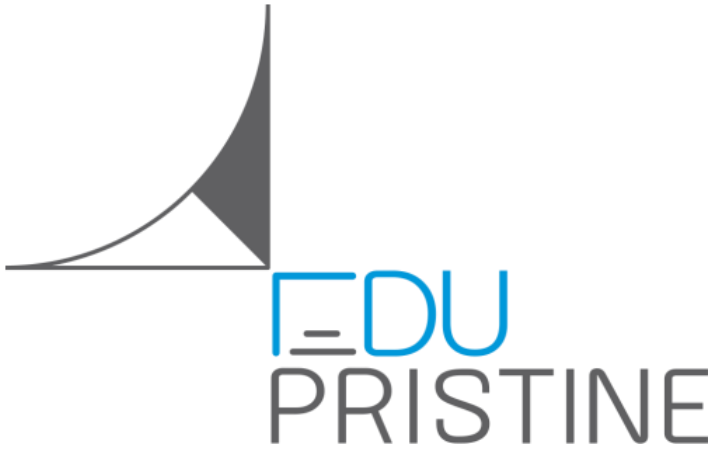


Map Reduce - Yarn

Map Reduce

- Programming model for processing large volumes.
- More relative to functional programming

Objectives

- Understand functional programming as it applies to MapReduce
- Understand the MapReduce program flow
- Understand how to write programs for Hadoop MapReduce
- Learn about additional features of Hadoop designed to aid software development.

MapReduce Basics

Functional Programming Concepts

List Processing

Mapping Lists

Reducing Lists

Putting them Together in MapReduce

An Example Application: Word Count

The Driver Method

FUNCTIONAL PROGRAMMING CONCEPTS

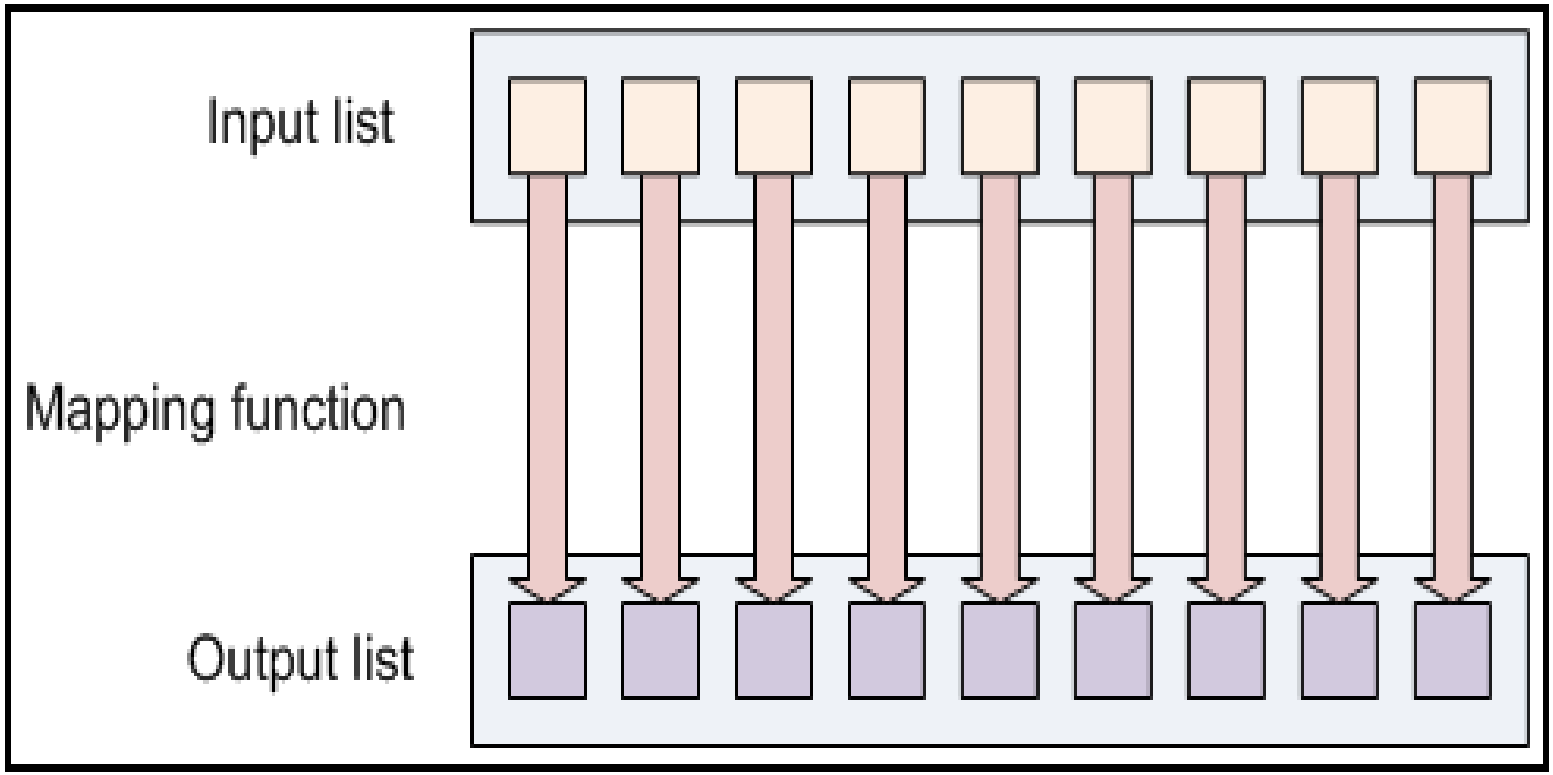
- Process large volumes of data.
- Data nodes share nothing.
- Elements in map reduce are immutable.

LIST PROCESSING

- Transform lists of input data elements into lists of output data elements.
- Processing idioms: *map*, and *reduce*.

MAPPING LISTS

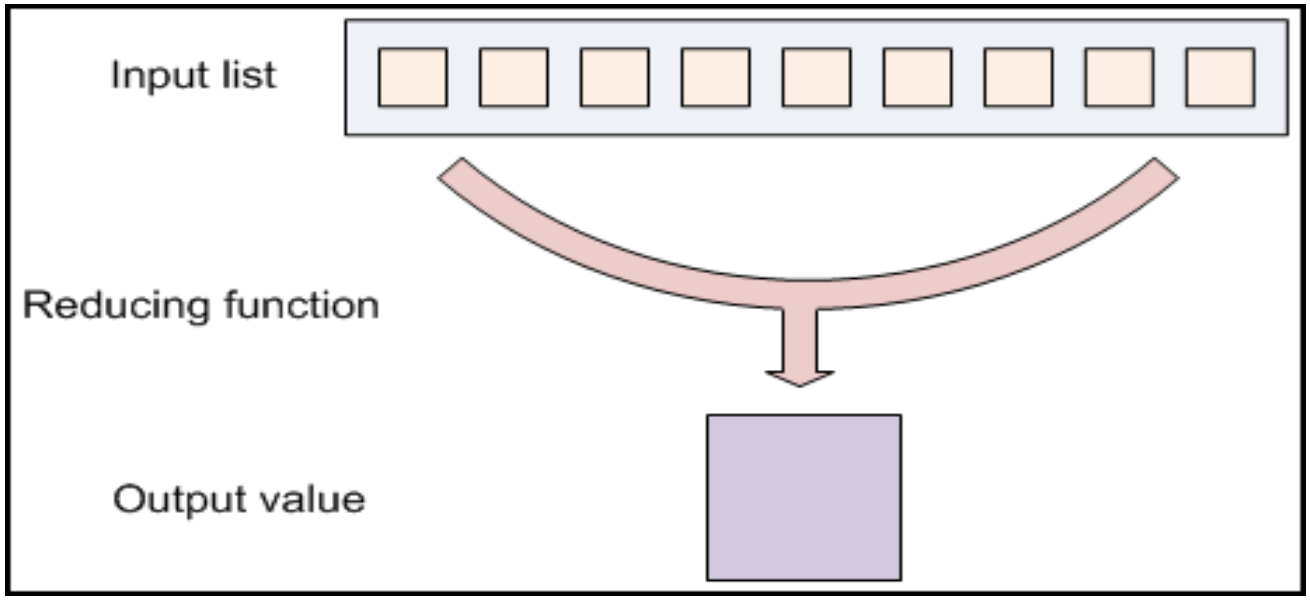
The first phase of a MapReduce program is called *mapping*.



REDUCING LISTS

Reducing lets you aggregate values together.

Iterating over values to get final results.



PUTTING THEM TOGETHER IN MAPREDUCE:

- Mapper
- Reducer
- Keys and values
- Keys divide the reduce space

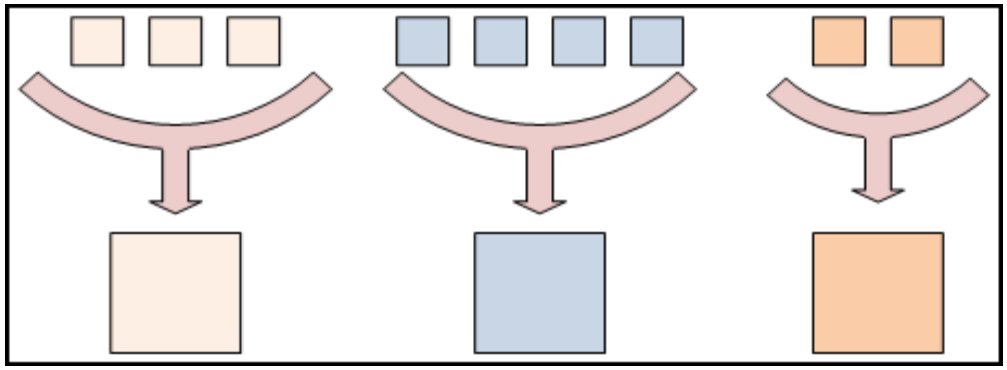
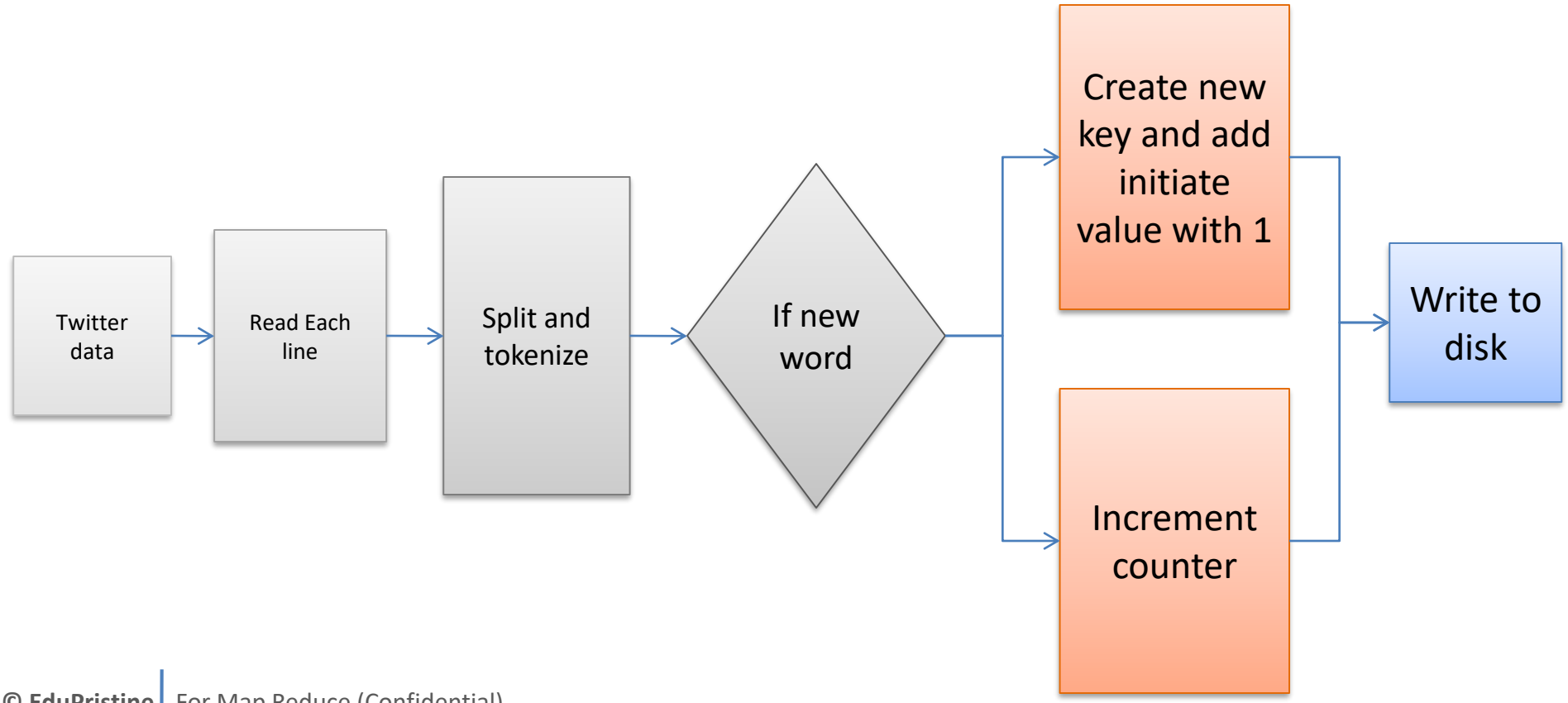


Figure 4.3: Different colors represent different keys. All values with the same key are presented to a single reduce task.

