Hadoop Cluster Setup
CSCI 3325: Distributed Systems
March 26, 2015

1. You should already have your private key and set of 4 IP addresses to your machines. If you are getting permissions errors logging in with your key, remember to set the permissions of your keyfile so it is not world- or group-readable:

```
$ chmod 600 username-keypair
```

2. As a first step, it's a good idea to apply all outstanding software updates to your machines (since these machines are imaged from a software snapshot that is typically several months old, even 'freshly installed' machines will often have many updates). SSH into your machines and run:

```
$ sudo yum update
```

You can open up 4 windows to run this simultaneously on all your machines. This will likely take a minute or two to run. Also install the full JDK if you haven't already:

```
$ sudo yum install java-1.7.0-openjdk-devel.x86_64
```

Also while you're logged into each server, run ifconfig and record the IP address given. This will NOT be the same address you used to SSH to the server (this is an internal-only address, while the IP you used to connect is the public IP); however, for the rest of the Hadoop setup, you should be using the internal IPs instead of the public IPs. The only time you should use the public IPs is when you're connecting from your local machine to one of the cluster machines.

3. First we need to download the Hadoop software. Unfortunately, this isn't quite as simple as running a one-line yum command. **On each of your machines,** download the Hadoop files using wget, unpack the archive, and store it at /usr/local/hadoop:

```
$ wget https://archive.apache.org/dist/hadoop/core/hadoop-1.2.1/
hadoop-1.2.1.tar.gz
$ tar xzvf hadoop-1.2.1.tar.gz
$ sudo mv hadoop-1.2.1 /usr/local/hadoop
```

While you're logged into each of your machines, you should check to make sure that the local disks are recognized (this is to make sure I provisioned your server correctly!). Run **lsblk** to see the attached drives: you should see a disk entry of about 160 GB (as

well as a smaller 8 GB drive for the main system files and a third entry for swap space). If the ~160 GB entry is not there, let me know.

4. You need to configure your machines so that you can SSH between them (this is required by Hadoop). Currently you are able to SSH into any of your machines, but once logged in you cannot SSH to a different machine. As we have done so far, we will configure SSH logins using your private key. You need to copy your private key file from your local machine into the **.ssh** directory within your home directory **on each of your Hadoop machines**. We'll also save it as the default name that SSH expects (**id_rsa**):

```
$ scp -i sb-keypair sb-keypair sb@1.2.3.4:~/.ssh/id_rsa
```

5. Then, we need to modify the SSH settings to allow root logins. Unfortunately (or perhaps fortunately), Linux makes this rather difficult for security's sake. **Be careful with these steps, as errors could lock you out of your machine entirely.**

Copy your private key file (id_rsa) to the root user, as well as your authorized_keys user controlling who is allowed to login. **Repeat this step on all your machines**.

```
$ sudo cp ~/.ssh/id_rsa /root/.ssh/
$ sudo cp ~/.ssh/authorized_keys /root/.ssh/
```

Next, open up the sshd (SSH server daemon) configuration file, also with sudo:

```
$ sudo nano /etc/ssh/sshd_config
```

Scroll down to the PermitRootLogin entry and you should see a snippet like this:

```
#PermitRootLogin yes
# Only allow root to run commands over ssh, no shell
PermitRootLogin forced-commands-only
#StrictModes yes
```

Change this by **commenting** the "forced-commands-only" line and **uncommenting** the "PermitRootLogin yes" line, i.e., change it to:

```
PermitRootLogin yes
# Only allow root to run commands over ssh, no shell
#PermitRootLogin forced-commands-only
#StrictModes yes
```

Save this file and quit your editor, and now restart the SSH server for the changes to take effect:

```
$ sudo service sshd restart
```

Remember to do all this entire sequence (step 5) on all of your machines!

6. Pick one of your 4 machines to configure as the Hadoop master. We're going to get all of the Hadoop configuration files setup on the master, then just copy them to the rest of your cluster. Make a note of the IP of the master.

7. On your chosen master, open up /usr/local/hadoop/conf/slaves in your favorite editor. You need to change this file to contain the (private) IP addresses of all your machines (1 per line). Remove all other lines. For example:

```
1.2.3.4
2.3.4.5
3.4.5.6
4.5.6.7
```

8. Edit the masters file in the same directory as the slaves file so that it contains just the IP address of the master, e.g.:

```
1.2.3.4
```

9. Create a directory for Hadoop to store most of its data. Most importantly, this directory is where Hadoop will be storing the local data blocks of the HDFS (Hadoop Distributed File System). Since we may be working with a lot of data, it's important to store this data on the larger attached disk as opposed to the small system drive:

```
$ sudo mkdir /media/ephemeral0/hadoop
```

Note: If you ever shut down or reboot your machine, the contents of this directory will be wiped out!

