CMR
INSTITUTE OF
TECHNOLOGY

USN | 1 | C | | | | | | | | |

Internal Assessment Test 3 – March 2023

Sub:	Big Data Analytics							Code:	20MCA352
Date:	15.03.2023	Duration:	90 mins	Max Marks:	50	Sem:	III A & B	Branch:	MCA

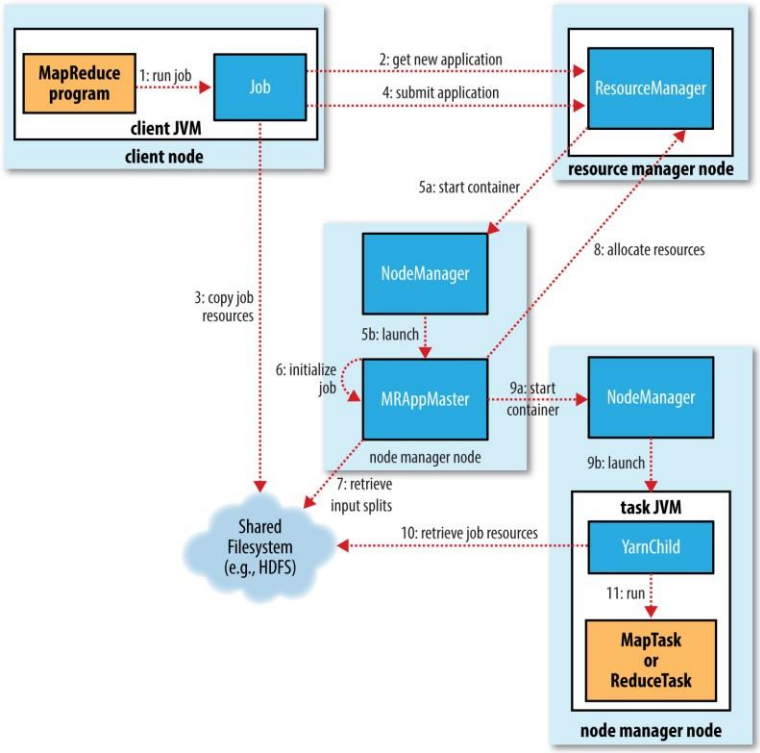
Note: Answer any full 5 questions. All questions carry equal marks. Total Marks: 50

1. a What is Map Reduce? Sketch a neat diagram and explain the logical data flow in Map Reduce?

Map Reduce:

MapReduce is a programming model and an associated implementation for processing and generating big data sets with a parallel, distributed algorithm on a cluster. A MapReduce program is composed of a map procedure, which performs filtering and sorting, and a reduce method, which performs a summary operation.