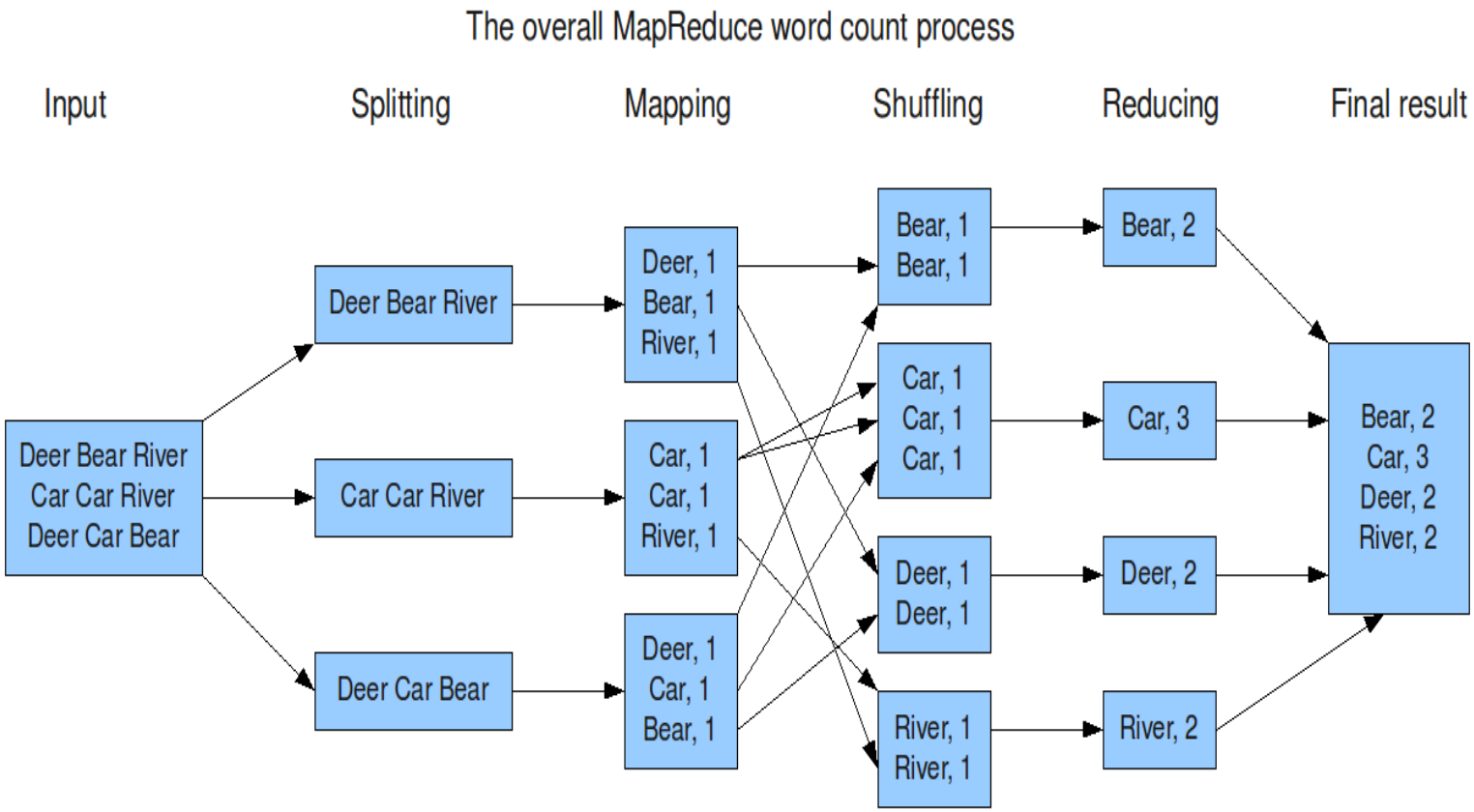
Word Count

- Sequential Programming
- 1 PB data to process in twitter to know the most popular actor.



Word Count

Parallel Programming



Retail Transactions

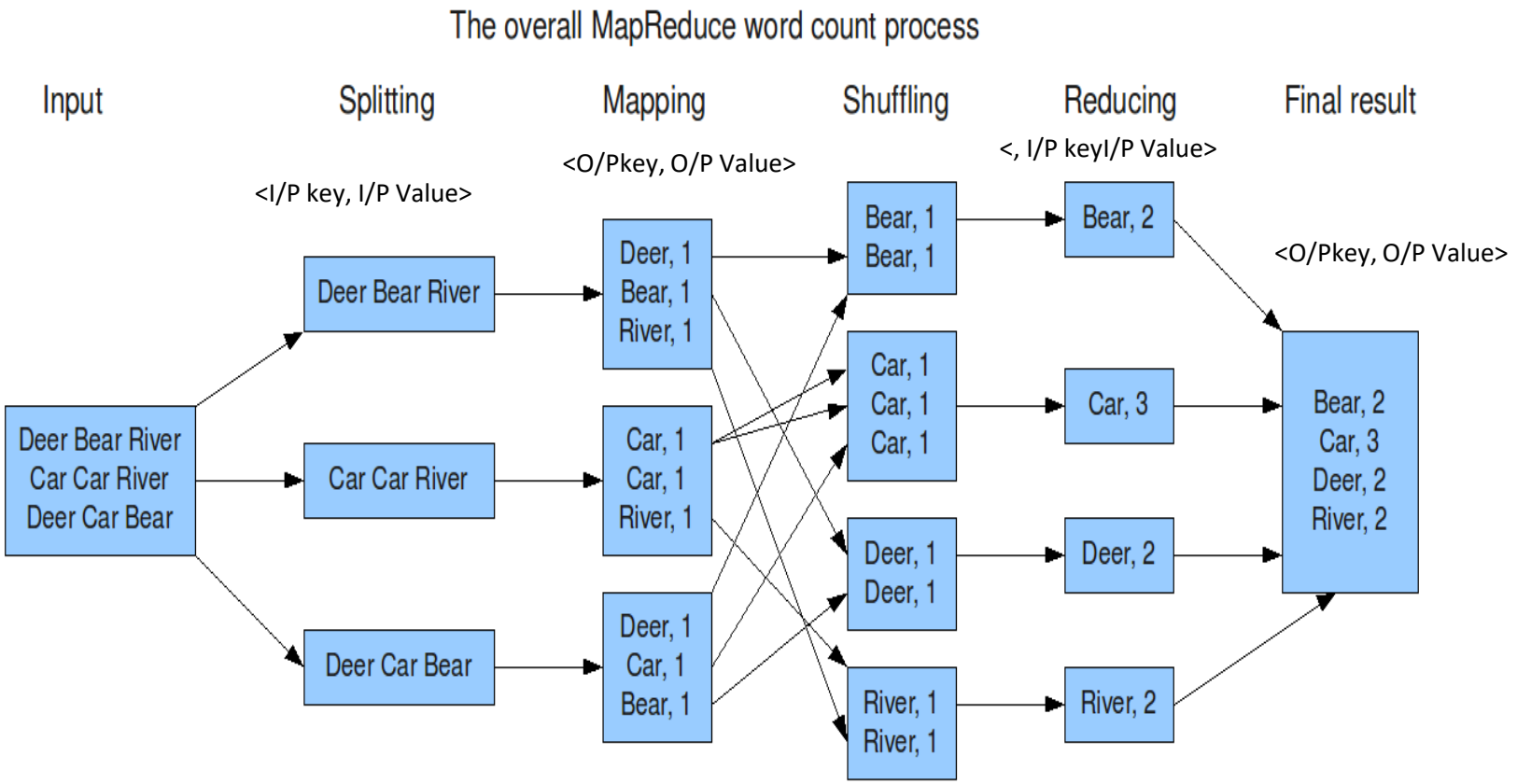
- Customer ID
- Date of purchase
- Brand
- Company
- Department
- Amount
- Quantity

Driving Patterns

- Driver ID
- Speed
- Time
- Geo location
- Delivered on time
- Delay

Word Count

Parallel Programming



Program

- Mapper
- Reducer
- Driver

Mapper code

```
public static class Map 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 line = value.toString();
       StringTokenizer tokenizer = new StringTokenizer(line);
       while (tokenizer.hasMoreTokens()) {
           word.set(tokenizer.nextToken());
           context.write(word, one);
       }
   }
}
```

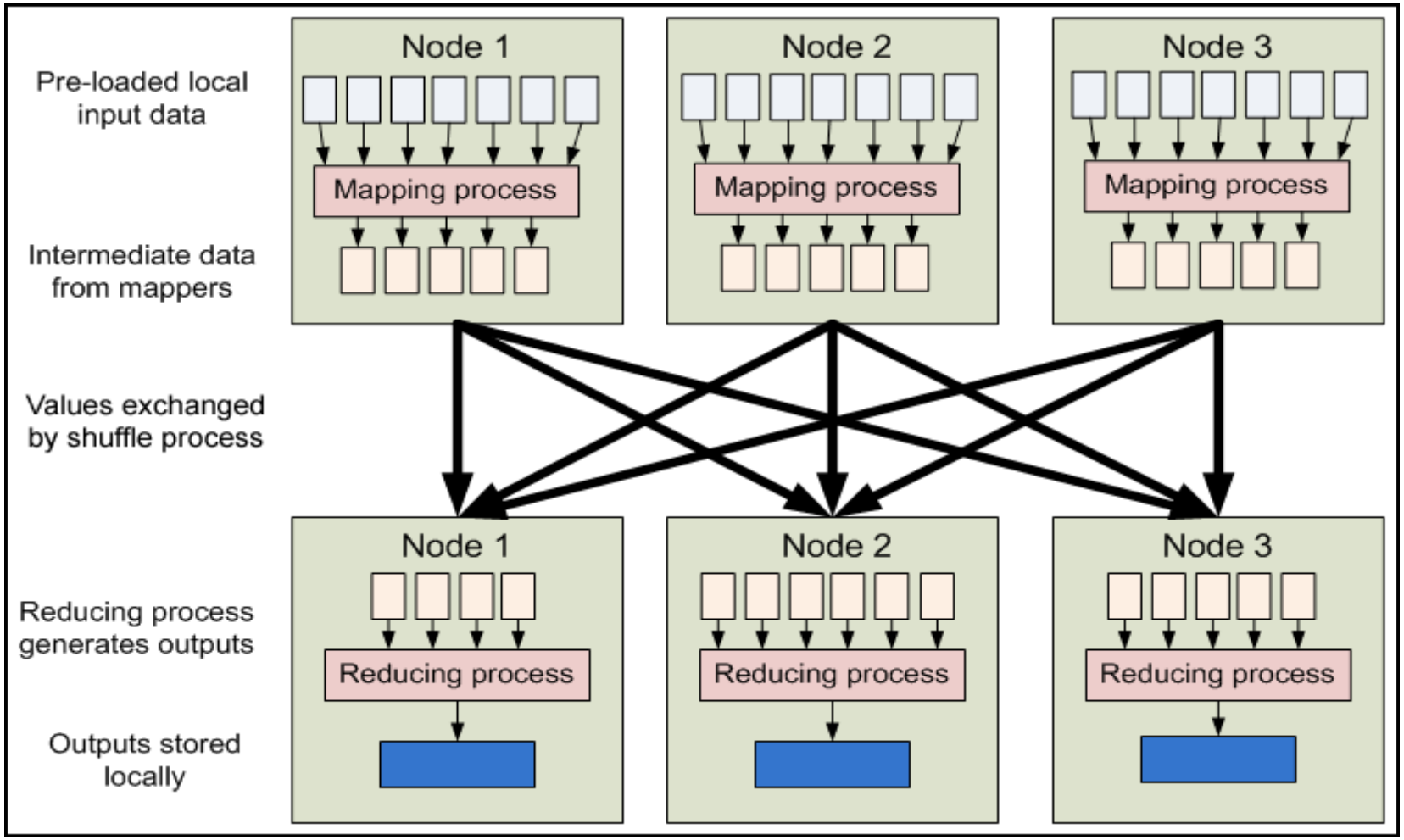
Reducer

```
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 val : values) {
           sum += val.get();
       }
       context.write(key, new IntWritable(sum));
   }
}
```

