

EE542 – Laboratory Assignment #6: Hadoop Tutorial

Instructor: Young H. Cho

For this lab we would be creating 1 namenode and 3 datanode (so 4 instances in total). (google for the naming convention of Hadoop). Go through below links for detailed understanding of Hadoop, containers and map-reduce architecture. (you will also need to implement same program mentioned in document below on MPI also). So, start earlier on doing lab.

<https://en.wikipedia.org/wiki/MapReduce>

<https://hadoop.apache.org/docs/r2.10.1/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html>

<https://hadoop.apache.org/docs/r2.10.1/hadoop-yarn/hadoop-yarn-site/YARN.html>

<https://hadoop.apache.org/docs/r2.10.1/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html>

Select a compatible **Ubuntu Server** (i.e. Ubuntu Server 16.04 LTE HVM) image on x86 and configure the instance with only one interface it being elastic ip interface. Instance type should be of t2.medium or any instance with two vcpus and at least 4 GB of ram and 12 GB of hard drive space (you may try out this in student account to reduce cost). In your selected VPC you have to enable **DNS Hostname** from the VPC tab. For the only subnet attached to the VPC you have to enable “**auto-assign ipv4**” under “Modify auto assign public ipv4 address”. Moreover, you have to “auto assign public ip” set to “**use subnet setting (enable)**”. Assign network interface IP statically from within the AWS. These options would automatically assign you a public ip since we have only one network interface per instance. (elastic ip not required)

Step 3: Configure Instance Details

No default subnet found

Please choose another subnet in your default VPC, or choose another VPC.

Configure the instance to suit your requirements. You can launch multiple instances from the same AMI, request Spot instances to take advantage c

Number of instances	<input type="text" value="1"/>	Launch into Auto Scaling Group
Purchasing option	<input type="checkbox"/> Request Spot instances	
Network	<input type="text" value="vpc-00548f134b4c44749 lab8_vpc_hadoop"/>	Create new VPC
Subnet	<input type="text" value="subnet-098a5f12e954a5390 hadoop us-west-2a"/> 247 IP Addresses available	Create new subnet
Auto-assign Public IP	<input type="text" value="Use subnet setting (Enable)"/>	

Modify auto-assign IP settings

Enable the auto-assign IP address setting to automatically request a public IPv4 or IPv6 address for an instance launched in this subnet. You can override the auto-assign IP settings for an instance at launch time.

Subnet ID subnet-078bb4ccff6581038

Auto-assign IPv4 ☒ Enable auto-assign public IPv4 address ⓘ

Auto-assign Co-IP ☐ Enable auto-assign customer-owned IPv4 address ⓘ

* Required

<input type="checkbox"/>	datanode-1	i-0755e71b3345bc1f4	t2.medium	us-west-2a	running	2/2 checks ...	None	ec2-34-222-27-154.us-...	34.222.27.154	-	west_vyos
<input type="checkbox"/>	datanode-2	i-0c6772e106647cc22	t2.medium	us-west-2a	running	2/2 checks ...	None	ec2-54-184-35-172.us-...	54.184.35.172	-	west_vyos
<input type="checkbox"/>	datanode-3	i-0af1e3954d85b4ffe	t2.medium	us-west-2a	running	2/2 checks ...	None	ec2-52-43-246-200.us-...	52.43.246.200	-	west_vyos
<input type="checkbox"/>	mpi_initiator_1_new	i-01f7d6fb8cc0b4162	t2.large	us-west-2a	stopped		None		-	-	west_vyos
<input type="checkbox"/>	namenode	i-08a5914772e4fc3e	t2.medium	us-west-2a	running	2/2 checks ...	None	ec2-35-160-60-48.us-w...	35.160.60.48	-	west_vyos

create a security group with rules to allow all kind of traffic within the private subnet and to your global ip only. (my elastic ip subnet was 10.0.0.0/24 and global ip 104.32.154.67)

Description	Inbound	Outbound	Tags
Edit			
Type ⓘ	Protocol ⓘ	Port Range ⓘ	Source ⓘ
All traffic	All	All	10.0.0.0/24
All traffic	All	All	104.32.154.67/32
SSH	TCP	22	0.0.0.0/0

Ensure that all the instances you have created uses same private key for ssh. We will be using only **ubuntu** user account for logging in through ssh.

