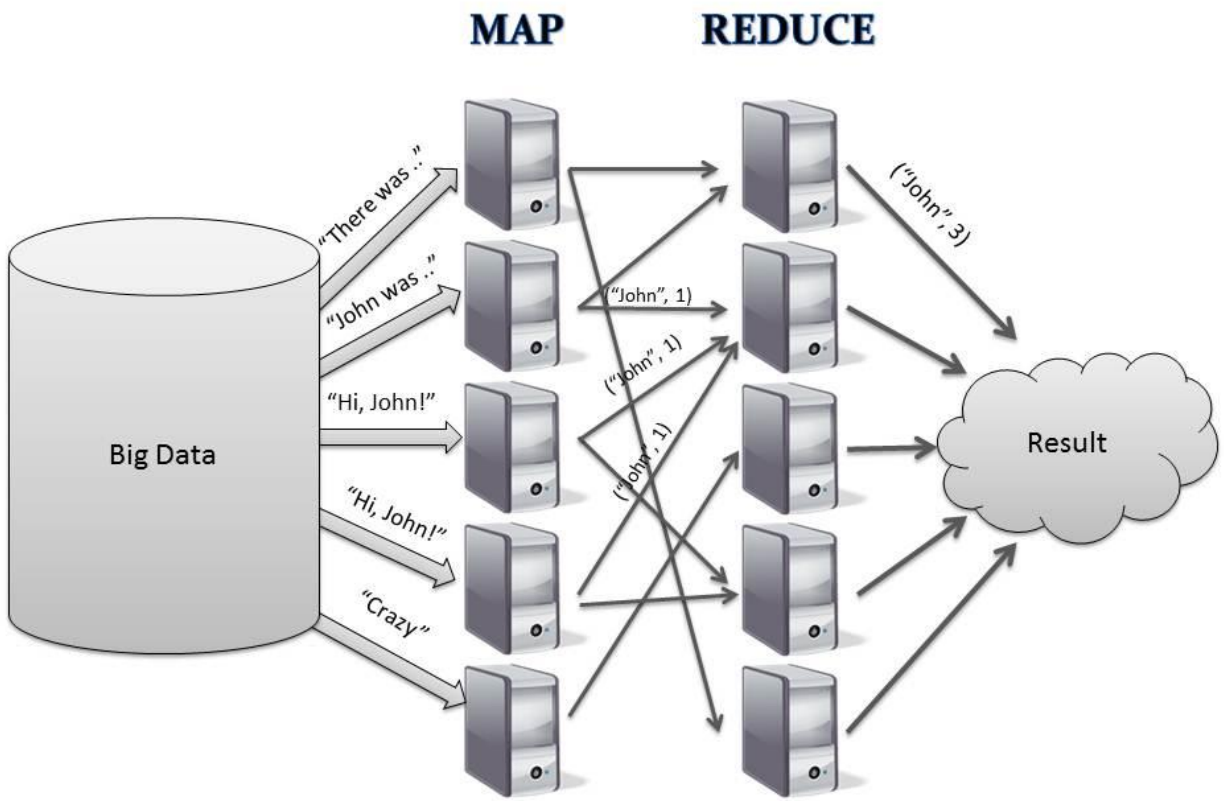
MapReduce

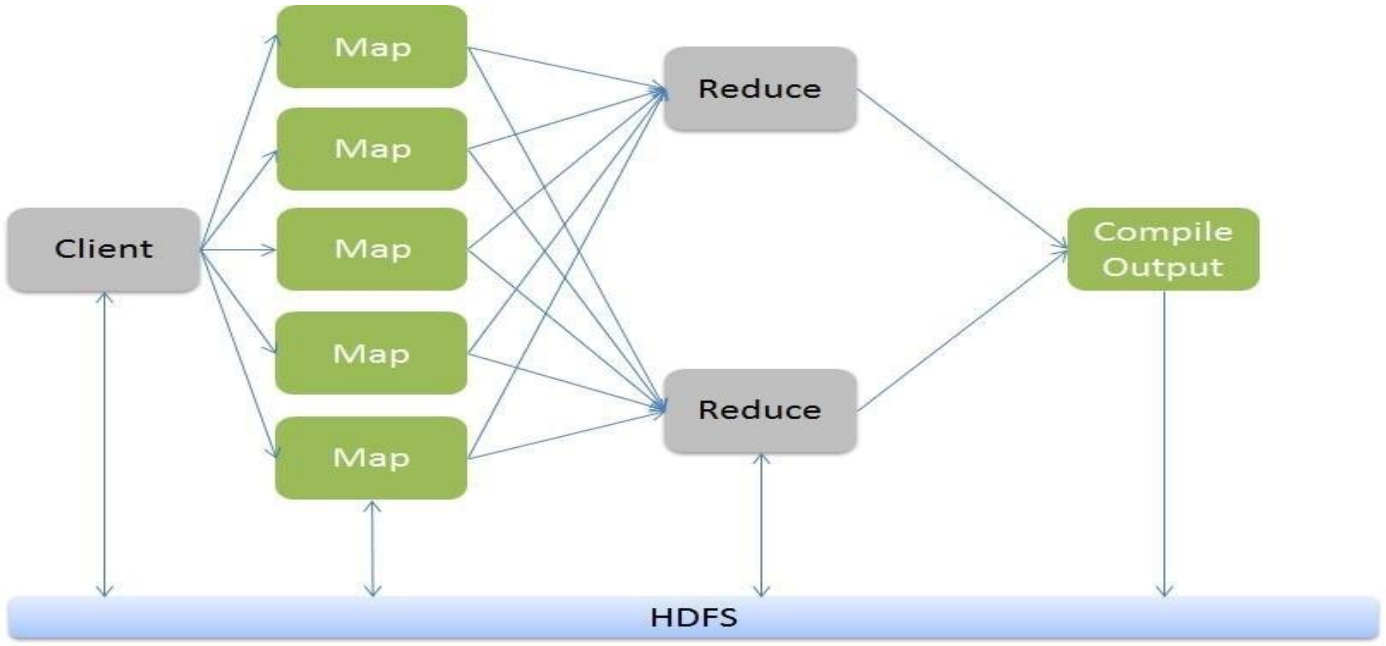

MAP
REDUCE
Big Data
"There was .."
"John was .."
"Hi, John!"
"Hi, John!"
"Crazy"
("John", 1)
("John", 1)
("John", 1)
("John", 3)
Result