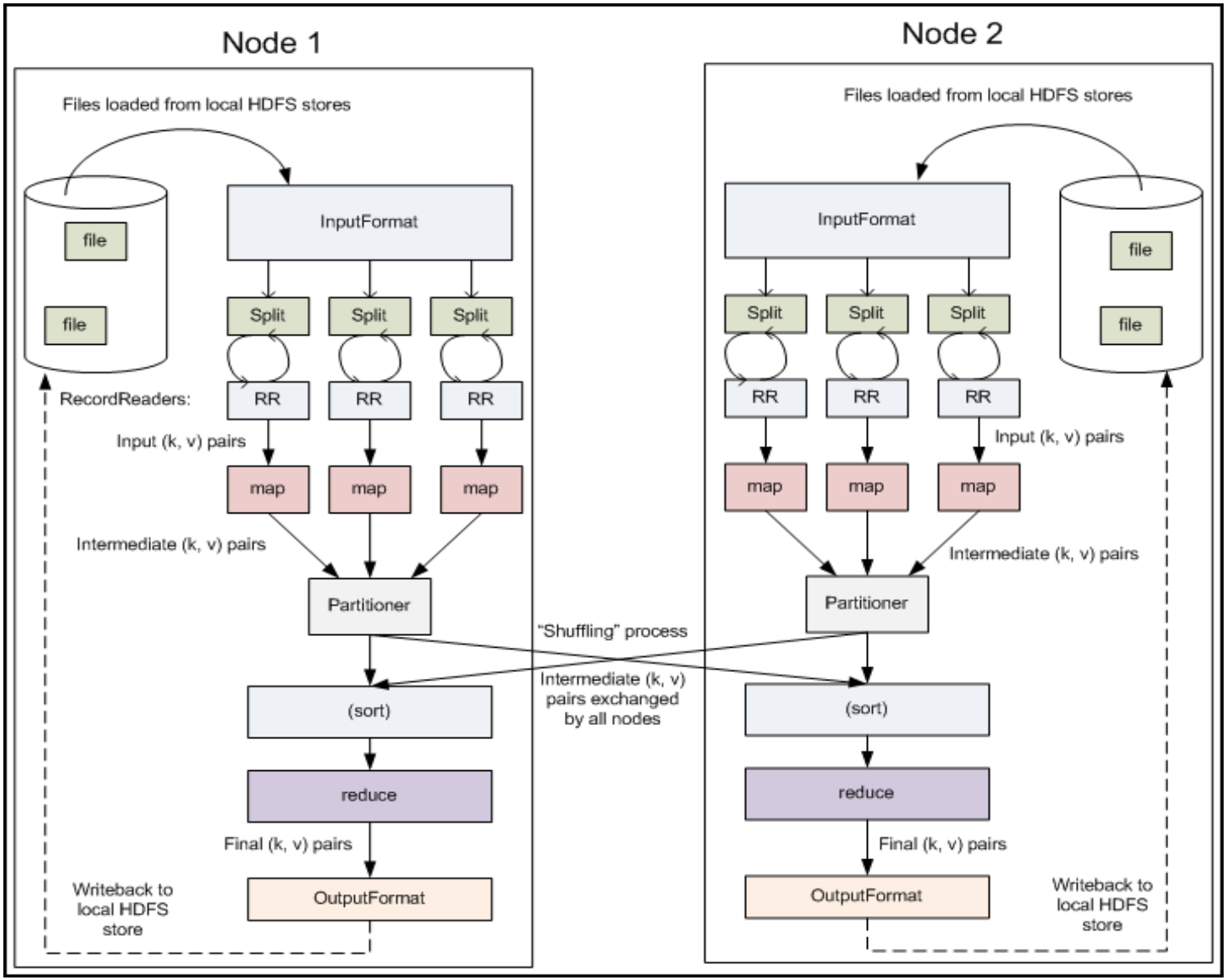
Driver

```
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();

        Job job = new Job(conf, "wordcount");

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

    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);

    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.waitForCompletion(true);
}
```

Map Reduce Data Flow



A closer look

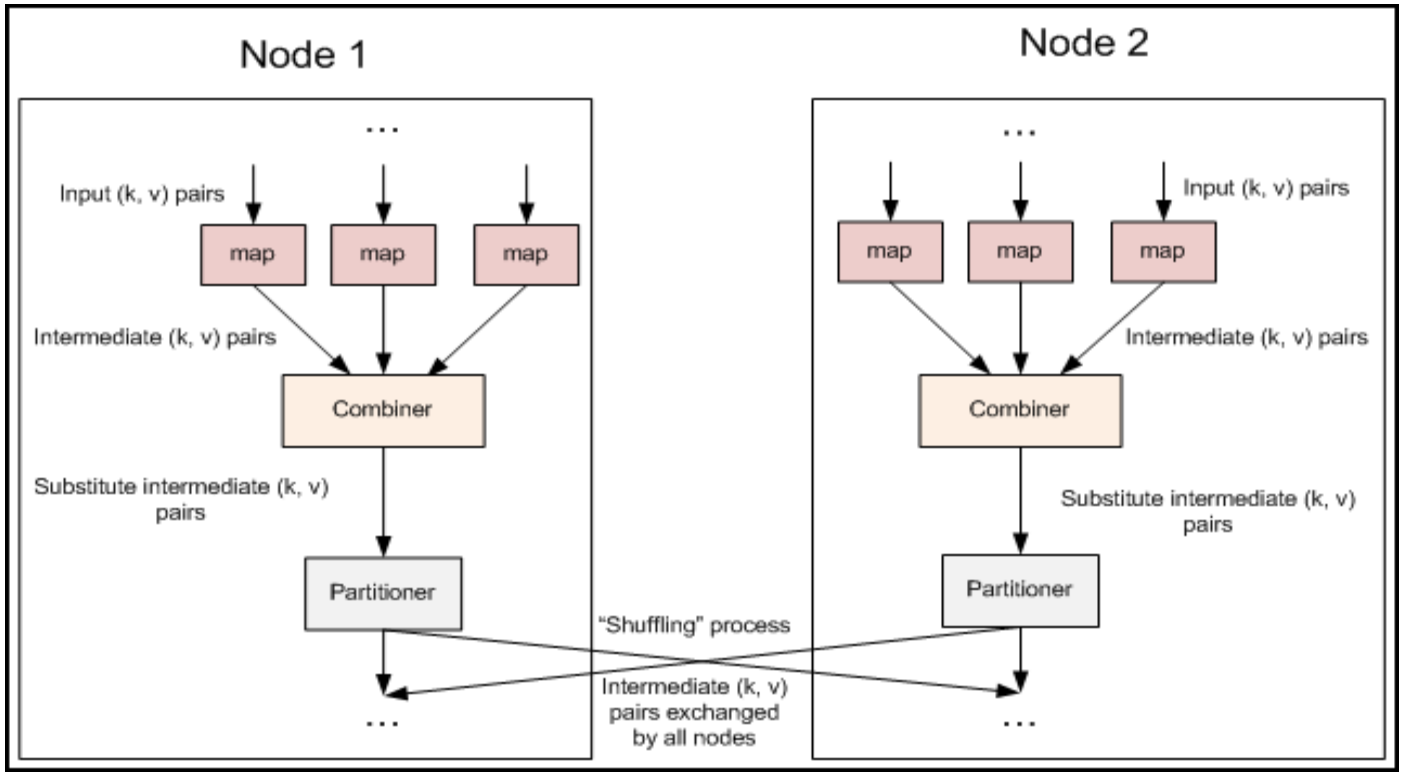


Input Format

InputFormat:	Description:	Key:	Value:
TextInputFormat	Default format; reads lines of text files	The byte offset of the line	The line contents
KeyValueInputFormat	Parses lines into key, val pairs	Everything up to the first tab character	The remainder of the line
SequenceFileInputFormat	A Hadoop-specific high-performance binary format	user-defined	user-defined

Table 4.1: InputFormats provided by MapReduce

Additional Map Reduce Functionality



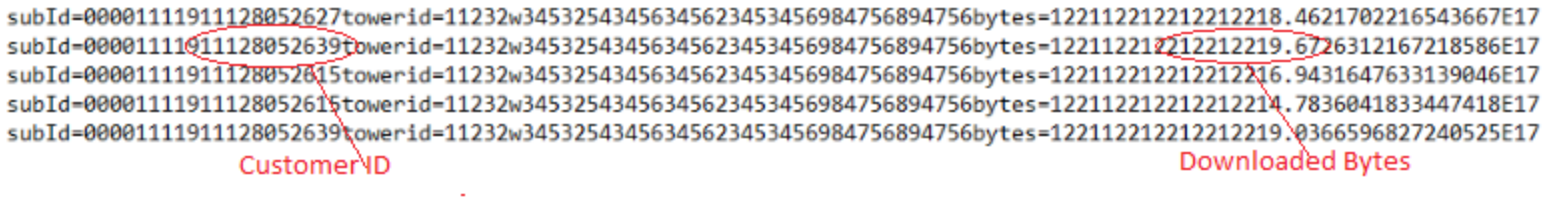
Writable, Comparable interfaces

- All keys are Comparable, Writable.
- All values are Writable.
- Most used:

  Text (which stores String data),
  IntWritable,
  LongWritable,
  FloatWritable,
  BooleanWritables

Case Study 1

- To find out subscribers and their corresponding downloaded bytes from sample logs of airmobile provided. Each line has information about subscriber (substring 15,26) the bytes downloaded (substring 87,97)



- Sample log files are present in above format. Data is present in line delimited format. From each line Customer ID and Downloaded Bytes have to be extracted for analysis.

Solution 1

- **Pseudo code**

```
Map<long, double> subscribertobytesMap;
for(each line in doc)
{
        parse line for subsriber id and bytes
        if(subscribertobytesMap.contains(subscriber id))
                subscribertobytesMap.get(subscriberid) + bytes
        else
        {
                subscribertobytesMap.put(subscriberid, bytes)
        }
}
```

When Documents Become Large

- When there are multiple documents
- We need to iterate through each document and calculate the map following solution 1

Issues

- Performance All the data is processed sequentially. There are millions of records which get processed and hence it takes hours and becomes pretty slow as the number of records increases
- Can run out of memory- Since we are maintaining a hashmap the heap memory can become insufficient

Solution 2

- Use multithreading to process the records in parallel, however this approach is also limited to number of CPUs / cores

Issues

- How to divide the data in equal parts
- Combining results will take further processing
- Still limited by processing power of 1 machine. Some data sets are beyond the memory of single machine

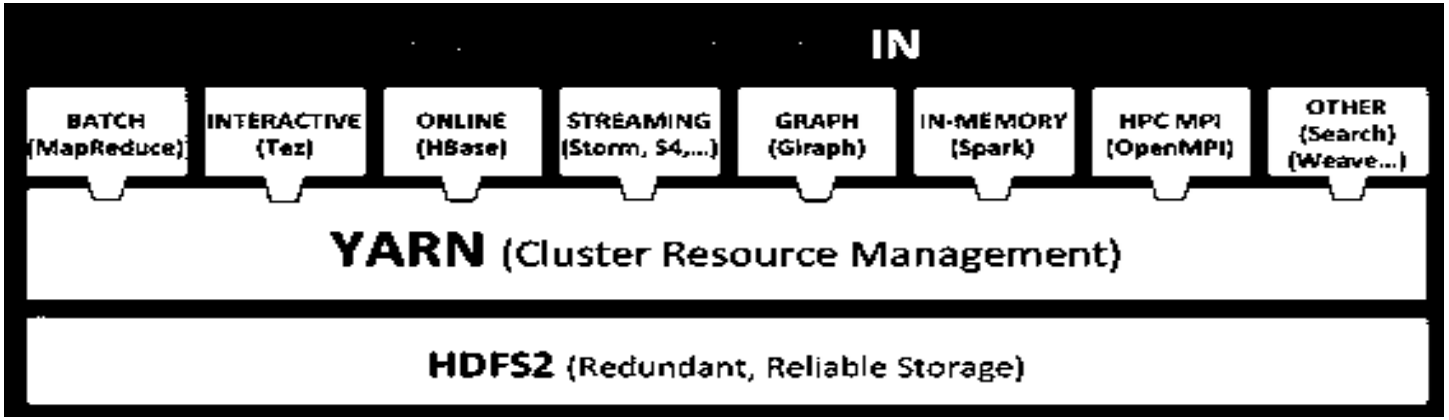
Solution 3:
Map Reduce v2 (Yarn)

What is Map Reduce v2?

- Split input files(e.g., by HDFS blocks)
- Operate on key / value pairs
- Mappers filter and transform input data
- Reducers operate on mapper output
- Finally, move code to data.
- The new map reduce i.e. YARN was born of a need to enable a broader array of interaction for data stored in HDFS beyond MapReduce. The YARN-based architecture of Hadoop 2.0 provides a more general processing platform that is not constrained to MapReduce.
- YARN :- **Yet Another Resource Negotiator** , allows other applications on top of it.

Why YARN ?

- Yarn provides programming beyond java application and makes it interactive to let other applicatons like GI-Raph, Hbase to work on it.



Source:-http://hortonworks.com/hadoop/yarn/

- The beauty of YARN's design is that different YARN applications can co-exist on the same cluster—so a MapReduce application can run at the same time as an MPI application, for example—which brings great benefits for managability and cluster utilization.

Components of Yarn

- **Client :-** For submitting MapReduce Jobs.
- **Resource manager :-** For coordinating the allocation of computer resources on the cluster.
- **Node managers :-** For launching and monitoring the computer containers on machines in the cluster.
- **The MapReduce application master** :- Checks tasks running the MapReduce job. The application master and the MapReduce tasks run in containers that are scheduled by the resource manager, and managed by the node managers.
- The distributed filesystem i.e HDFS which is used for sharing job files between the other entities.

Note:-- Previously installed hadoop had Jobtracker & Tasktracker which were responsible for handling resources and checking progress management. However, Hadoop 2.0 has Resource manager and Nodemanager to overcome the shortfall of Jobtracker & Tasktracker.

- JobTracker(JT) managed the task trackers & also kept track of resource availability and task life cycle management tracking its progress, fault tolerance whereas TaskTracker just does the job submitted by JT & gives updates to them about their progress.