After logging in, execute below commands on all of the instances:

```
# sudo cp /etc/ssh/ssh_host_rsa_key ~/.ssh/id_rsa
# sudo cp /etc/ssh/ssh_host_rsa_key.pub ~/.ssh/id_rsa.pub (change to ubuntu@ in last)
# sudo cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys (change to ubuntu@ in last)
# sudo chmod 777 ~/.ssh/id_rsa
# sudo chmod 777 ~/.ssh/id_rsa.pub
```

```
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQCsEyt+GlHYHkcwRg+ogiodeEH
E0VB1PNgG5SujFQcUIH6r5uVf6OHgZ+MhgAWZI+yom2kAkPo9GBGxlozNRmjcj
To358QZ0GyTGG1Wymn68++5qKxNT7Je8hLvqh1 ubuntu@ip-10-0-0-247
```

Ensure that in all remote hosts, the `authorized_key` file is modified to include public key of the namenode (namenode -> where you will be running Hadoop commands) and it contains `ubuntu@` for the host being added to it. My master node ip here was of 10.0.0.247. (these steps are same as the steps for establishing passwordless ssh connectivity as in previous labs).

```
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQACUNsdJ/pVLo4+PZL2cUV0hOKCG9aKqAOWqB4Fl+JytdLcSZlDm1tG3pyzYlRMMpeaUBWnOA/YLQV0bNgZVX2uv2ULvh
OAM0q85AfMhUoPnMTW8pFk5Ura/GJ7DXnCadxaByzbA9pp38BVLtWDVkv8sHszERYV0mFzE0+Ebokh3YyVPWaTwcRNNolhQsmI+QhtOoWftEzkYk9gknAfgK3JJxA8fl
xBwrmcUt6w3Gwa1fauwGcUmgIpx/dbygqsfBjz52iu4t6NXgehVDAw1F07xmQATPofeYmEptKTmROQDQnj12LXk5RYZbdhopljgASnw5wu3DzQ6o5dUYbM8B west_vyos
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQACbWiny4UhdBHwtqk5G53rz7yAhfaXD8zqact1HMALZ+z0kBlW/3rknSZWbqm7u0j9Ego3EDaAh0SiYx0iuOeKsRL5f+
A4Oui5/E7I6j0XPhAps6J5YgpJjIt2vhv6/FwisJzFjUaJE4mznyJgbgy9EFMhIANWlmbHSRPrUukDjHSbpNO9v1TbpG1J9H3Tt0ePlywB4ssL+DceUnmd1SJSHPKu/4Lj
ykYiJeIwPbFjddaGfZ7DF1Nb9Hm4vVMuyUGCJPQcM562HyVF+BBdvsYALCIGh6Zb0Q/e4kPjH8qNaCsElcddEHFahPNjEmFOXyqT6hFRBpjOG1JU9Eh9 ubuntu@ip
-10-0-0-115
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQACsEyt+G1HYHkcwRg+ogiodEHKqZMQ2l0d6eOj7EWwbG3GyvQuHz/7svsuC892/zSpPnfMZGhYh8QjC804rpf2N6G7RpH
MGa06FJ6e7XMaMKShizwm3kcANHAfZxu001SiFtDE0VB1PngG5SujFQcUIH6r5uVf6OHgz+MhgAW2I+yom2kAkPo9GBGxlozNRmjcxBoNqRcuksMxuYRzB12+fdqottj
5lK00HTwCAeVSqsA5LK22xOgSuJfNqiamln8kyPmHjx4jzvqmJl30iY3zMZwu9AvR+49d2vXOvdZTG+ybpTo358Q20GyTGG1Wymn68++5qKxNT7Je8hLvhl ubuntu@ip
-10-0-0-247
```

Now install java on all the instances by:

```
# sudo apt-get update
# sudo apt install default-jre
# sudo apt-get install default-jdk
# cd ~/
```

Now download hadoop by executing:

```
# wget https://mirrors.koehn.com/apache/hadoop/common/hadoop-2.10.1/hadoop-2.10.1.tar.gz
# sudo tar xzf hadoop-2.10.1.tar.gz
# sudo mv hadoop-2.10.1 /usr/local/hadoop
# sudo chown -R ubuntu /usr/local/hadoop/
```

