Sinhgad Institutes

Name of the Student:_____**Roll no:**_____

CLASS:- T.E.[I.T] **Division: A**

Course: - 2019

Subject: 314457: Data Science and Big Data Analytics Laboratory

PART_ A _Assignment No. 02

Marks: _____/10

Date of Performance: ___/____/_____ **Sign with Date: _____**

Part- A

ASSIGNMENT NO: 02

TITLE:

To Design a distributed application using MapReduce which processes a log file of a system.

AIM:

To List out the users who have logged for maximum period on the system. Use simple log file from the Internet and process it using a pseudo distribution mode on Hadoop platform.

OBJECTIVE:

1) To apply Big data primitives and fundamentals for application development.
2) To design algorithms and techniques for Big data analytics.

SOFTWARE USED: Hadoop 3.2.2, jdk11, Eclipse.

THEORY:

MapReduce:

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. Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs). Secondly, reduce task, which takes the output from a map as an input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce task is always performed after the map job.

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. Decomposing a data processing application into mappers and reducers is sometimes nontrivial. But, once we write an application in the MapReduce form, scaling the application to run over hundreds, thousands, or even tens of thousands of machines in a cluster is merely a configuration change. This simple scalability is what has attracted many programmers to use the MapReduce model.

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.

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.

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. Additionally, the key classes have to implement the Writable- Comparable interface to facilitate sorting by the framework. Input and Output types of a MapReduce job: (Input) <k1, v1> -> map -> <k2, v2>-> reduce -> <k3, v3>(Output).

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

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 - Schedules jobs and tracks the assign jobs to Task tracker.

Task Tracker - Tracks the task and reports status to JobTracker.

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.

Important Commands

All Hadoop commands are invoked by the $HADOOP_HOME/bin/hadoop command. Running the Hadoop script without any arguments prints the description for all commands.

Usage: hadoop [--config confdir] COMMAND

The following table lists the options available and their description.

Options	Description
namenode -format	Formats the DFS filesystem.
secondarynamenode	Runs the DFS secondary namenode.
namenode	Runs the DFS namenode.
datanode	Runs a DFS datanode.
dfsadmin	Runs a DFS admin client.
mradmin	Runs a Map-Reduce admin client.
fsck	Runs a DFS filesystem checking utility.

Fs	Runs a generic filesystem user client.
balancer	Runs a cluster balancing utility.
oiv	Applies the offline fsimage viewer to an fsimage.
fetchdt	Fetches a delegation token from the NameNode.
jobtracker	Runs the MapReduce job Tracker node.
pipes	Runs a Pipes job.
tasktracker	Runs a MapReduce task Tracker node.
historyserver	Runs job history servers as a standalone daemon.
job	Manipulates the MapReduce jobs.
queue	Gets information regarding JobQueues.
version	Prints the version.
jar <jar>	Runs a jar file.
distcp <srcurl> <desturl>	Copies file or directories recursively.
distcp2 <srcurl> <desturl>	DistCp version 2.
archive -archiveName NAME -p	Creates a hadoop archive.
<parent path> <src>* <dest>	
classpath	Prints the class path needed to get the Hadoop jar and the required libraries.
daemonlog	Get/Set the log level for each daemon

How to Interact with MapReduce Jobs

Usage: hadoop job [GENERIC_OPTIONS]

The following are the Generic Options available in a Hadoop job.

GENERIC_OPTIONS	**Description**
-submit <job-file>	Submits the job.
-status <job-id>	Prints the map and reduce completion percentage and all job counters.
-counter <job-id> <groupname> <countername>	Prints the counter value.
-kill <job-id>	Kills the job.
-events <job-id> <fromevent- #> <#-ofevents>	Prints the events' details received by jobtracker for the given range.

-history [all] <jobOutputDir> - history < jobOutputDir>	Prints job details, failed and killed tip details. More details
	about the job such as successful tasks and task attempts made for each task can be viewed by specifying the [all] option.
-list[all]	Displays all jobs. -list displays only jobs which are yet to complete.
-kill-task <task-id>	Kills the task. Killed tasks are NOT counted against failed attempts.
-fail-task <task-id>	Fails the task. Failed tasks are counted against failed attempts.
-set-priority <job-id> <priority>	Changes the priority of the job. Allowed priority values are VERY_HIGH, HIGH, NORMAL, LOW, VERY_LOW

CONCLUSION:

After the study of this assignment, we design a distributed application using MapReduce which processes a log file of a system.

