ENGR 516 ECC - Assignment 1

By

Shashwati (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.

```
hadoop@sdiware-ecc:~$ start-all.sh
WARNING: Attempting to start all Apache Hadoop daemons as hadoop in 10 seconds.
WARNING: This is not a recommended production deployment configuration.
WARNING: Use CTRL-C to abort.
Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [sdiware-ecc]
Starting resourcemanager
Starting nodemanagers
hadoop@sdiware-ecc:~$
```

Fig: Start-all.sh Output

```
hadoop@sdiware-ecc:~$ jps
3696 SecondaryNameNode
4529 Jps
4066 NodeManager
3941 ResourceManager
3303 NameNode
3447 DataNode
hadoop@sdiware-ecc:~$
```

Fig: All process Up and Running

```
export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
export HADOOP_HOME=/home/hadoop/hadoop
export HADOOP_INSTALL=$HADOOP_HOME
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export HADOOP_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"
```

Fig: Environment variables setup in .bashrc

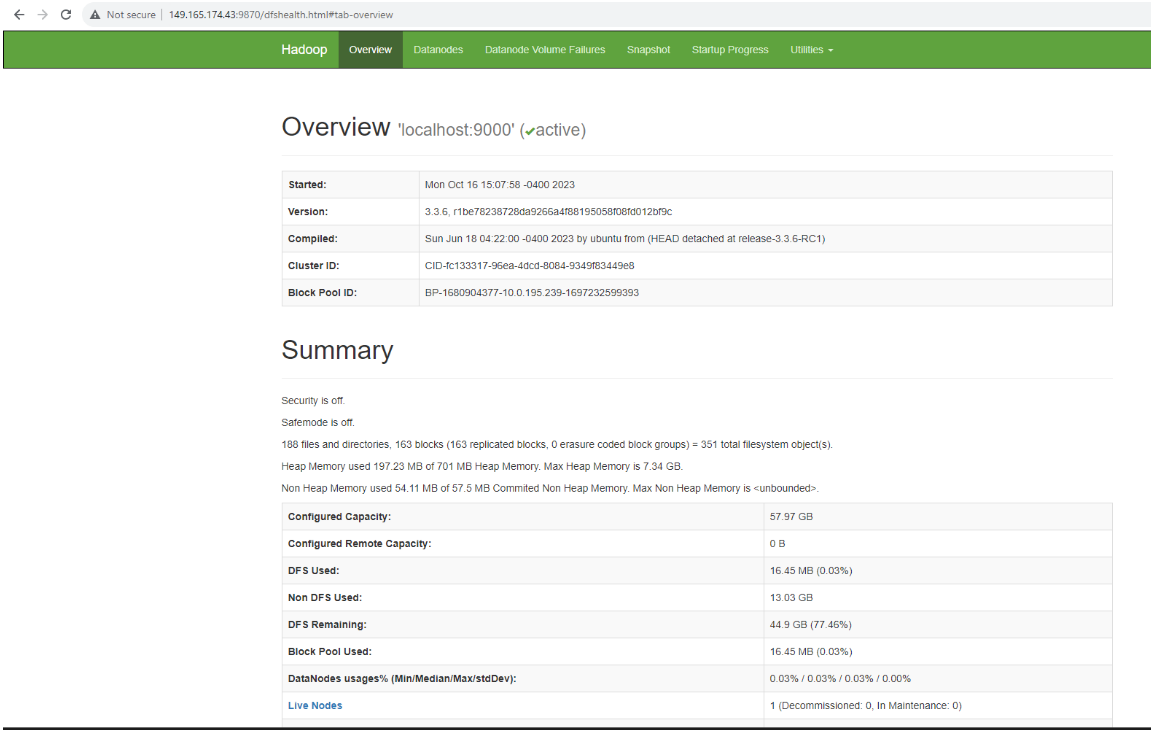


Fig: Hadoop UI

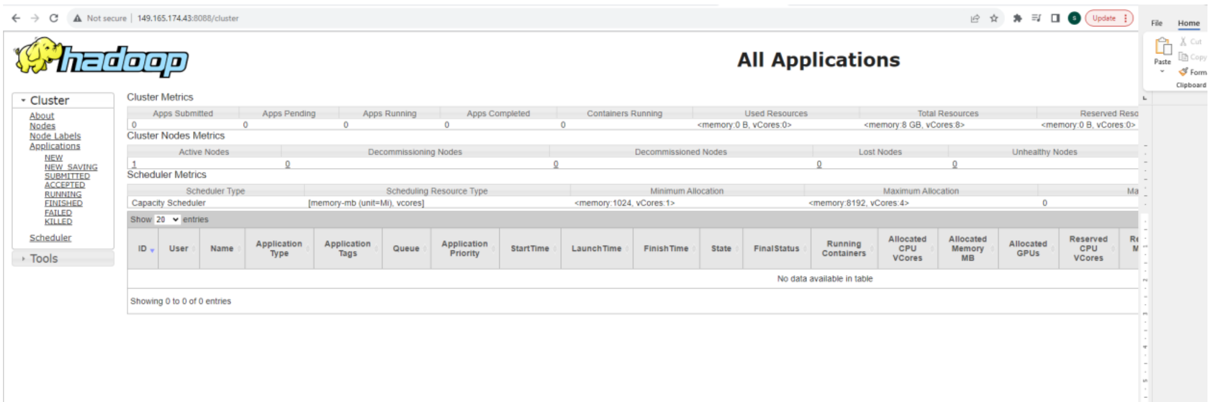


Fig Hadoop Cluster

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.

```
import re
import sys

# Regex given to extract IP and hour in the Assignment.
pattern = re.compile(r'(\d+\.\d+\.\d+\.\d+).*?(\d{4}:\d{2}):\d{2}')

for line in sys.stdin:
    patternFound = pattern.search(line)
    if patternFound:
        ip, hour = patternFound.groups()
        print(f'Hour: {hour}, IP: {ip}, 1')

~
~
~
~
~
~
```

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.

```
import sys
from collections import defaultdict

# To store hourly stats of IP address and their count
hourlyStats = defaultdict(lambda: defaultdict(int))

for line in sys.stdin:
    line = line.strip()
    hour, ip, count = line.split(',')
# Update the IP address with count of occurrences for specific hour
    hourlyStats[hour][ip] += int(count)

for hour, stats in hourlyStats.items():
    hourParts = hour.split(':')
    extractedHour = hourParts[2]
# Sorting in descending according the count and taking top 3 IP addresses
    topIPs = sorted(stats.items(), key=lambda x: x[1], reverse=True)[:3]
# Printing top IP adresses, count and Hour
    for ip, count in topIPs:
        print("-------------------------------------------------------")
        print(f"TOTAL COUNT: {count}, {ip}, Hour: {extractedHour}:00")
print("-------------------------------------------------------")
~
~
```

Fig: 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:

```
hadoop@sdiware-ecc:~$
hadoop@sdiware-ecc:~$ hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-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 /inputFo
lder/sample.log -output /outputFolder_part1_granularityHour
packageJobJar: [/tmp/hadoop-unjar1703074159230949628/] [] /tmp/streamjob1345808012063630l925.jar tmpDir=null
2023-10-16 04:48:21,368 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceManager at /0.0.0.0:8032
2023-10-16 04:48:21,476 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceManager at /0.0.0.0:8032
