**ENGR 516 ECC - Assignment 1**

By

Shashwati ([sdiware@iu.edu](mailto:sdiware@iu.edu))

* **Hadoop Installation:**

Below are the steps which I performed for Hadoop installation:

* Step 1: Created instance on Jetstream with Ubuntu to host Hadoop.
* Step 2: Updated JDK 11 as a prerequisite for running Hadoop.
* Step 3: Downloaded Hadoop 3.3.6.
* Step 4: I updated configuration files such as hdfs-site.xml, mapred-site.xml, they contain important settings for Hadoop’s distributed file system and MapReduce framework.
* Step 5: I setup environment variables such as JAVA\_HOME, HADOOP\_HOME in .bashrc.
* Step 6: I started Hadoop with start-all.sh script. This script includes NameNode, DataNode, NodeManager, ResourceManager startup process.

A screen shot of a computer program

Description automatically generated

Fig: Start-all.sh Output

A screen shot of a computer program

Description automatically generated

Fig: All process Up and Running

A computer screen shot of text

Description automatically generated

Fig: Environment variables setup in .bashrc

A screenshot of a computer

Description automatically generated

Fig: Hadoop UI

A screenshot of a computer

Description automatically generated

Fig Hadoop Cluster

* **PART 1. Output the top-3 IP addresses with the granularity of an hour**

**Mapper (Map\_granularityHour.py):**

* Below mapper code searches and extracts IP addresses and hours from input log file.
* For each match it prints the hour and IP along 1 to indicate the count which is passed to reducer.

A computer screen with text on it

Description automatically generated

Fig: Mapper (Map\_granularityHour.py)

**Reducer (Reduce\_Part1\_2.py):**

* It aggregates the count of each IP address for specific hour, then sorts the IP addresses according to their count in descending order.
* Finally prints the top 3 Ips with their count for each hour.

A computer screen shot of a program code

Description automatically generatedFig: Reducer (Reduce\_Part1\_2.py)

**Command to run Map Reduce:**

*Hadoop jar $HADOOP\_HOME/share/adoop/tools/lib/adoop-streaming-3.3.6.jar -files Map\_granularityHour.py,Reduce\_Part1\_2.py -mapper ‘python3 Map\_granularityHour.py’ -reducer ‘python3 Reduce\_Part1\_2.py’ -input /inputFolder/sample.log -output /outputFolder\_part1\_granularityHour*

**Logs:**

A computer screen with white and blue lines

Description automatically generated with medium confidence

Fig: Logs

**Command to check output*:*** *hdfs dfs -cat /outputFolder\_part1\_granularityHour/part-00000*

**Final Output*:***

A screenshot of a computer screen

Description automatically generated

Fig: Final Output

* **PART 2.1: Make your program like a database search. Your program should be able to accept parameters from users, such as 0-1, which means from time 00:00 to 01:00, and output the top-3 IP addresses in the given time period.**

**Mapper (Map\_databaseSearch.py):**

* I have modified the mapper to accept command line argument called time window.
* It is extracting the start hour and end hour from the time window mentioned in command line.
* And based on this, it filters and print entries that fall within the specified time window.

A computer screen shot of a code

Description automatically generated

Fig: Mapper (Map\_databaseSearch.py)

**Reducer (Reduce\_Part1\_2.py):**

* Reducer for this part is the same as part 1.
* I have taken two test cases; first time window is 01 to 02 which has no IP addresses and second time window is 01 to 04 which has IP addresses.

**Test case 1 Time Window 01 to 02:**

**Command to run Map Reduce:**

*hadoop jar $HADOOP\_HOME/share/hadoop/tools/lib/hadoop-streaming-3.3.6.jar -files Map\_databaseSearch.py,Reduce\_Part1\_2.py -mapper 'python3 Map\_databaseSearch.py' -reducer 'python3 Reduce\_Part1\_2.py' -input /inputFolder/sample.log -output /outputFolder\_part2\_databaseSearch\_test1 -****cmdenv timewindow="01-02"***

**Logs:**

A black screen with white and blue dots

Description automatically generated

Fig: Logs

**Command to check output:** *hdfs dfs -cat /outputFolder\_part2\_databaseSearch\_test1/part-00000*

**Final Output:**



Fig: Final Output

**Test Case 2 Time Window 01 to 04:**

**Command to run Map Reduce:**

*hadoop jar $HADOOP\_HOME/share/hadoop/tools/lib/hadoop-streaming-3.3.6.jar -files Map\_databaseSearch.py,Reduce\_Part1\_2.py -mapper 'python3 Map\_databaseSearch.py' -reducer 'python3 Reduce\_Part1\_2.py' -input /inputFolder/sample.log -output /outputFolder\_part2\_databaseSearch\_test2* ***-cmdenv timewindow="01-04"***

