

Department of Computer Science and Engineering

Parallel and Distributed Computing

Lab Manual

15 CS 3215

AY 2017-2018

List of Experiments

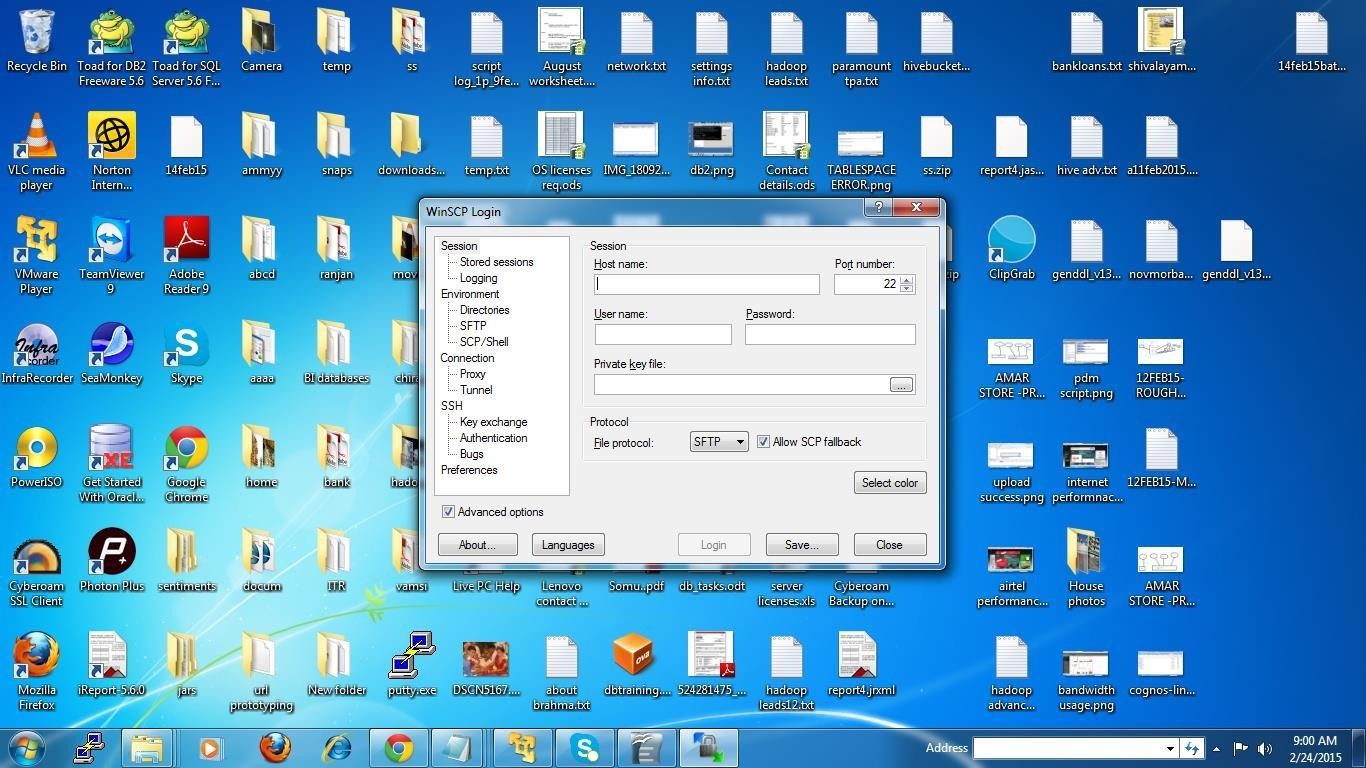
|  |  |
| --- | --- |
| **S.No** | **Program** |
|  | Installation of Hadoop and Practice Hadoop Commands |
|  | Implement Hadoop commands related to file system. |
|  | Implement Hadoop commands related to safe mode and Map Reduce. |
|  | Implement a Hadoop program to calculate sum of the n numbers using Map Reduce |
|  | Implement a Hadoop program to count number of words in a text file by Map Reduce |
|  | Implement a Hadoop program to find the max temperature in an year by Map Reduce |
|  | Implement Hadoop map reduce program to implement combiner logic for Map Reducer |
|  | Implement Partitioner usage in Hadoop Map Reduce program. |
|  | Implement wiki page count program in offline mode |
|  | Implement wiki page count program with online mode using Map Reduce. |
|  | Implement bank transaction logic to identify the failure transactions using Map Reduce. |
|  | Implement stock management with MR logic |

**Experiment No 1**

**Installation of Hadoop and Practice Hadoop Commands**

**Java Installation**

We have java software in windows machine, we have to install in linux machine. So, We have to copy software from windows to Linux. for this purpose, we are using Winscp. Winscp is a simple next, next.. finish installation in windows. after installing it will display the following screen.



Here we have to enter

Hostname :-- Linux system ip.

We can check Linux system ip by using ifconfig command in linux system

Login to your linux sysyem using the credentials

After login, you will get 2 windows. Left side is windows system right side is linux system.

