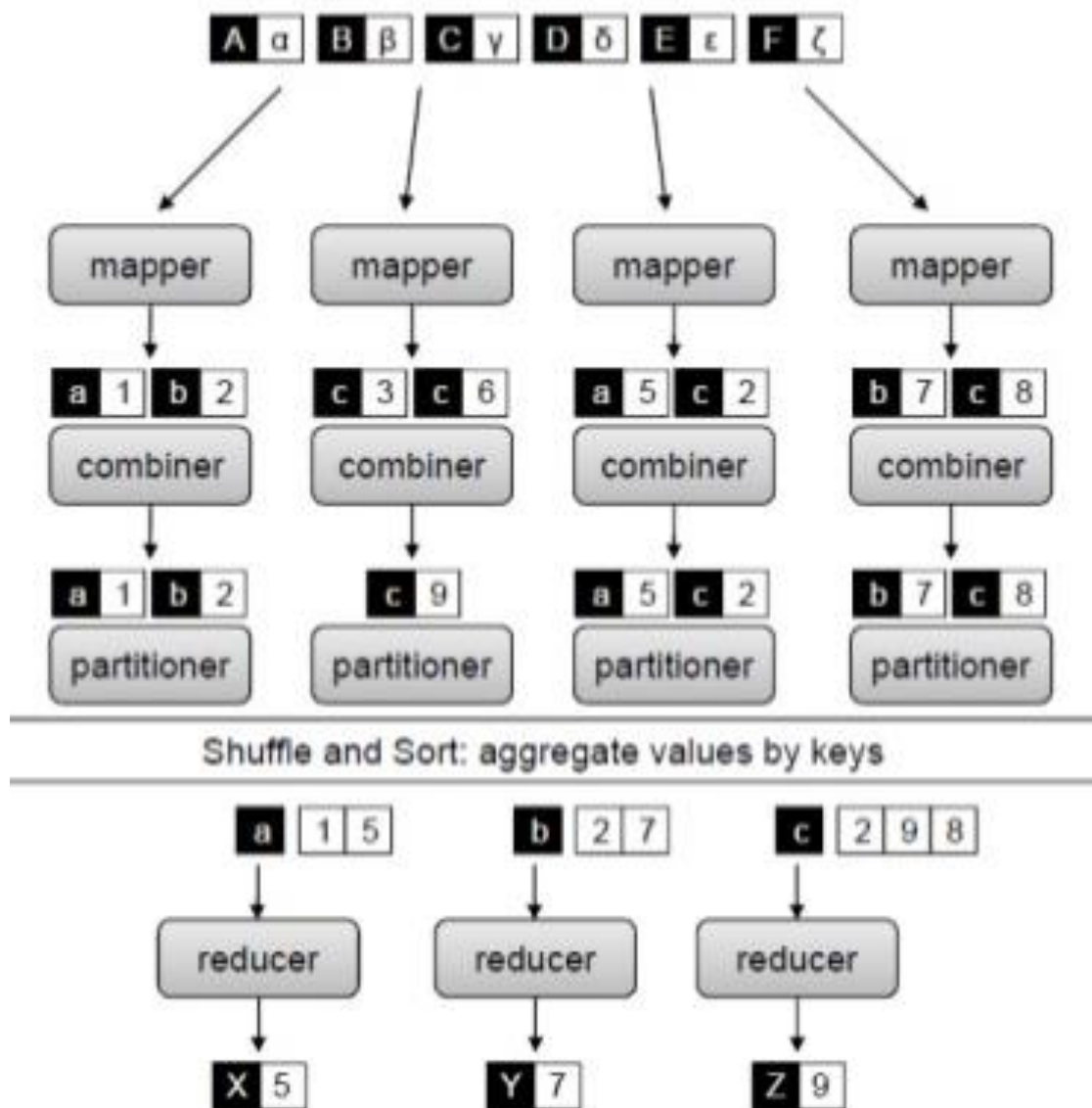
9. Combiner

If you are processing 1Gb of data it will create 10, 00,000 key,value pairs, If 1TB of data, 1000000000000 bytes Key,value pair have to create. So, Network traffic high, and Performance will Down. To resolve these issues Combiners are introduced.

