Assignment 5 – PageRank Algorithm

PageRank: PageRank is a web graph ranking algorithm that helps Internet users sort hits by their importance.

PageRank calculates a numerical value for each element of a hyperlinked set of web pages, which reflects the probability that a random surfer will access that page.

PageRank is a recursive join plus update of rank values plus aggregation of ranks.

Formula for PageRank:

$$PR(p_i) = \frac{1-\alpha}{N} + \alpha \sum_{p_j \text{ links to } p_i} \frac{PR(p_j)}{L(p_j)}$$

MapReduce:

Steps:-

Mapper: The master nodes takes the input, divide it into smaller sub-problems and distributes them to worker nodes.

```java
public class FinMapper extends Mapper<LongWritable, Text, DoubleWritable, Text> {

	public void map(LongWritable key, Text value, Context context)
			throws IOException, InterruptedException,
			IllegalArgumentException {
		String line = value.toString(); // Converts Line to a String
		/*
		 * TODO output key:-rank, value: node
		 * See IterMapper for hints on parsing the output of IterReducer.
		 */

		String[] sections = line.split("\t");
		String[] parts = sections[0].split(";");
		Double rank = Double.parseDouble(parts[1]);
		context.write(new DoubleWritable(-rank), new Text(parts[0]));

	}
}
```

Reducer: Reducer reduces a set of intermediate values which share a key to a smaller set of values.

```java
public class FinReducer extends Reducer<DoubleWritable, Text, Text, Text> {

	public void reduce(DoubleWritable key, Iterable<Text> values, Context context)
throws IOException,
			InterruptedException {
		/*
		 * TODO: For each value, emit: key:value, value:-rank
		 */
```

```
            Iterator<Text> iterator = values.iterator();
            String node;
            while(iterator.hasNext()) {
            node = iterator.next().toString();
            context.write(new Text(node), new Text(String.valueOf(0 - key.get())));
            }
      }
}
```

InitMapper:

```
public class InitMapper extends Mapper<LongWritable, Text, Text, Text> {

      public void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException,
                  IllegalArgumentException {
            String line = value.toString(); // Converts Line to a String

            /*
             * TODO: Just echo the input, since it is already in adjacency list
format.
             */

            String[] pair = line.split(":");
            if(pair != null && pair.length == 2) {
            context.write(new Text(pair[0].trim()), new Text(pair[1]));
            }
      }
}
```

InitReducer:

```
public class InitReducer extends Reducer<Text, Text, Text, Text> {

      public void reduce(Text key, Iterable<Text> values, Context context) throws
IOException, InterruptedException {
            /*
             * TODO: Output key: node+rank, value: adjacency list
             */
            int defualtrank = 1;
            Iterator<Text> v = values.iterator();
            while(v.hasNext()) {
            context.write(new Text(key + "+" + defualtrank), v.next());
            }
      }
}
```

Iterator Mapper:

```
public class IterMapper extends Mapper<LongWritable, Text, Text, Text> {

      public void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException,
                  IllegalArgumentException {
            String line = value.toString();
            String[] sections = line.split("\t");
```

```
		if (sections.length > 2)		{
			throw new IOException("Incorrect data format");
		}
		if (sections.length != 2) {
			return;
		}
		/*
		 * TODO: emit key: adj vertex, value: computed weight.
		 *
		 * Remember to also emit the input adjacency list for this node!
		 * Put a marker on the string value to indicate it is an adjacency list.
		 */
		String[] noderank = sections[0].split("\\+");
		String n = String.valueOf(noderank[0]);

		double rank = Double.valueOf(noderank[1]);
		String alist = sections[1].toString().trim();
		String[] anodes = alist.split(" ");

		int N = anodes.length;
		double weightOfPage = (double)1/N * rank;

		for(String ajacentnode : anodes) {
			context.write(new Text(ajacentnode), new
Text(String.valueOf(weightOfPage)));
		}
		context.write(new Text(n), new Text(PageRankDriver.MARKER +
sections[1]));
	}
}
```

Iterator Reducer:

```
public class IterReducer extends Reducer<Text, Text, Text, Text> {

	public void reduce(Text key, Iterable<Text> values, Context context) throws
IOException, InterruptedException {
		//double d = PageRankDriver.DECAY; // Decay factor
		/*
		 * TODO: emit key:node+rank, value: adjacency list
		 * Use PageRank algorithm to compute rank from weights contributed by
incoming edges.
		 * Remember that one of the values will be marked as the adjacency list
for the node.
		 */

		int defualtrank = 1;
		Iterator<Text> v = values.iterator();
		while(v.hasNext()) {
		context.write(new Text(key + "+" + defualtrank), v.next());
		}
	}
}
```

Shuffle: Input to the Reducer is the sorted output of the mappers. In this phase the framework fetches the relevant partition of the output of all the mappers, via http.