Left side, browse your jdk software, right side, come down and drag and drop the jdk software.

After drag and drop, it will ask copy. Click on copy.

Open linux system terminal.

Give ls command.

You can able to see jdk software.

chmod 777 jdk......

use the above command to give permissions to that jdk file. Now if you give ls, file will display in green colour. You can use TAB button for auto completion.

./jdk....

use the above command (dot slash jdk) for installing software.

Java installation will complete.

How to check java installed or not? java -version

Where java installed?

/usr/java/jdk1.6.0\_39

**User Creation**

So far what we have done is under root user. Now we need to create one more user for Hadoop based installation. To add new use follow the below commands in terminal

useradd Hadoop

passwd Hadoop

Give user name and password any names. After completing this step logout from root user ( system → logout from root). Then login with your new user. From now on, we will do everything in our newly created user only.

# Hadoop Installation

Copy the hadoop software from windows to linux by using winscp. Now, in winscp also you have to login with your newly created username and password only.

chmod 777 had.... // for permissions

tar -xzvf hado... /// For Extracting hadoop

Environmental variables:--

.bashrc or .bash\_profile

gedit .bashrc

export JAVA\_HOME=/usr/java/jdk1.6.0\_39 export PATH=$PATH:$JAVA\_HOME/bin

export HADOOP\_HOME=/home/hdp/hadoop-1.0.3 export PATH=$PATH:$HADOOP\_HOME/bin

source .bashrc

The above command is to update the Environmental variables

echo $JAVA\_HOME echo $HADOOP\_HOME

**Hadoop Configuration**

**Navigate to the folder Hadoop-1.0.3/conf/ and edit the following files as we given below.**

**coresite.xml**

<property>

<name>hadoop.tmp.dir</name>

<value>/home/hdp/data</value>

</property>

<property>

<name>fs.default.name</name>

<value>hdfs://localhost:6789</value>

</property>

**mapredsite.xml**

<property>

<name>mapred.job.tracker</name>

<value>localhost:1234</value>

</property>

**hdfs-site.xml**

<property>

<name>dfs.replication</name>

<value>1</value>

</property>

**hadoop-env.sh**

in 8th line.... we have to specify our java home export JAVA\_HOME=/usr/java/jdk1.6.0\_39

# Formatting the namenode and starting the hadoop

hadoop namenode -format

the above command to format the namenode

start-all.sh

for starting hadoop

jps

toview the list of services. It have to display 5 services namely.

Name Node

Secondary Name Node

Job Tracker

Task Tracker

Data Node

stop-all.sh

for stopping the Hadoop services

**Test Cases.**

1. The above procedure is created for Windows as host machine and RedHat Linux as virtual machine. Try to implement the same procedure following Ubuntu as your host machine and create a virtual machine with RHEL.
   1. Must use equivalent software package for WINSCP
   2. Should create a user called Hadoop
   3. Verify the number of service running using JPS command after Hadoop installation.

Experiment 2

Basic commands related to Hadoop file system.

1.To start all the Hadoop based services

>start-all.sh

2.To clear the screen

clear

3.To display the services

JPS

4.to create a file

cat >new\_lfs

5.To display the contents

cat new\_lfs

6.To edit the contents

gedit new\_lfs

7.To display the files/dir in HDFS

>hadoop fs –ls

8.To copy a lfs file into HDFS

>hadoop fs -copyFromLocal new\_lfs new\_hdfs

9.To view the contents of the file

>hadoop fs -cat new\_hdfs

10.To copy the file from HDFS to LFS

>hadoop fs -copyToLocal new\_hdfs new\_lfs2

11.To move a local file system file to HDFS

>hadoop fs -moveFromLocal new\_lfs new\_hdfs1

12.To copy file from LFS to HDFS

>hadoop fs -put new\_lfs1 new\_hdfs2

13.To copy HDFS to LFS

>hadoop fs -get new\_hdfs2 new\_lfs3

14. To view the name node logs and Browse the FS

goto browser-->type localhost:50070--->

browseFS

namenodelogs

15. To view the status of job

goto the browser

type localhost:50030--->

In case of writing--->rep factor-->3---> 3 times the file

In case of reading--->one time--->

block size

hadoop 1.o-->64 MB

hadoop 2.o-->128 MB

file1-->200 MB-->

Hadoop 1.O-->blocks--->4 blocks

Hadoop 2.O-->blocks-->2 blocks

file1-->3 times-->

each file is 200 MB-->how many blocks in Hadoop 1.O-->

rep -->3-->600 MB--->

file1-->4 blocks--->3 rep-->3\*4--->12 blocks

file1-->2 blocks---->3 rep-->3\*2--> 6 blocks

16. safemode exception

safemode OFF

17. To verify the safemode status

>hadoop dfsadmin -safemode get

18. To enter into safemode

>hadoop dfsadmin -safemode enter

19.To leave from the safemode

>hadoop dfsadmin -safemode leave

20. To make a dir in Hadoop file system

hadoop fs -mkdir uma

21. To copy the file

hadoop fs -cp new\_hdfs123 uma

22.To view the content

hadoop fs -cat uma/new\_hdfs123

23.To list dir

hadoop fs –lsr

24. To move a file to a dir

hadoop fs -mv new\_hdfs123 uma

25. To display the blocks and bytes info

hadoop fs -du

hadoop fs –du

hadoop fs -du uma

26.To count the dirs

hadoop fs -count uma

hadoop fs -count .