Write Short Answers for Following Questions:

1. What is HDFS & MAPREDUCE?
2. Which Package required for IntWritable, LongWritable, and Text etc.
3. To output which package and methods are used?
4. Which package is required for Mapper and Reducer?
5. Which are the yarn daemons?
6. How start daemons for Hadoop write commands?

Viva Questions:

1. What is use of Reporter?
2. How to put input file in HDFS?
3. Explain what is Speculative Execution?
4. Explain what are the basic parameters of a Mapper?
5. Explain what is the function of MapReducer practitioner?
6. Explain what is difference between an Input Split and HDFS Block?
7. Explain how JobTracker schedules a task?

Design Experiments/ Lab innovations:

Design a distributed application using MapReduce under Hadoop for: Counting no. of occurrences of every word in a given text file.

WordCountMapReduce Program Execution steps

https://mapr.com/blog/how-write-mapreduce-program/

1. Create Java Project, add new class store following code.
2.Right Click to java project add hadoop jar files from
/usr/local/haoop/share/hadoop/common and from mapreduce folder
Right Click to Java project --→ select Build Path ---→ Configure Build Path---→ Add External Jar files --→add jar files from common and mapreduce folder

Java Program

```
import java.io.IOException;
import java.util.Iterator;

import org.apache.hadoop.io.Text;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class WordCount extends Configured implements Tool {
        @Override
        public int run(String args[]) throws Exception {
                if (args.length < 2 )
                {
                        System.out.println("Please give input and output directories properly");
```

```
                return -1;
                }
    JobConf conf = new JobConf(WordCount.class);
    //Path inp = new Path(args[0]);

    FileInputFormat.setInputPaths(conf,new Path(args[0]));
    FileOutputFormat.setOutputPath(conf,new Path(args[1]));
    conf.setMapperClass(WordMapper.class);
    conf.setReducerClass(WordReducer.class);
    conf.setMapOutputKeyClass(Text.class);
    conf.setMapOutputValueClass(IntWritable.class);
    conf.setOutputKeyClass(Text.class);
    conf.setOutputValueClass(IntWritable.class);
    JobClient.runJob(conf);
    return 0;
}
        public static void main(String args[]) throws Exception
        {
         int exitCode = ToolRunner.run(new WordCount(),args);
         System.exit(exitCode);
        }
}

class WordMapper extends MapReduceBase
implements Mapper<LongWritable,Text,Text,IntWritable>
{
        @Override
        public void map(LongWritable key, Text value,
                        OutputCollector<Text, IntWritable> output, Reporter r)
                        throws IOException
        {
                String s = value.toString();
                for (String word:s.split(" "))
                {
                        if(word.length()>0)
                        {
                                output.collect(new Text(word),new IntWritable(1));
```

```
                        }

                }

        }

}
class WordReducer extends MapReduceBase
implements Reducer<Text ,IntWritable,Text,IntWritable>
{
        @Override
        public void reduce(Text key, Iterator<IntWritable> values,
                        OutputCollector<Text, IntWritable> output, Reporter r)
                        throws IOException {
                int count =0;
                while(values.hasNext())
                {
                        IntWritable i = values.next();
                        count+=i.get();
                }
                output.collect(key,new IntWritable(count));

        }

        }
```

2. Create Jar file

Right click to java folder ---→ Export --→ Select Java--→ Select Jar file --→ Give jar file name (e.g. wordcount.jar) –-click finish

3. Start hadoop

4. create input file

```
cat> input.txt
 Hello everyone
 DSBDA  is Good subject
 DSBDA includes hadoop study
```

5. put this file on hadoop file system using

hadoop fs -mkdir sam

hadoop fs -put input.txt /sam/input.txt

hadoop fs -chmod 777 /sam/input.txt

6. cd workspace (wordcount.jar file is copied in workspace)

7. Run map reduce program using commanad

hadoop jar WordCount.jar WordCount /sam/input.txt WCDir

8. hadoop fs -ls /WCDir

9.Display output using

hadoop -cat WCDir/part-00000