Sort: The framework groups Reducer inputs by keys in this stage.

Partitioner: Partitioner controls the partitioning of the keys of the intermediate map-outputs.

Composite Function:

```
public static void composite(String input, String output, String interim1,
                    String interim2, String diff, int reducers) throws Exception {
             /*
              * TODO
              */
             System.out.println("Keyur Doshi(10405923)");

             int counter = 0;
             init(input, interim1, reducers); // Initializes data
             counter++;
             double difference = 100000000; // sets ridiculously large default
difference
             int i = 0; // counter variables
             while (difference >= 30) // continues operation till difference is of
specified value
             {
                    if (i % 2 == 0) {
                           iter(interim1, interim2, reducers);
                           /*
                            * or every even iteration, interim1 is the input and
interim2 the output
                            */
                    } else {
                           iter(interim2, interim1, reducers);
                           // For odd iterations it is swapped
                    }
                    if (i % 3 == 0) // for every 3 iterations calculates the
difference
                    {
                           counter++;
                           diff(interim1, interim2, "diffout", reducers);
                           /*
                            * as difference is an absolute value, order of input1,
input2 directories doesn't matter
                            */
                           difference = readDiffResult("diffout"); // Reads
difference from "diffout"
                           System.out.println("Difference updates to:" + difference);
                           deleteDirectory("diffout"); // deletes diffout
                    }

                    if (i % 2 == 0) // on even iterations, delete input directory
interim1
                    {
                           deleteDirectory(interim1);
                    }
```

```
                if (i % 2 == 1) // on odd, delete interim2
                {
                        deleteDirectory(interim2);
                }
                counter++;
                i++; // increment i
        }

        if (i % 2 == 1) // As i increments at the last step, for odd i, interim2
                                // is the input directory
        {
                deleteDirectory(interim1); // deletes other directory
                counter++;
                finish(interim2, output, reducers);
                summarizeResult(output);
        } else
        {
                deleteDirectory(interim2); // Deletes other directory
                counter++;
                finish(interim1, output, reducers);
                summarizeResult(output);
        }
        System.out.println();
        System.out.println(counter);

    }
```

Here are the steps to setup standalone Hadoop cluster.

```
Install Java 8 :
sudo add-apt-repository ppa:webupd8team/java
Run Update : sudo apt-get update
sudo apt-get install oracle-java8-installer

Check java version:
java -version

Creating a Hadoop User for accessing HDFS and Mapreduce: (To avoid
security issues, it is recommend to setup new Hadoop user group)
sudo addgroup hadoop
sudo adduser --ingroup hadoop huser
Enter password : - Create password to access huser
optional ignore
press Y

Minimize terminal and open new terminal:

Go to root directly: sudo su root
cd (change directory)

Giving correctly and securely root priviliges to hadoop user "hdhuser" by
editing the /etc/sudoers file:
sudo gedit /etc/sudoers
```

Under user priviliges

```
huser ALL=(ALL:ALL) ALL
```

Install as well as configure SSH

```
sudo apt-get install openssh-server
```

Hadoop uses SSH to access its nodes. For single node setup of Hadoop, we therefore need to configure SSH access to localhost.

Login with hyser

```
sudo su hsuer
cd
ssh-keygen -t rsa -P ** -- Generate SSH key for huser account
ssh-keygen -t rsa -P ""
```

[If asked for a filename just leave it blank and press the enter key to continue]

Add the newly created key to the list of authorized keys tso that hadoop can use SSH without prompting for a password.

```
cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys
```

Since hadoop doesn't work on IPv6 we should disable it. for that you need to update /etc/sysctl.conf

```
# disable ipv6
net.ipv6.conf.all.disable_ipv6 = 1
net.ipv6.conf.default.disable_ipv6 = 1
net.ipv6.conf.lo.disable_ipv6 = 1
```

Command to reboot machine:

```
sudo reboot
```

Download Hadoop hadoop-2.7.0.tar.gz
Copy to desktop and extract

Terminal:

```
sudo su huser
cd
```

since the huser has root priviliges, we can move the hadoop installation to the /usr/local/hadoop directory without any problem.

```
sudo mv '/home/keyur/Desktop/hadoop-2.7.3' /usr/local/hadoop
```