27.To remove a file

hadoop fs -rm new\_hdfs1234

28. To remove a dir

hadoop fs -rmr uma

29. To change the repfactor

hadoop fs -setrep 5 new\_hdfs123

To change the repfactor to 1

hadoop fs -setrep 1 new\_hdfs123

**Test Cases.**

Try to execute the task given below with appropriate commands and note the observation.

1. Start the Hadoop filesystem and verify the services.
2. Create a file called test1.txt in the host operating system
3. Create a folder called “Example” in Hadoop filesystem and copy test1.txt
4. Display the content of the file stored in Hadoop file system
5. Rename the file test1.txt to test2.txt
6. Enter the safemode
7. Check the status of safemode
8. Delete the file test2.txt
9. Exit from the safemode
10. Delete the folder Example.
11. Stop all the services.

Experiment 3

Implement Hadoop commands related to safe mode and Map Reduce

1. Starting a single node cluster

$start-all.sh

This will startup a Namenode, Datanode, Jobtracker and a Tasktracker on your machine.

1. To start only namenode, datanode, secondary name node

$start-dfs.sh

1. To start only jobtracker,tasktracker

$start-mapred.sh

1. To stop single node cluster

$ stop-all.sh

1. To stop only namenode,datanode,secondary name node

$stop-dfs.sh

1. To stop only jobtracker,tasktracker

$stop-mapred.sh

1. To format the namenode(only once is preferable)

$hadoop namenode -format

1. To check status of namenode wheather it is in safemode or not

$hadoop dfsadmin -safemode get

1. To stop namenode temporarily

$hadoop dfsadmin -safemode enter

1. Start namenode

$hadoop dfsadmin -safemode leave

1. The FileSystem (FS) shell is invoked by

$hadoop fs <args>

All the FS shell commands take path URIs as arguments. The URI(uniform resource identifier) format is scheme://autority/path.

ForHDFS the scheme is hdfs, and for the local filesystem the scheme is file.

The scheme andauthority are optional. If not specified, the default scheme specified in the configuration is used.

Most of the commands in FS shell behave like corresponding Unix commands.

Input will be sent to stdin (standard input),Error information is sent to stderr(standard error), and the output is sent to stdout(standard output) .These are called as standard I/O streams.

All hadoop commands are invoked by the bin/hadoop script

1. Running the hadoop script without any arguments prints the description for all commands.

$hadoop --config confdir

1. To Know the list hadoop file system commands

$hadoop fs

1. o know the list of directories in hadoop file system

$hadoop fs -ls

1. To list all the files in particular dir

$hadoop fs -ls <dir name>

1. To know the list of dir's and also file which present in dir's. Recursive version of ls. Similar to Unix ls -R

$hadoop fs -lsr

$hadoop fs -lsr <dir name>

1. The du command displays aggregate length of files contained in the directory or the length of a file in case its just a file.

$hadoop fs -du

1. The hadoop dus command prints the summary of file lengths

$hadoop fs -dus

1. To count and displays the number of directories, files and bytes under the paths that match the specified file pattern. The output columns are:DIR\_COUNT, FILE\_COUNT, CONTENT\_SIZE FILE\_NAME.

$hadoop fs -count <dir name>

1. The output columns with -q are:QUOTA, REMAINING\_QUATA, SPACE\_QUOTA, REMAINING\_SPACE\_QUOTA, DIR\_COUNT, FILE\_COUNT, CONTENT\_SIZE, FILE\_NAME.

$hadoop fs -count -q <dir name>

1. To Move permanently the file from local hard drive to hadoop dir.src is deleted after it's copied

$hadoop fs -moveFromLocal <src> <dst>

1. To move files from source to destination. This command allows multiple sources as well in which case the destination needs to be a directory. Moving files across filesystems is not permitted.

$hadoop fs -mv <src> <dst>

1. To copy the file from local hard drive to hadoop dir. Put and -copyFromLocal – These commands are used to put files from local file system to the destination file system. The difference is that put allows reading from stdin while copyFromLocal allows only local file reference as a source.

$hadoop fs -copyFromLocal <sour path> <dest path>

$hadoop fs -put <sour> <dest>

1. To copy the file from one hadoop dir to another hadoop dir

$hadoop fs -cp <sour path> <dest path>

$hadoop fs -cp /firstdir/abc.txt secondir

1. Delete files specified as args. Only deletes non empty directory and files. Refer to rmr for recursive deletes

$hadoop fs -rm <particular path>

1. Hadoop fs -rmr:Recursively deletes the files and sub directories.

$hadoop fs -rmr <dir name>

1. Hadoop fs expunge:Used to empty the trash.

$hadoop fs -expunge

1. Get and -copyToLocal – These commands are used to copy files from HDFS to the local file system.

$hadoop fs -get <surc path> <dest path>

$hadoop fs -copyToLocal <surc> <dest>

1. getmerge-Retrieves all files identified by SRC, merges them, and writes the single merged file to LOCALDST in the local filesystem. The option addnl will add a newline character to the end of each file.