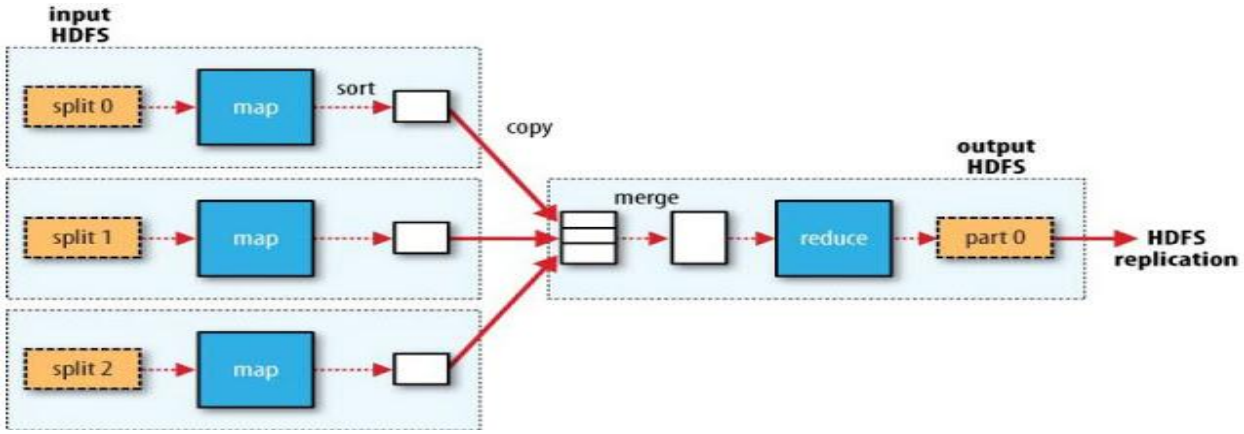
- One map task is created for each split which then executes map function for each record in the split.
- It is always beneficial to have multiple splits, because time taken to process a split is small as compared to the time taken for processing of the whole input. When the splits are smaller, the processing is better load balanced since we are processing the splits in parallel.
- However, it is also not desirable to have splits too small in size. When splits are too small, the overload of managing the splits and map task creation begins to dominate the total job execution time.
- For most jobs, it is better to make split size equal to the size of an HDFS block (which is 64 MB, by default).
- Execution of map tasks results into writing output to a local disk on the respective node and not to HDFS.
- Reason for choosing local disk over HDFS is, to avoid replication which takes place in case of HDFS store operation.
- Map output is intermediate output which is processed by reduce tasks to produce the final output.
- Once the job is complete, the map output can be thrown away. So, storing it in HDFS with replication becomes overkill.
    - In the event of node failure before the map output is consumed by the reduce task, Hadoop reruns the map task on another node and re-creates the map output.
    - Reduce task don't work on the concept of data locality. Output of every map task is fed to the reduce task. Map output is transferred to the machine where reduce task is
    - running.
- On this machine the output is merged and then passed to the user defined reduce function.
- Unlike to the map output, reduce output is stored in HDFS (the first replica is stored on the local node and other replicas are stored on off-rack nodes). So, writing the reduce output



2 a How does a Map reduce model works with a single reduce task? Explain with a neat diagram.

A **Map Task** is a single instance of a MapReduce app. These tasks determine which records to process from a data block. The input data is split and analyzed, in parallel, on the assigned compute resources in a Hadoop cluster. This step of a MapReduce job prepares the <key, value> pair output for the reduce step.

A **Reduce Task** processes an output of a map task. Similar to the map stage, all reduce tasks occur at the same time, and they work independently. The data is aggregated and combined to deliver the desired output. The final result is a reduced set of <key, value> pairs which MapReduce, by default, stores in HDFS.



The Map and Reduce stages have two parts each.

The **Map** part first deals with the **splitting** of the input data that gets assigned to individual map tasks. Then, the **mapping** function creates the output in the form of intermediate key-value pairs.

The **Reduce** stage has a shuffle and a reduce step. **Shuffling** takes the map output and creates a list of related key-value-list pairs. Then, **reducing** aggregates the results of the shuffling to produce the final output that the MapReduce application requested.

3. a Discuss the data format of Weather Dataset and write a Unix code to retrieve the maximum temperature.

Data Format

- NCDC data (National Climatic Data Center)
- The data is stored using a line-oriented ASCII format, in which each line is a record
- Focus is on basic elements such as temperature
- Data files are organized by date and weather station