"Combiner is a mini-reducer, No of Mappers = No.of Combiners

- Operates by accepting the inputs from the Map class and thereafter passing the output key-value pairs to the Reducer class.
- The Combiner class is used in between the Map class and the Reduce class to reduce the volume of data transfer between Map and Reduce.

- Usually, the output of the map task is large and the data transferred to the reduce task is high



How Combiner Works?

- A combiner does not have a predefined interface and it must implement the Reducer interface's **reduce()** method.
- A combiner operates on each map output key. It must have the same output key-value types as the Reducer class.

- A combiner can produce summary information from a large dataset because it replaces the original Map output.

we have the following input text file named **input.txt** for MapReduce.

```
What do you mean by Object
What do you know about Java
What is Java Virtual Machine
How Java enabled High Performance
```

1. Record Reader

This is the first phase of MapReduce where the Record Reader reads every line from the input text file as text and yields output as key-value pairs.

Input – Line by line text from the input file.

Output – Forms the key-value pairs. The following is the set of expected key-value pairs.

```
<1, What do you mean by Object>
<2, What do you know about Java>
<3, What is Java Virtual Machine>
<4, How Java enabled High Performance>
```

2. Map Phase

The Map phase takes input from the Record Reader, processes it, and produces the output as another set of key-value pairs.

Input – the above key-value pair is the input taken from the Record Reader.

Output – the expected output is as follows

```
<What,1> <do,1> <you,1> <mean,1> <by,1> <Object,1>
<What,1> <do,1> <you,1> <know,1> <about,1> <Java,1>
<What,1> <is,1> <Java,1> <Virtual,1> <Machine,1>
<How,1> <Java,1> <enabled,1> <High,1> <Performance,1>
```

3. Combiner Phase

The Combiner phase takes each key-value pair from the Map phase, processes it, and produces the output as **key-value collection** pairs.

Input – the above key-value pair is the input taken from the Map phase.

The Combiner phase reads each key-value pair, combines the common **words as key** and **values as collection**. it is similar to that of a Reducer. Following is the code snippet for Mapper, Combiner and Reducer class declaration.

```
job.setMapperClass(TokenizerMapper.class);
job.setCombinerClass(IntSumReducer.class);
job.setReducerClass(IntSumReducer.class);
```

Output – The expected output is as follows

```
<What,1,1,1> <do,1,1> <you,1,1> <mean,1> <by,1> <Object,1>
<know,1> <about,1> <Java,1,1,1>
<is,1> <Virtual,1> <Machine,1>
<How,1> <enabled,1> <High,1> <Performance,1>
```

4. Reducer Phase

The Reducer phase takes each key-value collection pair from the Combiner phase, processes it, and passes the output as key-value pairs. Note that the Combiner functionality is same as the Reducer.

Input – the following key-value pair is the input taken from the Combiner phase.

Output – the expected output from the Reducer phase is as

```
<What,3> <do,2> <you,2> <mean,1> <by,1> <Object,1>
<know,1> <about,1> <Java,3>
<is,1> <Virtual,1> <Machine,1>
```

```
<How,1> <enabled,1> <High,1> <Performance,1>
```

5. Record Writer

This is the last phase of MapReduce where the Record Writer writes every key-value pair from the Reducer phase and sends the output as text.

Input – Each key-value pair from the Reducer phase along with the Output format.