$hadoop fs –getmerge SRC [SRC ...] LOCALDST [addnl]

$hadoop fs -getmerge /user/ubuntu/uni1 /home/ubuntu/ addnl

(in this uni1 is dir in which some file are present (like file1.txt,file2.txt) all are merged into single file and copies to localdir)

1. Displays the file content.

$hadoop fs -cat <file path>

$hadoop fs -cat /user/hadoop/file4.txt

1. Displays the textual content of files. Identical to catif files are text files. Files in known compressed format (gzip and Hadoop’s binary sequence file format) are uncompressed first.

$hadoop fs -text <file path>

$hadoop fs -text /user/ubuntu/uni2/Sampledata.tar.gz

Output:

Warning: $HADOOP\_HOME is deprecated.

file30000600000175000017500000000005412341043561011417 0ustar ubuntuubuntuthe quick brown fox jumps over the lazy dog

file20000600000175000017500000000003412341043544011415 0ustar ubuntuubuntuHello Hadoop Goodbye Hadoop

file10000600000175000017500000000002612341043532011412 0ustar ubuntuubuntuHello World Bye World

1. moveToLocal-Displays a “not implemented yet” message.

$hadoop fs –moveToLocal <src> <dest>

1. mkdir – This command can be used to create a directory in HDFS.//mkdir is particularly useful when you have multiple users on your hadoop system. It really helps to have separate user directories on HDFS the same way on a Unix system. So remember to create HDFS directories for your Unix users who need to access Hadoop as well.

$hadoop fs -mkdir <dir name>

1. setrep-Sets the target replication factor to REPfor given files. The -R option will recursively apply the target replication factor to files in directories identified by PATH. The replication factor will take some time to get to the target. The -w option will wait for the replication factor to match the target.

$hadoop fs –setrep [-R] [-w] REP PATH [PATH ...]

$hadoop fs -setrep -w 3 -R /user/ubuntu/uni1

1. touchz-Create a file of zero length.Fails if files already exist and have nonzero length.

$hadoop fs –touchz FILEname

1. Test-Performs one of the following type checks on PATH:

-e PATH existence. Returns 0 if PATH exists.

-z Empty file. Returns 0 if file length is 0.

-d Returns 0 if PATH is a directory.

$hadoop fs -test -e filename

1. Displays “statistical” information on files. The FORMAT string is printed exactly but with the following format specifiers replaced. %b Size of file in blocks %F The string “directory” or “regular file” depending on file type %n Filename %o Block size %r Replication %y UTC date in yyyy-MM-dd HH:mm:ss format %Y Milliseconds since January 1, 1970 UTC

$hadoop fs -stat <path>

1. Tail-Displays last kilobyte of the file to stdout. -f option can be used as in Unix.

$hadoop fs -tail [-f] filepath

1. chgrp-Changes the ownership of files and directories. The -Roption applies the change recursively. The user must be the files owner or a superuser Additional information is in the HDFS Admin Guide: Permissions.

$hadoop fs -chgrp [-R] GROUP URI [URI …]

1. chmod-Changes the permissions of files and directories.

Similar to its Unix equivalent, MODE can be a 3-digit octal mode, or {augo}+/-{rwxX}.

The -R option applies the change recursively. The user must be the files’ owner or a superuser.

$hadoop fs –chmod [-R] MODE[,MODE ...] PATH [PATH ...]

1. chown-Changes the ownership of files and directories. The -Roption applies the change recursively. The user must be a superuser.

$hadoop fs –chown [-R] [OWNER][:[GROUP]] PATH [PATH...]

1. Displays usage information for the command CMD.

If CMD is omitted, it displays usage information for all commands.

$hadoop fs –help [CMD]

1. To check to get block information

$hadoop fsck -block

1. To check file system checkup/health check

$hadoop fsck -report

1. To empty the trash

$ hadoop fs -expunge

1. To see how much space this directory occupies in HDFS

$hadoop fs -du -s -h /user/ajay/retail

**Experiment 4**

**Implement a Hadoop program to calculate sum of the n numbers using Map Reduce**

**Sum\_Mapper.java**

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.mapred.MapReduceBase;

import org.apache.hadoop.mapred.Mapper;

import org.apache.hadoop.mapred.OutputCollector;

import org.apache.hadoop.mapred.Reporter;

public class sum\_mapper extends MapReduceBase im plements Mapper<LongWritable,Text,Text,IntWritable>

{

public void map(LongWritable key, Text value,OutputCollector<Text,IntWritable> output,

Reporter reporter) throws IOException{

String line = value.toString();