	Input	Output
Map	<k1, v1>	list(<k2, v2>)
Reduce	<k2, list(v2)>	list(<k3, v3>)

Steps in Map-reduce Phase:

- Map takes a data in the form of <key , value> pairs and returns a list of <key , value> pairs. The keys will not be unique in this case
- Using the output of Map, sort and shuffle are applied by the hadoop architecture. This sort and shuffle acts on these list of <key , value> pairs and sends out unique keys and a list of values associated with this unique key <key , list(values)>
- Output of sort and shuffle will be sent to reducer phase. Reducer will perform a defined function on list of values for unique keys and Final output will <key , value> will be stored/displayed.

Sample Input File

- subId=000011119111128052639towerid=11232w3453254345634562345345698475689475 6bytes=12211221221221219.67263121672185 86E17
- subId=000011119111128052615towerid=11232w3453254345634562345345698475689475 6bytes=12211221221221216.94316476331390 46E17
- subId=000011119111128052615towerid=11232w3453254345634562345345698475689475 6bytes=12211221221221214.78360418334474 18E17
- subId=000011119111128052639towerid=11232w3453254345634562345345698475689475 6bytes=12211221221221219.03665968272405 25E17
- subId=000011119111128052619towerid=11232w3453254345634562345345698475689475 6bytes=12211221221221218.06862800145404 67E17
- subId=000011119111128052658towerid=11232w3453254345634562345345698475689475 6bytes=12211221221221216.98608904961789 44E17

k1, v1

- (0 , subId=000011119111280526 39towerid=11232w345325434563456234534569847568947 56bytes =1221122122122122 19.67263121672185 86E17)
- (121 , subId=000011119111280526 15towerid=11232w345325434563456234534569847568947 56bytes =1221122122122122 16.94316476331390 46E17
- (242 , subId=000011119111280526 15towerid=11232w345325434563456234534569847568947 56bytes =1221122122122122 14.78360418334474 18E17)

Data as input is sent through “TextInput” format (to be discussed later) . Mapper takes input in form of (k1,v1). Using this format, one particular line is read and sent as value with key as offset from start

First line has zero offset from start, so key will be 0

Second line has offset equal to length of first line, so offset 121. Similarly each line is sent in the form of <key , value> pairs.

list(K2, V2)

- (Output from mapper)
- (28052627, 8.4621702216543)
- (28052639, 9.672631216721858)
- (28052627, 8.64072609693471)

In our mapper function for each <k1,v1> we have extracted Subscriber's ID and Downloaded bytes from 'v1' using Substring() method. Then output is sent in the form of < ID, Downloaded bytes> as <k2,v2>.

(K2, list(V2))

- (“28052627”, (8.4621702216543, 8.64072609693471)
- (“28052639”, (9.672631216721858))

Sort and shuffle

Using the input from each mapper <k2,v2> , we collect all the values for each unique key k2. This output from shuffle phase in the form of <k2,list(v2)> is sent as input to reducer phase.

* Reducer phase cannot start unless all the mapper functions have been completed.

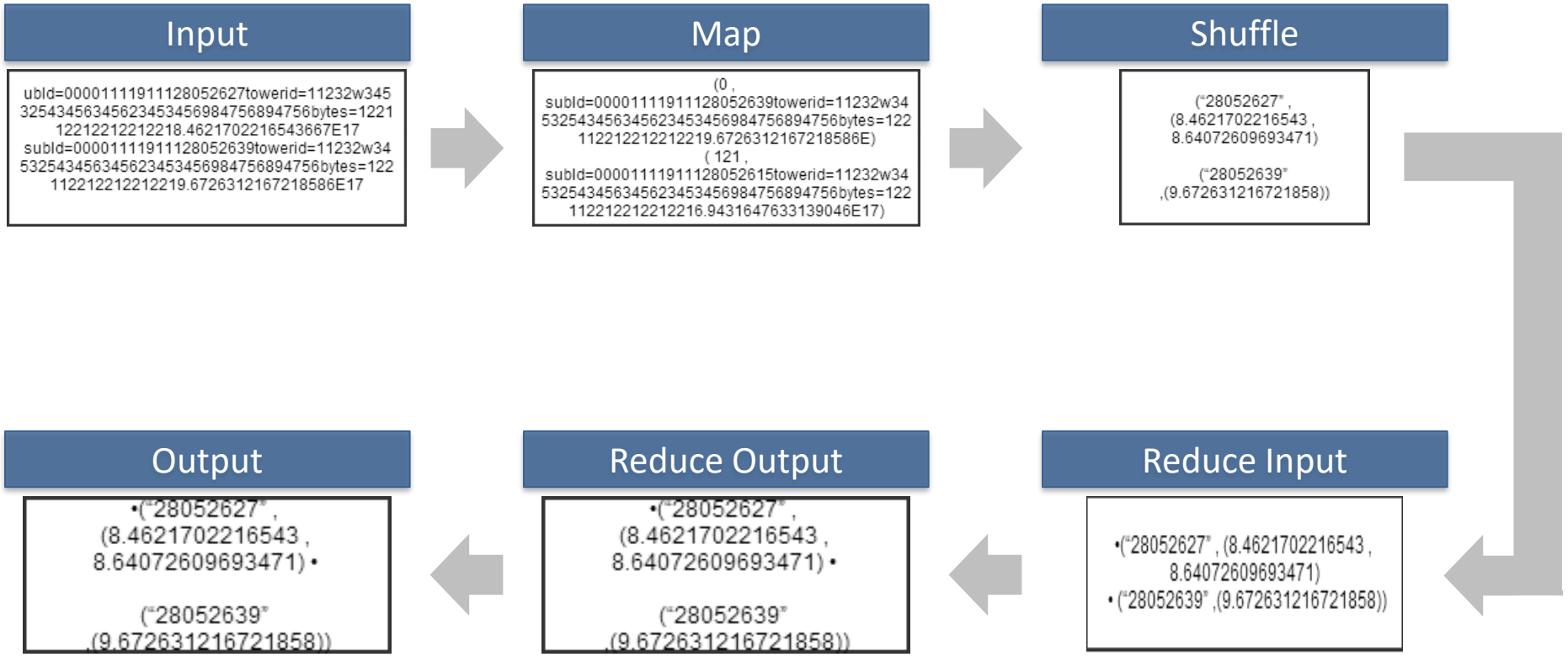
list(K3, V3)

- (Output of reducer)
- (“28052627”, (17.10289631858901))
- (“28052639”, (9.672631216721858))

In our reducer phase, Sum is applied on list(v2) to get total downloaded bytes.

<k3,v3> = <k2,Sum(list(v2))>

Flow chart of data flow through MAP-REDUCE phase

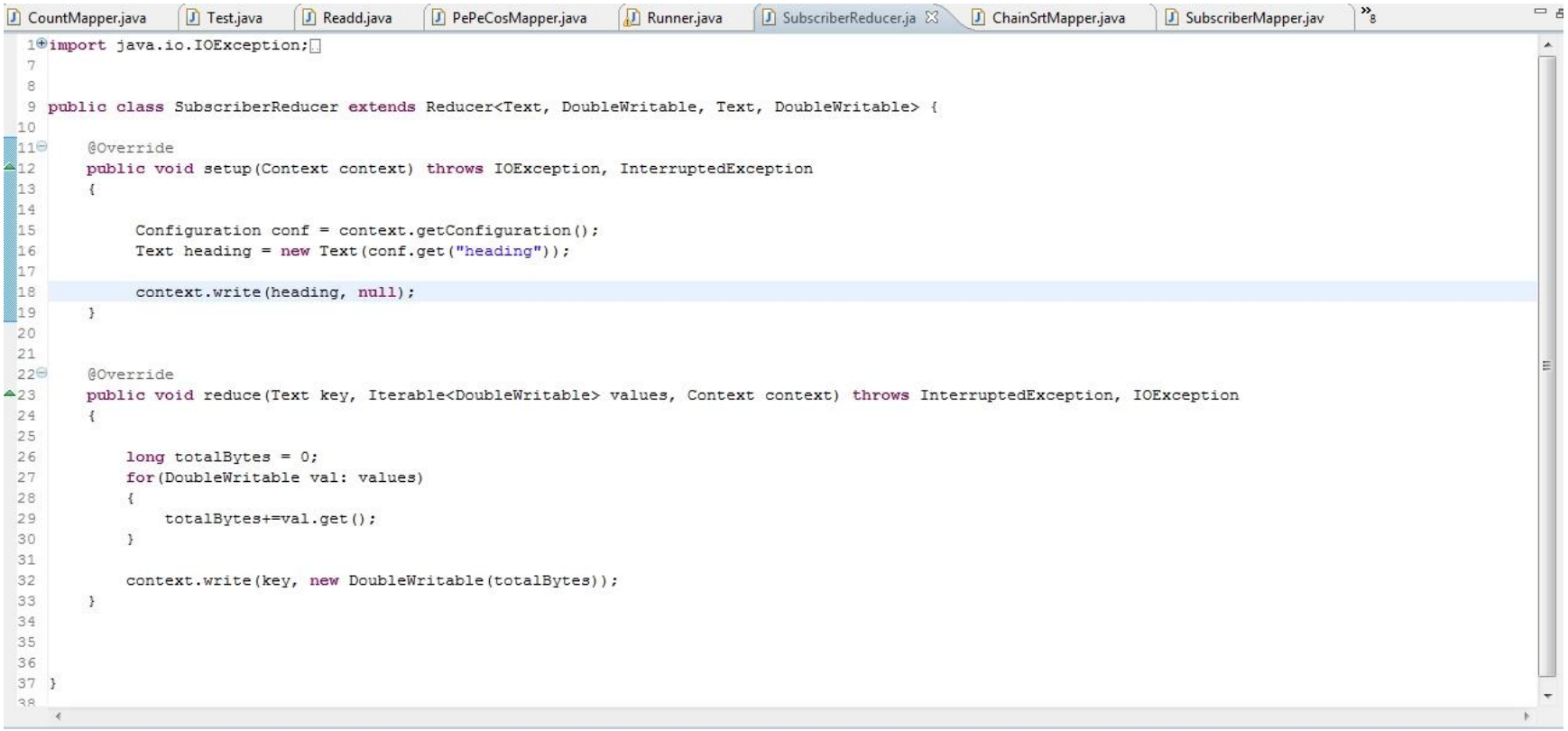


Mapper

CountMapper.java | SubscriberMapper.java | TestLine.java | Test.java | Readd.java

```
1 import java.io.IOException;
7
8
9 public class SubscriberMapper extends Mapper<LongWritable, Text, Text, DoubleWritable> {
10
11
12     private static final double MISSING = 0;
13
14     public void map(LongWritable key, Text value, Context context)throws IOException, InterruptedException
15     {
16         String line = value.toString();
17         String subId = line.substring(15,26);
18
19         Double bytes = Double.parseDouble(line.substring(87,97));
20
21         if(bytes==null)
22         {
23             bytes = MISSING;
24         }
25
26         context.write(new Text(subId), new DoubleWritable(bytes));
27
28
29
30     }
31
32 }
33
```

Reducer


CountMapper.java
Test.java
Readd.java
PePeCosMapper.java
Runner.java
SubscriberReducer.ja
ChainSrtMapper.java
SubscriberMapper.jav
import java.io.IOException;
public class SubscriberReducer extends Reducer<Text, DoubleWritable, Text, DoubleWritable> {
@Override
public void setup(Context context) throws IOException, InterruptedException
{
Configuration conf = context.getConfiguration();
Text heading = new Text(conf.get("heading"));
context.write(heading, null);
}
@Override
public void reduce(Text key, Iterable<DoubleWritable> values, Context context) throws InterruptedException, IOException
{
long totalBytes = 0;
for(DoubleWritable val: values)
{
totalBytes+=val.get();
}
context.write(key, new DoubleWritable(totalBytes));
}
}