Agenda

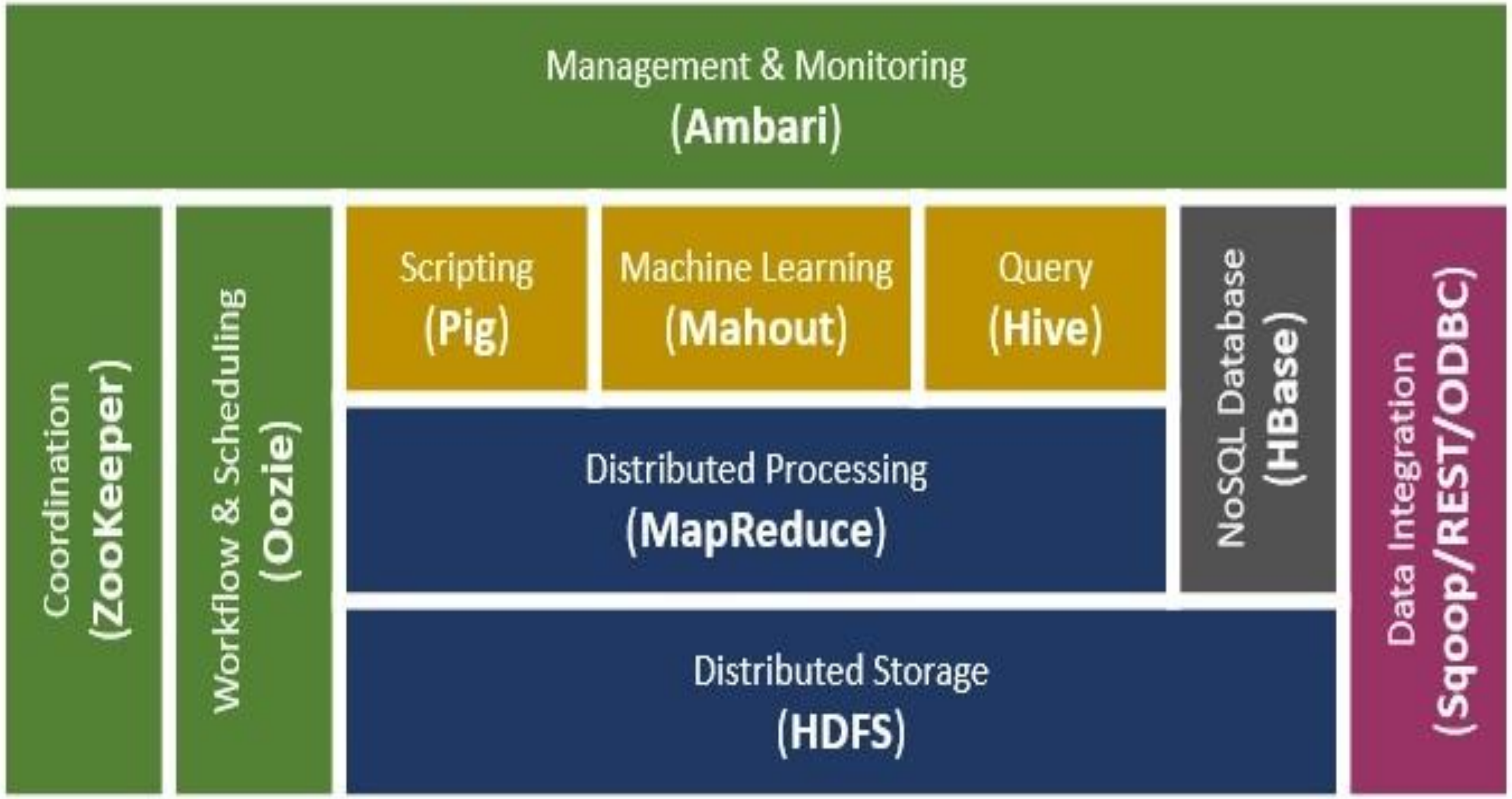
- What is Map Reduce and concept
- Terminology
- Eco System of Map Reduce
- Bear, Deer, River and Car Example
- Java Hands on Lab
- Advantage of MapReduce
- Uses
- The Algorithm
- FAQ

What is MapReduce?

- MapReduce is a framework using which we can write applications to process huge amounts of data, in parallel, on large clusters of commodi
- MapReduce is a processing technique and a program model for distributed computing based on java. The MapReduce algorithm contains two important tasks, namely Map and Reduce.



Apache Hadoop Ecosystem



What is MapReduce?

- Hadoop MapReduce (Hadoop Map/Reduce) is a software framework for distributed processing of large data sets on computing clusters. It is a sub-project of the Apache Hadoop project.
- Apache Hadoop is an open-source framework that allows to store and process big data in a distributed environment across clusters of computers using simple programming models.
- MapReduce is the core component for data processing in Hadoop framework. In layman's term Mapreduce helps to split the input data set into a number of parts and run a program on all data parts parallel at once. The term MapReduce refers to two separate and distinct tasks. The first is the map operation, takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs). The reduce operation combines those data tuples based on the key and accordingly modifies the value of the key.

What is MapReduce?

- MapReduce is a programming framework that allows us to perform distributed and parallel processing on large data sets in a distributed environment.
- MapReduce consists of two distinct tasks – Map and Reduce.
- As the name MapReduce suggests, reducer phase takes place after mapper phase has been completed.
- So, the first is the map job, where a block of data is read and processed to produce key-value pairs as intermediate outputs.
- The output of a Mapper or map job (key-value pairs) is input to the Reducer.
- The reducer receives the key-value pair from multiple map jobs.
- Then, the reducer aggregates those intermediate data tuples (intermediate key-value pair) into a smaller set of tuples or key-value pairs which is the final output.