StringTokenizer tokenizer = new StringTokenizer(line);

while (tokenizer.hasMoreTokens()){

String s=tokenizer.nextToken("+");

int p=Integer.parseInt(s);

output.collect(value,new IntWritable(p));

}

}

}

**Sum\_ reducer.java**

import java.io.IOException;

import java.util.Iterator;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapred.MapReduceBase;

import org.apache.hadoop.mapred.OutputCollector;

import org.apache.hadoop.mapred.Reducer;

import org.apache.hadoop.mapred.Reporter;

public class sum\_reducer extends MapReduceBase implements Reducer<Text,IntWritable,Text,IntWritable> {

public v oid reduce(Text key, Iterator<IntWritable> values,OutputCollector<Text,IntWritable> output,

Reporter reporter) throws IOException {

int sum=0;

while (values.hasNext()) {

sum+=values.next().get();

}

output.collect(key,new IntWri table(sum)); }}

**Sum\_driver .java**

import java.io.IOException;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

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.TextInputFormat;

import org.apache.hadoop.mapred.TextOutputFormat ;

public class sum\_runner {

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

JobConf conf = new JobConf(sum\_runner.class);

conf.setJobName("Sumofthedigits");

conf.setOutputKeyClass(Text.class);

conf.setOutputValueClass(IntWritable.class);

conf.setMapperClass(sum\_mapper.class);

conf.setReducerClass(sum\_reducer.class);

conf.setInputFormat(TextInputFormat.class);

conf.setOutputFormat(TextOut putFormat.class);

FileInputFormat.setInputPaths(conf,new Path(args[0]));

FileOutputFormat.setOutputPath(conf,new Path(args[1]));

JobClient.runJob(conf);

}

}

**Test Cases**

Implement a Hadoop map reduce program to find

* 1. The smallest among n numbers
  2. The largest among n numbers
  3. The product of n numbers
  4. Find the multiples of 5 from the input file

**Viva Questions**

What is the difference between Hadoop and Map Reduce?

What is the use of mapper?

What is the use of reducer?

What is the use of runner?

What is string tokenizer?

Explain the procedure to run a Hadoop map reduce program?

What is output collector?

What does ‘jps’ command do?

How to restart Namenode?

Which are the three modes in which Hadoop can be run?

**Experiment -5**

**Aim:**

**Implement Word Count Program using MapReduce.**

WordCountMapper:--

package com.WordCount;

import java.io.IOException;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Mapper.Context;

public class WordCountMapper extends Mapper<LongWritable, Text,Text, LongWritable>

{

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

{

output.write(value, new LongWritable(1));

}

}

WordCountReducer:--

package com.WordCount;

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.Reducer;

import org.apache.hadoop.mapreduce.Reducer.Context;

public class WordCountReducer extends Reducer<Text, LongWritable, Text, IntWritable>

{

public void reduce(Text key, Iterable<LongWritable> value, Context output) throws IOException, InterruptedException

{

int sum=0;

for(LongWritable val:value){

sum +=val.get();

}

output.write(key, new IntWritable(sum));

}

}

WordCountdriver:--

package com.WordCount;

import org.apache.hadoop.conf.Configuration;

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.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Reducer;

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.util.Tool;

import java.io.IOException;

public class WordCountDriver{

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

Job job=new Job(new Configuration());

job.setJobName("WC");

job.setJarByClass(WordCountDriver.class);

job.setInputFormatClass(TextInputFormat.class);

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(LongWritable.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

job.setMapperClass(WordCountMapper.class);

job.setReducerClass(WordCountReducer.class);

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

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

job.waitForCompletion(true);

}

}

Test Cases:

1. Create a file in LFS with repeated words in LFS?
2. Load the file to HDFS and Run the same program and explain your observations?
3. Create a file with special characters, alphabets and numbers and load the same to HDFS?
4. Run the code with the above input and what is your observation?

Viva Questions:

1. What is the use of reducer?

2. How driver class is helpful in Map reduce?

3. What is Hadoop API?

4. What is build path?

**Experiment -6**

**Implement a hadoop program to find the max temperature in an year by mapreduce**

**TempMR2.java**

import java.io.IOException;

import java.util.Iterator;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapred.MapReduceBase;

import org.apache.hadoop.mapred.OutputCollector;

import org.apache.hadoop.mapred.Reducer;

import org.apache.hadoop.mapred.Reporter;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.mapred.Mapper;

import org.apache.hadoop.fs.Path;

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.TextInputFormat;

import org.apache.hadoop.mapred.TextOutputFormat;

public class TempMR2 {

public static class TempMap extends MapReduceBase implements Mapper<LongWritable,Text,Text,IntWritable> {

public void map(LongWritable key, Text value,OutputCollector<Text,IntWritable> output,

Reporter reporter) throws IOException {

String record = value.toString();

String[] parts = record.split(",");

output.collect(new Text(parts[0]), new IntWritable(Integer.parseInt(parts[1])));

}

}

public static class TempReduce extends MapReduceBase implements Reducer<Text,IntWritable,Text,IntWritable> {

public void reduce(Text key, Iterator<IntWritable> values,OutputCollector<Text,IntWritable> output,