Edit ~/.profile file with below contents added as shown in figure:

```
export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-amd64
export HADOOP_HOME=/usr/local/hadoop
PATH="$HOME/bin:$HOME/.local/bin:$JAVA_HOME/bin:$HADOOP_HOME/bin:$PATH"
```

```
# if running bash
if [ -n "$BASH_VERSION" ]; then
    # include .bashrc if it exists
    if [ -f "$HOME/.bashrc" ]; then
        . "$HOME/.bashrc"
    fi
fi

export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-amd64
export HADOOP_HOME=/usr/local/hadoop
PATH="$HOME/bin:$HOME/.local/bin:$JAVA_HOME/bin:$HADOOP_HOME/bin:$PATH"
```

Then execute:

```
# source ~/.profile
```

This results in environment variables of your current ssh session to be updated. When you login later over ssh, ubuntu automatically source this file.

Edit following files under "/usr/local/hadoop/etc/hadoop/" for all the instances (namenode + datanodes).

1. `hadoop-env.sh`

replace "export JAVA_HOME=" with "export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-amd64"

```
# The java implementation to use.
export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-amd64

# The jsvc implementation to use. Jsvc is required to run security
# that bind to privileged ports to provide authentication of
# protocol. Jsvc is not required if SASL is configured for a
# data transfer protocol using non-privileged ports.
export JSVC_HOME=${JSVC_HOME}

export HADOOP_CONF_DIR=${HADOOP_CONF_DIR:-"/etc/hadoop"}
```

2. `core-site.xml`.

Edit it with the below content (remove: `<configuration>` `</configuration>` and update with below contents)

```
<configuration>
  <property>
    <name>fs.default.name</name>
    <value>hdfs://<namenode_private_IP>:9000</value>
  </property>
  <property>
    <name>hadoop.tmp.dir</name>
    <value>/home/ubuntu/hadooptmp</value>
    <description>A base for other temporary directories.</description>
  </property>
</configuration>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Put site-specific property overrides in this file. -->
<configuration>
  <property>
    <name>fs.default.name</name>
    <value>hdfs://10.0.0.247:9000</value>
  </property>
  <property>
    <name>hadoop.tmp.dir</name>
    <value>/home/ubuntu/hadooptmp</value>
    <description>A base for other temporary directories.</description>
  </property>
</configuration>
```

Change the **namenode_private_IP** to eth0 interface ip of the namemode for all of the instances (namenode + datanodes). (this ip should not change in-between instance reboots, so better to assign static in aws network interface creation phase)

Also **create "/home/ubuntu/hadooptmp"** (as it's specified in the above configuration for temp directory for hdfs)

core-site.xml contains the configuration settings for Hadoop Core (eg I/O) that are common to HDFS and MapReduce. It also informs Hadoop daemon where NameNode (the master) runs in the cluster. So each node(instance) must have this file completed.

3. hdfs-site.xml

Edit with below content: (remove: <configuration> </configuration> and update with below contents)

```
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>3</value>
  </property>
  <property>
    <name>dfs.namenode.name.dir</name>
    <value>/usr/local/hadoop/hdfs/namenode</value>
  </property>
  <property>
    <name>dfs.namenode.data.dir</name>
    <value>/usr/local/hadoop/hdfs/datanode</value>
  </property>
</configuration>
```

```
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>3</value>
  </property>
  <property>
    <name>dfs.namenode.name.dir</name>
    <value>/usr/local/hadoop/hdfs/namenode</value>
  </property>
  <property>
    <name>dfs.namenode.data.dir</name>
    <value>/usr/local/hadoop/hdfs/datanode</value>
  </property>
</configuration>
```

4. mapred-site.xml.template

First rename mapred-site.xml.template to mapred-site.xml.

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

Then put below contents in it.

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

```
<!-- Put site-specific property overrides in this file. -->
<configuration>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>
</configuration>
```

5. yarn-site.xml

Edit with below content: (remove: <configuration> </configuration> and update with below contents)

```
<configuration>
  <property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
  </property>
  <property>
    <name>yarn.nodemanager.auxservices.mapreduce.shuffle.class</name>
    <value>org.apache.hadoop.mapred.ShuffleHandler</value>
  </property>
  <property>
    <name>yarn.resourcemanager.hostname</name>
    <value><namenode_private_IP></value>
    <description>The hostname of the Resource Manager.</description>
  </property>
</configuration>
```


Give namenode eth0 interface ip in <namenode_private_IP> for all of the instances (namenode + datanodes).

