Practice Tasks using Hadoop

Task 0: Installation (Single Node)

For installation, any of the following options can be availed:

- **Option 1; Install Apache Hadoop directly:** Download and install Apache Hadoop directly on your own machine. Visit the download section on their website https://hadoop.apache.org and follow the instructions (varies from distribution to distribution).
- **Option 2; Create an instance on Google Cloud Platform (GPC):** Create an instance on Google Cloud platform by visiting http://cloud.google.com and create a Compute Engine Instance with your username. In Boot Disk type, specify Ubuntu LTS latest version. In Firewall, allow http traffic. When done, add your ssh key (public key of your local machine) to SSH Keys when clicking on your created instance.

If you choose any online cloud platform, you would need to login to the platform through SSH.

```
ssh <ip address>
```

Perform some routine installations by running the following:

```
sudo apt update
sudo apt install openjdk-8-jdk
```

Create a non-root user for Hadoop with any password you like (e.g. test123), and then login to it using su.

```
sudo adduser hdoop

su - hdoop
```

To prevent frequent login or for password less login, use ssh-keygen (for the hdoop user) and add it to authorized keys as follows:

```
ssh-keygen -t rsa

cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys

chmod 0600 ~/.ssh/authorized_keys
```

Now, download the latest hadoop version from their website and extract it (Note: The latest version is 3.4.0. The following instructions are for version 3.3.6. Modify accordingly, or follow with the older version)

```
wget https://dlcdn.apache.org/hadoop/common/hadoop-3.3.6/hadoop-3.3.6.tar.gz

tar xzf hadoop-3.3.6.tar.gz
```

Set your environment variables by editing the file ~/.bashrc using nano or vim, and add the contents to the end of the file.

```
nano ~/.bashrc

export HADOOP_HOME=/home/hdoop/hadoop-3.3.6
export HADOOP_INSTALL=$HADOOP_HOME
export HADOOP_MAPRED_HOME=$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 PATH=$PATH:$HADOOP_HOME/sbin:$HADOOP_HOME/bin
```

```
export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib/native"
```

Specify the Java home location by running the following few lines. The location should end with openjdk-amd64 only.

```
readlink -f /usr/bin/javac
```

Specify the JAVA_HOME variable in the following:

```
nano ~/hadoop-3.3.6/etc/hadoop/hadoop-env.sh

export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
```

Perform some basic hadoop configurations using the following:

```
vim ~/hadoop-3.3.6/etc/hadoop/core-site.xml

<configuration>
<property>
  <name>hadoop.tmp.dir</name>
  <value>/home/omar/hadoop-3.3.6/tmpdata</value>
</property>
<property>
  <name>fs.default.name</name>
  <value>hdfs://127.0.0.1:9000</value>
</property>
</configuration>
```

With the above, make sure you create the tmpdata directory in the right location you specified.

Then, perform some basic HDFS settings using:

```
vim ~/hadoop-3.3.6/etc/hadoop/hdfs-site.xml

<configuration>
<property>
  <name>dfs.data.dir</name>
  <value>/home/omar/hadoop-3.3.6/dfsdata/namenode</value>
</property>
<property>
  <name>dfs.datanode.data.dir</name>
  <value>/home/omar/hadoop-3.3.6/dfsdata/datanode</value>
</property>
<property>
  <name>dfs.replication</name>
  <value>1</value>
</property>
</configuration>
```

Then, select the resource manager for your Map Reduce jobs as Yarn using:

```
vim ~/hadoop-3.3.6/etc/hadoop/mapred-site.xml

<configuration>
<property>
  <name>mapreduce.framework.name</name>
  <value>yarn</value>
</property>
</configuration>
```

Then, give the following for resource manager Yarn settings:

```
vim ~/hadoop-3.3.6/etc/hadoop/yarn-site.xml
```

```
<configuration>
<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>
<property>
  <name>yarn.resourcemanager.hostname</name>
  <value>34.125.156.43</value>
</property>
<property>
  <name>yarn.resourcemanager.webapp.address</name>
  <value>${yarn.nodemanager.hostname}:5349</value>
</property>
<property>
    <name>yarn.resourcemanager.admin.address</name>
    <value>${yarn.nodemanager.hostname}:5350</value>
</property>

<property>
    <name>yarn.resourcemanager.resource-tracker.address</name>
    <value>${yarn.nodemanager.hostname}:5351</value>
</property>
<property>
    <name>yarn.resourcemanager.scheduler.address</name>
    <value>${yarn.nodemanager.hostname}:5352</value>
</property>
<property>
    <name>yarn.resourcemanager.address</name>
    <value>${yarn.nodemanager.hostname}:5353</value>
</property>
<property>
  <name>yarn.acl.enable</name>
  <value>0</value>
</property>
<property>
  <name>yarn.nodemanager.env-whitelist</name>
<value>JAVA_HOME,HADOOP_COMMON_HOME,HADOOP_HDFS_HOME,HADOOP_CONF_DIR,CLASSPATH_PERPEND_DI
STCACHE,HADOOP_YARN_HOME,HADOOP_MAPRED_HOME</value>
</property>
</configuration>
```

With the settings done, we can now format our HDFS file system using:

```
hdfs namenode -format
```

Now, we start our all services individually, or start all in bulk using either of the following

```
~/hadoop-3.3.6/sbin/start-dfs.sh

~/hadoop-3.3.6/sbin/start-yarn.sh

~/hadoop-3.3.6/sbin/start-all.sh
```

When started, you can check which processes are active by running Java Processes (jps) in the end:

```
jps
```

If using google cloud platform, you will need to Create firewall rule for your system IP address and port 9870 (for Hadoop) and 5349 (for Yarn Resource Manager) in Firewall settings of Google Cloud Platform.