Reporter reporter) throws IOException {

int maxValue = 0;

//Looping and calculating Max for each year

while (values.hasNext()) {

maxValue = Math.max(maxValue, values.next().get());

}

output.collect(key, new IntWritable(maxValue));

}

}

public static void main(String[] args) throws Exception {

JobConf job = new JobConf(TempMR2.class);

job.setJarByClass(TempMR2.class);

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(IntWritable.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

job.setMapperClass(TempMap.class);

job.setReducerClass(TempReduce.class);

job.setInputFormat(TextInputFormat.class);

job.setOutputFormat(TextOutputFormat.class);

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

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

JobClient.runJob(job);

}

}

**Test Cases:**

**1. Find out the minimum temperature from the given input?**

**2. Create the different jar file to run the code?**

**3. Use the same jar file to find out the minimum temperature.**

**Viva Questions:**

1. **Describe various input formats available?**
2. **List out the Hadoop jar files required to run map reduce code?**
3. **What is the function of Job Client.**
4. **List out the functionalities of IntWritable?**

**Experiment -7**

**Aim:**

**Implement Word Count Program using MapReduce with Combiner Logic hence explain the benefits of Combiner.**

package com.sample;

import java.io.IOException;

import java.util.Iterator;

import java.util.StringTokenizer;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

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.mapred.TextInputFormat;

import org.apache.hadoop.mapred.TextOutputFormat;

public class WordCountWithCombiner {

public static class Map extends MapReduceBase implements 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, OutputCollector<Text, IntWritable> output, Reporter reporter)

throws IOException {

String line = value.toString();

StringTokenizer tokenizer = new StringTokenizer(line);

while (tokenizer.hasMoreTokens()) {

word.set(tokenizer.nextToken());

output.collect(word, one);

}

}

}

public static class Reduce extends MapReduceBase implements Reducer<Text, IntWritable, Text, IntWritable> {

public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text, IntWritable> output,

Reporter reporter) throws IOException {

int sum = 0;

while (values.hasNext()) {

sum += values.next().get();

}

output.collect(key, new IntWritable(sum));

}

}

public static void main(String[] args) throws Exception {

JobConf conf = new JobConf(WordCountWithCombiner.class);

conf.setJobName("WordCountWithCombiner");

conf.setOutputKeyClass(Text.class);

conf.setOutputValueClass(IntWritable.class);

conf.setMapperClass(Map.class);

conf.setCombinerClass(Reduce.class);

conf.setReducerClass(Reduce.class);

conf.setInputFormat(TextInputFormat.class);

conf.setOutputFormat(TextOutputFormat.class);

FileInputFormat.setInputPaths(conf, new Path(args[0]));

FileOutputFormat.setOutputPath(conf, new Path(args[1]));

JobClient.runJob(conf);

}

}

**Test Cases:**

1. Create a file as input in LFS without duplicate data?

2. Load the same and run the program?

3. Observe the time taken to run the program without duplicates?

4. Observe the time taken to run the program with duplicates?

5. Run the same program with duplicates but without combiner and Justify?

**Viva Questions:**

1. What is use of combiner?

2. What if combiner is used for the input without duplicates in the input?

3. How combiner reduces the cpu time while working with duplicate data?

4. How combiner usage in Map Reduce is different with normal Map reduce?

**Experiment -8**

Implement Partitioner usage in Hadoop map reduce program.

package com.sample;

import java.io.IOException;

import java.util.Iterator;

import java.util.StringTokenizer;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

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.Partitioner;

import org.apache.hadoop.mapred.Reducer;

import org.apache.hadoop.mapred.Reporter;

import org.apache.hadoop.mapred.TextInputFormat;

import org.apache.hadoop.mapred.TextOutputFormat;

public class WordCountWithPartitioner {

// Output types of Mapper should be same as arguments of Partitioner

public static class MyPartitioner implements Partitioner<Text, IntWritable> {

public int getPartition(Text key, IntWritable value, int numPartitions) {

//key - Deer

String myKey = key.toString().toLowerCase();

//myKey = deer

int c = myKey.charAt(0);

int whichReducerToPass = c % numPartitions;

return whichReducerToPass;

/\*

\* String myKey = key.toString().toLowerCase();

\*

\* if (myKey.startsWith("s") || myKey.startsWith("S")) { return 0; }

\* if (myKey.startsWith("m") || myKey.startsWith("M") ) { return 1;

\* } else { return 2; }

\*/

}

public void configure(JobConf arg0) { // Gives you a new instance of JobConf if you want to change Job

// Configurations

}

}

public static class Map extends MapReduceBase implements 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, OutputCollector<Text, IntWritable> output, Reporter reporter)

throws IOException {

String line = value.toString();

StringTokenizer tokenizer = new StringTokenizer(line);

while (tokenizer.hasMoreTokens()) {

word.set(tokenizer.nextToken());

output.collect(word, one);