Main Class

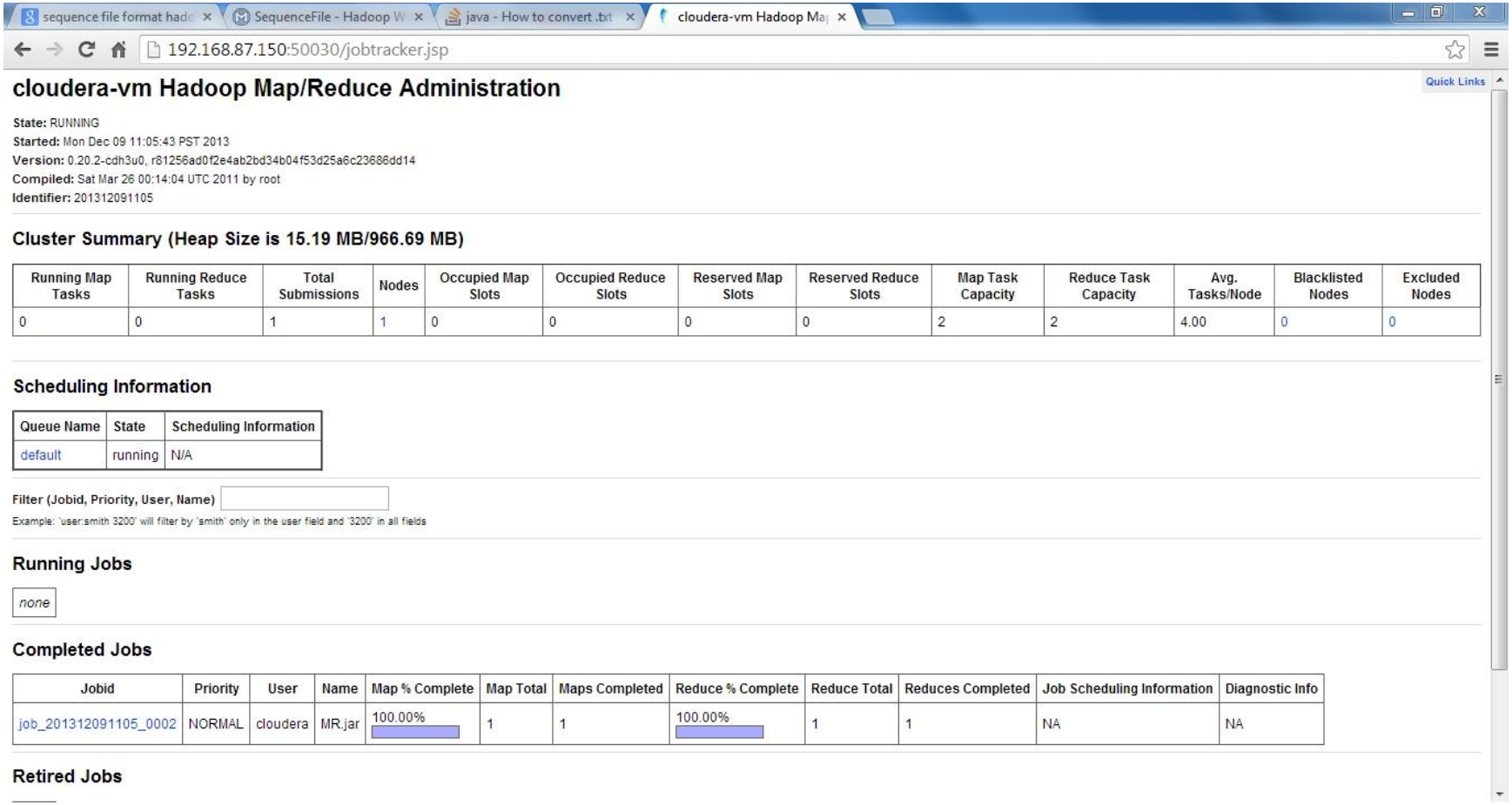
CountMapper.java | Test.java | Readd.java | PePeCosMapper.java | Runner.java | SubscriberReducer.ja | ChainSrtMapper.java | SubscriberMapper.jav | 8

```
13
14 public class Runner {
15
16     public static void main(String[] args) throws Exception {
17
18         Configuration conf = new Configuration();
19
20
21
22         conf.set("heading", "This is subscriberMR");
23
24         Job job = new Job(conf);
25
26         job.setJarByClass(Runner.class);
27
28         FileInputFormat.addInputPath(job, new Path("hdfs://cloudera-vm:8020//pristine"));
29         FileOutputFormat.setOutputPath(job, new Path("hdfs://cloudera-vm:8020//output"));
30
31         job.setMapperClass(SubscriberMapper.class);
32         job.setReducerClass(SubscriberReducer.class);
33         job.setCombinerClass(SubscriberReducer.class);
34         job.setInputFormatClass(TextInputFormat.class);
35         job.setOutputFormatClass(SequenceFileOutputFormat.class);
36
37
38         job.setOutputKeyClass(Text.class);
39         job.setOutputValueClass(DoubleWritable.class);
40
41         System.exit(job.waitForCompletion(true)?0:1);
42     }
43
44 }
45
```

Run Job Command (Hadoop jar “MR.jar” Runner)

```
cloudera@cloudera-vm:~$ hadoop jar my.jar Runner
13/12/09 11:10:34 WARN mapred.JobClient: Use GenericOptionsParser for parsing the arguments. Applications should implement Tool for the same.
13/12/09 11:10:35 INFO input.FileInputFormat: Total input paths to process : 1
13/12/09 11:10:35 INFO mapred.JobClient: Running job: job_201312091105_0002
13/12/09 11:10:36 INFO mapred.JobClient:  map 0% reduce 0%
13/12/09 11:11:28 INFO mapred.JobClient:  map 100% reduce 0%
13/12/09 11:11:36 INFO mapred.JobClient:  map 100% reduce 100%
13/12/09 11:11:38 INFO mapred.JobClient: Job complete: job_201312091105_0002
13/12/09 11:11:38 INFO mapred.JobClient: Counters: 22
13/12/09 11:11:38 INFO mapred.JobClient:   Job Counters
13/12/09 11:11:38 INFO mapred.JobClient:     Launched reduce tasks=1
13/12/09 11:11:38 INFO mapred.JobClient:     SLOTS_MILLIS_MAPS=51114
13/12/09 11:11:38 INFO mapred.JobClient:     Total time spent by all reduces waiting after reserving slots (ms)=0
13/12/09 11:11:38 INFO mapred.JobClient:     Total time spent by all maps waiting after reserving slots (ms)=0
13/12/09 11:11:38 INFO mapred.JobClient:     Launched map tasks=1
13/12/09 11:11:38 INFO mapred.JobClient:     Data-local map tasks=1
13/12/09 11:11:38 INFO mapred.JobClient:     SLOTS_MILLIS_REDUCES=8054
13/12/09 11:11:38 INFO mapred.JobClient:   FileSystemCounters
13/12/09 11:11:38 INFO mapred.JobClient:     FILE_BYTES_READ=219984
13/12/09 11:11:38 INFO mapred.JobClient:     HDFS_BYTES_READ=1198678
13/12/09 11:11:38 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=545242
13/12/09 11:11:38 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=1466
13/12/09 11:11:38 INFO mapred.JobClient:   Map-Reduce Framework
13/12/09 11:11:38 INFO mapred.JobClient:     Reduce input groups=51
13/12/09 11:11:38 INFO mapred.JobClient:     Combine output records=0
13/12/09 11:11:38 INFO mapred.JobClient:     Map input records=9999
13/12/09 11:11:38 INFO mapred.JobClient:     Reduce shuffle bytes=219984
13/12/09 11:11:38 INFO mapred.JobClient:     Reduce output records=51
13/12/09 11:11:38 INFO mapred.JobClient:     Spilled Records=19998
13/12/09 11:11:38 INFO mapred.JobClient:     Map output bytes=199980
13/12/09 11:11:38 INFO mapred.JobClient:     Combine input records=0
13/12/09 11:11:38 INFO mapred.JobClient:     Map output records=9999
13/12/09 11:11:38 INFO mapred.JobClient:     SPLIT_RAW_BYTES=98
13/12/09 11:11:38 INFO mapred.JobClient:     Reduce input records=9999
cloudera@cloudera-vm:~$
```

Status can be Checked with WEB GUI IP:50030(URL)



Code Walk Through Subscribermapper

- This class extends org.apache.hadoop.mapreduce.Mapper class
- The Mapper class has methods cleanup(),map(),setup() and run() that can be overridden by the child class
- Framework first calls startup , map and then cleanup
- In SubscriberMapper class the map() method is overridden.
- Typically with TextInputFormat the input key for Mapper function is offset bytes for each line. This is not needed for further processing so we can ignore this field
- Input value for Mapper function is Text which is string representation of each line.
- We can extract subscriber id and their corresponding bytes using substring.
- The output key value pair is subscriber id <LongWritable> and corresponding bytes <DoubleWritable>
- The output is written in Mapper context which can be retrieved by Reducer

Subscriber Reducer

- This class extends org.apache.hadoop.mapreduce.Reducer
- The Reducer class also has methods setup(), cleanup() and reduce methods which can be overridden
- Framework first calls startup(), reduce() and then cleanup()
- In SubscriberReducer reduce() method is overridden
- The input key value pair of reducer are output of SubscriberMapper. Hence, <subscriberid, bytes> LongWritable and Iterable of DoubleWritable are input for reducer
- The output key is subscriber Id
- The output value i.e., total bytes for each subscriber is calculated by applying aggregate function on the bytes

Runner Class

- org.apache.hadoop.conf.Configuration -> The hadoop configuration parameters can be set by making object of Configuration
- The Mapper, Reducer, Combiner, InputFormat, OutputFormat need to be set in the main class

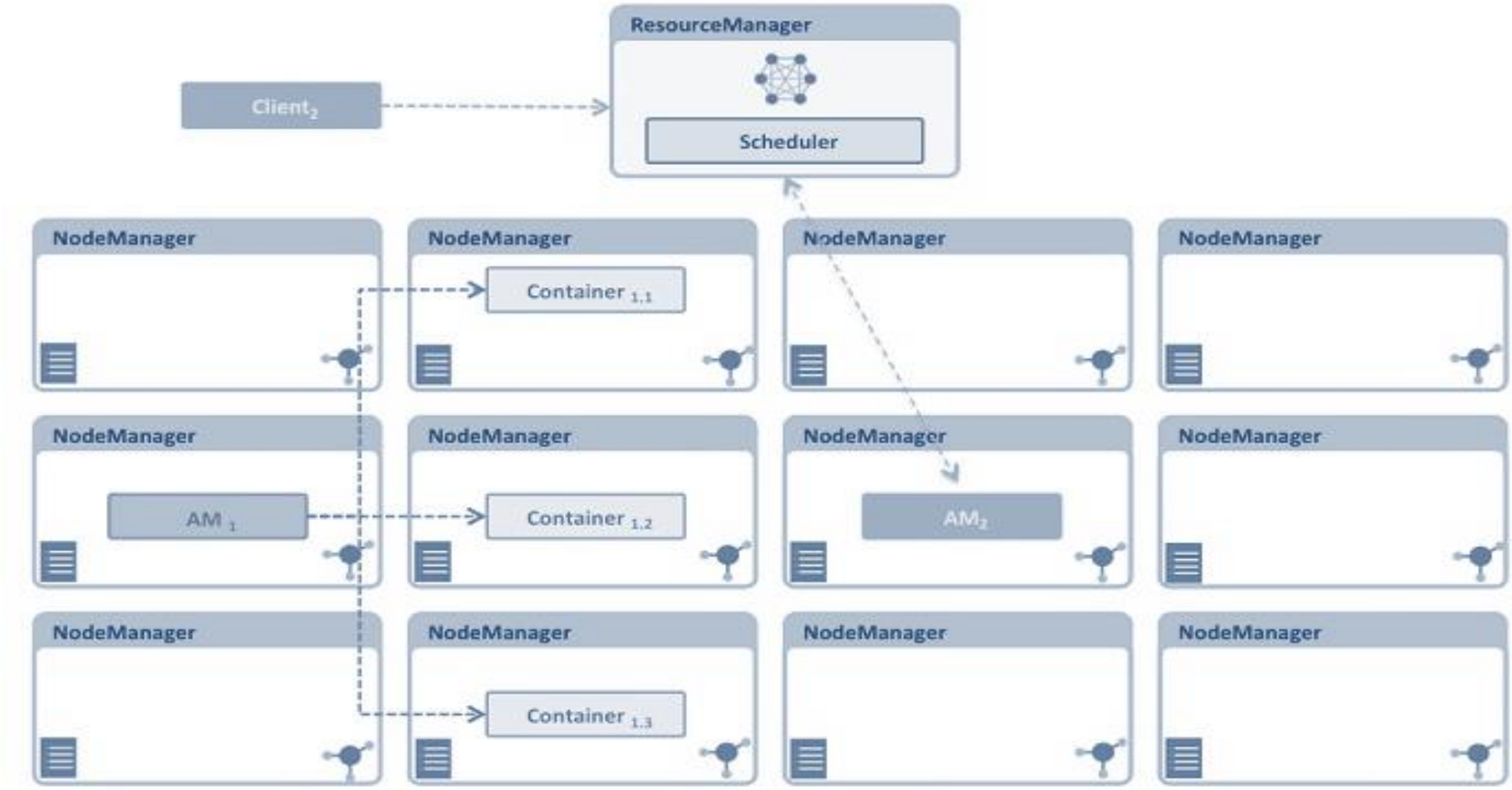
Architecture

- Split input data into independent chunks
- MAP Phase (input/output key value pair)
- Shuffling and sorting
- Reduce Phase(input/output key value pair)
- Compute and storage nodes are same. i.e., MR and HDFS run on same nodes.
- Schedule tasks on nodes where data is already present