**Logs:**

A screen shot of a computer

Description automatically generated

Fig: Logs

**Command to check output*:*** *hdfs dfs -cat /outputFolder\_part2\_databaseSearch\_test2/part-00000*

**Final Output:**

A screenshot of a computer screen

Description automatically generated

Fig: Final Output

* **PART 2.2: Run it along with three other examples, WordCount, Sort, Grep, at the same time, and test fair and capacity schedulers.**

**Running all task:**

* To run all the task in parallel that are wordcount, sort, database search and grep I developed a script called runAll.sh.
* I ran these jobs individually and added them in script with & at the end to make it run in background and concurrently.
* I have also added individual mapper and reducer code and output for each of the above task.
* I have added each queue name in each command with the flag -D mapreduce.jobs.queuename. Below is the screenshot of runAll.sh.

**A computer screen shot of text

Description automatically generated**

**Fig: runAll.sh**

* **Sort Task:**

**Mapper Sort (Map\_sort.py):**

* It extracts the first word as IP and remaining words as string.
* It then prints IP along with rest words in tab-separated format and sends it to reducer.

A computer screen with white text

Description automatically generated

Fig: Mapper Sort (Map\_sort.py)

**Reducer Sort (Reduce\_sort.py):**

* It stores log entries in dictionary and then sorts the IP address and prints it alongside the corresponding log entries**.**

A computer screen with text and numbers

Description automatically generated

Fig: Reducer Sort (Reduce\_sort.py)

**Command to run Map Reduce:**

*hadoop jar $HADOOP\_HOME/share/hadoop/tools/lib/hadoop-streaming-3.3.6.jar -files Map\_sort.py,Reduce\_sort.py -mapper 'python3 Map\_sort.py' -reducer 'python3 Reduce\_sort.py' -input /inputFolder/sample.log -output /outputFolder\_sort*

**Command to check output*:*** *hdfs dfs -cat /outputFolder\_sort/part-00000*

**Final Output:**

* It shows logs sorted according to IP address.

A black background with white and blue lines

Description automatically generated

Fig: Final Output

* **Word Count Task:**

**Mapper Word Count (Map\_wc.py):**

* It uses dictionary to store word counts and sends the word counts to reducer.

A computer screen shot of a black screen with colorful text

Description automatically generated

Fig Mapper Word Count (Map\_wc.py)

**Reducer Word Count (Reduce\_wc.py):**

* It processes word count pairs and consolidates the counts for each word.
* It sums the count for each word and when word changes it prints the word and count.

A screen shot of a computer code

Description automatically generated

Fig: Reducer Word Count (Reduce\_wc.py)

**Command to run Map Reduce:**

*hadoop jar $HADOOP\_HOME/share/hadoop/tools/lib/hadoop-streaming-3.3.6.jar -files Map\_wc.py,Reduce\_wc.py -mapper 'python3 Map\_wc.py' -reducer 'python3 Reduce\_wc.py' -input /inputFolder/sample.log -output /outputFolder\_wc*

**Command to check output*:*** *hdfs dfs -cat /outputFolder\_wc/part-00000*

**Final Output:**

* It shows word count of each word from the log file.

A screenshot of a computer program

Description automatically generated

Fig: Final Output

* **Grep Task:**

**Mapper Grep (Map\_grep.py):**

* It takes the input for specific search pattern from user via command line and checks if it matches.
* If the line matches the search pattern it sends it to reducer.

A computer screen shot of a black background with white text

Description automatically generated

Mapper Grep (Map\_grep.py)

**Reducer Grep (Reduce\_grep.py):**

* It prints out the line which it receives from the mapper after filtering according to search pattern.

