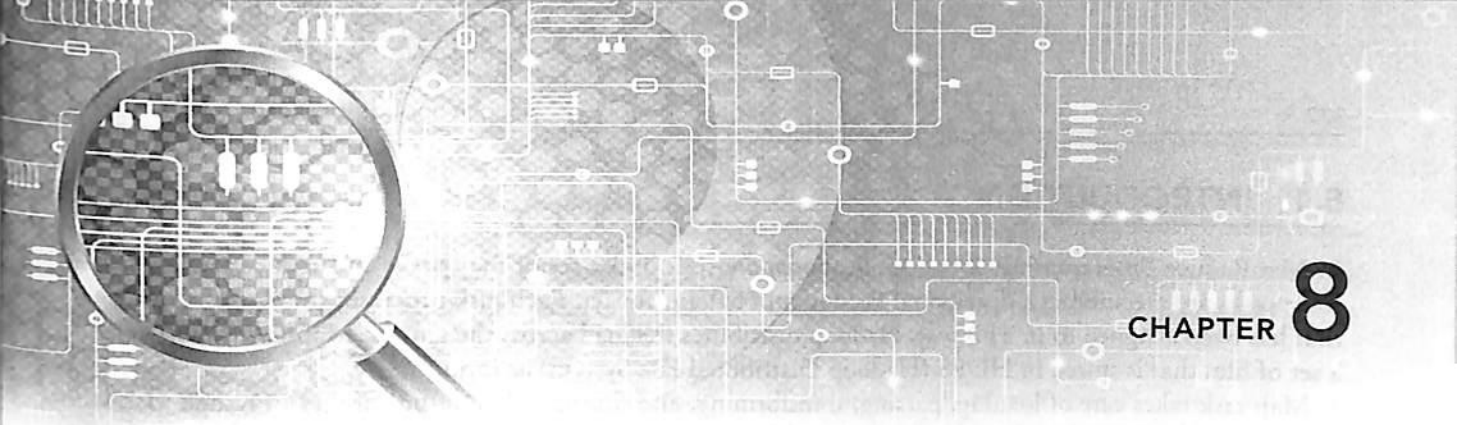

CHAPTER 8

Introduction to MAPREDUCE Programming

BRIEF CONTENTS

- What's in Store?
- Introduction
- Mapper
  - RecordReader
  - Map
  - Combiner
  - Partitioner
- Reducer
  - Shuffle
  - Sort
  - Reduce
  - Output Format
- Combiner
- Partitioner
- Searching
- Sorting
- Compression

"The alchemists in their search for gold discovered many other things of greater value."

– Arthur Schopenhauer, German Philosopher

WHAT'S IN STORE?

We assume that you are familiar with the basic concepts of HDFS and MapReduce Programming discussed in Chapters 4 and 5. The focus of this chapter will be to build on this knowledge to understand optimization techniques of MapReduce Programming such as combiner, partitioner, and compression. We will also discuss how to write MapReduce Programming for sorting and searching.

We suggest you refer to some of the learning resources provided at the end of this chapter for better learning and comprehension.

8.1 INTRODUCTION

In MapReduce Programming, Jobs (Applications) are split into a set of map tasks and reduce tasks. Then these tasks are executed in a distributed fashion on Hadoop cluster. Each task processes small subset of data that has been assigned to it. This way, Hadoop distributes the load across the cluster. MapReduce job takes a set of files that is stored in HDFS (Hadoop Distributed File System) as input.

Map task takes care of loading, parsing, transforming, and filtering. The responsibility of reduce task is grouping and aggregating data that is produced by map tasks to generate final output. Each map task is broken into the following phases:

1. RecordReader.
2. Mapper.
3. Combiner.
4. Partitioner.

The output produced by map task is known as intermediate keys and values. These intermediate keys and values are sent to reducer. The reduce tasks are broken into the following phases:

1. Shuffle.
2. Sort.
3. Reducer.
4. Output Format.

Hadoop assigns map tasks to the DataNode where the actual data to be processed resides. This way, Hadoop ensures data locality. Data locality means that data is not moved over network; only computational code is moved to process data which saves network bandwidth.

8.2 MAPPER

A mapper maps the input key–value pairs into a set of intermediate key–value pairs. Maps are individual tasks that have the responsibility of transforming input records into intermediate key–value pairs.