Terminology

- **PayLoad** - Applications implement the Map and the Reduce functions, and form the core of the job.
- **Mapper** - Mapper maps the input key/value pairs to a set of intermediate key/value pair.
- **NamedNode** - Node that manages the Hadoop Distributed File System (HDFS).
- **DataNode** - Node where data is presented in advance before any processing takes place.
- **MasterNode** - Node where JobTracker runs and which accepts job requests from clients.
- **SlaveNode** - Node where Map and Reduce program runs.
- **JobTracker** - Is the Master of the System which manages the Job and Resource on the cluster. Schedules jobs and tracks the assign jobs to Task tracker.
- **Task Tracker** - Are the Slaves. Deploying in every machines. Which is responsible for running the map and reduce task as instrcuted by the jobTrackers
- **Job** - A program is an execution of a Mapper and Reducer across a dataset.
- **Task** - An execution of a Mapper or a Reducer on a slice of data.
- **Task Attempt** - A particular instance of an attempt to execute a task on a SlaveNode.

MapReduce consists of 2 steps:

- It takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (Key-Value pair).

Reduce Function

Takes the output from Map as an input and combines those data tuples into a smaller set of tuples.

Workflow of MapReduce

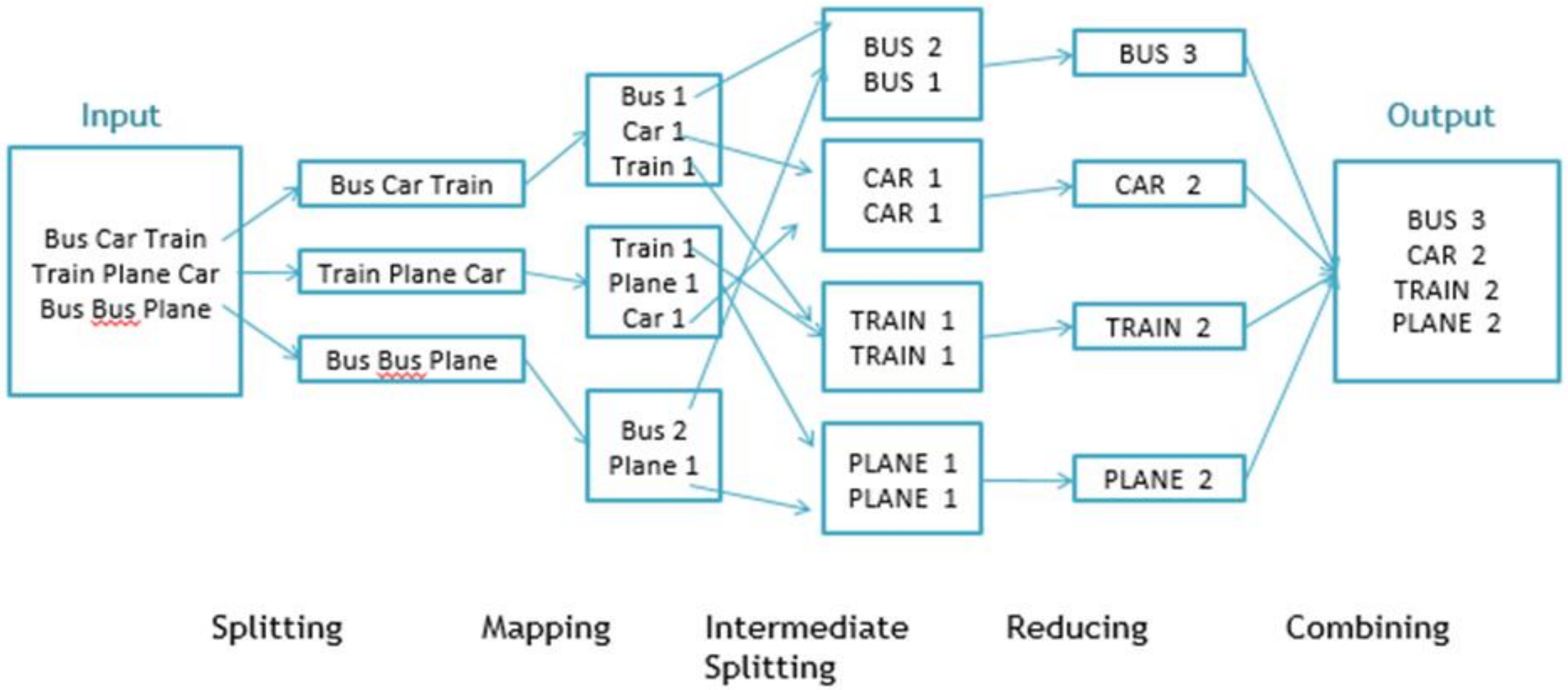
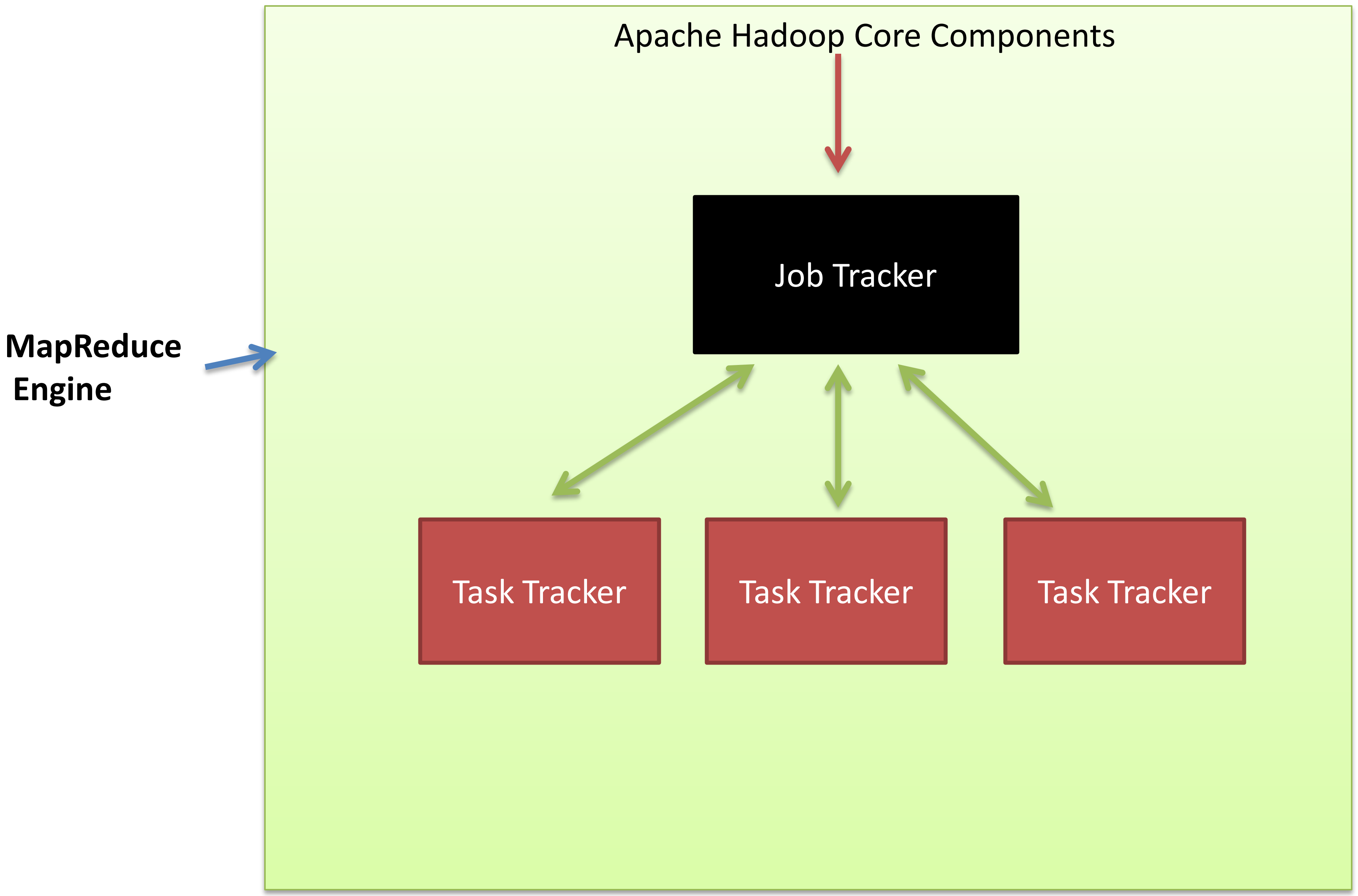
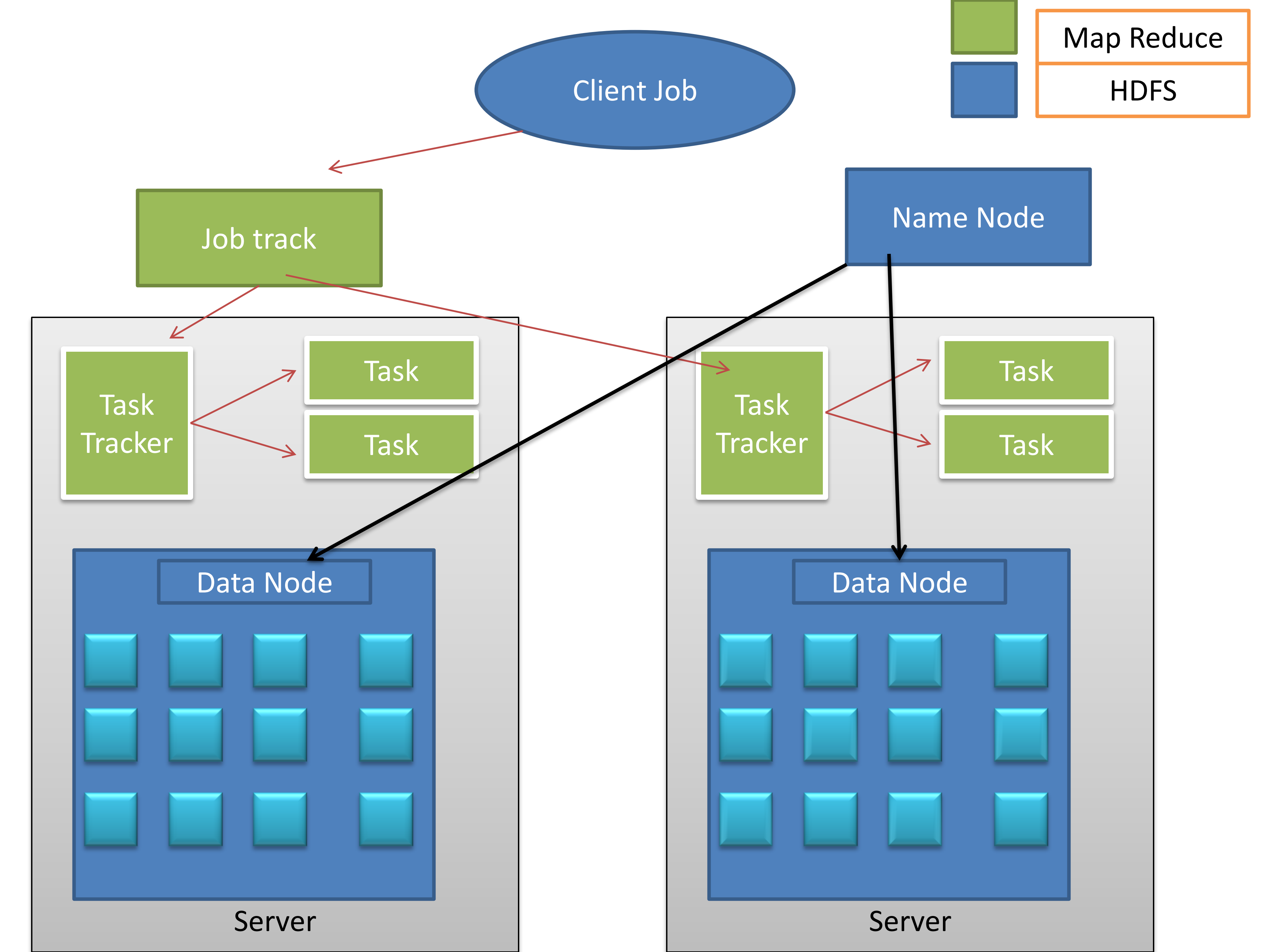