Replicate the above configurations in 3 **datanode** instances also.

Now log into each of the **datanode** instances. Create:

1. empty master files using "touch /usr/local/hadoop/etc/hadoop/masters"
2. vi /usr/local/hadoop/etc/hadoop/slaves

Specify eth0 ip of the current datanode instance (where you are executing the vi command) in "/usr/local/hadoop/etc/hadoop/slaves". Remove all other contents. (only one ip should be present)

```
ubuntu@ip-10-0-0-125: /usr/local/hadoop/logs
10.0.0.125
```

My one of the datanode's IP was 10.0.0.125.

Now in the **namenode** instance. Edit

1. vi /usr/local/hadoop/etc/hadoop/masters

Give eth0 interface ip of the namenode in the file.

```
ubuntu@ip-10-0-0-247: /usr/local/hadoop/etc/hadoop
10.0.0.247
```

My master node ip was 10.0.0.247.

2. vi /usr/local/hadoop/etc/hadoop/slaves

Give eth0 IP of all of the 3 data-nodes in this file. Remove all other contents. (add line by line each IP)

```
ubuntu@ip-10-0-0-247: /usr/local/hadoop/etc/hadoop
10.0.0.125
10.0.0.115
10.0.0.68
```

Now perform from master-node below commands:

```
# ssh -v -x ubuntu@<IP>
```

to all the 3 data-nodes from namenode. If it asks for a message confirmation type yes. This should establish a passwordless ssh connectivity to remote host. Type exit to bring back control to the source host. (Only proceed further if you are able to do passwordless ssh access from namenode to all of the datanodes).

Execute below command only on **namenode** to create hdfs file system (directory is specified in hdfs-site.xml):

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

Now we will start hdfs and yarn daemons of Hadoop. Hdfs is responsible for maintaining hdfs distributed file system and yarn for containers and resource management.

```
# /usr/local/hadoop/sbin/start-dfs.sh
```

```
# /usr/local/hadoop/sbin/start-yarn.sh
```

If it asks for yes/no message, type yes and press enter (it's self-logging in namenode instance using localhost).

For debugging purposes, logs are located under "/usr/local/hadoop/logs" in datanodes and namenodes. If you face any issues first check the logs under these folder and make an inference what could be the issue by googling.

Execute below command to verify that datanode and namenode is up and configured:

```
# hdfs dfsadmin -report
```

This should show your namenode and 3 datanodes.

```
ubuntu@ip-10-0-0-247:/usr/local/hadoop/etc/hadoop$ hdfs dfsadmin -report
Configured Capacity: 37267918848 (34.71 GB)
Present Capacity: 28386066432 (26.44 GB)
DFS Remaining: 24840458240 (23.13 GB)
DFS Used: 3545608192 (3.30 GB)
DFS Used%: 12.49%
Under replicated blocks: 0
Blocks with corrupt replicas: 0
Missing blocks: 0
Missing blocks (with replication factor 1): 0
Pending deletion blocks: 0

-----
Live datanodes (3):

Name: 10.0.0.115:50010 (ip-10-0-0-115.us-west-2.compute.internal)
Hostname: ip-10-0-0-115.us-west-2.compute.internal
Decommission Status : Normal
Configured Capacity: 12422639616 (11.57 GB)
DFS Used: 1181773824 (1.10 GB)
Non DFS Used: 2943717376 (2.74 GB)
DFS Remaining: 8280371200 (7.71 GB)
DFS Used%: 9.51%
DFS Remaining%: 66.66%
Configured Cache Capacity: 0 (0 B)
Cache Used: 0 (0 B)
Cache Remaining: 0 (0 B)
Cache Used%: 100.00%
Cache Remaining%: 0.00%
Xceivers: 1
Last contact: Mon Oct 12 03:21:13 UTC 2020
Last Block Report: Sun Oct 11 21:54:48 UTC 2020

Name: 10.0.0.125:50010 (ip-10-0-0-125.us-west-2.compute.internal)
Hostname: ip-10-0-0-125.us-west-2.compute.internal
Decommission Status : Normal
Configured Capacity: 12422639616 (11.57 GB)
DFS Used: 1181782016 (1.10 GB)
Non DFS Used: 2944299008 (2.74 GB)
DFS Remaining: 8279781376 (7.71 GB)
DFS Used%: 9.51%
DFS Remaining%: 66.65%
Configured Cache Capacity: 0 (0 B)
Cache Used: 0 (0 B)
Cache Remaining: 0 (0 B)
Cache Used%: 100.00%
Cache Remaining%: 0.00%
```