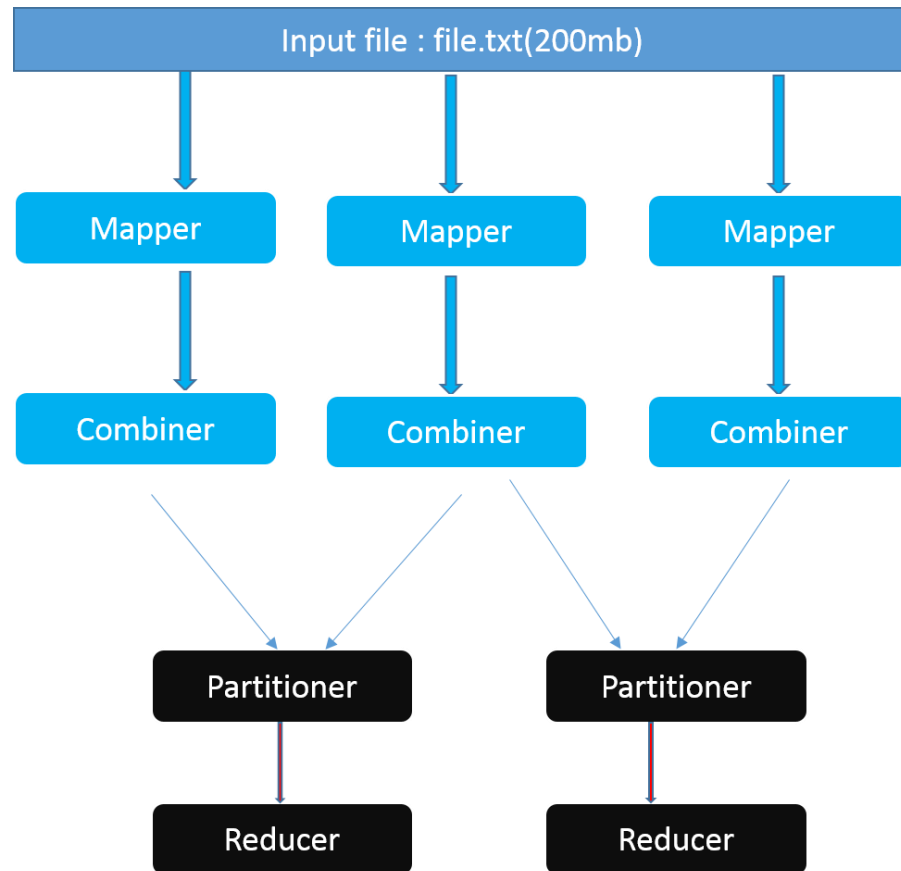
Output – It gives you the key-value pairs in text format. Following is the expected output.

What	3
do	2
you	2
mean	1
by	1
Object	1
know	1
about	1
Java	3
is	1
Virtual	1
Machine	1
How	1
enabled	1
High	1
Performance	1

10. Partitioner

If we are using 3 or more reducers. Then **key, value** pairs are divided to these Reducers. HashPartitioner takes care about distributing **K,V** pairs to Reducers. If you want to pass specific K, V pairs to Specific reduces, then we have to write Partitioner class.

The partition phase takes place after the Map phase and before the Reduce phase



The number of partitioners is equal to the number of reducers. That means a partitioner will divide the data according to the number of reducers.

It partitions the data using a user-defined condition, which works like a hash function

1. Map Tasks

The map task accepts the key-value pairs as input while we have the text data in a text file.

Output – you will get the gender data and the record data value as key-value pairs

Input.txt

1201	gopal	45	Male	50000
1202	manisha	40	Female	51000
1203	khaleel	34	Male	30000
1204	prasanth	30	Male	31000
1205	kiran	20	Male	40000
1206	laxmi	25	Female	35000
1207	bhavya	20	Female	15000
1208	reshma	19	Female	14000
1209	kranthi	22	Male	22000
1210	Satish	24	Male	25000
1211	Krishna	25	Male	26000
1212	Arshad	28	Male	20000
1213	lavanya	18	Female	8000

2. Partitioner Task

The partitioner task accepts the key-value pairs from the map task as its input

Input – The whole data in a collection of key-value pairs.

Method – The process of partition logic runs as follows.

Read the age field value from the input key-value pair.

```
String[] str = value.toString().split("\t");
int age = Integer.parseInt(str[2]);
```

Check the age value with the following conditions.

- **Age less than or equal to 20**
- **Age Greater than 20 and Less than or equal to 30.**
- **Age Greater than 30.**

Output – the whole data of key-value pairs are segmented into three collections of key-value pairs. The Reducer works individually on each collection.

3. Reduce Tasks

The number of Partitioner tasks is equal to the number of reducer tasks. Here we have three Partitioner tasks and hence we have three Reducer tasks to be executed.

Input – The Reducer will execute three times with different collection of key-value pairs.