Fig. WorkFlow of MapReducing

Workflow of MapReduce consists of 5 steps

- **Splitting** – The splitting parameter can be anything, e.g. splitting by space, comma, semicolon, or even by a new line (‘\n’).
- **Mapping** – as explained above
- **Intermediate splitting** – the entire process in parallel on different clusters. In order to group them in “Reduce Phase” the similar KEY data should be on same cluster.
- **Reduce** – it is nothing but mostly group by phase
- **Combining** – The last phase where all the data (individual result set from each cluster) is combine together to form a Result


Apache Hadoop Core Components
Job Tracker
MapReduce Engine
Task Tracker
Task Tracker
Task Tracker


Map Reduce
HDFS
Client Job
Job track
Name Node
Task Tracker
Task
Task
Data Node
Server
Task Tracker
Task
Task
Data Node
Server

Workflow of MapReduce consists of 5 steps

- **Splitting** – The splitting parameter can be anything, e.g. splitting by space, comma, semicolon, or even by a new line (‘\n’).
- **Mapping** – as explained above
- **Intermediate splitting** – the entire process in parallel on different clusters. In order to group them in “Reduce Phase” the similar KEY data should be on same cluster.
- **Reduce** – it is nothing but mostly group by phase
- **Combining** – The last phase where all the data (individual result set from each cluster) is combine together to form a Result

The Algorithm

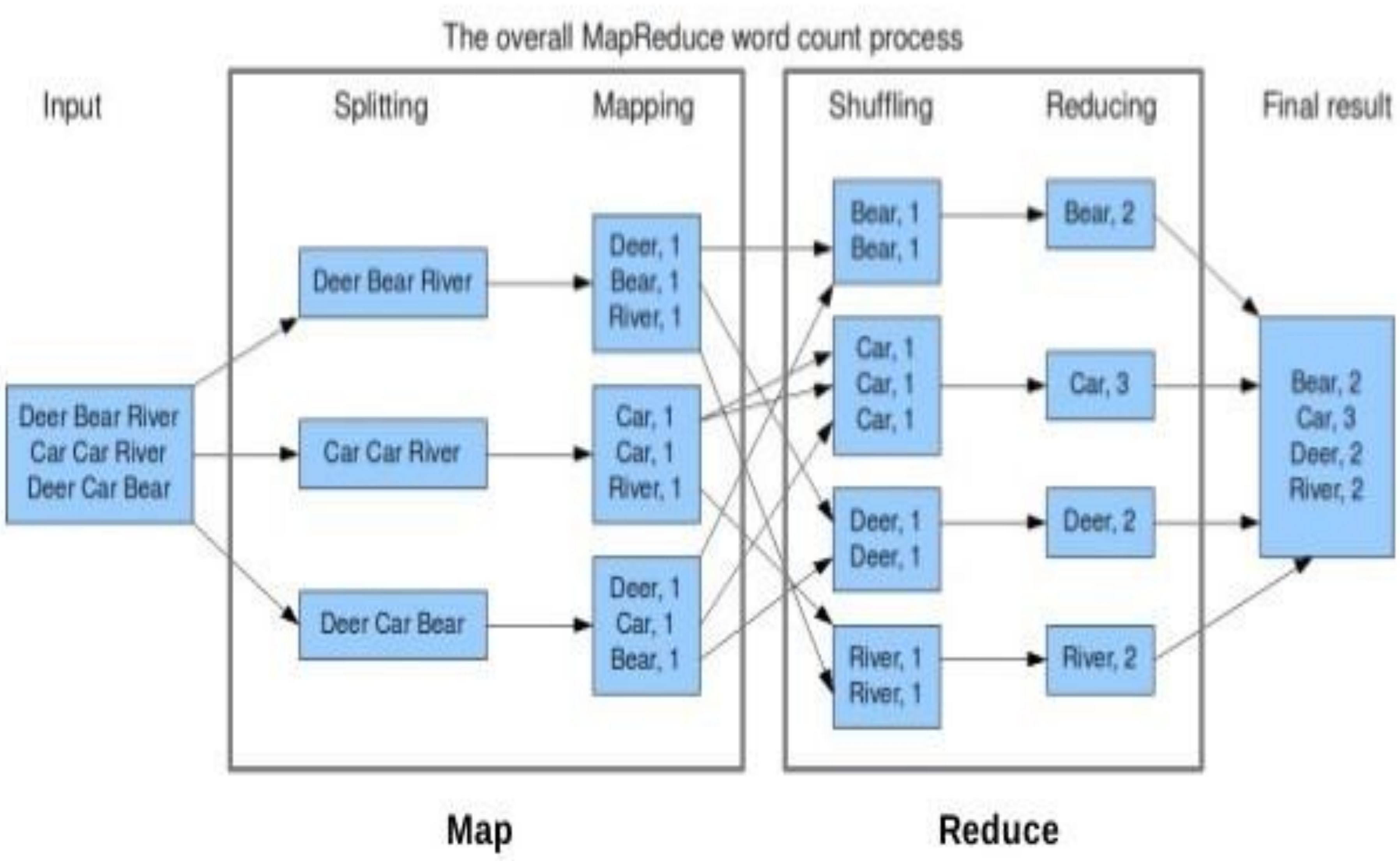
- Generally MapReduce paradigm is based on sending the computer to where the data resides!
- MapReduce program executes in three stages, namely map stage, shuffle stage, and reduce stage.

1. **Map stage** : The map or mapper's job is to process the input data. Generally the input data is in the form of file or directory and is stored in the Hadoop file system (HDFS). The input file is passed to the mapper function line by line. The mapper processes the data and creates several small chunks of data.
2. **Reduce stage** : This stage is the combination of the **Shuffle** stage and the **Reduce** stage. The Reducer's job is to process the data that comes from the mapper. After processing, it produces a new set of output, which will be stored in the HDFS.