Title to come

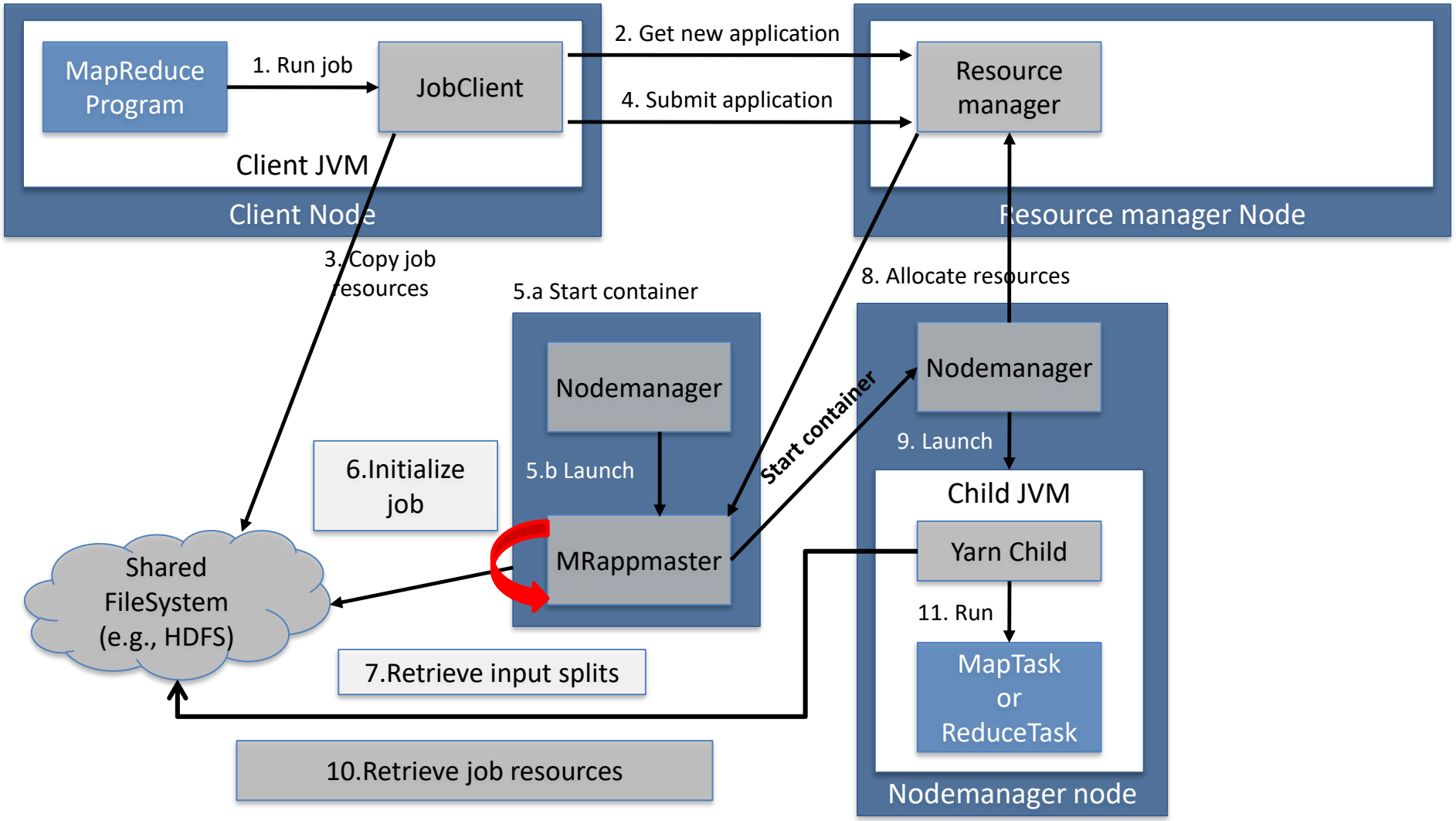
- Resource Manager-> YARN combines a central resource manager that reconciles the way applications use Hadoop system resources with node manager agents that monitor the processing operations of individual cluster nodes.
- Application master – 1 application master per application(job)-> executes tasks as directed by master.

Architecture – Resource Manager

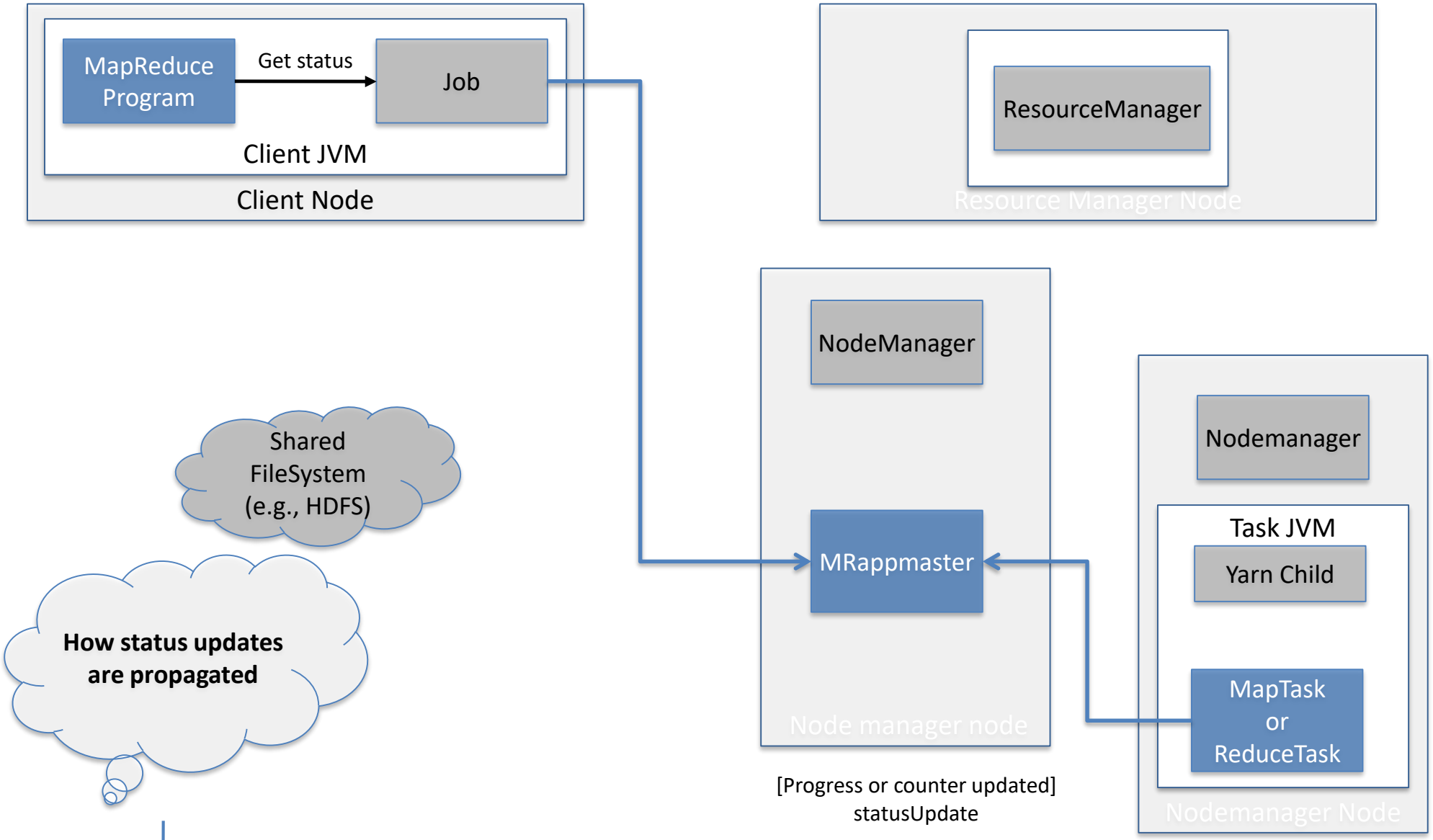


Source: Yarn webinar by Hortonworks

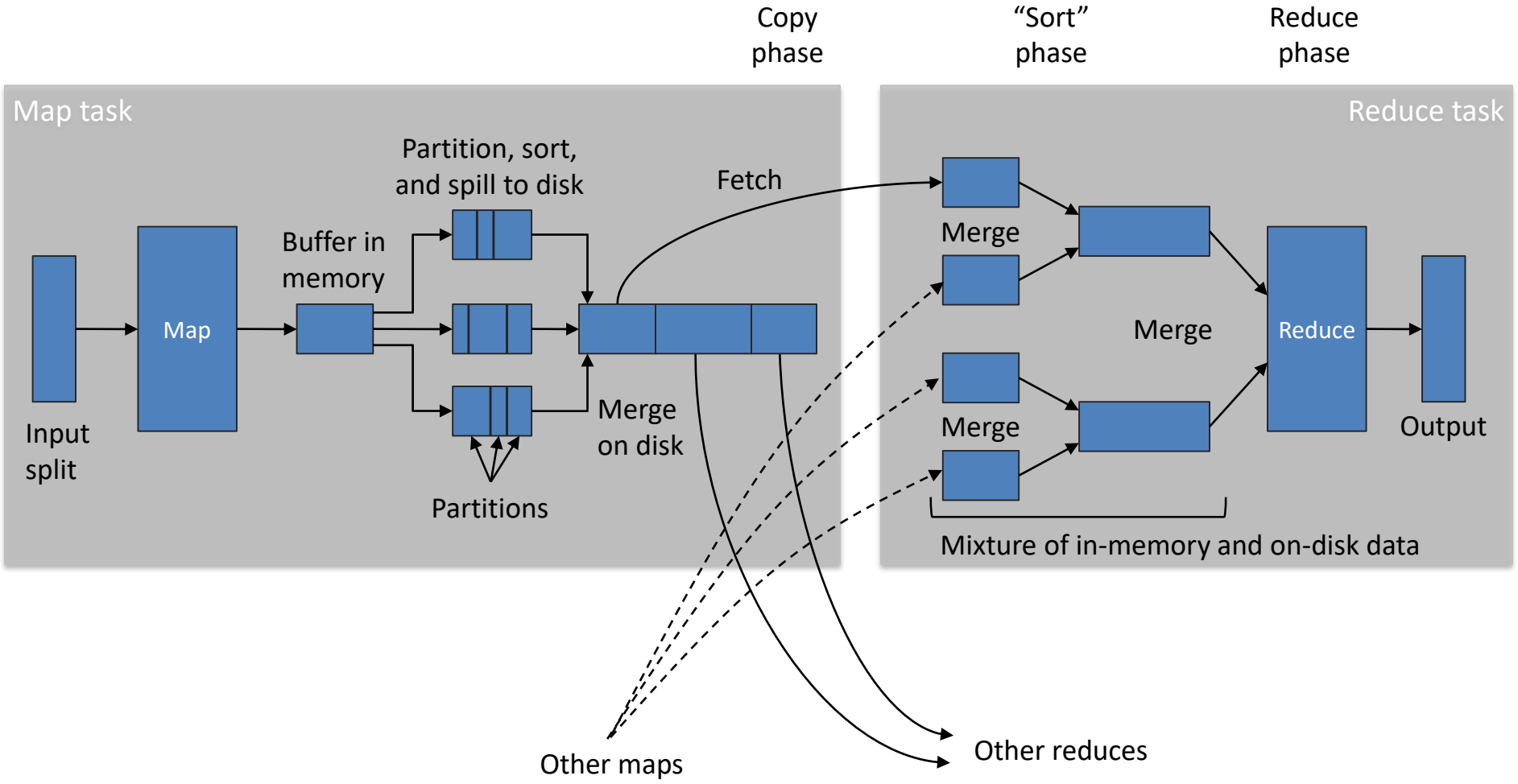
Architecture - MapReduce



Status Updation



Shuffle and Sort in Map Reduce



Features

- Process data in-parallel on a cluster of nodes
- Fault tolerant
- Commodity hardware

Implementation

- Applications specify input/output location, MR functions, job parameters (job configuration)
- Hadoop job client submits the job(jar/executable).
- Resource manager now takes the responsibility of distributing software, monitoring and scheduling tasks.

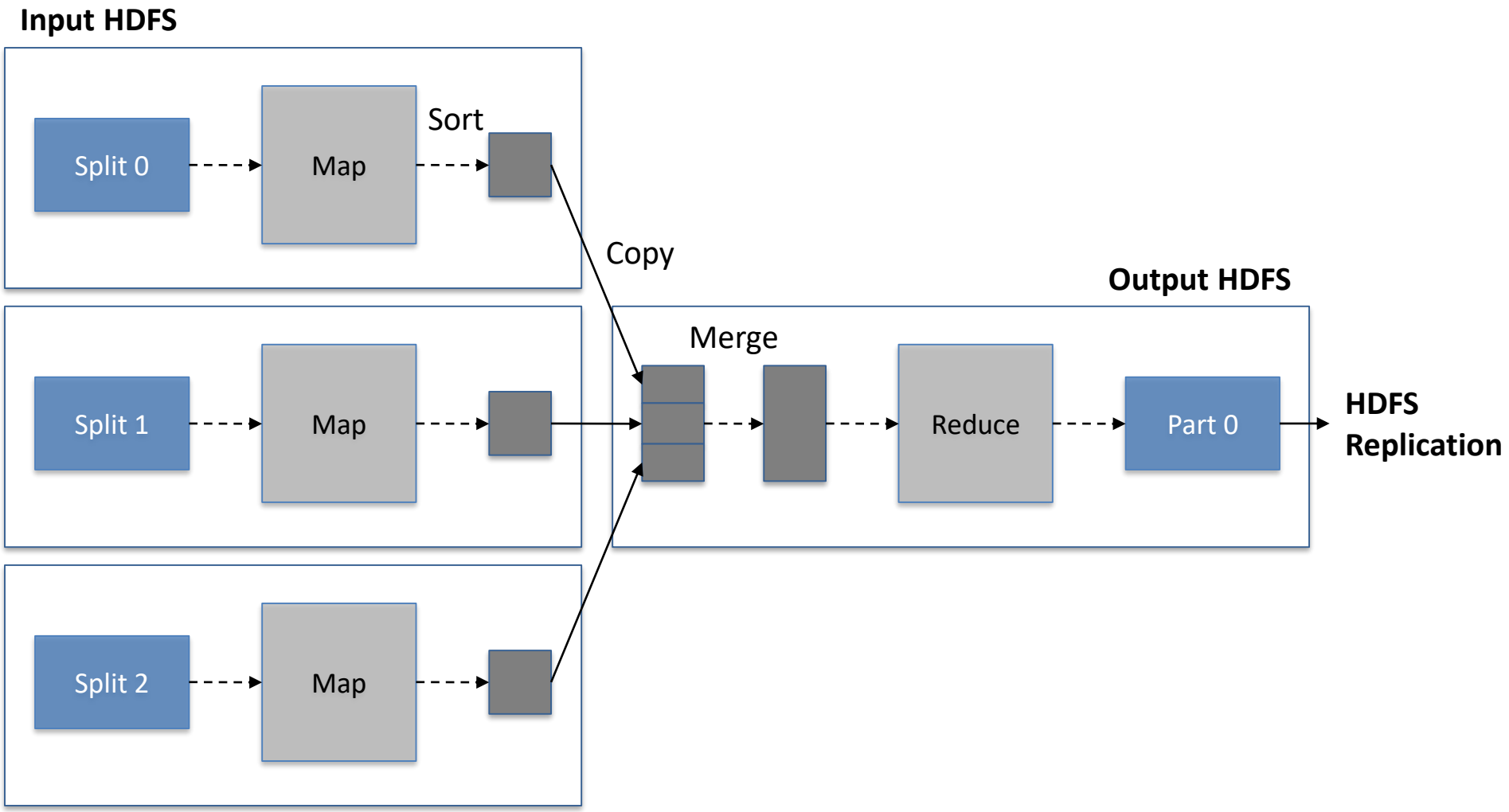
How Many Maps?