Now from the namenode, execute below sample program:

```
#hadoop jar /usr/local/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.10.1.jar pi 10 1000
```

If above program ran successfully you should be seeing an output like below:

```
Bytes Read=1180
File Output Format Counters
Bytes Written=97
Job Finished in 27.421 seconds
Estimated value of Pi is 3.14080000000000000000
```

You can access hadoop Master dashboard from:

<http://<namenode instance public ip>:8088/cluster>

The screenshot shows the Hadoop Master dashboard at the URL <http://35.160.60.48:8088/cluster/apps/FINISHED>. The dashboard displays the Hadoop logo and a sidebar with navigation options: Cluster, About, Nodes, Node Labels, Applications, NEW, NEW SAVING, SUBMITTED, ACCEPTED, RUNNING, FINISHED, FAILED, KILLED, Scheduler, and Tools. The main content area is titled 'FINISHED Applications' and shows 'Cluster Metrics' and 'Scheduler Metrics'. The 'Cluster Metrics' table includes columns for Apps Submitted, Apps Pending, Apps Running, Apps Completed, Containers Running, and Used Resources. The 'Scheduler Metrics' table includes columns for Scheduler Type, Scheduling Resource Type, Capacity Scheduler, and a list of applications. Two applications are listed: 'application_1602451219882_0007' and 'application_1602451219882_0006', both in 'FINISHED' state with 'SUCCEEDED' final status.

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Used Resources
7	0	0	7	0	<memory:0, vCores:0>

Scheduler Type	Scheduling Resource Type
Capacity Scheduler	[<name=memory-mb default-unit=M type=COUNTABLE>, <name=vcores default-unit= type=COUNTABLE>]

ID	User	Name	Application Type	Queue	Application Priority	StartTime	LaunchTime	FinishTime	State	FinalStatus	Running Container
application_1602451219882_0007	ubuntu	streamjob6461858192224585556.jar	MAPREDUCE	default	0	Sun Oct 11 17:00:06 -0700 2020	Sun Oct 11 17:00:07 -0700 2020	Sun Oct 11 17:07:27 -0700 2020	FINISHED	SUCCEEDED	N/A
application_1602451219882_0006	ubuntu	streamjob2316964470512303092.jar	MAPREDUCE	default	0	Sun Oct 11 16:35:55 -0700 2020	Sun Oct 11 16:35:55 -0700 2020	Sun Oct 11 16:36:15 -0700 2020	FINISHED	SUCCEEDED	N/A

datanode instance access from:

<http://<datanode instance public ip>:8042/cluster>

The screenshot shows the Hadoop datanode dashboard at the URL <http://34.222.27.154:8042/node>. The dashboard displays the Hadoop logo and a sidebar with navigation options: Resource Manager, Node Manager, Node Information, List of Applications, List of Containers, and Tools. The main content area is titled 'NodeManager Information' and shows various metrics and status information for the NodeManager. The metrics include Total Vmem allocated for Containers (16.80 GB), Vmem enforcement enabled (true), Total Pmem allocated for Container (8 GB), Pmem enforcement enabled (true), Total Vcores allocated for Containers (8), NodeHealthyStatus (true), LastNodeHealthTime (Mon Oct 12 04:36:19 UTC 2020), NodeHealthReport, NodeManager started on (Sun Oct 11 21:20:18 UTC 2020), NodeManager Version (2.10.1 from 1827467c9a56f133025f28557bfc2c562d78e816 by centos source checksum 2da9946ffe56799794b77621f6e0be1a on 2020-09-14T13:24Z), and Hadoop Version (2.10.1 from 1827467c9a56f133025f28557bfc2c562d78e816 by centos source checksum 3114edef868f1f3824e7d0f68be03650 on 2020-09-14T13:17Z).