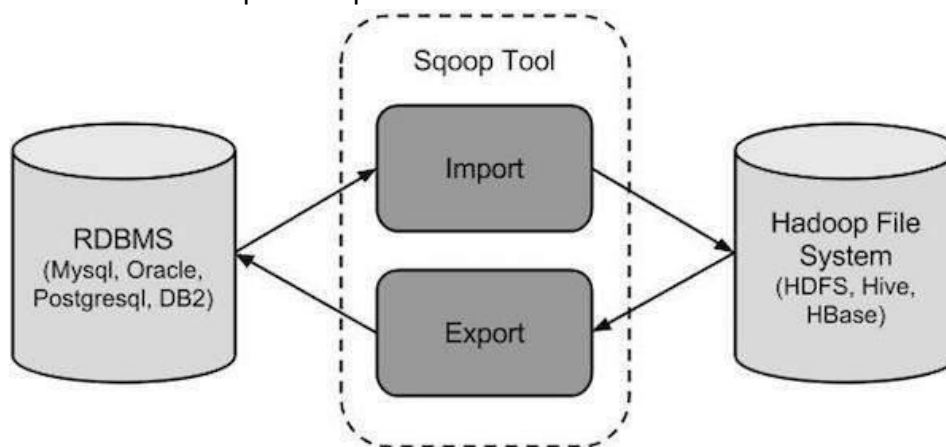
Output – finally, you will get a set of key-value pair data in three collections of different age groups.

11. Sqoop

Introduction

Sqoop is a tool, acts as bridge between RDBMS Hadoop

It runs on the Top of MapReduce



By Default it is working with 4 Mappers & Not working with Reducers

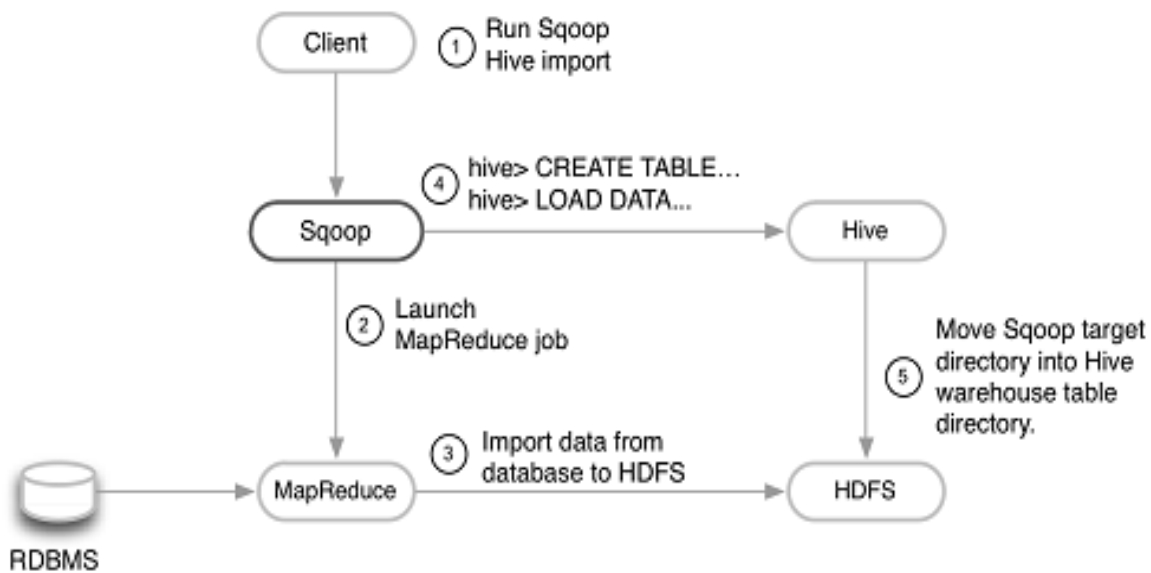
Output also comes from MapReduce as 4 output files

If you want to working your own number of Mappers

`-m <no. of Mappers>`

We can import

- Import entire table by sqoop
- Import part of the table using WHERE or COLUMNS clause
- Import all tables from specific Database
- Export data from HDFS to RDBMS



Ref: <http://wideskills.com/hadoop/sqoop-data-transfer-utility>

1. How to display databases from RDMS (MySQL) using Sqoop

\$sqoop list-databases

- --connect jdbc:mysql://<hostname> \
- -- <uname> --<password> ←| ENTER |

2. How to import all tables from a Database Mysql to HDFS

\$sqoop import-all-tables \

- --connect jdbc:mysql://<hostname> /<database name>\
- -- <uname> --<password> \
- -- fields-terminated-by '\t'