Example 2-1. Format of a National Climate Data Center record

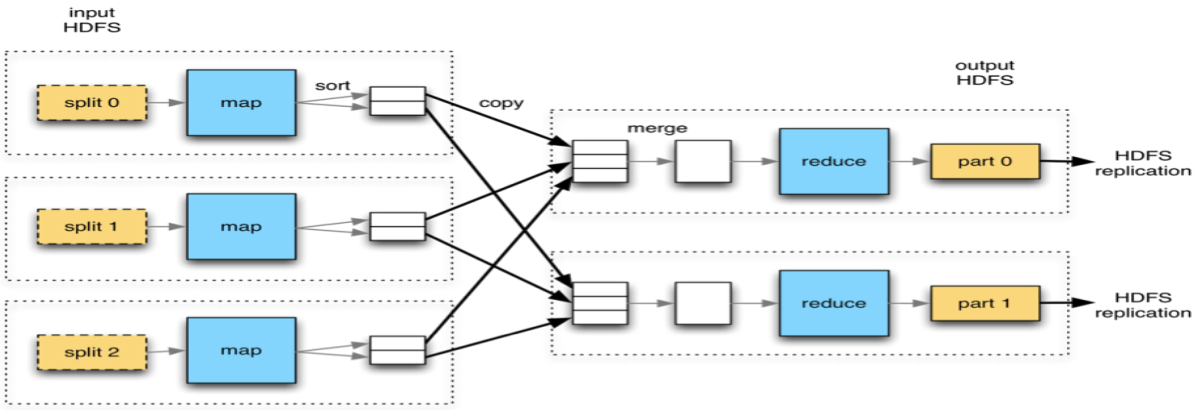
```
0057
332130   # USAF weather station identifier
99999    # WBAN weather station identifier
19500101 # observation date
0300     # observation time
4
+51317   # latitude (degrees x 1000)
+028783  # longitude (degrees x 1000)
FM-12
+0171    # elevation (meters)
99999
V020
320      # wind direction (degrees)
1        # quality code
N
0072
1
00450    # sky ceiling height (meters)
1        # quality code
C
N
010000   # visibility distance (meters)
1        # quality code
N
9
-0128    # air temperature (degrees Celsius x 10)
1        # quality code
-0139    # dew point temperature (degrees Celsius x 10)
1        # quality code
10268    # atmospheric pressure (hectopascals x 10)
1        # quality code
```

Example 2-2. A program for finding the maximum recorded temperature by year from NCDC weather records

```
#!/usr/bin/env bash
for year in all/*
do
  echo -ne `basename $year .gz`"\t"
  gunzip -c $year | \
    awk '{ temp = substr($0, 88, 5) + 0;
           q = substr($0, 93, 1);
           if (temp !=9999 && q ~ /[01459]/ && temp > max) max = temp }
         END { print max }'
done
```

4. a How does a Map reduce model works with a multi- reduce tasks? Explain with a neat diagram.

- MapReduce program executes in three stages, namely map stage, shuffle stage, and reduce stage.
  - 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.
  - 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.
- During a MapReduce job, Hadoop sends the Map and Reduce tasks to the appropriate servers in the cluster.
- The framework manages all the details of data-passing such as issuing tasks, verifying task completion, and copying data around the cluster between the nodes.
- Most of the computing takes place on nodes with data on local disks that reduces the network traffic.
- After completion of the given tasks, the cluster collects and reduces the data to form an appropriate result, and sends it back to the Hadoop server.



5. a Write a Java Map Reduce code to find maximum temperature from the weather data set.

Example 2-3. Mapper for maximum temperature example

```
import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class MaxTemperatureMapper
  extends Mapper<LongWritable, Text, Text, IntWritable> {

  private static final int MISSING = 9999;

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

    String line = value.toString();
    String year = line.substring(15, 19);
    int airTemperature;
    if (line.charAt(87) == '+') { // parseInt doesn't like leading plus signs
      airTemperature = Integer.parseInt(line.substring(88, 92));
    } else {
      airTemperature = Integer.parseInt(line.substring(87, 92));
    }
    String quality = line.substring(92, 93);
    if (airTemperature != MISSING && quality.matches("[01459]")) {
      context.write(new Text(year), new IntWritable(airTemperature));
    }
  }
}
```

Example 2-4. Reducer for maximum temperature example

```
import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class MaxTemperatureReducer
  extends Reducer<Text, IntWritable, Text, IntWritable> {

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

    int maxValue = Integer.MIN_VALUE;
    for (IntWritable value : values) {
      maxValue = Math.max(maxValue, value.get());
    }
    context.write(key, new IntWritable(maxValue));
  }
}
```