Bear, Deer, River and Car Example

- The following word count example explains MapReduce method. For simplicity, let's consider a few words of a text document.
- We want to find the number of occurrence of each word. First the input is split to distribute the work among all the map nodes as shown in the figure. Then each word is identified and mapped to the number one. Thus the pairs also called as tuples are created.
- In the first mapper node three words Deer, Bear and River are passed.
- Thus the output of the node will be three key, value pairs with three distinct keys and value set to one. The mapping process remains the same in all the nodes. These tuples are then passed to the reduce nodes. A partitioner comes into action which carries out shuffling so that all the tuples with same key are sent to same node.

Bear, Deer, River and Car Example



Description of Bear, Deer, River and Car Example

- First, we divide the input in three splits as shown in the figure. This will distribute the work among all the map nodes.
- Then, we tokenize the words in each of the mapper and give a hardcoded value (1) to each of the tokens or words. The rationale behind giving a hardcoded value equal to 1 is that every word, in itself, will occur once.
- Now, a list of key-value pair will be created where the key is nothing but the individual words and value is one. So, for the first line (Dear Bear River) we have 3 key-value pairs – Dear, 1; Bear, 1; River, 1. The mapping process remains the same on all the nodes.
- After mapper phase, a partition process takes place where sorting and shuffling happens so that all the tuples with the same key are sent to the corresponding reducer.
- So, after the sorting and shuffling phase, each reducer will have a unique key and a list of values corresponding to that very key. For example, Bear, [1,1]; Car, [1,1,1].., etc.
- Now, each Reducer counts the values which are present in that list of values. As shown in the figure, reducer gets a list of values which is [1,1] for the key Bear. Then, it counts the number of ones in the very list and gives the final output as – Bear, 2.
- Finally, all the output key/value pairs are then collected and written in the output file.

Bear, Deer, River and Car Example

- The Reducer node processes all the tuples such that all the pairs with same key are counted and the count is updated as the value of that specific key. In the example there are two pairs with the key ‘Bear’ which are then reduced to single tuple with the value equal to the count. All the output tuples are then collected and written in the output file.

Hands On- Explanation of MapReduce Program

- The entire MapReduce program can be fundamentally divided into three parts:

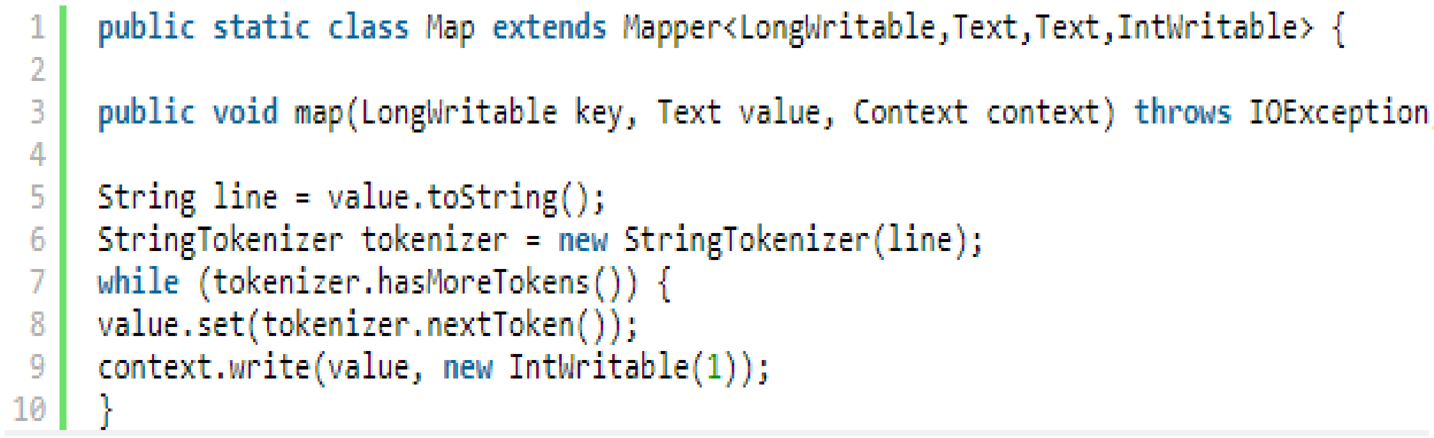
1. Mapper Phase Code
2. Reducer Phase Code
3. Driver Code

- We will understand the code for each of these three parts sequentially.

Inputs and Outputs (Java Perspective)

- The MapReduce framework operates on <key, value> pairs, that is, the framework views the input to the job as a set of <key, value> pairs and produces a set of <key, value> pairs as the output of the job, conceivably of different types.
- The key and the value classes should be in serialized manner by the framework and hence, need to implement the Writable interface sorting by the framework. Input and Out. Additionally, the key classes have to implement the Writable-Comparable interface to facilitateput types of a MapReduce job:
- **(Input) <k1, v1> -> map -> <k2, v2>-> reduce -> <k3, v3>(Output).**

Mapper code:



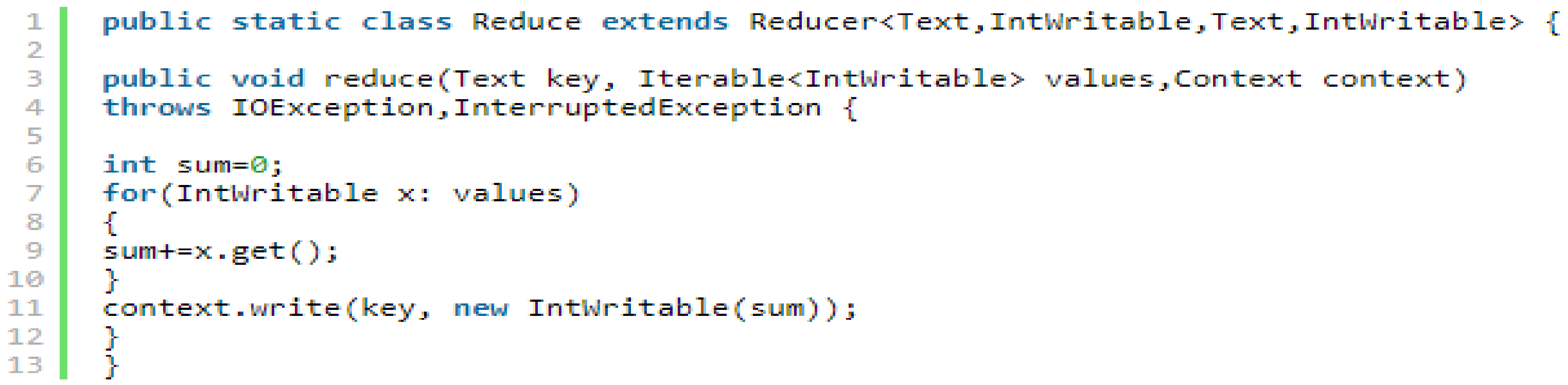
```java
public static class Map extends Mapper<LongWritable,Text,Text,IntWritable> {

public void map(LongWritable key, Text value, Context context) throws IOException,InterruptedException {

String line = value.toString();
StringTokenizer tokenizer = new StringTokenizer(line);
while (tokenizer.hasMoreTokens()) {
value.set(tokenizer.nextToken());
context.write(value, new IntWritable(1));
}
```

Input Text File

Key	**Value**
0	Dear Bear River
121	Car Car River
226	Deer Car Bear

- **We have created a class Map that extends the class Mapper which is already defined in the MapReduce Framework.**
- **We define the data types of input and output key/value pair after the class declaration using angle brackets.**
- **Both the input and output of the Mapper is a key/value pair.**
- **Input:**
  - **The *key* is nothing but the offset of each line in the text file:*LongWritable***
  - **The *value* is each individual line (as shown in the figure at the right): *Text***
- **Output:**
  - **The *key* is the tokenized words: *Text***
  - **We have the hardcoded *value* in our case which is 1: *IntWritable***
  - **Example – Dear 1, Bear 1, etc.**
- **We have written a java code where we have tokenized each word and assigned them a hardcoded value equal to *1*.**

Reducer Code:



```
public static class Reduce extends Reducer<Text,IntWritable,Text,IntWritable> {

public void reduce(Text key, Iterable<IntWritable> values,Context context)
throws IOException,InterruptedException {

int sum=0;
for(IntWritable x: values)
{
sum+=x.get();
}
context.write(key, new IntWritable(sum));
}
}
```

We have created a class Reduce which extends class Reducer like that of Mapper.
We define the data types of input and output key/value pair after the class declaration using angle brackets as done for Mapper.
Both the input and the output of the Reducer is a key-value pair.
Input:

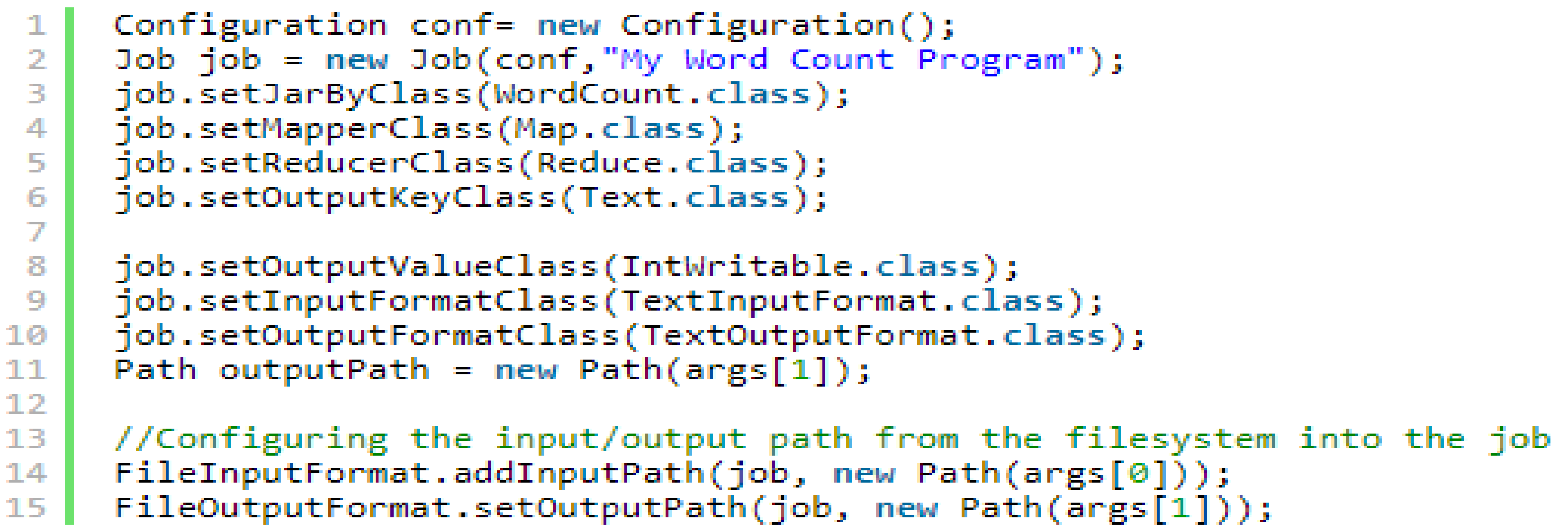
- The *key* nothing but those unique words which have been generated after the sorting and shuffling phase: *Text*
- The *value* is a list of integers corresponding to each key: *IntWritable*
- Example – Bear, [1, 1], etc.

Output:

- The *key* is all the unique words present in the input text file: *Text*
- The *value* is the number of occurrences of each of the unique words: *IntWritable*
- Example – Bear, 2; Car, 3, etc.

We have aggregated the values present in each of the list corresponding to each key and produced the final answer.
In general, a single reducer is created for each of the unique words, but, you can specify the number of reducer in mapred-site.xml.