2023-10-16 04:48:21,643 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/hadoop/.staging/job_1697430851016_0001
2023-10-16 04:48:22,368 INFO mapred.FileInputFormat: Total input files to process : 1
2023-10-16 04:48:22,418 INFO mapreduce.JobSubmitter: number of splits:2
2023-10-16 04:48:22,559 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1697430851016_0001
2023-10-16 04:48:22,559 INFO mapreduce.JobSubmitter: Executing with tokens: []
2023-10-16 04:48:22,665 INFO conf.Configuration: resource-types.xml not found
2023-10-16 04:48:22,666 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2023-10-16 04:48:22,978 INFO impl.YarnClientImpl: Submitted application application_1697430851016_0001
2023-10-16 04:48:23,000 INFO mapreduce.Job: The url to track the job: http://sdiware-ecc:8088/proxy/application_1697430851016_0001/
2023-10-16 04:48:23,001 INFO mapreduce.Job: Running job: job_1697430851016_0001
2023-10-16 04:48:29,069 INFO mapreduce.Job: Job job_1697430851016_0001 running in uber mode : false
2023-10-16 04:48:29,070 INFO mapreduce.Job:  map 0% reduce 0%
2023-10-16 04:48:33,112 INFO mapreduce.Job:  map 50% reduce 0%
2023-10-16 04:48:34,117 INFO mapreduce.Job:  map 100% reduce 0%
2023-10-16 04:48:39,139 INFO mapreduce.Job:  map 100% reduce 100%
2023-10-16 04:48:39,146 INFO mapreduce.Job: Job job_1697430851016_0001 completed successfully
2023-10-16 04:48:39,210 INFO mapreduce.Job: Counters: 55
        File System Counters
                FILE: Number of bytes read=12201
                FILE: Number of bytes written=864983
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=106687
                HDFS: Number of bytes written=376
                HDFS: Number of read operations=11
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
                HDFS: Number of bytes read erasure-coded=0
        Job Counters
                Killed map tasks=1
                Launched map tasks=2
                Launched reduce tasks=1
                Data-local map tasks=2
                Total time spent by all maps in occupied slots (ms)=3728
                Total time spent by all reduces in occupied slots (ms)=1892
                Total time spent by all map tasks (ms)=3728
                Total time spent by all reduce tasks (ms)=1892
                Total vcore-milliseconds taken by all map tasks=3728
                Total vcore-milliseconds taken by all reduce tasks=1892
                Total megabyte-milliseconds taken by all map tasks=3817472
                Total megabyte-milliseconds taken by all reduce tasks=1937408
        Map-Reduce Framework
                Map input records=319
                Map output records=319
                Map output bytes=11557
                Map output materialized bytes=12207
                Input split bytes=192
                Combine input records=0
                Combine output records=0
                Reduce input groups=46
                Reduce shuffle bytes=12207
                Reduce input records=319
                Reduce output records=7
                Spilled Records=638
```