3. How to import a table from a Database Mysql to HDFS

```
$sqoop import \
```

- --connect jdbc:mysql://<hostname> /<database name>\
- -- <uname> --<password> \
- -- table <tablename> -- fields-terminated-by '\t'
- --columns "id,name" //or// --where "sal>1000"

4. How to Import Part of the table from RDBMS mysql to HDFS (Hadoop)

```
$sqoop import \
```

- --connect jdbc:mysql://<hostname> /<database name>\
- -- <uname> --<password> \
- -- table <tablename> -- fields-terminated-by '\t'

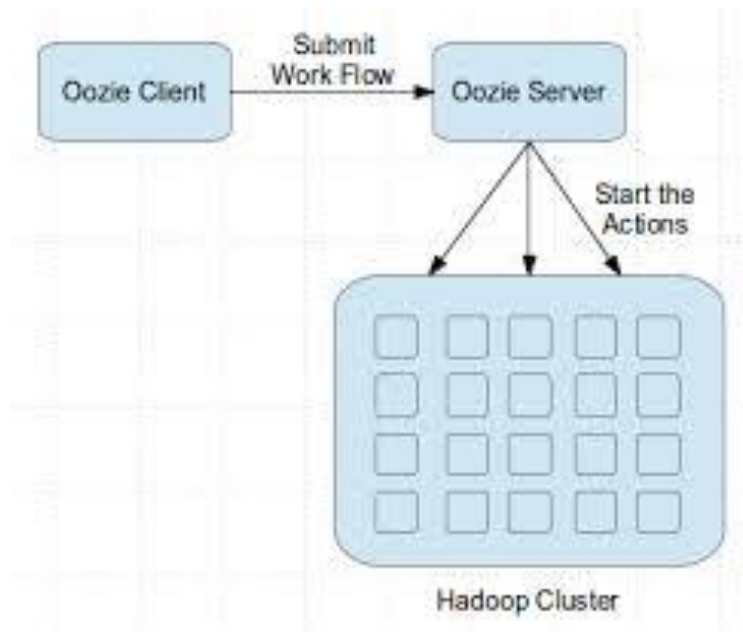
5. How to export data from HDFS to RDBMS mysql (Hadoop)

```
$sqoop export \
```

- --connect jdbc:mysql://<hostname> /<database name>\
- -- <uname> --<password> \
- -- table <tablename> \
- --input -- fields-terminated-by '\t'
- -- export-dir <directory path>

12. Oozie

- If you are working with large datasets one job is not enough
- If no. of jobs are working, one job output is require for another job.
- So we need some co-ordination between these Jobs
- The **Co-ordination between Jobs are taken by 'Oozie'**
- Oozie is a server called as Workflow Engine
- Oozie have two nodes
 - **Control node** → taken care where Job is started(one only)
 - **Action node** → Job is running by Action node (No.of nodes)



- And we need to configure **workflow.xml** to work with Oozie
- All Oozie configurations taken care by workflow.xml
- Workflow.xml is similar to web.xml in Servlets
- For running Oozie we need run **BOOSTRAP** service

\$sudo /etc/init.d/Oozie start

13. Hive(HiveQL)

- Hadoop accepts Structure , semistructure, unstruvture data
- Hive is used to work with **Structured Data**
- Varchar,char,int datatypes in Mysql , Hive has also it's datatypes
- Here every table created as a directory
- We cannot use insert/ update / Delete commands

Tables in Hive:

There are two types of tables in hive:

- 1) Managed Tables
- 2) External Tables

Managed Tables:

- Controls metadata and lifecycle of data.
- data stored in hive.metastore.warehouse.dir
- Dropping managed tables delete all data and related metadata.

For practice of commands, before creating table first you have to create file which contains data in local file system. For example create file kb.txt in "/home/demo/kb.txt" which contains information like:

```
101,raj
102,sham
103,dipak
104,sachin
105,ravi
```

create a table according to content of file i.e id and name.

