CSC 463: Project Assignment Four

John Rosso

I. INTRODUCTION

In this report I will go over my implementation of a hadoop cluster system using three CentOs 7 virtual machines. As well as testing the hadoop cluster using a custom written map reduce program that counts the words of input files. As mentioned the hadoop cluster will run using three CentOs 7 machines, each running through Oracle's VirtualBox program. The custom map reduce program will be written using python. As for the report format I will start with this small introduction section. Then the implementation section which will go over the cluster setup and include screenshots of the system finished and running. Lastly the report will end with the test section where I will run the custom map reduce program and talk about the results.

II. IMPLEMENTATION

To start setting up the hadoop cluster system I first need to create a CentOs 7 virtual machine, to act as the master node for this hadoop cluster. The hadoop cluster will work with a master node to manage and two worker nodes for work, node1 and node2. Once I have the master node running I edit the network file to configure a static IP address for this node. I then edit the hosts and hostname file to give this node a convenient name, master, and tell this node which machines to look for on the local network.

Now that the master node is ready for configuration I can start by installing some needed tools to run the hadoop programs. The first being a devel version of java runtime, this is what will run the hadoop program. Next is the wget tool that will allow me to easily download the hadoop program files. Both of these tools are downloaded using the yum package installer.

We now have the needed tools to run our hadoop program and we can begin to set up the necessary operating system configurations. First we need to set up a hadoop user and a password for this user. To do this we can simply run the useradd and passwd commands. Once our user is set up and we have switched to this user we can then set up ssh keys to allow us to easily communicate and connect to the other two nodes we will use. To do this we generate a rsa ssh key and save this key in an authorized_keys file that allows us to remote connect to the master node’s hadoop user without needing the password.

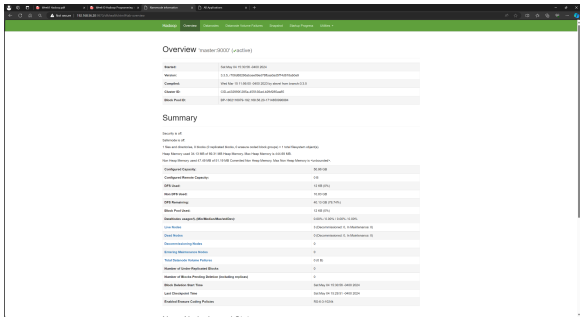
After we have the operating system configured we can now begin to deploy the hadoop program. To start this process we need to install the hadoop program files, which we can do using the hadoop download link and the wget tool we installed earlier. Once we have the install zip file we just need to unzip the file using tar and rename the install directory to something convenient like hadoop.

With the hadoop program installed, we can start the configuration process for the hadoop program. First we need to set the file paths for the hadoop program. We do this inside the .bashrc file where we can set variables with names and file paths. After we set these we then need to locate the location of the java runtime we installed earlier. To find this location we can run the “alternatives –display java command” and copy the path that is given. With this file path we can edit the JAVA_HOME variable inside the hadoop-env.sh file located inside the hadoop program directory.

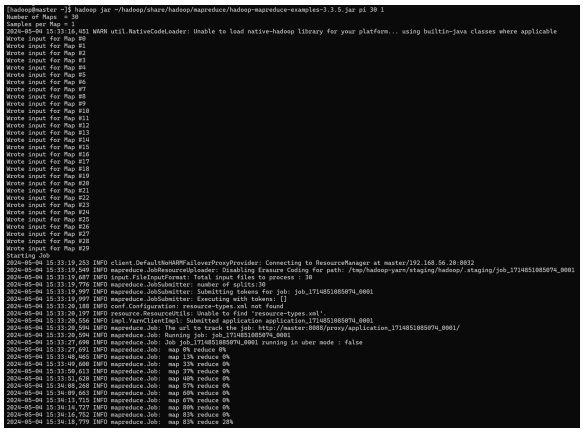
Now that we have set the files path for both the hadoop program and the java runtime we can begin to configure our hadoop cluster by editing a few configuration files. The first being the core-site.xml file. Here we are defining the master node that will manage the cluster. Next is the hdfs-site.xml file, where we will configure where the file system will be stored. After that is the mapred-site.xml file, here we will define the map-reduce system's amount of memory to use from each worker node. Lastly we need to edit the yarn-site.xml file where we define the maximum amount of memory each worker node will give to the cluster system. Now that we have our system configured we just need to define the worker nodes in the worker file and we should be good to go for our master node.