Task 1: Creating Your Directory Space

The first thing is to create your own directory space on HDFS. Issue the following from your terminal:

```
hdfs dfs -mkdir /usr/$(whoami)
```

Verify that it exists by the following:

```
hdfs dfs -ls /usr

drwxr-xr-x   - omar omar          0 2020-05-16 09:49 /usr/omar
```

Note that the initial column contains drwxr-xr-x. This can be broken into multiple sections:

1. d, meaning that this is a directory
2. rwx, meaning that the user “omar” has read/write/execute permissions
3. r-x, meaning that all users in the hadoop group have read/execute permissions
4. r-x, meaning that everybody else on the system have read/execute permissions

You may set your folder to restricted mode to avoid other users tampering with your data:

```
hdfs dfs -chmod -R 700 /usr/omar
```

Task 2: Understanding the System

Using the following commands, address the questions:

```
hdfs dfsadmin -printTopology

hdfs dfsadmin -report

hdfs fsck /

hadoop fsck / -files -blocks -locations
```

Questions

1. How many datanodes are part of the hadoop topology?
2. What are the IP addresses of these datanodes?
3. What is the configured and present capacity of the HDFS?
4. What is the default file replication count?

Task 3: Expanding to Multiple Nodes

Inside your master node (created in Task 0), edit the /etc/hosts file and add IP addresses of both machines.

```
vim /etc/hosts

192.168.1.153 hadoop-master
192.168.1.154 hadoop-slave
```

Follow the same steps on the new machine you are using. i.e., inside your data node (created now), edit the /etc/hosts file also and add IP addresses of both machines.

```
vim /etc/hosts

192.168.1.153 hadoop-master
192.168.1.154 hadoop-slave
```

Next, setup ssh in the data node to facilitate password less authentication. On your data node, run the following:

```
ssh-keygen -t rsa

cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys

ssh-copy-id -i ~/.ssh/id_rsa.pub hdoop@hadoop-master
ssh-copy-id -i ~/.ssh/id_rsa.pub hdoop@hadoop-slave

chmod 0600 ~/.ssh/authorized_keys
```

Above will copy the public key to both hadoop-master and hadoop-slave. On the master node, the public key is already placed in authorized keys. You need to add the public key of the master to the authorized key of the data node. For this, on the master node, run the following:

```
ssh-copy-id -i ~/.ssh/id_rsa.pub hdoop@hadoop-slave
```

Then, follow the same steps on an additional machine that are given in Task 0: Single Node Cluster for installation of hadoop.

In the data node, the following settings need to be modified

```
vim ~/hadoop-3.3.6/etc/hadoop/core-site.xml

<property>
 <name>fs.default.name</name>
 <value>hdfs://hadoop-master:9000</value>     # Modified from 127.0.0.1 in Task 0
</property>
<property>                                    # New property added
 <name>dfs.permissions</name>
 <value>false</value>
</property>
```

In the mapred-site.xml external file, make the following changes:

```
vim ~/hadoop-3.3.6/etc/hadoop/mapred-site.xml

<configuration>
 <property>
 <name>mapred.job.tracker</name>
 <value>hdfs://hadoop-master:9001</value>
 </property>
</configuration>
```

Task 4: Word Count Application

Take any sufficiently long text file (txt) format and move it to your HDFS. Suppose the file is called test.txt.

Copy over your data using:

```
hdfs dfs -put test.txt /test.txt
```

Verify that it exists by:

```
hdfs dfs -ls /test.txt

Found 1 items

-rw-r--r-- 3 omar omar ..... 10:16 /test.txt
```

Answer the following questions:

	Question	Answer
1	What is the default block size (in Mb) of the test.txt file?	
2	Is there any missing replicas for the file test.txt?	
3	What command will you use to change this block size? (remember to convert into bytes)	
4	How many blocks are used by test.txt after changing block size above?	
5	How many missing replicas are there for file test.txt after block change?	
6	Why are there missing replicas?	

Note: You can use the following to get data for some of the above answers:

```
hdfs fsck /test.txt
```

Setup your map reduce job by taking the mapper.py code from lecture slides.

```
#!/usr/bin/python

import sys

for line in sys.stdin:

        data = line.strip().split(",")
        key = data[0]
        value = 1
        print ("{0}\t{1}".format(key, value) )
```

Also do the same for reducer.py:

```
#!/usr/bin/python

import sys
total = 0
oldkey = None

for line in sys.stdin:

        data = line.strip().split("\t")
        thiskey = data[0]
        value = data[1]

        if thiskey != oldkey and oldkey != None:

                print ("{0}\t{1}".format(oldkey, total))
                oldkey = thiskey
                total = 0

        oldkey = thiskey
        total += float(value)

if oldkey != None:

        print ("{0}\t{1}".format(oldkey, total))
```

Give both the mapper.py and Reducer.py executable permissions:

```
chmod u+x mapper.py

chmod u+x reducer.py
```

Test the mapper and reducer on your local directory first, without map reduce:

```
cat test.txt | ./mapper.py | sort | ./reducer.py
```

If it works, you should be able to see the word count in action. You now need to migrate towards hadoop. For this, run the following:

```
hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-3.2.3.jar -file
./mapper.py -file ./reducer.py -mapper mapper.py -reducer reducer.py -input /test.txt
-output /query1_output
```

You will see plenty of output generated on the screen. Give answers to the following:

	Question	Answer
1	What was the <key,value> pair used in this query?	
2	How many mapper threads were used?	
3	How many reducer threads were used?	
4	What was the time spent by all mapper threads?	
5	What was the time spent by all reducer threads?	
6	What is the file name in which your output is located?	