1. **RecordReader:** RecordReader converts a byte-oriented view of the input (as generated by the InputSplit) into a record-oriented view and presents it to the Mapper tasks. It presents the tasks with keys and values. Generally the key is the positional information and value is a chunk of data that constitutes the record.
2. **Map:** Map function works on the key–value pair produced by RecordReader and generates zero or more intermediate key–value pairs. The MapReduce decides the key–value pair based on the context.
3. **Combiner:** It is an optional function but provides high performance in terms of network bandwidth and disk space. It takes intermediate key–value pair provided by mapper and applies user-specific aggregate function to only that mapper. It is also known as local reducer.
4. **Partitioner:** The partitioner takes the intermediate key–value pairs produced by the mapper, splits them into shard, and sends the shard to the particular reducer as per the user-specific code. Usually, the key with same values goes to the same reducer. The partitioned data of each map task is written to the local disk of that machine and pulled by the respective reducer.

8.3 REDUCER

The primary chore of the Reducer is to reduce a set of intermediate values (the ones that share a common key) to a smaller set of values. The Reducer has three primary phases: Shuffle and Sort, Reduce, and Output Format.

1. **Shuffle and Sort:** This phase takes the output of all the partitioners and downloads them into the local machine where the reducer is running. Then these individual data pipes are sorted by keys which produce larger data list. The main purpose of this sort is grouping similar words so that their values can be easily iterated over by the reduce task.
2. **Reduce:** The reducer takes the grouped data produced by the shuffle and sort phase, applies reduce function, and processes one group at a time. The reduce function iterates all the values associated with that key. Reducer function provides various operations such as aggregation, filtering, and combining data. Once it is done, the output (zero or more key–value pairs) of reducer is sent to the output format.
3. **Output Format:** The output format separates key–value pair with tab (default) and writes it out to a file using record writer.

Figure 8.1 describes the chores of Mapper, Combiner, Partitioner, and Reducer for the word count problem. The Word Count problem has been discussed under "Combiner" and "Partitioner".

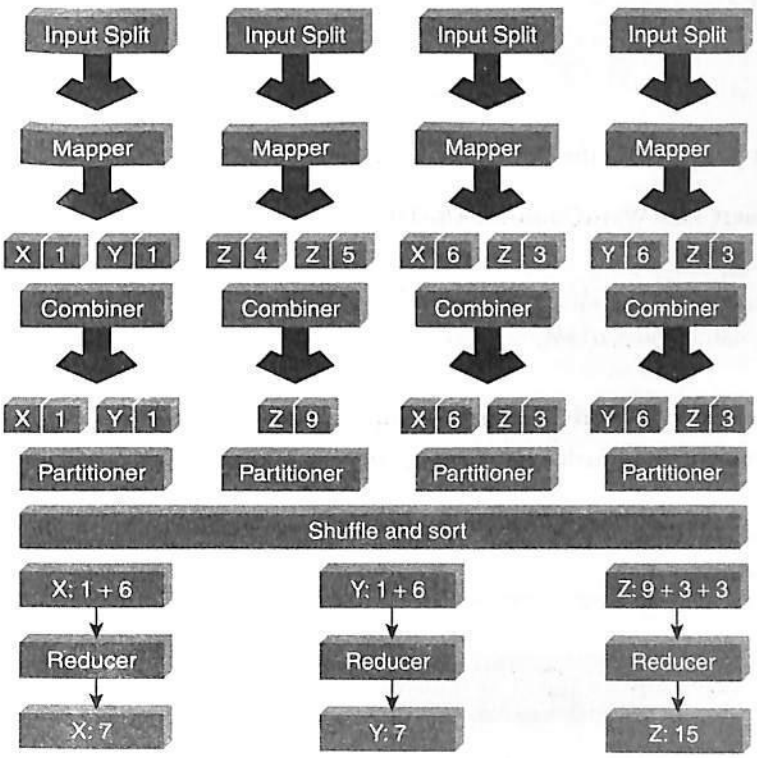

Figure 8.1 The chores of Mapper, Combiner, Partitioner, and Reducer.

8.4 COMBINER

It is an optimization technique for MapReduce Job. Generally, the reducer class is set to be the combiner class. The difference between combiner class and reducer class is as follows:

1. Output generated by combiner is intermediate data and it is passed to the reducer.
2. Output of the reducer is passed to the output file on disk.

The sections have been designed as follows:

Objective: What is it that we are trying to achieve here?
Input Data: What is the input that has been given to us to act upon?
Act: The actual statement/command to accomplish the task at hand.
Output: The result/output as a consequence of executing the statement.

Objective: Write a MapReduce program to count the occurrence of similar words in a file. Use combiner for optimization.

Note: Refer Chapter 5 – Hadoop for Mapper Class and Reduce Class and Driver Program.