Metric	Value
Total Vmem allocated for Containers	16.80 GB
Vmem enforcement enabled	true
Total Pmem allocated for Container	8 GB
Pmem enforcement enabled	true
Total Vcores allocated for Containers	8
NodeHealthyStatus	true
LastNodeHealthTime	Mon Oct 12 04:36:19 UTC 2020
NodeHealthReport	
NodeManager started on	Sun Oct 11 21:20:18 UTC 2020
NodeManager Version	2.10.1 from 1827467c9a56f133025f28557bfc2c562d78e816 by centos source checksum 2da9946ffe56799794b77621f6e0be1a on 2020-09-14T13:24Z
Hadoop Version	2.10.1 from 1827467c9a56f133025f28557bfc2c562d78e816 by centos source checksum 3114edef868f1f3824e7d0f68be03650 on 2020-09-14T13:17Z

You can view hadoop filesystem with webgui by following below link:

<http://<namenode public ip>:50070>

Browse Directory

Show entries
 Search:

<input type="checkbox"/>	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name	
<input type="checkbox"/>	-rw-r--r--	ubuntu	supergroup	0 B	Oct 11 17:07	3	128 MB	_SUCCESS	
<input type="checkbox"/>	-rw-r--r--	ubuntu	supergroup	42.48 MB	Oct 11 17:07	3	128 MB	part-00000	

Showing 1 to 2 of 2 entries

You can explore more programs under "hadoop jar /usr/local/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.10.1.jar"

⇒ If you get error while clicking some text under page, **replace the private ip** in http link with corresponding instance **public ip**.

Debugging Tips:

use command "jps" to view java running processes on each instances.

"netstat -a -p | grep -i port" and then "kill pid". For releasing the port. (To troubleshoot port error obtained from logs inside for hdfs and yarn daemons)

hadoop job -kill <job_id> --> to kill job (from namenode)

Now we will be discussing a technique in which you can use **python** to write your map and reduce functions. (normally in hadoop it's in java as you have seen in above examples). We will be using streaming api support of Hadoop to achieve this. (even c/c++ program can also be done in this way)

Install python on all datanodes and namenode.

```
# sudo apt-get install python
```

Here we will be implementing a word count map-reduce program. It reads text files and counts how often words occur. The input is text files and the output is text files, each line of which contains a word and the count of how often it occurred, separated by a tab.

Create a mapper and reduce function in python as explained below (under home directory).

1. Create mapper.py which would be used for map function with below contents. Also change permission to 777 (by executing chmod 777 mapper.py):

```
#!/usr/bin/env python
"""mapper.py"""
```

```
import sys
```

```
# input comes from STDIN (standard input)
```

```

for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    # split the line into words
    words = line.split()
    # increase counters
    for word in words:
        # write the results to STDOUT (standard output);
        # what we output here will be the input for the
        # Reduce step, i.e. the input for reducer.py
        #
        # tab-delimited; the trivial word count is 1
        print '%s\t%s' % (word, 1)

```

2. Create `reduce.py` for reduce functionality of map-reduce with below contents. Also change permission to 777 (by executing `chmod 777 reduce.py`):

```

#!/usr/bin/env python
"""reducer.py"""

from operator import itemgetter
import sys

current_word = None
current_count = 0
word = None

# input comes from STDIN
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()

    # parse the input we got from mapper.py
    word, count = line.split('\t', 1)

    # convert count (currently a string) to int
    try:
        count = int(count)
    except ValueError:
        # count was not a number, so silently
        # ignore/discard this line
        continue

    # this IF-switch only works because Hadoop sorts map output

```

```

# by key (here: word) before it is passed to the reducer
if current_word == word:
    current_count += count
else:
    if current_word:
        # write result to STDOUT
        print '%s\t%s' % (current_word, current_count)
    current_count = count
    current_word = word

# do not forget to output the last word if needed!
if current_word == word:
    print '%s\t%s' % (current_word, current_count)

```

3. Download an input dataset which is a sample text file for testing map-reduce program by executing:

```
# wget https://www.gutenberg.org/files/20417/20417-8.txt
```