}

}

}

public static class Reduce extends MapReduceBase implements Reducer<Text, IntWritable, Text, IntWritable> {

public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text, IntWritable> output,

Reporter reporter) throws IOException {

int sum = 0;

while (values.hasNext()) {

sum += values.next().get();

}

output.collect(key, new IntWritable(sum));

}

}

public static void main(String[] args) throws Exception {

JobConf conf = new JobConf(WordCountWithPartitioner.class);

conf.setJobName("WordCountWithPartitioner");

conf.setOutputKeyClass(Text.class);

conf.setOutputValueClass(IntWritable.class);

conf.setNumReduceTasks(6);

conf.setMapperClass(Map.class);

conf.setCombinerClass(Reduce.class);

conf.setReducerClass(Reduce.class);

conf.setPartitionerClass(MyPartitioner.class);

conf.setInputFormat(TextInputFormat.class); conf.setOutputFormat(TextOutputFormat.class);

FileInputFormat.setInputPaths(conf, new Path(args[0]));

FileOutputFormat.setOutputPath(conf, new Path(args[1]));

JobClient.runJob(conf);

}

}

**Test Cases:**

1. Change the data set in HDFS and run the code?

2. Copy the data set from the local file system to HDFS?

3. Change the Input format and justify your observation?

4. Implement the code by creating 3 different classes?

5.Create a file with 3 different kinds of data like alphabets, special characters and numbers?

**Viva Questions:**

1. What is Map reduce, in what way it achieves parallel and distributed processing?

2. Describe various Input Formats supported by Hadoop?

3. Explain the concept of block size in memory allocation of Hadoop?

4. What is Functionality of Mapper Class?

5.What do you mean by job configuration?

Experiment:9

Implement wiki page count program in offline mode

WikiMapper:--

package com.wikidatamining;

import java.io.IOException;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Mapper.Context;

public class WikiMapper extends Mapper<LongWritable, Text,Text, LongWritable>

{

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

{

String Line= value.toString(); //for storing into string

String a[]=Line.split(" "); // for spli based on space

if(a[0].equalsIgnoreCase("en")){

output.write(new Text(a[1]), new LongWritable(Long.parseLong(a[2])));

}

}

}

WikiReducer:--

package com.wikidatamining;

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.Reducer;

import org.apache.hadoop.mapreduce.Reducer.Context;

public class WikiReducer extends Reducer<Text, LongWritable, Text, IntWritable>

{

public void reduce(Text key, Iterable<LongWritable> value, Context output) throws IOException, InterruptedException

{

int sum=0;

for(LongWritable val:value){

sum +=val.get();

}

output.write(key, new IntWritable(sum));

}

}

Wiki driver:--

package com.wikidatamining;

import org.apache.hadoop.conf.Configuration;

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.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Reducer;

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.util.Tool;

import java.io.IOException;

public class WikiDriver{

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

Job job=new Job(new Configuration());

job.setJobName("Wiki datamining");

job.setJarByClass(WikiDriver.class);

job.setInputFormatClass(TextInputFormat.class);

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(LongWritable.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

job.setMapperClass(WikiMapper.class);

job.setReducerClass(WikiReducer.class);

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

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

job.waitForCompletion(true);

}

}

Test cases:

1. Create different search engine clicks and apply the logic?

2. Generate the output of the search engines where clicks are >40K

Viva Questions:

1. What is page counts data refers?

2. How map reduce is suitable to click streams count?

Experiment: 10

Implement wiki page count program with online mode

WikiMapper

package com.Wiki2;

import java.io.IOException;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class WikiMapper extends

Mapper<LongWritable, Text, Text, LongWritable> {

public void map(LongWritable key, Text value, Context output)

throws IOException, InterruptedException {

String line = value.toString();

String a[] = line.split(" ");

// wiki program only english

if(a[0].equalsIgnoreCase("en")){

output.write(new Text(a[1]), new LongWritable(Long.parseLong(a[2])));

}

// wiki count only 40000

if(Long.parseLong(a[3])>40000){

output.write(new Text(a[1]), new LongWritable(1));

}

}

}

WikiReducer

package com.Wiki2;

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.Reducer;

public class WikiReducer extends

Reducer<Text, LongWritable, Text, IntWritable> {

public void reduce(Text key, Iterable<LongWritable> value, Context output)

throws IOException, InterruptedException {

int sum = 0;

for (LongWritable val : value) {

sum += val.get();

}

output.write(key, new IntWritable(sum));

}

}

WikiDriver

package com.Wiki2;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

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;

public class WikiDriver{

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

Job job=new Job(new Configuration());

job.setJobName("WIKI");

job.setJarByClass(WikiDriver.class);

job.setInputFormatClass(TextInputFormat.class);

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(LongWritable.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

job.setMapperClass(WikiMapper.class);

job.setReducerClass(WikiReducer.class);

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

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

job.waitForCompletion(true);

}

}

Test cases:

1. Download another data set from wiki page counts web site?

2. Apply the logic of wiki page counts in online mode with the data?

3. Show the logical difference between offline and online mode of wiki page counts?

Experiment: 11

Implement bank transaction logic to identify the failure transactions.