Input Data:

```
Welcome to Hadoop Session
Introduction to Hadoop
Introducing Hive
Hive Session
Pig Session
```

Act: In the driver program, set the combiner class as shown below.

```
job.setCombinerClass(WordCounterRed.class);

// Input and Output Path
FileInputFormat.addInputPath(job, new Path("/mapreducedemos/lines.txt"));
FileOutputFormat.setOutputPath(job, new Path("/mapreducedemos/output/wordcount/"));
```

hadoop jar <<jar name>> <<driver class>> <<input path>> <<output path>>

Here driver class name, input path, and output path are optional arguments.

Output:

```
[root@volgalnx010 mapreducedemos]# hadoop jar wordcount.jar
```

Contents of directory /mapreducedemos

Goto : /mapreducedemos go

Go to parent directory

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
lines.txt	file	91 B	3	128 MB	2015-03-01 21:05	rw-r--r--	root	supergroup
output	dir				2015-03-01 23:21	rwxr-xr-x	root	supergroup

Go back to DFS home

Local logs

Contents of directory /mapreducedemos/output

Goto : /mapreducedemos/output go

Go to parent directory

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
wordcount	dir				2015-03-01 23:21	rwxr-xr-x	root	supergroup

Go back to DFS home

Local logs

The reducer output will be stored in part-r-00000 file by default.

Contents of directory /mapreducedemos/output/wordcount

Goto : /mapreducedemos/output/wo go

Go to parent directory

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
_SUCCESS	file	0 B	3	128 MB	2015-03-01 23:21	rw-r--r--	root	supergroup
part-r-00000	file	76 B	3	128 MB	2015-03-01 23:21	rw-r--r--	root	supergroup

Go back to DFS home

Local logs

File: /mapreducedemos/output/wordcount/part-r-00000

Goto : /mapreducedemos/output/wo go

Go back to dir listing
Advanced view/download options

```
Hadoop  2
Hive    2
Introducing     1
Introduction    1
Pig     1
Session 3
Welcome 1
to      2
```

8.5 PARTITIONER

The partitioning phase happens after map phase and before reduce phase. Usually the number of partitions are equal to the number of reducers. The default partitioner is hash partitioner.

Objective: Write a MapReduce program to count the occurrence of similar words in a file. Use partitioner to partition key based on alphabets.
Note: Refer Chapter 5 – Hadoop for Mapper Class and Reduce Class and Driver Program.

Input Data:

```
Welcome to Hadoop Session
Introduction to Hadoop
Introducing Hive
Hive Session
Pig Session
```

Act:

WordCountPartitioner.java

```
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Partitioner;

public class WordCountPartitioner extends Partitioner<Text, IntWritable> {
        @Override
        public int getPartition(Text key, IntWritable value, int numPartitions) {
                String word = key.toString();
                char alphabet = word.toUpperCase().charAt(0);
                int partitionNumber = 0;
                switch(alphabet) {
                case 'A': partitionNumber = 1; break;
                case 'B': partitionNumber = 2; break;
                case 'C': partitionNumber = 3; break;
                case 'D': partitionNumber = 4; break;
                case 'E': partitionNumber = 5; break;
                case 'F': partitionNumber = 6; break;
                case 'G': partitionNumber = 7; break;
                case 'H': partitionNumber = 8; break;
                case 'I': partitionNumber = 9; break;
                case 'J': partitionNumber = 10; break;
                case 'K': partitionNumber = 11; break;
                case 'L': partitionNumber = 12; break;
                case 'M': partitionNumber = 13; break;
                case 'N': partitionNumber = 14; break;
                case 'O': partitionNumber = 15; break;
                case 'P': partitionNumber = 16; break;
                case 'Q': partitionNumber = 17; break;
                case 'R': partitionNumber = 18; break;
                case 'S': partitionNumber = 19; break;
                case 'T': partitionNumber = 20; break;
                case 'U': partitionNumber = 21; break;
                case 'V': partitionNumber = 22; break;
                case 'W': partitionNumber = 23; break;
                case 'X': partitionNumber = 24; break;
                case 'Y': partitionNumber = 25; break;
                case 'Z': partitionNumber = 26; break;
                default: partitionNumber = 0; break;
                }
                return partitionNumber;
        }
}
```

In the driver program, set the partitioner class as shown below:

```
job.setNumReduceTasks(27);
job.setPartitionerClass(WordCountPartitioner.class);

// Input and Output Path
FileInputFormat.addInputPath(job, new Path("/mapreducedemos/lines.txt"));
FileOutputFormat.setOutputPath(job, new Path("/mapreducedemos/output/
wordcountpartitioner/"));
```

Output:

You can see 27 partitions in the below output.