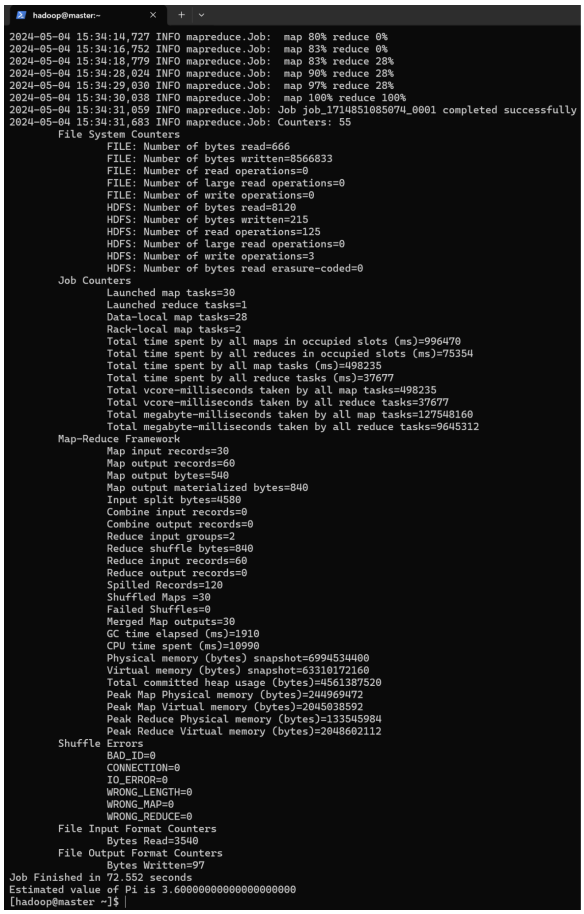
Once we have configured the master node to this point we now need to create the worker nodes. To do this we simply shut down the master node and clone two new virtual machines from the master node. Since we already installed and configured everything on the master node we only need to change the names and IP addresses of these new nodes. Which can be done by editing the network and hostname file.

Finally we are ready to fully deploy and run our hadoop cluster system. After turning the firewall off on each node we run the following command on the master node, “hdfs namenode -format”, this formats all our configuration settings and creates the file system for the cluster. Next we run the following two commands, “start-dfs.sh” & “start-yarn.sh”. This will begin to start the cluster system which I can confirm by going to my web browser and typing the IP address of my master node followed by :9870, where I see the following.



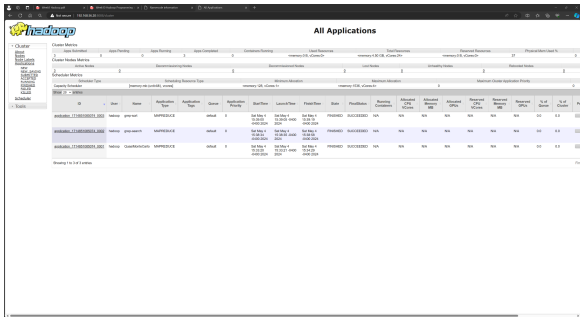
The screenshot above is the homepage for my hadoop cluster system where I can see that three nodes are live in the system and ready to be used. If I try running a test using the hadoop map-reduce examples I can see the following happening within my master node.





```
hadoop@master:~

2024-05-04 15:34:14,727 INFO mapreduce.Job:  map 80% reduce 0%
2024-05-04 15:34:16,752 INFO mapreduce.Job:  map 83% reduce 0%
2024-05-04 15:34:18,779 INFO mapreduce.Job:  map 83% reduce 28%
2024-05-04 15:34:28,024 INFO mapreduce.Job:  map 90% reduce 28%
2024-05-04 15:34:29,030 INFO mapreduce.Job:  map 97% reduce 28%
2024-05-04 15:34:30,038 INFO mapreduce.Job:  map 100% reduce 100%
2024-05-04 15:34:31,059 INFO mapreduce.Job: Job job_1714851085074_0001 completed successfully
2024-05-04 15:34:31,683 INFO mapreduce.Job: Counters: 55
        File System Counters
                FILE: Number of bytes read=666
                FILE: Number of bytes written=8566833
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=8120
                HDFS: Number of bytes written=215
                HDFS: Number of read operations=125
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=3
                HDFS: Number of bytes read erasure-coded=0
        Job Counters
                Launched map tasks=30
                Launched reduce tasks=1
                Data-local map tasks=28
                Rack-local map tasks=2
                Total time spent by all maps in occupied slots (ms)=996470
                Total time spent by all reduces in occupied slots (ms)=75354
                Total time spent by all map tasks (ms)=498235
                Total time spent by all reduce tasks (ms)=37677
                Total vcore-milliseconds taken by all map tasks=498235
                Total vcore-milliseconds taken by all reduce tasks=37677
                Total megabyte-milliseconds taken by all map tasks=1275548160
                Total megabyte-milliseconds taken by all reduce tasks=9645312
        Map-Reduce Framework
                Map input records=30
                Map output records=60
                Map output bytes=540
                Map output materialized bytes=840
                Input split bytes=4580
                Combine input records=0
                Combine output records=0
                Reduce input groups=2
                Reduce shuffle bytes=840
                Reduce input records=60
                Reduce output records=0
                Spilled Records=120
                Shuffled Maps =30
                Failed Shuffles=0
                Merged Map outputs=30
                GC time elapsed (ms)=1910
                CPU time spent (ms)=10990
                Physical memory (bytes) snapshot=6994534400
                Virtual memory (bytes) snapshot=63310172160
                Total committed heap usage (bytes)=4561387520
                Peak Map Physical memory (bytes)=244969472
                Peak Map Virtual memory (bytes)=2045038592
                Peak Reduce Physical memory (bytes)=133545984
                Peak Reduce Virtual memory (bytes)=2048602112
        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=3540
        File Output Format Counters
                Bytes Written=97
Job Finished in 72.552 seconds
Estimated value of Pi is 3.60000000000000000000
[hadoop@master ~]$
```

The screenshots above show the cluster working to complete the map-reduce task and finishing with an incorrect estimate of the number PI, but still, the system worked as it should have. I can also utilize the job tracker web page to display jobs the cluster is and has run, like seen in the following screenshot.