4. Now run:

```
# cat 20417-8.txt | ./mapper.py | sort -k1,1 | ./reducer.py
```

It should produce output like below: (ie <text> <word count>) (this is manual way to simulate operations which map-reduce does in background, so that the program can be tested before starting it as map-reduce job)

```

=      2
|      136
|-----|-----|-----|-----|      2
|-----|-----|-----|-----|      3
|-----|-----|-----|-----|      1
|-----|-----|-----|-----|      1
|-----|-----|-----|-----|      7
|-----|-----|-----|-----|
-      7
--      2
----- 4
-,      1
{      3
***      6
*****  2
&      12
+      1
$      77
0      2
.001    1
0.24    1
0.62    1
1      24

```

Now download 3 input text file for passing input to map reduce job. First create a directory under home and cd to that and download the txt files using wget:

```
# mkdir ~/test_txt
# cd ~/test_text
# wget http://www.gutenberg.org/files/5000/5000-8.tx
# wget http://www.gutenberg.org/files/4300/4300-0.txt
# wget http://www.gutenberg.org/cache/epub/20417/pg20417.txt
```

Execute below command to copy from local file system to hadoop filesystem.

```
#hdfs dfs -copyFromLocal ~/test_text /user/ubuntu/gutenberg
```

For Viewing files under hdfs execute:

```
#hdfs dfs -ls /user/ubuntu
```

```
ubuntu@ip-10-0-0-247:~/test_txt$ hdfs dfs -ls
Found 4 items
drwxr-xr-x  - ubuntu supergroup          0 2020-10-11 23:20 gutenberg
drwxr-xr-x  - ubuntu supergroup          0 2020-10-11 21:39 test
drwxr-xr-x  - ubuntu supergroup          0 2020-10-11 21:42 test_result
drwxr-xr-x  - ubuntu supergroup          0 2020-10-11 21:36 tt
ubuntu@ip-10-0-0-247:~/test_txt$ hdfs dfs -ls /user/ubuntu
Found 4 items
drwxr-xr-x  - ubuntu supergroup          0 2020-10-11 23:20 /user/ubuntu/gutenberg
drwxr-xr-x  - ubuntu supergroup          0 2020-10-11 21:39 /user/ubuntu/test
drwxr-xr-x  - ubuntu supergroup          0 2020-10-11 21:42 /user/ubuntu/test_result
drwxr-xr-x  - ubuntu supergroup          0 2020-10-11 21:36 /user/ubuntu/tt
ubuntu@ip-10-0-0-247:~/test_txt$
```

This should show a gutenberg folder as show above. Same thing should be visible from web gui access also on port 50070.

4. Now start map-reduce job by:

```
# hadoop jar /usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-2.10.1.jar -file /home/ubuntu/mapper.py -mapper /home/ubuntu/mapper.py -file /home/ubuntu/reducer.py -reducer /home/ubuntu/reducer.py -input /user/ubuntu/gutenberg/* -output /user/ubuntu/gutenberg-output
```

If everything was succesfull, you would be seeing a directory created at /user/ubuntu/gutenberg-output and contents visible under it.


```

Shuffled Maps=3
Failed Shuffles=0
Merged Map outputs=3
GC time elapsed (ms)=524
CPU time spent (ms)=7370
Physical memory (bytes) snapshot=1045188608
Virtual memory (bytes) snapshot=7702974464
Total committed heap usage (bytes)=695205888

Shuffle Errors
BAD_ID=0
CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0

File Input Format Counters
Bytes Read=3689899
File Output Format Counters
Bytes Written=887453
20/10/12 04:54:08 INFO streaming.StreamJob: Output directory: /user/ubuntu/gutenberg-output5

```

Here I have specified output as /user/ubuntu/gutenberg-output5.

====> Now modify the above program to find the **most** and **least** frequent words. (if multiple words with same frequency, print all such occurrences) and append it to last of the generated text output along with normal output. Execute new map-reduce job with your new modified program. (you might want to delete previous file at output path, google for the Hadoop command to remove file)

Perform below command to verify that output file is created: (as shown below)

```

#hdfs dfs -ls /user/ubuntu/gutenberg-output/
ubuntu@ip-10-0-0-247:~$ hdfs dfs -ls /user/ubuntu/gutenberg-output/
Found 2 items
-rw-r--r--  3 ubuntu supergroup          0 2020-10-11 23:36 /user/ubuntu/gutenberg-output/_SUCCESS
-rw-r--r--  3 ubuntu supergroup    887453 2020-10-11 23:36 /user/ubuntu/gutenberg-output/part-00000
ubuntu@ip-10-0-0-247:~$