Driver Code:



```
Configuration conf= new Configuration();
Job job = new Job(conf,"My Word Count Program");
job.setJarByClass(WordCount.class);
job.setMapperClass(Map.class);
job.setReducerClass(Reduce.class);
job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);
job.setInputFormatClass(TextInputFormat.class);
job.setOutputFormatClass(TextOutputFormat.class);
Path outputPath = new Path(args[1]);

//Configuring the input/output path from the filesystem into the job
FileInputFormat.addInputPath(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));
```

In the driver class, we set the configuration of our MapReduce job to run in Hadoop.
We specify the name of the job , the data type of input/output of the mapper and reducer.
We also specify the names of the mapper and reducer classes.
The path of the input and output folder is also specified.
The method setInputFormatClass () is used for specifying that how a Mapper will read the input data or what will be the unit of work. Here, we have chosen TextInputFormat so that single line is read by the mapper at a time from the input text file.
The main () method is the entry point for the driver. In this method, we instantiate a new Configuration object for the job.

Advantages of MapReduce

- **Parallel Processing:**
- In MapReduce, we are dividing the job among multiple nodes and each node works with a part of the job simultaneously. So, MapReduce is based on Divide and Conquer paradigm which helps us to process the data using different machines. As the data is processed by multiple machine instead of a single machine in parallel, the time taken to process the data gets reduced by a tremendous amount

Advantages of MapReduce

- **Data Locality:**
- Instead of moving data to the processing unit, we are moving processing unit to the data in the MapReduce Framework. In the traditional system, we used to bring data to the processing unit and process it. But, as the data grew and became very huge, bringing this huge amount of data to the processing unit posed following issues:
- Moving huge data to processing is costly and deteriorates the network performance.
- Processing takes time as the data is processed by a single unit which becomes the bottleneck.
- Master node can get over-burdened and may fail.
- Now, MapReduce allows us to overcome above issues by bringing the processing unit to the data. So, as you can see in the above image that the data is distributed among multiple nodes where each node processes the part of the data residing on it. This allows us to have the following advantages:
- It is very cost effective to move processing unit to the data.
- The processing time is reduced as all the nodes are working with their part of the data in parallel.
- Every node gets a part of the data to process and therefore, there is no chance of a node getting overburdened.

Advantage of MapReduce:

- The major advantage of MapReduce is that it is easy to scale data processing over multiple computing nodes. Under the MapReduce model, the data processing primitives are called mappers and reducers.
- This simple scalability is what has attracted many programmers to use the MapReduce model.

Uses

- On a general note it is used in scenario of needle in a haystack or for continuous monitoring of a huge system statistics. One such example is monitoring the traffic in a country road network and handling the traffic flow to prevent a jam. One common example is analyzing and storage of twitter data. It is also used in Log analysis which consists of various summations.
- Log Analysis: Trouble shooting, Audit and Security checks
- Breath Fast Search
- Votes Casting

Questions and Answers

Q1: what is Sequencefileinputformat?

Answer : Sequencefileinputformat is used for reading files in sequence. It is a specific compressed binary file format which is optimized for passing data between the output of one MapReduce job to the input of some other MapReduce job.

Q2:What does conf.setMapper Class do?

Answer : Conf.setMapperclass sets the mapper class and all the stuff related to map job such as reading a data and generating a key-value pair out of the mapper.

Q3: What are the methods in the Reducer class and order of their invocation?

Answer: The Reducer class contains the run() method, which call its own setup() method only once, it also call a reduce() method for each input and finally calls it cleanup() method.

Q4 what is the purpose of RecordReader in Hadoop?

Answer: In Hadoop, the RecordReader loads the data from its source and converts it into key, value pairs suitable for reading by the Mapper.

Questions and Answers

Q5 Which interface needs to be implemented to create Mapper and Reducer for the Hadoop?

Answer: 1)apache.hadoop.mapreduce.Mapper

2)apache.hadoop.mapreduce.Reducer

Q6 Mention what are the main configuration parameters that user need to specify to run MapReduce Job ?

Answer: The user of MapReduce framework needs to specify the following:

- Job's input locations in the distributed file system
- Job's output location in the distributed file system
- Input format
- Output format
- Class containing the map function
- Class containing the reduce function
- JAR file containing the mapper, reducer and driver classes

Questions and Answers

Q7: What Mapper does?

Answer: Mapper is the first phase of MapReduce phase which process map task.Mapper reads key/value pairs and emit key/value pair.Maps are the individual tasks that transform input records into intermediate records.The transformed intermediate records do not need to be of the same type as the input records. A given input pair may map to zero or many output pairs.

Q8: What are the four basic parameters of a mapper?

Answer: The four basic parameters of a mapper are LongWritable, text; text and IntWritable. The first two represent input parameters and the second two represent intermediate output parameters.

Q9: What is the input type/format in MapReduce by default?

Answer: By default the type input type in MapReduce is 'text'.

Q10: Is it mandatory to set input and output type/format in MapReduce?

Answer: No, it is not mandatory to set the input and output type/format in MapReduce. By default, the cluster takes the input and the output type as 'text'.

Questions and Answers

Q11: How to set mappers and reducers for Hadoop jobs?

Answer: Users can configure JobConf variable to set number of mappers and reducers.job.setNumMaptasks() and job.setNumreduceTasks().

Q12: Is it important for Hadoop MapReduce jobs to be written in Java?

Answer: It is not necessary to write Hadoop MapReduce jobs in java but users can write MapReduce jobs in any desired programming language like Ruby, Perl, Python, R, Awk, etc. through the Hadoop Streaming API.

Reference Links

- https://www.youtube.com/watch?v=ht3dNvdNDzI
- https://www.youtube.com/watch?v=SqvAaB3vK8U
- https://www.youtube.com/playlist?list=PL9ooVrP1hQOFrYxqxb0NJCdCABPZNo0pD
- https://www.youtube.com/watch?v=bcjSe0xCHbE
- https://www.youtube.com/watch?v=SqvAaB3vK8U
- https://www.edureka.co/blog/mapreduce-tutorial/
- http://www.bigdatatrunk.com/top-50-interview-quiz-mapreduce/