Contents of directory /mapreducedemos/output/wordcountpartitioner

Goto : /mapreducedemos/output/wo go

Go to parent directory

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
_SUCCESS	file	0 B	3	128 MB	2015-03-01 23:40	rw-r--r--	root	supergroup
part-r-00000	file	0 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00001	file	0 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00002	file	0 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00003	file	6 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00004	file	0 B	3	128 MB	2015-03-01 23:40	rw-r--r--	root	supergroup
part-r-00005	file	0 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00006	file	0 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00007	file	0 B	3	128 MB	2015-03-01 23:40	rw-r--r--	root	supergroup
part-r-00008	file	16 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00009	file	29 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00010	file	0 B	3	128 MB	2015-03-01 23:40	rw-r--r--	root	supergroup
part-r-00011	file	0 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00012	file	0 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00013	file	0 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00014	file	0 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00015	file	0 B	3	128 MB	2015-03-01 23:40	rw-r--r--	root	supergroup
part-r-00016	file	6 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00017	file	0 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00018	file	0 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00019	file	10 B	3	128 MB	2015-03-01 23:40	rw-r--r--	root	supergroup
part-r-00020	file	5 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00021	file	0 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00022	file	0 B	3	128 MB	2015-03-01 23:40	rw-r--r--	root	supergroup
part-r-00023	file	10 B	3	128 MB	2015-03-01 23:40	rw-r--r--	root	supergroup
part-r-00024	file	0 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00025	file	0 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00026	file	0 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup

The output file part-r-00008 is associated with alphabet **'H'**.

File: /mapreducedemos/output/wordcountpartitioner/part-r-00008

Goto : /mapreducedemos/output/wo go

Go back to dir listing
Advanced view/download options

```
Hadoop  2
Hive    2
```

8.6 SEARCHING

Objective: To write a MapReduce program to search for a specific keyword in a file.

Input Data:

```
1001,John,45
1002,Jack,39
1003,Alex,44
1004,Smith,38
1005,Bob,33
```

Act:

WordSearcher.java

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
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.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
public class WordSearcher {
        public static void main(String[] args) throws IOException,
                        InterruptedException, ClassNotFoundException {
                Configuration conf = new Configuration();
                Job job = new Job(conf);
                job.setJarByClass(WordSearcher.class);
                job.setOutputKeyClass(Text.class);
                job.setOutputValueClass(Text.class);
                job.setMapperClass(WordSearchMapper.class);
                job.setReducerClass(WordSearchReducer.class);
                job.setInputFormatClass(TextInputFormat.class);
                job.setOutputFormatClass(TextOutputFormat.class);
                job.setNumReduceTasks(1);
                job.getConfiguration().set("keyword", "Jack");
                FileInputFormat.setInputPaths(job, new Path("/mapreduce/student.csv"));
```

```
                FileOutputFormat.setOutputPath(job, new Path("/mapreduce/output/search"));
                System.exit(job.waitForCompletion(true) ? 0 : 1);
        }
}
```

WordSearchMapper.java

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.InputSplit;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;
public class WordSearchMapper extends Mapper<LongWritable, Text, Text, Text> {
        static String keyword;
        static int pos = 0;
        protected void setup(Context context) throws IOException,
                        InterruptedException {
                Configuration configuration = context.getConfiguration();
                keyword = configuration.get("keyword");
        }
        protected void map(LongWritable key, Text value, Context context)
                        throws IOException, InterruptedException {
                InputSplit i = context.getInputSplit(); // Get the input split for this map.
                FileSplit f = (FileSplit) i;
                String fileName = f.getPath().getName();
                Integer wordPos;
                pos++;
                if (value.toString().contains(keyword)) {
                        wordPos = value.find(keyword);
                        context.write(value, new Text(fileName + ","+ new IntWritable(pos).
toString() + ", "+ wordPos.toString()));
                }
        }
}
```

WordSearchReducer.java

```
import java.io.IOException;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
public class WordSearchReducer extends Reducer<Text, Text, Text, Text> {
        protected void reduce(Text key, Text value, Context context)
                        throws IOException, InterruptedException {
                context.write(key, value);
        }
}
```

Output:

File: /mapreduce/output/search/part-r-00000

Goto : /mapreduce/output/search go

Go back to dir listing
Advanced view/download options

```
1002,Jack,39    student.csv,2, 5
```

8.7 SORTING

Objective: To write a MapReduce program to sort data by student name (value).