Example 2-5. Application to find the maximum temperature in the weather dataset

```
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class MaxTemperature {

  public static void main(String[] args) throws Exception {
    if (args.length != 2) {
      System.err.println("Usage: MaxTemperature <input path> <output path>");
      System.exit(-1);
    }

    Job job = new Job();
    job.setJarByClass(MaxTemperature.class);
    job.setJobName("Max temperature");

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

    job.setMapperClass(MaxTemperatureMapper.class);
    job.setReducerClass(MaxTemperatureReducer.class);

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

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

6. a What is Hadoop Streaming? Find maximum temperature from NCDC dataset by writing code in Python and Ruby.

Hadoop Streaming is a utility in Hadoop that allows developers to write MapReduce programs in any programming language that can read and write to standard input/output streams, such as Python, Ruby, Perl, or even shell scripts.

With Hadoop Streaming, developers can write MapReduce programs without having to write code in Java, the primary language for Hadoop development. Instead, they can use their preferred language to write the MapReduce functions, such as the mapper and reducer functions, as long as they conform to the standard input/output format of Hadoop.

Hadoop Streaming works by reading the input data from Hadoop Distributed File System (HDFS) and

piping it to the external program specified by the developer. The program processes the input data and writes the output to standard output, which is then read by Hadoop and written back to HDFS.

Example 2-10. Map function for maximum temperature in Python

```
#!/usr/bin/env python

import re
import sys

for line in sys.stdin:
  val = line.strip()
  (year, temp, q) = (val[15:19], val[87:92], val[92:93])
  if (temp != "+9999" and re.match("[01459]", q)):
    print "%s\t%s" % (year, temp)
```

Example 2-11. Reduce function for maximum temperature in Python

```
#!/usr/bin/env python

import sys

(last_key, max_val) = (None, 0)
for line in sys.stdin:
  (key, val) = line.strip().split("\t")
  if last_key and last_key != key:
    print "%s\t%s" % (last_key, max_val)
    (last_key, max_val) = (key, int(val))
  else:
    (last_key, max_val) = (key, max(max_val, int(val)))

if last_key:
  print "%s\t%s" % (last_key, max_val)
```

```
#!/usr/bin/env ruby

STDIN.each_line do |line|
  val = line
  year, temp, q = val[15,4], val[87,5], val[92,1]

  puts "#{year}\t#{temp}" if (temp != "+9999" && q =~ /[01459]/)
end
```

```
#!/usr/bin/env ruby

last_key, max_val = nil, 0
STDIN.each_line do |line|
  key, val = line.split("\t")
  if last_key && last_key != key
    puts "#{last_key}\t#{max_val}"
    last_key, max_val = key, val.to_i
  else
    last_key, max_val = key, [max_val, val.to_i].max
  end
end
puts "#{last_key}\t#{max_val}" if last_key
```

7. a Explain in detail the steps involved in running the map reduce program in a cluster

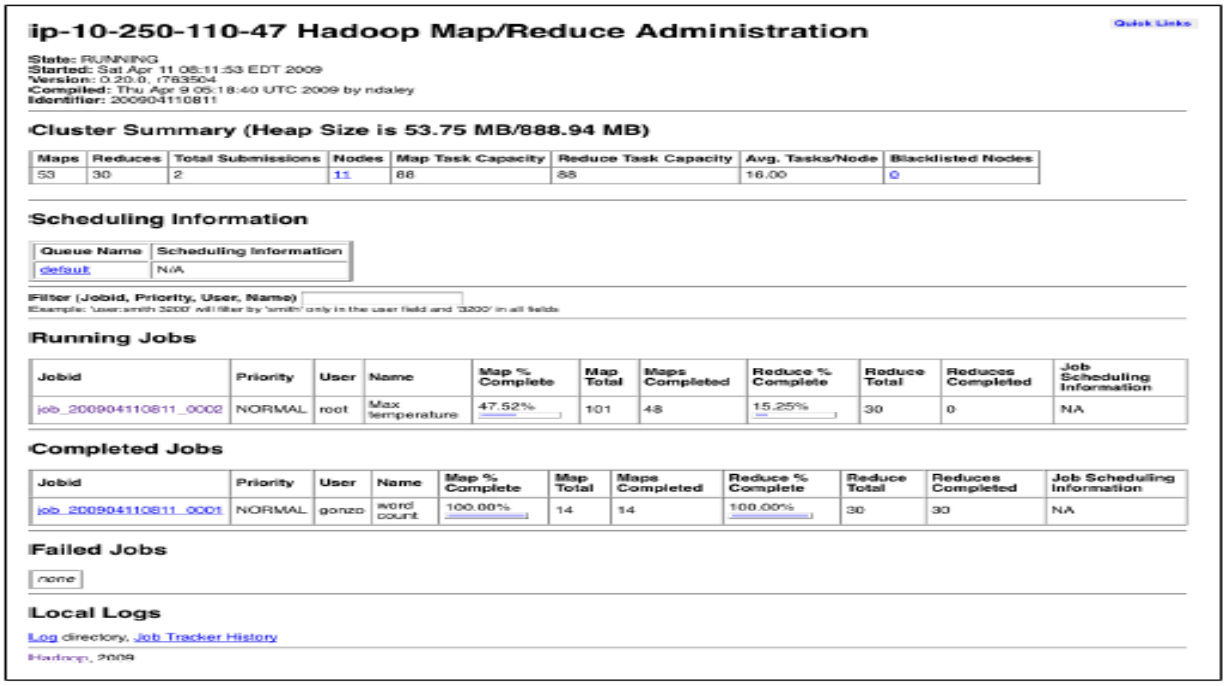
- Packaging
- Launching a job
- The map reduce web UI
- Retrieving the results
- Debugging a job

Packaging

- The program need not be modified to run on a cluster.
- The programs have to be packages as JAR files.
- It is done through Ant tool