```

Verify content by executing:

#hdfs dfs -cat /user/ubuntu/gutenberg-output/part-00000 (here most and least is not shown)

```

2
-'Tis 2
-'lldo! 1
'46. 1
'92 1
'Slife, 1
'Tis 8
'Tis, 1
'Twas 6
'Twixt 1
'em. 2
'mid 1
'neath 1
'pon 1
's 1
'tis 4
'twas 4
'twas. 1
'twere, 1
"Come 1
"J" 1
"Viator" 1
"YOU 1
. 1
* 1
: 3
: 1
crit 1
ubuntu@ip-10-0-0-247:~$

```

Now we optimize mapper.py and reducer.py with below contents, which uses iterators and generators.
(for improved performance)

1. mapper.py

```
#!/usr/bin/env python
"""A more advanced Mapper, using Python iterators and generators."""

import sys

def read_input(file):
    for line in file:
        # split the line into words
        yield line.split()

def main(separator='\t'):
    # input comes from STDIN (standard input)
    data = read_input(sys.stdin)
    for words in data:
        # write the results to STDOUT (standard output);
        # what we output here will be the input for the
        # Reduce step, i.e. the input for reducer.py
        #
        # tab-delimited; the trivial word count is 1
        for word in words:
            print '%s%s%d' % (word, separator, 1)

if __name__ == "__main__":
    main()
```

2. reducer.py

```
#!/usr/bin/env python
"""A more advanced Reducer, using Python iterators and generators."""

from itertools import groupby
from operator import itemgetter
import sys

def read_mapper_output(file, separator='\t'):
    for line in file:
        yield line.rstrip().split(separator, 1)

def main(separator='\t'):
    # input comes from STDIN (standard input)
```

```

data = read_mapper_output(sys.stdin, separator=separator)
# groupby groups multiple word-count pairs by word,
# and creates an iterator that returns consecutive keys and their group:
# current_word - string containing a word (the key)
# group - iterator yielding all ["<current_word>", "<count>"] items
for current_word, group in groupby(data, itemgetter(0)):
    try:
        total_count = sum(int(count) for current_word, count in group)
        print "%s%s%d" % (current_word, separator, total_count)
    except ValueError:
        # count was not a number, so silently discard this item
        pass
if __name__ == "__main__":
    main()

```

Now download input dataset from:

```

# wget
http://www.i3s.unice.fr/~jplozi/hadooplab_lsds_2015/datasets/gutenberg-1G.txt.gz
# gunzip gutenber-1G.txt.gz

```

Then copy this file to hadoop hdfs by executing:

```
# hdfs dfs -copyFromLocal ~/<filepath> /user/ubuntu/gutenberg2
```

==> 1. Then run map-reduce job with your modified code with this input dataset by executing: (I will be using this input dataset for grading purpose with your python changes present on map and reduce functions and it should use iterators and generators)(your new .py files should be uploaded along with report or video submission))

```

# hadoop jar /usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-2.10.1.jar -file /home/ubuntu/mapper.py -mapper /home/ubuntu/mapper.py -file /home/ubuntu/reducer.py -reducer /home/ubuntu/reducer.py -input /user/ubuntu/gutenberg2/gutenberg-1G.txt -output /user/ubuntu/gutenberg-output2

```

REFERENCES:

<https://en.wikipedia.org/wiki/MapReduce>
<https://hadoop.apache.org/docs/r2.10.1/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html>
<https://hadoop.apache.org/docs/r2.10.1/hadoop-yarn/hadoop-yarn-site/YARN.html>
<https://hadoop.apache.org/docs/r2.10.1/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html>
https://www.tutorialspoint.com/hadoop/hadoop_mapreduce.htm
<https://hadoop.apache.org/docs/r2.10.1/hadoop-project-dist/hadoop-common/ClusterSetup.html>
<https://www.michael-noll.com/tutorials/writing-an-hadoop-mapreduce-program-in-python/>
<https://blog.gaelfoppolo.com/lets-try-hadoop-on-aws-13a23e641490>
<https://developer.ibm.com/articles/l-pycon/>