Input Data:

```
1001,John,45
1002,Jack,39
1003,Alex,44
1004,Smith,38
1005,Bob,33
```

Act:

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;
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 SortStudNames {

    public static class SortMapper extends
            Mapper<LongWritable, Text, Text, Text> {

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

            String[ ] token = value.toString().split(",");
            context.write(new Text(token[1]), new Text(token[0]+" - "+token[1]));
        }
    }
  // Here, value is sorted...
    public static class SortReducer extends
            Reducer<Text, Text, NullWritable, Text> {

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

            for (Text details : values) {
                context.write(NullWritable.get(), details);
            }
        }
    }

    public static void main(String[ ] args) throws IOException,
            InterruptedException, ClassNotFoundException {

        Configuration conf = new Configuration();
        Job job = new Job(conf);
        job.setJarByClass(SortEmpNames.class);
        job.setMapperClass(SortMapper.class);
        job.setReducerClass(SortReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);
        FileInputFormat.setInputPaths(job, new Path("/mapreduce/student.csv"));
        FileOutputFormat.setOutputPath(job, new
Path("/mapreduce/output/sorted/"));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }

}
```

Output:

File: /mapreduce/output/search/part-r-00000

Goto : /mapreduce/output/search go

Go back to dir listing
Advanced view/download options

```
1002,Jack,39    student.csv,2, 5
```

8.8 COMPRESSION

In MapReduce programming, you can compress the MapReduce output file. Compression provides two benefits as follows:

1. Reduces the space to store files.
2. Speeds up data transfer across the network.

You can specify compression format in the Driver Program as shown below:

```
conf.setBoolean("mapred.output.compress", true);
conf.setClass("mapred.output.compression.codec", GzipCodec.class,CompressionCodec.class);
```

Here, codec is the implementation of a compression and decompression algorithm. GzipCodec is the compression algorithm for gzip. This compresses the output file.

REMIND ME

- Mapper maps the input key–value pairs to intermediate key–value pairs.
- Reducer then reduces the set of key–value pairs that share a common key to a smaller set of values.
- The Reducer has three primary phases:
  - Shuffle and Sort
  - Reduce
  - Output Format
- Combiner and Partitioner are optimization techniques.

POINT ME (BOOK)

- MapReduce Design Patterns, O'REILLY, Donald Miner and Adam Shook.

CONNECT ME (INTERNET RESOURCES)

- http://hadooptutorial.wikispaces.com/MapReduce
- http://bigdataanalyticsnews.com/anatomy-mapreduce-job/
- http://bigdataconsultants.blogspot.in/2013/11/secondary-sort-in-hadoop-actor.html

TEST ME

A. Fill Me

1. Partitioner phase belongs to _____ task.
2. Combiner is also known as _____.
3. RecordReader converts byte-oriented view into _____ view.
4. MapReduce sorts the intermediate value based on _____.
5. In MapReduce Programming, reduce function is applied _____ group at a time.

Answers:

1. map
2. local reducer
3. record-oriented
4. keys
5. one

ASSIGNMENT FOR HANDS-ON PRACTICE

ASSIGNMENT 1

Objective: To learn about MapReduce Programming using Java.
Problem Description: Write a MapReduce Program to arrange the data on user id, then within the user id sort them in increasing order of the page count.

Input:

User_id	count	URL
12398	5	http://www.cbtnuggets.com/
23487	9	http://www.xda-developers.com/
34576	3	http://www.w3schools.com/
45665	6	https://www.google.co.in/
56754	4	http://www.encyclopedia.com/
67843	6	http://tutorialspoint.com/
78932	7	http://stackoverflow.com/
89021	3	http://www.wikipedia.org/
91210	2	http://www.cisce.org/results
82391	4	http://www.slideshare.net/

ASSIGNMENT 2

Objective: To learn about MapReduce Programming using Java.
Problem Description: Write a MapReduce Program to find unitwise salary.

Input:

Empno	Empname	Unit	Designation	Salary	Location
1001	John	IMST	TA	30000	Trivandrum
1002	Jack	CLOUD	PM	80000	Bangalore
1003	Joshi	FNPR	TA	35000	Trivandrum
1004	Josh	ECSSAP	PM	75000	Bangalore
1005	Jim	FSADM	SPM	60000	Bangalore
1006	Smith	ICS	TA	24000	Chandigarh
1007	Tiger	IMST	SPM	56000	Trivandrum
1008	Kate	FNPR	PM	76000	Chennai
1009	Cassy	MFGADM	TA	40000	Bangalore
1010	Ronald	ECSSAP	SPM	65000	Chennai