A black screen with red lines

Description automatically generated

Reducer Grep (Reduce\_grep.py)

**TestCase 1 “Dual” as search pattern:**

**Command to run Map Reduce:**

*hadoop jar $HADOOP\_HOME/share/hadoop/tools/lib/hadoop-streaming-\*.jar -files Map\_grep.py,Reduce\_grep.py -mapper 'python3 Map\_grep.py' -reducer 'python3 Reduce\_grep.py' -input /inputFolder/sample.log -output /outputFolder\_grep -cmdenv search\_pattern="Dual"*

**Command to check output*:***hdfs dfs -cat /outputFolder\_grep/part-00000

**Final Output for Test Case 1:**

A black background with many small colored lines

Description automatically generated

Fig: Final Output

**Test Case 2 “telegram” as search pattern:**

**Command to run Map Reduce:**

*hadoop jar $HADOOP\_HOME/share/hadoop/tools/lib/hadoop-streaming-\*.jar -files Map\_grep.py,Reduce\_grep.py -mapper 'python3 Map\_grep.py' -reducer 'python3 Reduce\_grep.py' -input /inputFolder/sample.log -output /outputFolder\_grep\_test2 -cmdenv search\_pattern="telegram"*

**Command to check output*:*** *hdfs dfs -cat /outputFolder\_grep\_test2/part-00000*

**Final Output:**

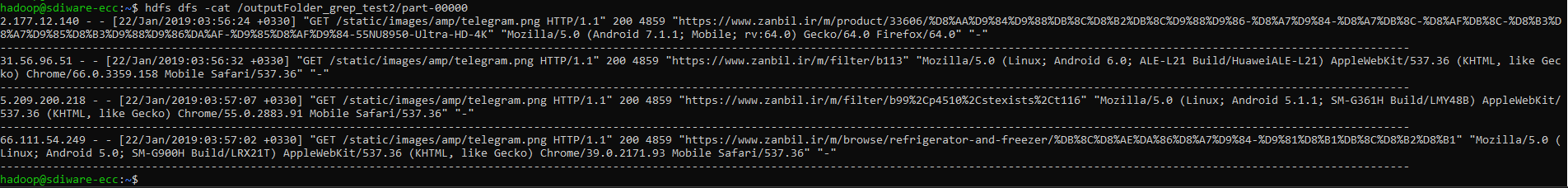


Fig: Final Output

* **Testing the Fair Scheduler:**
* To test the fair scheduler, I added below properties in $HADOOP\_HOME/yarn-site.xml. It is fair scheduler module which is required to change the scheduler to fair.

A computer screen with white text

Description automatically generated

Fig: yarn-site.xml for fair scheduler

* I also created fair-sheduler.xml file and allocated below queues in it.

1. Database Search
2. Word Count
3. Sort
4. Grep

A computer screen shot of a program

Description automatically generated

Fig: fair-scheduler.xml

* Below are the queues which are created in Hadoop.

A screenshot of a computer

Description automatically generated

Fig: Initial Queues in Hadoop

* In fair-scheduler.xml, I have given more weight database search as it was consuming more resources and initially it failed hence, I increased the weight.
* In the below screenshot, we can see DatabaseSearch queue is running first.

A screenshot of a computer

Description automatically generated

Fig: DatabaseSearch Queue running first

* I also gave name WordCount slightly less weight than DatabaseSearch but more than other two and hence it is running on second number as seen in below screenshot.

A screenshot of a computer

Description automatically generated

Fig: WordCount queue running second

* I gave equal weights to both Sort and Grep as they were relatively less bulky and executed fast. As seen in below screenshot they are running concurrently.

A screenshot of a computer

Description automatically generated

Fig: Grep and Sort queue running concurrently

* **Testing the Capacity Scheduler:**
* To implement capacity scheduler, I modified capacity-scheduler.xml.
* I have configured databaseSearch, wordcount, sort, grep queues in my capacity-scheduler.xml as shown in below screenshot.

A computer screen shot of a program

Description automatically generated

Fig: Capacity-scheduler.xml with sort, grep, wordcount and DatabaseSearch Queue

A screenshot of a computer

Description automatically generated

Fig: Capacity Scheduler with sort, grep, wordcount and DatabaseSearch Queue

* Soon I noticed from below that queues reached at max capacity and got stuck hence I implemented priority handling in fair scheduling.
* I saw the queues executed smoothy in fair scheduling.

**A screenshot of a computer

Description automatically generated**

* **Conclusion:**

I got to learn many topics related to scheduling and map-reduce framework along with Hadoop.

* **Reference:**

1. [How to Install Apache Hadoop on Ubuntu 22.04 – TecAdmin](https://tecadmin.net/how-to-install-apache-hadoop-on-ubuntu-22-04/)