package com.FailedTransaction;

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;

import org.apache.hadoop.mapreduce.Mapper.Context;

public class FailedTransactionMapper extends Mapper<LongWritable, Text, Text,Text>

{

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

{

String Line = value.toString();

String BankData[] = Line.split(" ");

String DateTime[] = BankData[3].split("-");

String Date[] = DateTime[0].split("/");

if(BankData[6].equalsIgnoreCase("error")){

//output.write(new Text(BankData[6].toString()), new LongWritable(Integer.parseInt(BankData[4]+" "+BankData[1]+" "+BankData[5]+" "+BankData[3])));

output.write(new Text(BankData[6].toString()), new Text((BankData[4]+" "+BankData[1]+" "+BankData[5]+" "+BankData[3]).toString()));

}

}

}

package com.FailedTransaction;

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.Reducer;

import org.apache.hadoop.mapreduce.Reducer.Context;

public class FailedTransactionReducer extends Reducer<Text, Text, Text, Text>

{

public void reduce(Text key, Iterable<Text> value, Context output) throws IOException, InterruptedException

{

/\*for (LongWritable val:value)

{

if(val.get()>max){

max=(int) val.get();

}

}

output.write(key, new LongWritable(max));

\*/

for(Text val:value){

output.write(null,new Text(val));

}

}

}

package com.FailedTransaction;

import org.apache.hadoop.conf.Configuration;

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.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Reducer;

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.util.Tool;

import java.io.IOException;

public class FailedTransactionDriver{

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

Job job=new Job(new Configuration());

job.setJobName("Bank Failed Transaction");

job.setJarByClass(FailedTransactionDriver.class);

job.setInputFormatClass(TextInputFormat.class);

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(Text.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(Text.class);

job.setMapperClass(FailedTransactionMapper.class);

job.setReducerClass(FailedTransactionReducer.class);

//job.setCombinerClass(VoterMapper.class);

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

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

job.waitForCompletion(true);

}

}

Viva Questions:

1. What is Shuffling and Sorting in MapReduce?
2. What main configuration parameters are specified in MapReduce?
3. What is JobTracker?
4. Explain job scheduling through JobTracker.
5. How to set mappers and reducers for Hadoop jobs?

Experiment:12

Implement stock management with MR logic

package com.hirw.maxcloseprice;

/\*\*

\* MaxClosePriceMapper.java

\* \* This is a Mapper program to calculate Max Close Price from stock dataset using MapReduce

\*/

import java.io.IOException;

import org.apache.hadoop.io.FloatWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class MaxClosePriceMapper extends Mapper<LongWritable, Text, Text, FloatWritable> {

@Override

public void map(LongWritable key, Text value, Context context)

throws IOException, InterruptedException {

String line = value.toString();

String[] items = line.split(",");

String stock = items[1];

Float closePrice = Float.parseFloat(items[6]);

context.write(new Text(stock), new FloatWritable(closePrice));

}

}

package com.hirw.maxcloseprice;

/\*\* \* MaxClosePriceReducer.java

\* www.hadoopinrealworld.com

\* This is a Reduce program to calculate Max Close Price from stock dataset using MapReduce

\*/

import java.io.IOException;

import org.apache.hadoop.io.FloatWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

public class MaxClosePriceReducer

extends Reducer<Text, FloatWritable, Text, FloatWritable> {

@Override

public void reduce(Text key, Iterable<FloatWritable> values, Context context)

throws IOException, InterruptedException {

float maxClosePrice = Float.MIN\_VALUE;

//Iterate all temperatures for a year and calculate maximum

for (FloatWritable value : values) {

maxClosePrice = Math.max(maxClosePrice, value.get());

}

//Write output

context.write(key, new FloatWritable(maxClosePrice));

}

}

/\*\*

\* MaxClosePrice.java

\* This is a driver program to calculate Max Close Price from stock dataset using MapReduce

\*/

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.FloatWritable;

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 MaxClosePrice {

public static void main(String[] args) throws Exception {

if (args.length != 2) {

System.err.println("Usage: MaxClosePrice <input path> <output path>");

System.exit(-1);

}

//Define MapReduce job

Job job = new Job();

job.setJarByClass(MaxClosePrice.class);

job.setJobName("MaxClosePrice");

//Set input and output locations

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

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

//Set Input and Output formats

job.setInputFormatClass(TextInputFormat.class);

job.setOutputFormatClass(TextOutputFormat.class);

//Set Mapper and Reduce classes

job.setMapperClass(MaxClosePriceMapper.class);

job.setReducerClass(MaxClosePriceReducer.class);

//Output types

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(FloatWritable.class);

//Submit job

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

}

}

Viva Questions:

1. What are the parameters of mappers and reducers?
2. What platform and Java version is required to run Hadoop?
3. **Compare RDBMS with Hadoop Map Reduce.**
4. **Explain the differences between a combiner and reducer.**
5. **Explain what is Speculative Execution?**