Omkar Gurav
TE IT Batch B T3
Roll No:- 8048

Assignment - 3

- Aim:- Design a distributed application using MapReduce which processes a logfile of a system. List out the users who have logged maximum times of the system. Use simple log file and process it using pseudo distribution mode on Hadoop platform.

- Theory:-

Q.1. Explain job execution in Hadoop.

→ MapReduce is a programming model designed to process huge amounts of data by dividing the job into independent local tasks.

When user submits a mapreduce job to Hadoop, the loc job client prepares the job for submission & hands it off to the job tracker.

The job tracker schedules the job & distributes the m work amongst multiple task trackers for parallel processing.

Each task tracker issues a map task. These tasks are assigned with task IDs. Job initialization & job

cleanup task are created & run by these task trackers.

Once mapping phase results are available, job tracker distributes the reduce work among the task trackers for parallel processing.

Each task tracker issues a reduce task to perform the work. Job tracker receives progress information from task trackers. Job client keeps polling the job tracker for progress.

Once job is completed, cleanup task gets processed. Task tracker sends the job completion status to the job tracker. Job tracker sends job completion message to the client. The tool process causes Job Clients waitForJobToComplete method to return.

Q.2. Explain following classes:-

① IntWritable

→

① IntWritable is the wrapper class in Hadoop which is similar to Integer class in Java. It is optimized to provide serialization in Hadoop.

② It implements Comparable Writable & WritableComparable interfaces.

② Iterable

→ ① Java iterable interface represents a collection of objects which can be iterable. A class implementing this interface can have its elements iterated.

eg. Iterable<string> = new Iterable[];

③ Context

→ ① The context object allows the Mapper/Reducer to interact with the rest of the Hadoop system.

② It includes configuration for the job and provides functions to write to an area of memory the outputs of various tasks.

eg: Context con;

con.write(key-val pair)

- Conclusion:- Map Reduce application to process log file is successfully implemented.

Driver class : Driver.java

```
package log;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FileStatus;
import org.apache.hadoop.fs.FileSystem;
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 Driver {

              @SuppressWarnings("deprecation")
           public static void main(String[] args) throws Exception{
           //creating object of configuration class
           Configuration c = new Configuration();

           //Assigning job to new configuration object
           Job job = new Job(c);

           //setting jar class
           job.setJarByClass(log.Driver.class);

           job.setMapperClass(log.LogMapper.class);

           job.setReducerClass(log.LogReducer.class);

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

           //Adding a Path to the list of inputs
           FileInputFormat.addInputPath(job, new Path(args[0]));

           //Setting the Path of the output directory
           FileOutputFormat.setOutputPath(job,new Path(args[1]));

           //wait till job is completed
           job.waitForCompletion(true);

              //file system object
              FileSystem fs = FileSystem.get(c);

              FileStatus[] status = fs.listStatus(new
Path("hdfs://localhost:9000"+args[1]));
              FSDataInputStream fd = fs.open(status[1].getPath());

              String str = fd.readLine();
              String ip = "";
              int max = 0;

              while(str != null)
              {

                     String parts[] = str.split("\t");

                     if(max<Integer.parseInt(parts[1])) {

                            max = Integer.parseInt(parts[1]);
                            ip = parts[0];
                     }

              str = fd.readLine();
       }

              System.out.println("IP address : " + ip);
              System.out.println("No. of occurrences : " + max);
       }
}
```

Mapper class : LogMapper.java

```
package log;

import java.io.IOException;
import java.util.StringTokenizer;
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 LogMapper extends Mapper <LongWritable,Text,Text,IntWritable> {

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

              String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);

        con.write(new Text((tokenizer.nextToken())),new IntWritable(1));

       }
}
```

Reducer class : LogReducer.java

```
package log;

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

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

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

              int sum=0;

           for(IntWritable value : values)
           {
              sum += value.get();
           }

           con.write(word, new IntWritable(sum));
       }



}
```

Output Screenshots