1) hive > create table student(id int, name string)

> row format delimited fields terminated by ','

> stored as textfile;

('student' table is created with 2 fields id and name. Type of each columns must be specified. row format fields indicates that each row in the data file is , delimited text.)

2) hive > show tables;

3) hive > describe student;

4) hive > describe extended student;

//show all details of table.(owner, creation type, data types, table type(managed/external), creation time, stored data format, location where the table file is stored and so on...)

5) **hive > describe formatted student;**

// show details of table in formatted manner.

6) Alter table commands have so many options:

(rename, drop, add column, change column name, replace column)

hive > alter table student rename to stud;

7) **hive > drop table if exists stud;**

8) **hive > load data local inpath '/home/demo/kb.txt' into table student;**

//file is present in local file system then use 'local' keyword.

9) **hive > load data inpath '/input/kb.txt' into table student;**

// file kb.txt present in hdfs. so dont use 'local' keyword.

10) **hive > select * from student;**

11) **select count(*) from student;**

External Tables:

- Stored in directory outside of hive.
- Useful if sharing data with other tools.
- Dropping delete only metadata not actual data.
- must add external and location keywords to create statement.

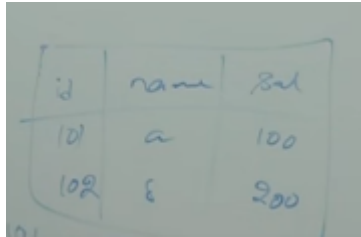
14. HBase

It is a NoSQL database

All data is taken as [KEY, VALUE] pair

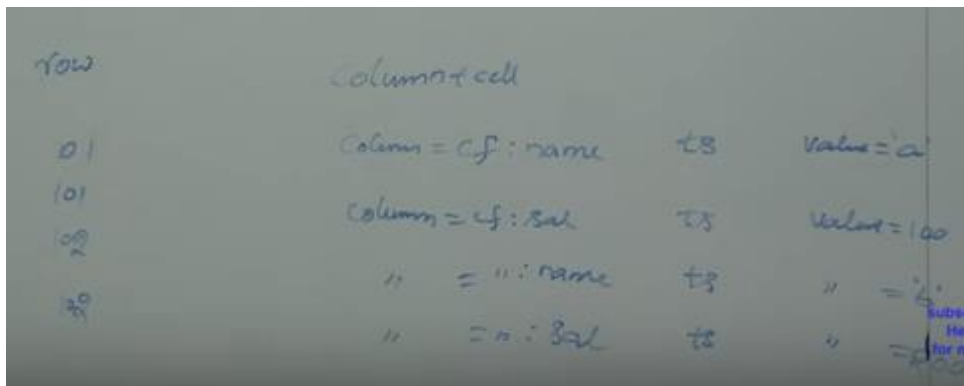
[KEY, VALUE] pair are in the form of **ByteArray** type

In mysql data stored in row format



id	name	Sal
101	a	100
102	b	200

In HBase data stored in column format



rowid	column+cell
01	Column = cf: name ts Value = 'a'
101	Column = cf: Sal ts Value = 100
102	" = " : name ts " = 'b'
20	" = " : Sal ts " = 200