9. Now we have to change some configuration files in the same /usr/local/hadoop/conf directory. Insert the following XML for the following 3 files (leaving the xml header intact), but replacing the placeholder IP with your actual master (private) IP:

core-site.xml:

```
<configuration>
    <property>
        <name>hadoop.tmp.dir</name>
```

```
        <value>/media/ephemeral0/hadoop/tmp</value>
    </property>
    <property>
        <name>fs.default.name</name>
        <value>hdfs://1.2.3.4:9000</value>
    </property>
</configuration>
```

mapred-site.xml:

```
<configuration>
    <property>
        <name>mapred.job.tracker</name>
        <value>hdfs://1.2.3.4:9001</value>
    </property>
</configuration>
```

hdfs-site.xml:

```
<configuration>
    <property>
        <name>dfs.replication</name>
        <value>2</value>
    </property>
</configuration>
```

10. Copy the configuration files out to the rest of your cluster. **For each of your slave nodes,** copy the files from the master:

```
$ scp -r /usr/local/hadoop/conf/* 1.2.3.4:/usr/local/hadoop/conf/
```

11. Before we actually start the HDFS cluster, run a quick test to make sure that the master can talk to the slaves. From within your root shell (on the master), try SSHing to each of your machines (including the master itself -- i.e., SSHing from the master to itself). Remember to use the internal IPs and not the public IPs. If the login succeeds, just run 'exit' to get back to the master shell and then try the next machine. If you get a "Permission denied" error when connecting to any of your machines, you probably made a mistake during step #5 for that machine, so go back and double check.

12. Almost done! First we need to start a new distributed file system across our cluster. **On the master**, open up a new shell as the root user and start the Hadoop "NameNode", which is the central controller of a HDFS filesystem that maintains the directory tree and locations of data across the cluster.

```
$ sudo bash
# /usr/local/hadoop/bin/hadoop namenode -format
```

(Note: every command you run in a "sudo" shell is run with root permissions, i.e., the equivalent of running the command with sudo)

If you're thinking that the NameNode seems rather undistributed of Hadoop, you're right! More recent versions of Hadoop provide failover capabilities for the NameNode to minimize the dependency on a single server.

If the above command works, it will run for a few seconds and then exit with a success message (having formatted the distributed filesystem).

13. Assuming all the connections worked, go back to the master, make sure no java processes are running, then start the Hadoop daemons. On the master, run:

```
# killall -9 java
# /usr/local/hadoop/bin/start-dfs.sh
# /usr/local/hadoop/bin/start-mapred.sh
```

If everything worked, you should be able to see the various Java components of Hadoop running using **jps** -- on the master, this should show something like:

```
29888 DataNode
30087 JobTracker
29995 SecondaryNameNode
30203 TaskTracker
30244 Jps
```

On the slaves, running jps should show something like this:

```
29671 TaskTracker
29732 Jps
29564 DataNode
```

If you aren't seeing a DataNode and TaskTracker on each node, you most likely have an error in your configuration files (or forgot to copy them across all machines, etc). Hadoop logfiles are written to /usr/local/hadoop/logs/, which may tell you something about what went wrong.

14. If everything's running, you can test the distributed filesystem now. First let's copy some files into the filesystem. Let's just use the files in our conf directory:

```
# /usr/local/hadoop/bin/hadoop dfs -put /usr/local/hadoop/conf input
```

This command put our directory into HDFS, which we can then view:

```
# /usr/local/hadoop/bin/hadoop dfs -ls /user/root/input
```

15. Now let's run one of the examples provided. Here's a distributed grep example that greps for a pattern in the input directory we just stored:

```
# /usr/local/hadoop jar /usr/local/hadoop/hadoop-examples-1.2.1.jar
grep input output 'dfs[a-z.]+'
```

This command will run a MapReduce job and print out status information as it goes.

16. Poke around in the logs directory, which should have a bunch of files dumped by the job we just ran.

17. To see the actual output files of the job, we can fetch them from HDFS back into the local filesystem and then view them:

```
# /usr/local/hadoop/bin/hadoop dfs -get output output
# cat output/*
```

Or we can just view them directory from HDFS:

```
# /usr/local/hadoop/bin/hadoop dfs -cat output/*
```

Hopefully, the output contents match the pattern we grepped for!

18. To shut down the MapReduce cluster, we need to stop both both daemons we started:

```
# /usr/local/hadoop/bin/stop-mapred.sh
# /usr/local/hadoop/bin/stop-dfs.sh
```

General tips on using the cluster:

The cluster machines are unreliable, not backed up, and data may be lost if the machines even just restart. If you are curious, the machines are actually **virtual machines** (VMs) running in an Amazon-operated datacenter in Virginia, and are running on hardware shared by many other users. As such, make sure you are not leaving important files (e.g., code) on only the cluster machines.

You will likely save a lot of typing by either cding to /usr/local/hadoop to run your commands or by adding /usr/local/hadoop/bin to your PATH in .bashrc.

The cluster setup is a one-time procedure (assuming your machines are not subsequently inadvertently corrupted)! Once Hadoop is set up, you can easily start and stop the cluster using the two commands above.

Your cluster is unlikely to surprise you with its blazing speed, seeing as it is composed of only four relatively low-powered machines. Remember that real MapReduce clusters are generally comprised of hundreds to thousands of more powerful machines! However, the technology underlying your 'toy' cluster is exactly the same as could be run on a real cluster.