```
<jar
destfile="hadoop-examples.jar" basedir="${classes.dir}"/>
```

The map reduce web UI



ip-10-250-110-47 Hadoop Map/Reduce Administration

State: RUNNING
Started: Sat Apr 11 08:11:53 EDT 2009
Version: 0.20.0, r763504
Compiled: Thu Apr 9 05:18:40 UTC 2009 by ndaley
Identifier: 200904110811

Cluster Summary (Heap Size is 53.75 MB/888.94 MB)

Maps	Reduces	Total Submissions	Nodes	Map Task Capacity	Reduce Task Capacity	Avg. Tasks/Node	Blacklisted Nodes
53	30	2	11	88	88	16.00	0

Scheduling Information

Queue Name	Scheduling Information
default	N/A

Filter (Jobid, Priority, User, Name)

Running Jobs

Jobid	Priority	User	Name	Map % Complete	Map Total	Maps Completed	Reduce % Complete	Reduce Total	Reduces Completed	Job Scheduling Information
job_200904110811_0002	NORMAL	root	Max temperature	47.52%	101	48	15.25%	30	0	NA

Completed Jobs

Jobid	Priority	User	Name	Map % Complete	Map Total	Maps Completed	Reduce % Complete	Reduce Total	Reduces Completed	Job Scheduling Information
job_200904110811_0001	NORMAL	gonzo	word count	100.00%	14	14	100.00%	30	30	NA

Failed Jobs

none

Local Logs

Log directory, Job Tracker History

Hadoop, 2009

Debugging a Job:

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

  parser.parse(value);
  if (parser.isValidTemperature()) {
    int airTemperature = parser.getAirTemperature();
    if (airTemperature > 1000) {
      System.err.println("Temperature over 100 degrees for input: " + value);
      context.setStatus("Detected possibly corrupt record: see logs.");
      context.getCounter(Temperature.OVER_100).increment(1);
    }
    context.write(new Text(parser.getYear()), new IntWritable(airTemperature));
  }
```

8. a Explain in detail the configuration file, its APIs and access Properties

The *hadoop-local.xml* file contains the default Hadoop configuration for the default filesystem and the jobtracker:

```
<?xml version="1.0"?>
<configuration>

  <property>
    <name>fs.default.name</name>
    <value>file:///</value>
  </property>

  <property>
    <name>mapred.job.tracker</name>
    <value>local</value>
  </property>

</configuration>
```

The settings in *hadoop-localhost.xml* point to a namenode and a jobtracker both running on localhost:

```
<?xml version="1.0"?>
<configuration>

  <property>
    <name>fs.default.name</name>
    <value>hdfs://localhost/</value>
  </property>

  <property>
    <name>mapred.job.tracker</name>
    <value>localhost:8021</value>
  </property>

</configuration>
```