- The total number of maps required to run a job depends on the size of data to be processed. i.e., total number of blocks of input files
- If the input file is of the size 1000Mbytes and typically block size is 128 Mbytes, the number of mappers will be 8

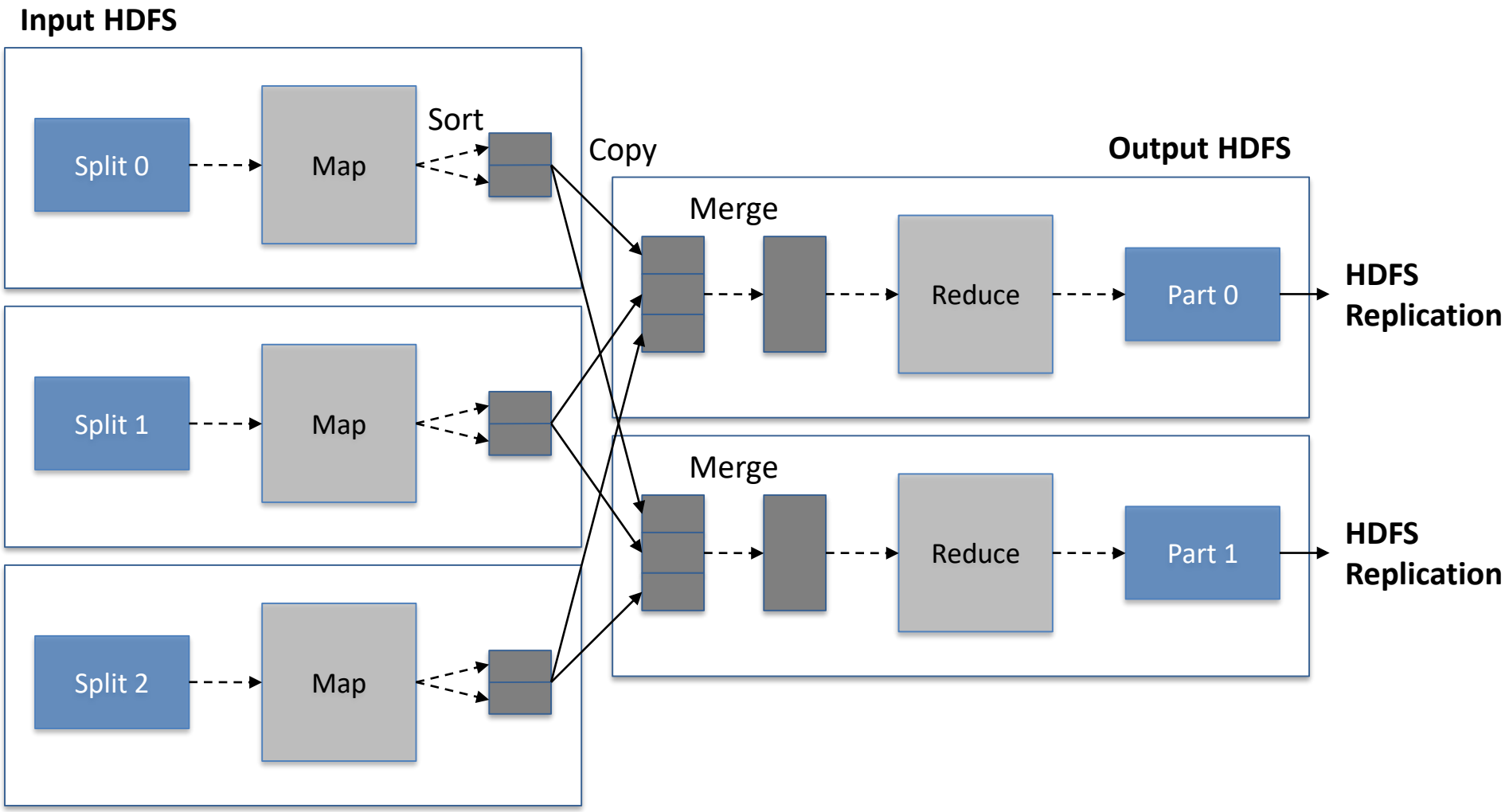
How Many Reduces?

- The number of reduces is approximately 0.95 or 1.75*(<numofnodes>*mapred.tasktracker.reduce.tasks.maximum)
- Increasing the number of reduces increases the framework overhead, but the load balancing improves
- The scaling factor is not kept a whole number to reserve slots for failed tasks

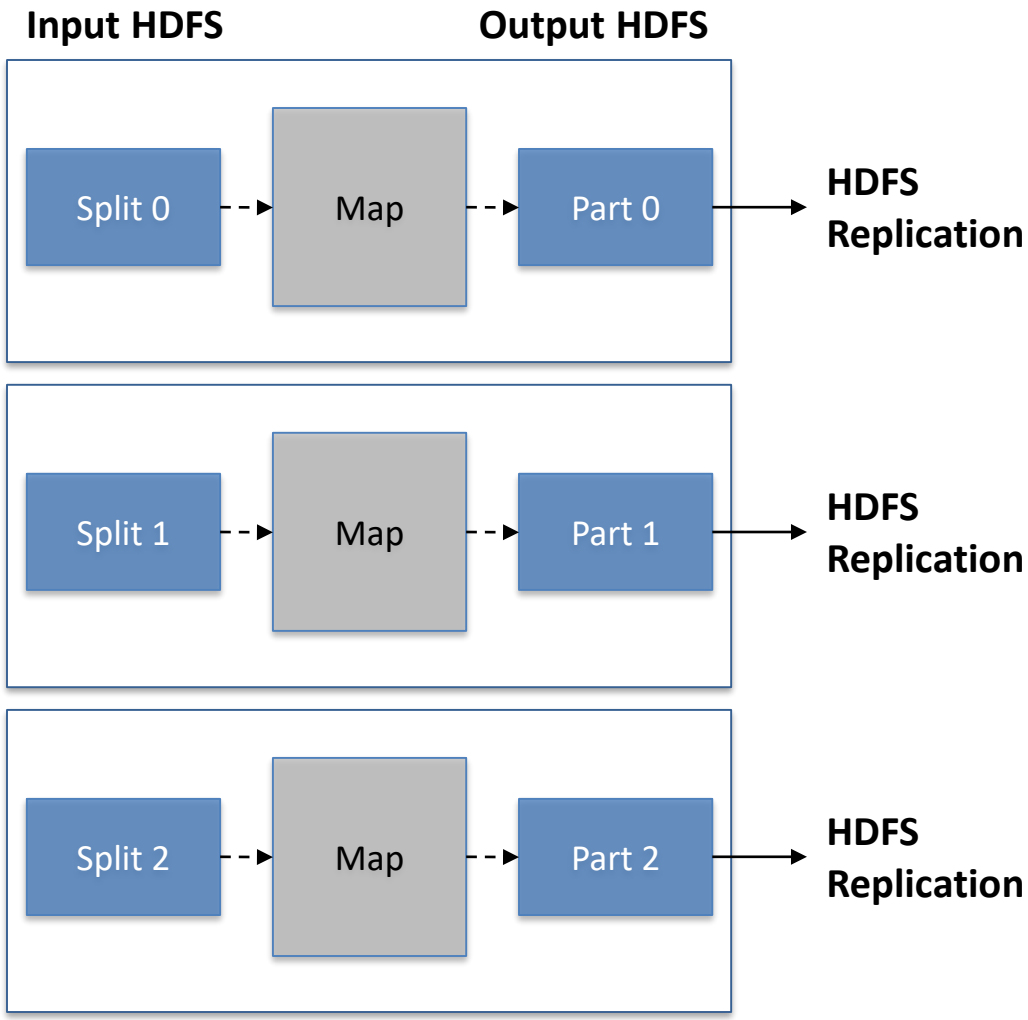
Data Flow with Single Reducer



Data Flow with Multiple Reducers



Data Flow with No Reducer



No Reducer

- It is possible to set no reducer, if reduce phase is not required
- In this case the outputs of map tasks go directly to filesystem, in the path specified by setOutputPath(Path)

Partitioner

- Controls the partioning of the keys of intermediate map-outputs. The key is used to derive the partition, typically by hashfunction
- Total number of partitions is same as number of reduce tasks for the job
- Controls which intermediate key and hence record is sent to which reduce task for reduction

Serialization / Deserialization

Serialization / Deserialization

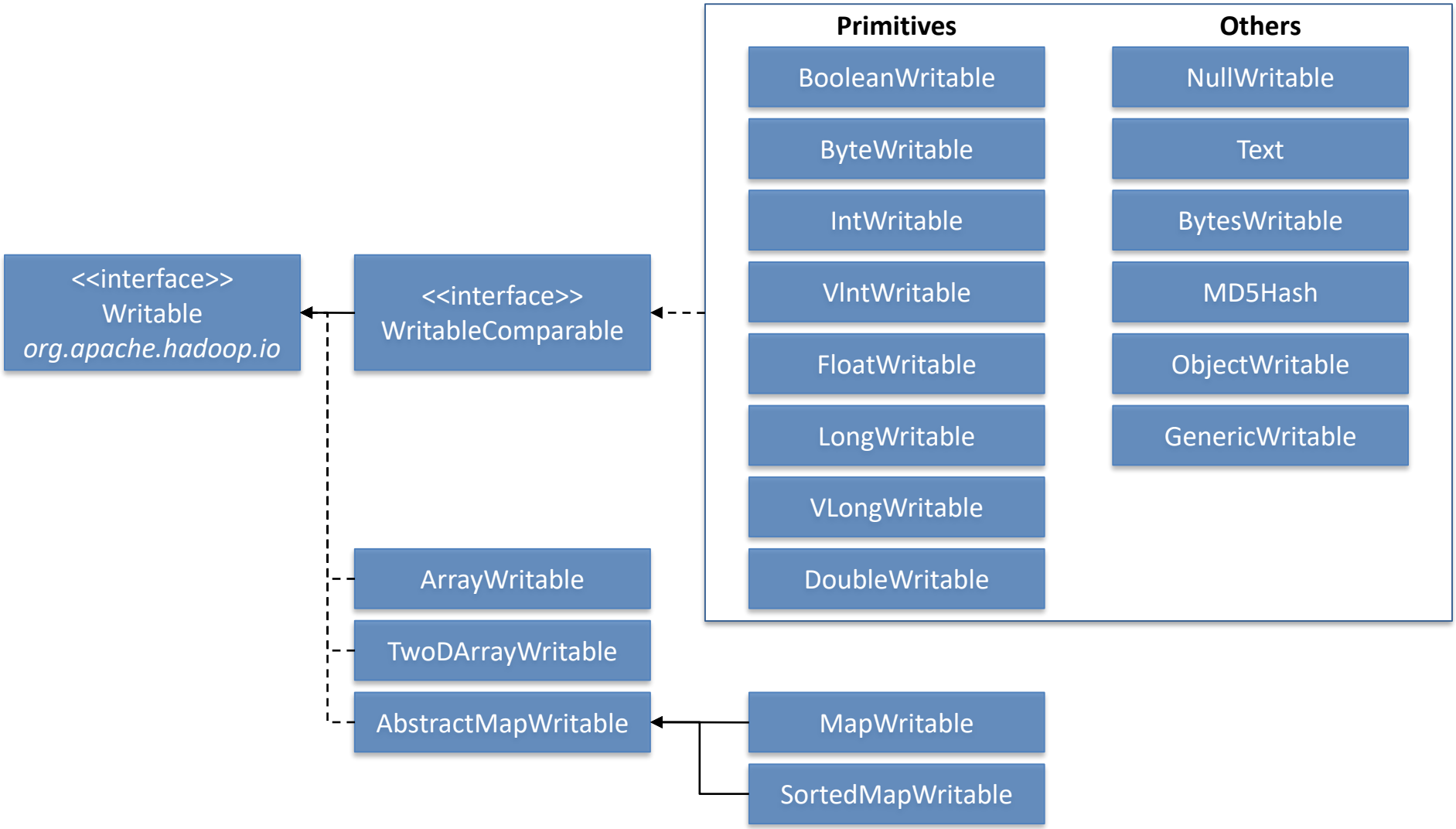
- **Serialization:** Process of turning objects into stream of bytes for transmission across network or for writing into some storage(e.g., disk). Hadoop uses it's own format Writables which are compact and fast