Assign ownership of this folder to Hadoop user:

```
sudo chown huser:hadoop -R /usr/local/hadoop
```

Creat hadoop temp directories for Namenode and Datanode

```
sudo mkdir -p /usr/local/hadoop_tmp/hdfs/namenode
sudo mkdir -p /usr/local/hadoop_tmp/hdfs/datanode
```

Again asign ownership of this hadoop temp folder to hadoopr user

```
sudo chown huser:hadoop -R /usr/local/hadoop_tmp
sudo gedit .bashrc
```

Add the following lines at the end of the file bashrc file

```
# -- HADOOP ENVIRONMENT VARIABLES START -- #
export JAVA_HOME=/usr/lib/jvm/java-8-oracle
export HADOOP_HOME=/usr/local/hadoop
export PATH=$PATH:$HADOOP_HOME/bin
export PATH=$PATH:$HADOOP_HOME/sbin
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib"
export PATH=$PATH:/usr/local/hadoop/bin/
# -- HADOOP ENVIRONMENT VARIABLES END -- #
```

Now we going to configure hadoop-env.sh - This file contains some environment variable settings used by Hadoop such as where log files are stored, the maximum amount of heap used etc.

```
cd /usr/local/hadoop/etc/hadoop
sudo gedit hadoop-env.sh
```

Comment export JAVA_HOME = ${JAVA_HOME}

Add or change the vairable JAVA_HOME which specifies the path to the Java Installation used by hadoop
```
export JAVA_HOME='/usr/lib/jvm/java-8-oracle'
```

Configuration of core-site.xml file: core-site.xml contains configuration information that overrides the default values for core HAdoop properties.
```
sudo gedit core-site.xml
```

```
## Name node runs on localhost on port 9000
<property>
<name>fs.default.name</name>
<value>hdfs://localhost:9000</value>
</property>
```

Configuration of hdfs-site.xml file: The hdfs-site.xml file enables you to overwrite a number of default values that control the HDFS
```
sudo gedit hdfs-site.xml
```

```
<property>
<name>dfs.replication</name>
<value>1</value>
</property>
<property>
<name>dfs.namenode.name.dir</name>
<value>file:/usr/local/hadoop_tmp/hdfs/namenode</value>
</property>
<property>
```

```
<name>dfs.datanode.data.dir</name>
<value>file:/usr/local/hadoop_tmp/hdfs/datanode</value>
</property>
```

*dfs.replication - The default replication factor to use for each block of a file.
*dfs.namenode.name.dir - The namenode stores the metadata of the file system in the local file system in the speified directory.
*dfs.datanode.name.dir - The datanode stores the actual blocks of file data in the local fule system in the specified directory.

Configuration of yarn-site.xml: yarn-site.xml file enables you to overwrite a number of default values controlling the YARN components.

```
sudo gedit yarn-site.xml
```

```
<property>
<name>yarn.nodemanager.aux-services</name>
<value>mapreduce_shuffle</value>
</property>
<property>
<name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
<value>org.apache.hadoop.mapred.ShuffleHandler</value>
</property>
```

*yarn.nodemanager.aux-services - The name of an auxiliary service being added to the Node Manager.
*yarn.nodemanager.aux-services.mapreduce_shuffle.class - Implements the mapreduce_shuffle service

Configuration of yarn-site.xml: yarn-site.xml file enables you to overwrite a number of default values controlling the YARN components.

```
cp /usr/local/hadoop/etc/hadoop/mapred-site.xml.template
/usr/local/hadoop/etc/hadoop/mapred-site.xml
```

Configuration of mapred-site.xml: mapred-site.xml file enables you to overwrite a number of default values controlling the Mapreduce job execution.components.

```
sudo gedit mapred-site.xml
```

```
## This runtime framework that is used to execute MapReduce jobs.
<property>
<name>mapreduce.framework.name</name>
<value>yarn </value>
</property>
```

```
cd
clear
```

Prepare to start the hadoop cluster by formatting the namenode, which needs to be done only once.

```
hdfs namenode -format or
/usr/local/hadoop/bin/hadoop namenode -format
```

```
cd /usr/local/hadoop
```

```
Start hdfs daemons/services
start-dfs.sh or
sbin/start-dfs.sh
```

Format the New Hadoop Filesystem

```
hduser@ubuntu:~$ hadoop namenode --format
```

Starting Hadoop

```
hduser@ubuntu:~$ Start-all.sh
```

Stopping Hadoop

```
hduser@ubuntu:~$ Stop-all.sh
```

Checking if it’s really up and running:

```
hduser@ubuntu:~$ jps
```