This shows my cluster has three jobs in its history of running. It displays when they stated, when they finished, and other relevant information if provided.

Now that the cluster is running, able to perform tasks, and displays information about itself. I can begin to test if the system can run a custom written map-reduce program.

III. TEST

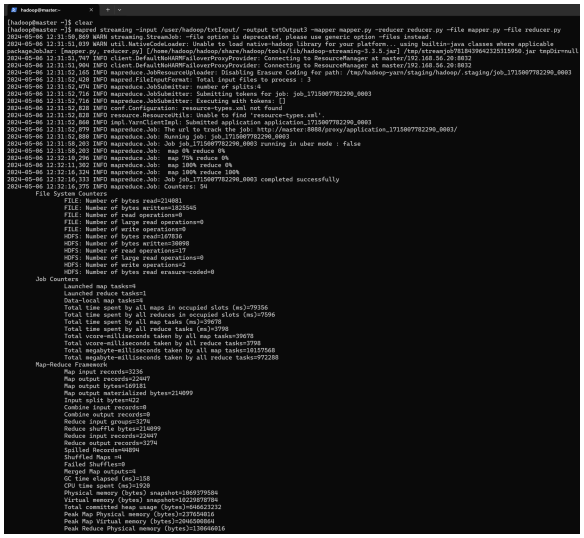
For this test I need to create a map-reduce program that can take an input file, separate the words inside the file, and count each iteration of a word. To do this I use python and create the following scripts.

```python
#!/usr/bin/env python
import sys

special = ("~`!@#$%^&*()_-+={[]}|\\:;'\",<.>/?0123456789")

for line in sys.stdin:
    line = line.strip()
    line = line.strip("~`!@#$%^&*()_-+={[]}|\\:;'\",<.>/?0123456789")
    for c in special:
        line = line.replace(c, ' ')

    words = line.split()

    for word in words:
        #print '%s\t%s' % (word, "1")
        print(word, "1")
```

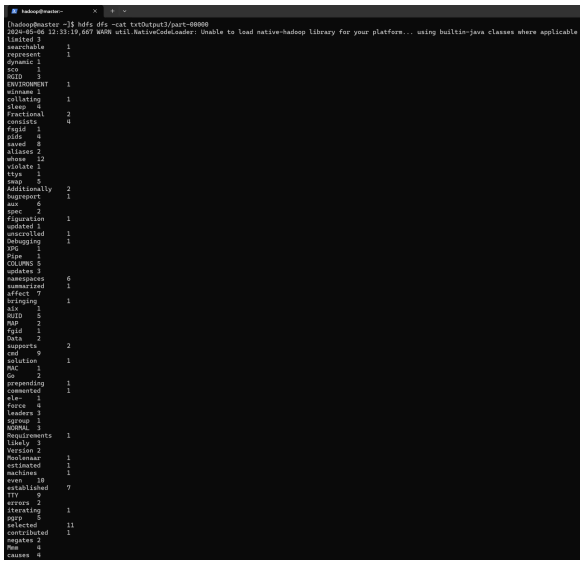
```python
#!/usr/bin/env python
import sys

wordCount = {}

for line in sys.stdin:
    line = line.strip()
    word, count = line.split('\t', 1)

    try:
        count = int(count)
    except ValueError:
        continue

    try:
        wordCount[word] = wordCount[word] + count
    except:
        wordCount[word] = count

    for word in wordCount.keys():
        #print '%s\t%s' % (word, wordCount[word])
        print(word, wordCount[word])
```

The first screenshot is the code for the mapper where I separate the words from special characters and numbers. Then the next screenshot is the reducer where I take each count of a word from the mapper program and set up a key value pair for each word and its iteration number. This is then printed out to the system.

Now that I have my map-reduce program set up, I need to create some input files to test my program out on. I do this by taking the manual for the ps, top, and vi commands and place them inside a .txt file. I then upload these files into the cluster filesystem under a directory called txtInput. Next I try to run the custom map-reduce program using the mapred command where I specify the map and reduce files as well as the input files and where to store the output. After I run the command I am shown that the program is running as seen here.



I then wait for the program to finish and check the results, which are displayed below.



Here we can see that the program ran as it should by separating the words in each file and counting the iterations for each.

With this I have now fully set up a hadoop cluster system using three nodes, a master and two workers. I have made sure the system works by testing the cluster with a custom made map-reduce program and I have shown you my step by step process and the results for each test.