- **Deserialzaton:** Reverse of serialization turning byte stream back into objects

Writable

- Defines methods for writing its state to DataOutput binary stream, and for reading its state from DataInput binary stream

Title to come



Properties of Writable Wrappers for Primitives

- All have a get() and a set() method for retrieving and sorting the wrapped value

org.apache.hadoop.mapreduce.Job

- The job submitter's view of the Job
- It allows user to configure job, submit it and query the state
- Parameters like job name, input/output path, Mapper, Reducer, priority etc can be set

Task Execution

- Tasktracker executes MR task as separate child process in a separate JVM

Job Submission and Monitoring

- Job Client is the interface by which user interacts with Job tracker
- Job submission involves
- Checking input/output specifications of job
- Computing input split values for the job
- Setting up accounting information for DistributedCache of the job
- Copying the job's jar and configuration to MR system directory on FileSystem
- Submitting job to jobtracker and optionally monitoring it's status

Job Control

- To chain MR jobs so that output of one MR job becomes input to another DistributedCache can be used
- The job can be submitted by job.submit

Job Input

- InputFormat describes input specification for MR job
- It validates input specification of the job
- Split up the input file into logical InputSplit instances, each of which can be assigned to an individual Mapper
- A lower bound on split size can be set mapred.min.split.size
- Default InputFormat is TextInputFormat

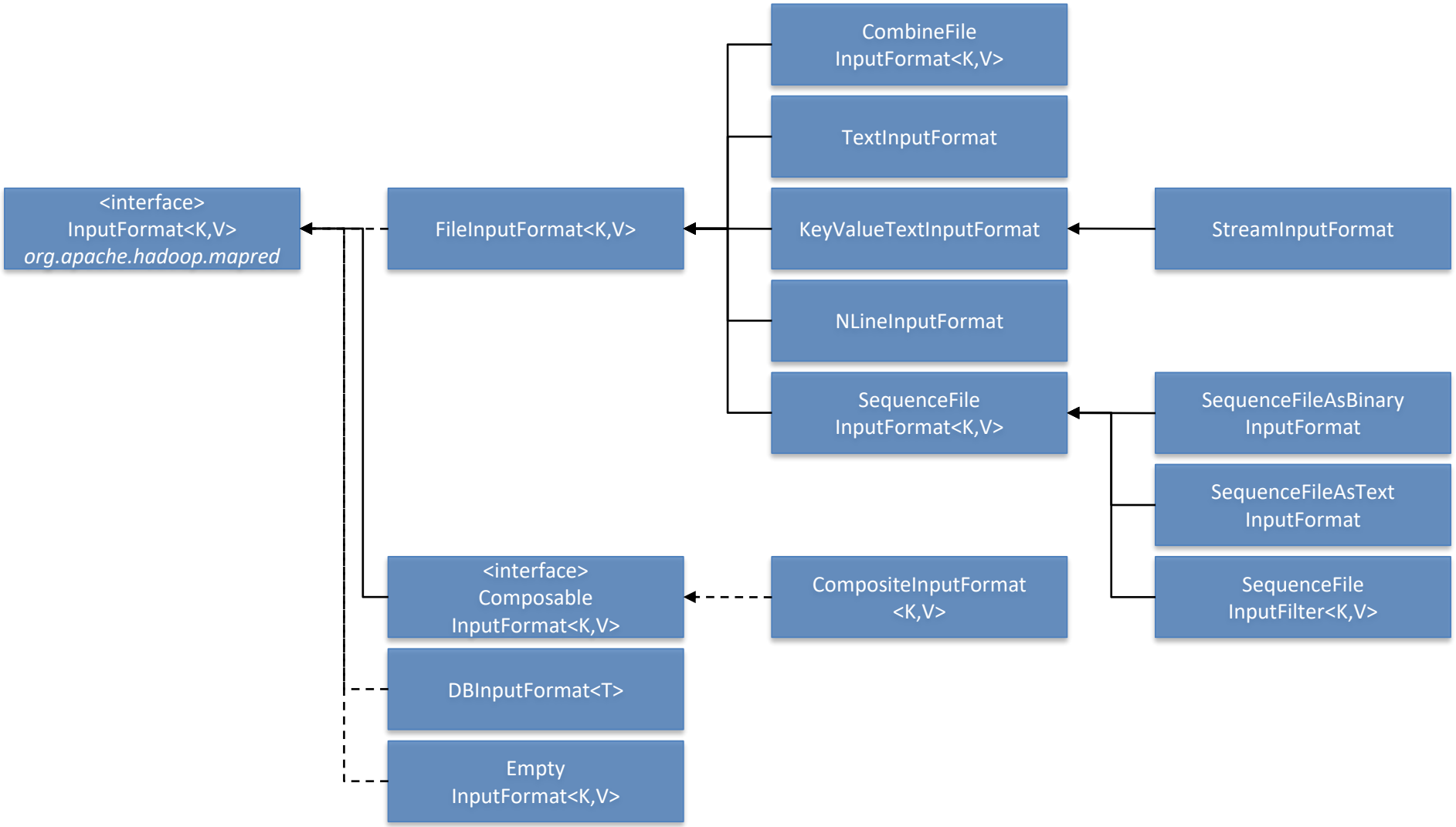
FileInputFormat

- Base class that uses file as it's data source
- Defines which files are included as input to a job
- Implementation for generating splits for the input files

Methods for Setting Jobconf's Input Paths

- Public static void addInputPath(JobConf conf, Path path)
- Public static void addInputPaths(JobConf conf, String commaSeparatedPaths)
- Public static void setInputPaths(JobConf conf, Path... inputPaths)
- Public static void setInputPaths(JobConf conf, String commaSeparatedPaths)
- **Note:** A path represents file/directory/collectionof files/collection of directories

InputFormat Class Hierarchy



How Does FileInputFormat Split?

- Splits only files larger than HDFS block size
- Applications can set a minimum split size also: by setting mapred.min.split.size this value larger than block size

CombineFileInputFormat

- MR framework optimizes mappers and reducers when files are large. However for cases where there are large number of small files the performance degrades
- CombineInputFormat alleviates the above problem by packing many files into each split, so that each mapper has more to process
- The best solution is always to keep small number of large files. This can be done my merging files using SequenceFile: keys file names and values file contents

Preventing Splitting

- Can be needed when a single mapper is needed to process single file entirely
- E.g., to check if records in a file are sorted
- Can be done by setting minimum split size as the largest Long value, or by overriding isSplittable in FileInputFormat as false

TextInputFormat

- Default input format
- Key LongWritable, byte offset within file of the beginning of line
- Value is content of line, packaged as Text Object

KeyValueTextInputFormat

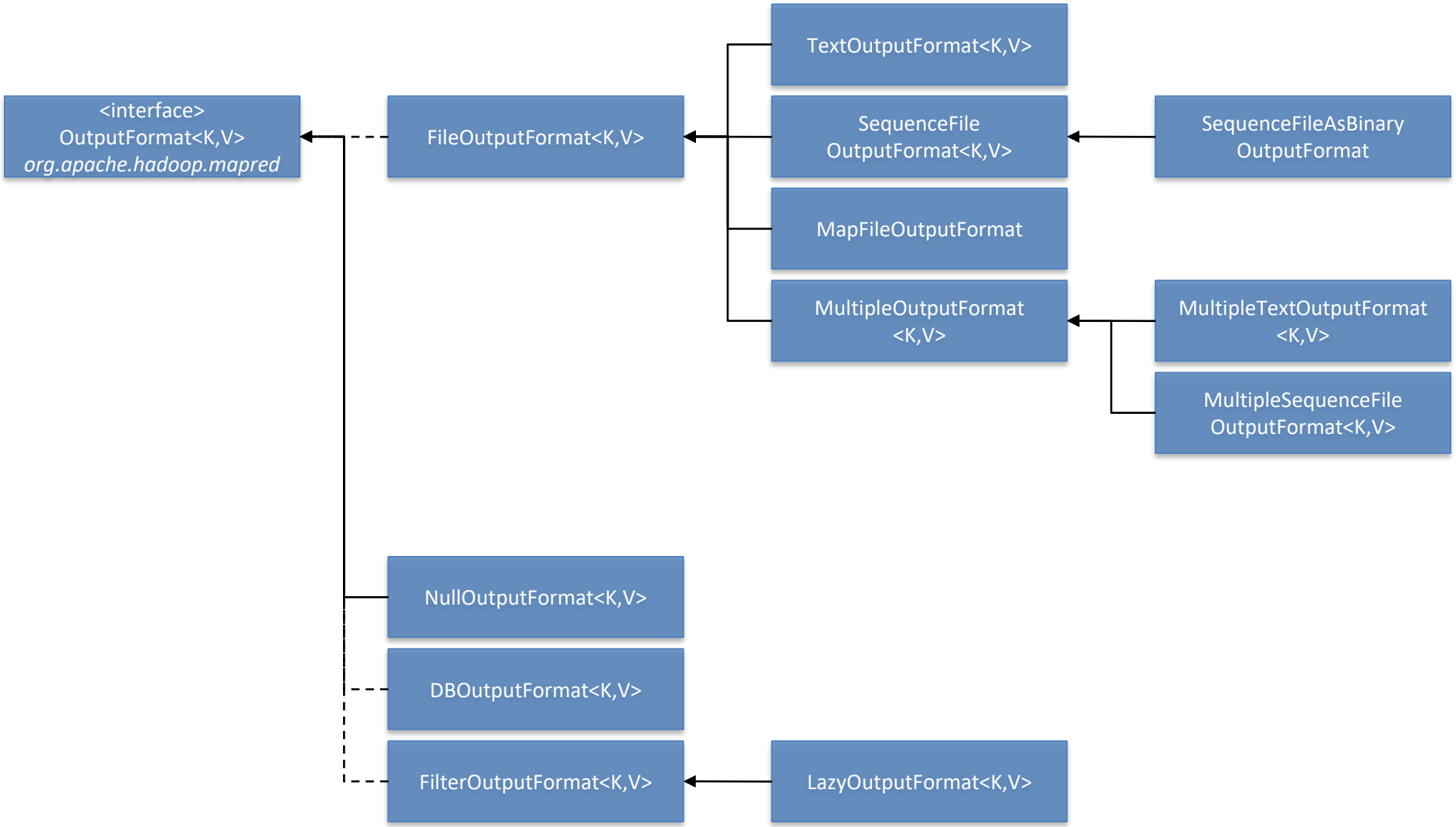
- To interpret files where each line in a file is in key value format

SequenceFileInputFormat

- Stores sequence of binary key value pairs
- If sequence file has IntWritable Keys and Text values, then Map signature will be <IntWritable, Text, K,V> where K and V are types of the map's output keys and values

Output Format

OutputFormat Class Hierarchy



Text Output

- Default output format
- Writes records as lines of text

SequenceFileOutputFormat

- Writes sequence files for its output
- Can be used if the output is further input for chained job

Exercise

- Use the output of case study 1 and sort the subscribers based on their data usage
- What would be the key value pair if we want to calculate the occurrence of the words in the file (e.g., if the file has content “The brown fox jumped over the brown tree”) “The” and “brown” appears twice, others have 1 occurrence

Solution 1

PePeCosReducer.java | Runner.java | SubscriberReducer.ja | ChainSrtMapper.java | SubscriberMapper.jav | SortRunner.java | SrtInputFormat.java | RgeSequenceFileInput

```
1  import java.io.IOException;
2
3  import org.apache.hadoop.io.DoubleWritable;
4
5  import org.apache.hadoop.io.Text;
6  import org.apache.hadoop.mapreduce.Mapper;
7
8
9
10 public class ChainSrtMapper extends Mapper<Text, DoubleWritable, DoubleWritable, Text>  {
11
12     public void map(Text key, DoubleWritable value, Context context)throws IOException, InterruptedException
13     {
14         context.write(value, key);
15
16     }
17
18 }
19
```

Solution 1

PePeCosReducer.java | Runner.java | SubscriberReducer.ja | ChainSrtMapper.java | SubscriberMapper.jav | SortRunner.java | SrtInputFormat.java | RgeSequenceFileInput

```
12
13
14
15
16 public class SortRunner {
17
18     public static void main(String[] args) throws Exception {
19
20         Configuration conf = new Configuration();
21
22         conf.set("mapred.max.split.size", "4294967296");
23
24         Job job = new Job(conf);
25
26         job.setJarByClass(SortRunner.class);
27
28         FileInputFormat.addInputPath(job, new Path("hdfs://cloudera-vm:8020//output"));
29         FileOutputFormat.setOutputPath(job, new Path("hdfs://cloudera-vm:8020//sort"));
30
31         job.setMapperClass(ChainSrtMapper.class);
32         job.setReducerClass(Reducer.class);
33
34         job.setInputFormatClass(SequenceFileInputFormat.class);
35
36
37         job.setMapOutputKeyClass(DoubleWritable.class);
38         job.setMapOutputValueClass(Text.class);
39
40         System.exit(job.waitForCompletion(true)?0:1);
41     }
42
43 }
44
```

Thank you!

Contact:

EduPristine

702, Raaj Chambers, Old Nagardas Road, Andheri (E), Mumbai-400 069. INDIA

www.edupristine.com

Ph. +91 22 3215 6191