There are several notable differences between the two APIs:

- The new API favors abstract classes over interfaces, since these are easier to evolve. For example, you can add a method (with a default implementation) to an abstract class without breaking old implementations of the class[2]. For example, the `Mapper` and `Reducer` interfaces in the old API are abstract classes in the new API.
- The new API is in the `org.apache.hadoop.mapreduce` package (and subpackages). The old API can still be found in `org.apache.hadoop.mapred`.
- The new API makes extensive use of context objects that allow the user code to communicate with the MapReduce system. The new `Context`, for example, essentially unifies the role of the `JobConf`, the `OutputCollector`, and the `Reporter` from the old API.
- In both APIs, key-value record pairs are pushed to the mapper and reducer, but in addition, the new API allows both mappers and reducers to control the execution flow by overriding the `run()` method. For example, records can be processed in batches, or the execution can be terminated before all the records have been processed. In the old API this is possible for mappers by writing a `MapRunnable`, but no equivalent exists for reducers.
- Configuration has been unified. The old API has a special `JobConf` object for job configuration, which is an extension of Hadoop's vanilla `Configuration` object (used for configuring daemons; see "The Configuration API" on page 146). In the new API, this distinction is dropped, so job configuration is done through a `Configuration`.
- Job control is performed through the `Job` class in the new API, rather than the old `JobClient`, which no longer exists in the new API.

The old and the new Java MapReduce APIs

The Java MapReduce API used in the previous section was first released in Hadoop 0.20.0. This new API, sometimes referred to as "Context Objects," was designed to make the API easier to evolve in the future. It is type-incompatible with the old, however, so applications need to be rewritten to take advantage of it.

9. a Write a short note on: Remote Debugging and Hadoop Logs

Remote Debugging

Remote debugging in Hadoop refers to the process of debugging Hadoop applications running on a remote cluster. Hadoop is a distributed computing framework, which means that the application code runs on a cluster of multiple nodes, each with its own CPU and memory resources. Remote debugging allows developers to debug their Hadoop applications without having to physically access the cluster.

To perform remote debugging in Hadoop, developers typically use an integrated development environment (IDE) such as Eclipse or IntelliJ IDEA. The IDE is configured to connect to the Hadoop cluster using the Hadoop Distributed File System (HDFS) and the MapReduce framework. Once the IDE is connected to the cluster, developers can set breakpoints and step through the code to identify and fix errors.

Remote debugging in Hadoop is essential for troubleshooting complex Hadoop applications, as it allows developers to inspect the state of the application and the data it processes while it is running on the cluster. It also helps to reduce the time required to identify and fix errors, as developers can debug their code in real-time without having to run the application repeatedly.

Hadoop Logs:

Logs	Primary audience	Description	Further information
System daemon logs	Administrators	Each Hadoop daemon produces a logfile (using log4j) and another file that combines standard out and error. Written in the directory defined by the HADOOP_LOG_DIR environment variable.	"System logfiles" on page 307 and "Logging" on page 349.
HDFS audit logs	Administrators	A log of all HDFS requests, turned off by default. Written to the namenode's log, although this is configurable.	"Audit Logging" on page 344.

Logs	Primary audience	Description	Further information
MapReduce job history logs	Users	A log of the events (such as task completion) that occur in the course of running a job. Saved centrally on the jobtracker, and in the job's output directory in a *_logs/history* subdirectory.	"Job History" on page 166.
MapReduce task logs	Users	Each tasktracker child process produces a logfile using log4j (called *syslog*), a file for data sent to standard out (*stdout*), and a file for standard error (*stderr*). Written in the *userlogs* subdirectory of the directory defined by the HADOOP_LOG_DIR environment variable.	This section.

10. a Write a map reduce program for word count problem

```
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
```

```
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {
  public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable>{
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context) throws IOException, InterruptedException
{
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }

  public static class IntSumReducer extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();

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

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```