Each column is saved as separate row

Can have duplicate key, but unique timestamp

To work with HBase, HMaster service should be started

```
Hbase -start.all
```

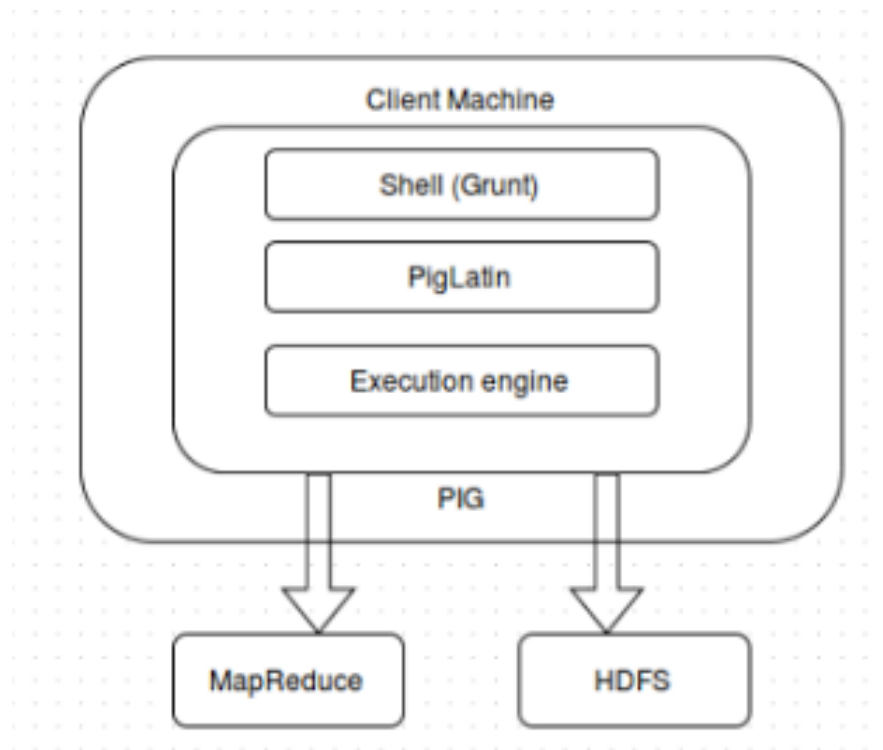
```
$sudo /etc/init.d/hadoop-hbase-master start ←|
```

15. PIG

PIG – has own language PIG LATIN

If People don't have knowledge on SQL,java they can directly use PIG

Built on Top of MapReduce



Handwritten table illustrating data structure:

id	name	sal
101	a	10
102	b	20
103	c	30

Annotations: 'row' points to a row, 'attribute (or) tuple' points to a column.

Handwritten table illustrating data structure as tuples:

101	a	10
102	b	20
103	c	30

Annotations: 'tuple' points to a row, 'atom' points to a value. A watermark 'subscribe here for more' is visible.

PIG don't have metadata, it wont store any data

Pig is divided into two parts:

1. PigLatin - language for expressing data flow. Operates like ETL (Extract , Transform and Load)
2. Execution environment to run PigLatin. Which divides into two parts:

1. Local execution in single JVM
2. Distributed execution on Hadoop Cluster

PIG has its own datatypes

Data Types in Pig:

1. Scalar Data type : int, long, float, double
2. Array Data type : bytearray, chararray
3. Composite Data Type : tuple, bag, atom (table -> bag, row/record -> tuple, attribute/tuple -> atom)

Pig Execution types:

1. local mode : `$ pig -x local` //Stores file in local file system (not on hdfs) and then process it.
2. Hadoop cluster mode : `$ pig -x mapreduce` // stores file in HDFS and then process it.
3. Executing pig script : Script is a file contains pig commands for execution, it runs in both modes.

`$ pig -x local demoscrypt.pig` (local mode)

`$ pig demoscrypt.pig` (mapreduce mode)

Running Pig Programs:

There are 3 ways to run pig programs all work in both local and hadoop mode:

1. Script : Pig can run script file that contains Pig commands. ex. `pig.script.pig`
2. Grunt : Interactive shell for running pig commands.
3. Embedded : Run Pig program from Java using the `PigServer` class like use JDBC to run SQL programs.

PIG Data:

- Scalar :
ex. int 2, chararray two, double 2.0
 - tuple : (fields are grouped together into tuples)
ex. (101,ram,25000.0)
 - bag : (tuples are grouped into bags)
ex. (101,ram,25000.0)
 (102,sham,35000.0)
 (103,ajay)
- All tuples need not contain same fields.
- Map :
ex. [1#ram,2#sham] (key-value format)
 - Relation : (outer_bag{inner_bag})

(Sachin {bat,ball,field})
(Rahul {bat,field})
(Kumar {ball,field})

INPUT functions in PIG:

Several built in operation for input. All functions are case sensitive.

1) BinStorage()

Data in machine readable format. Used internally by pig.

2) PigStorage('fields_delimiter')

Default delimiter is tab. Default input function.

3) TextLoader()

Load unstructured text.

4) JsonLoader()

Load data in a slandered JSON format.

REF.

<https://www.youtube.com/watch?v=u8HarSx7gIE&list=PLpc4L8tPSURCdIXH5FspLDUesmTGRQ39I&index=9>