Fig: Logs

Command to check output*:* *hdfs dfs -cat /outputFolder_part1_granularityHour/part-00000*

Final Output*:*

```
hadoop@sdiware-ecc:~$ hdfs dfs -cat /outputFolder_part1_granularityHour/part-00000
-------------------------------------------------------
TOTAL COUNT: 38,  IP: 66.111.54.249, Hour: 03:00
-------------------------------------------------------
TOTAL COUNT: 36,  IP: 5.211.97.39, Hour: 03:00
-------------------------------------------------------
TOTAL COUNT: 31,  IP: 66.249.66.194, Hour: 03:00
-------------------------------------------------------
hadoop@sdiware-ecc:~$
```

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.

```
import re
import sys
import os

# Extracting starting hour and ending hour from env variable time window
startHour, endHour = map(int, os.environ.get('timewindow').split("-"))
pattern = re.compile(r'(\d+\.\d+\.\d+\.\d+).*?(\d{4}:\d{2}):\d{2}')

for line in sys.stdin:
    patternFound = pattern.search(line)
    if patternFound:
        ip, hour = patternFound.groups()
        hourParts = hour.split(':')
        _, _, extractedHour = hour.partition(':')
# Extract the hour as integer value
        extractedHour = int(extractedHour)
# Checked if the extracted hour is withing starting and ending hour window
        if startHour <= extractedHour < endHour:
            print('Hour:', hour, ', IP:', ip, ', 1', sep='')
        else:
            continue
```

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:

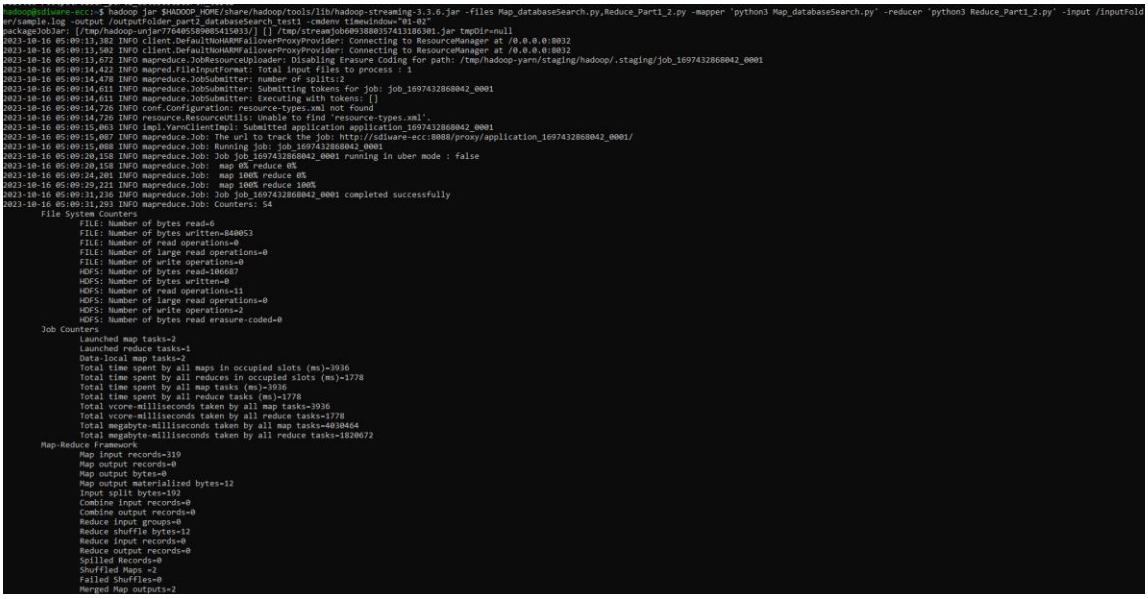


Fig: Logs

Command to check output: *hdfs dfs -cat /outputFolder_part2_databaseSearch_test1/part-00000*

Final Output:

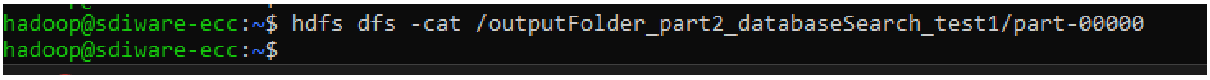


Fig: Final Output

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:

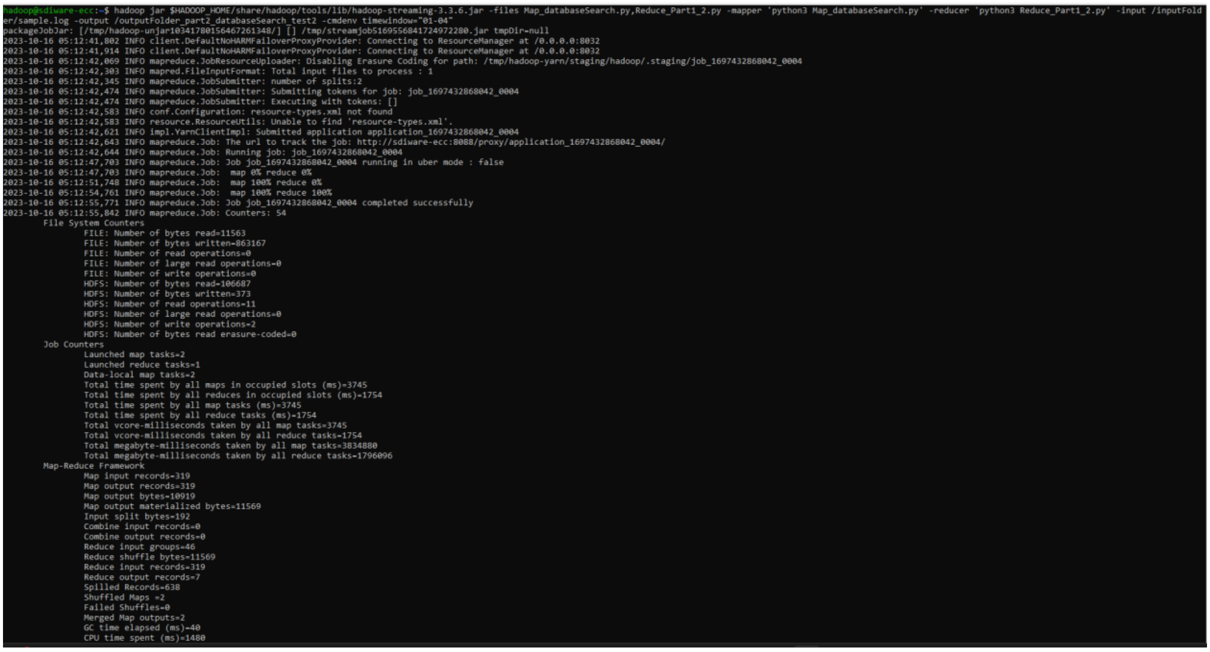


Fig: Logs

Command to check output*:* *hdfs dfs -cat /outputFolder_part2_databaseSearch_test2/part-00000*

Final Output:

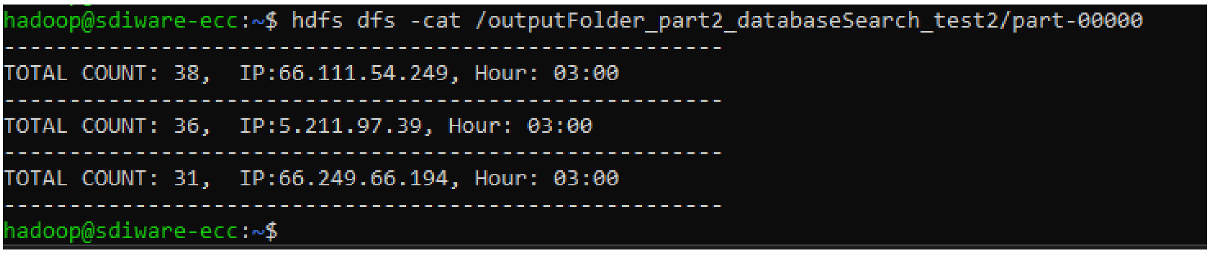


Fig: Final Output

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.

```
#!/bin/bash

# Database Search Job

hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-3.3.6.jar -D mapreduce.job.queuename=DatabaseSearch  -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_parallel -cmdenv timewindow="01-02" &

# Word Count Job

hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-3.3.6.jar -D mapreduce.job.queuename=WordCount  -files Map_wc.py,Reduce_wc.py -mapper 'python3 Map_wc.py' -reducer 'python3 Reduce_wc.py' -input /inputFolder/sample.log -output /outputFolder_wc_parallel &

# Sort Job
hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-3.3.6.jar -D mapreduce.job.queuename=Sort -files Map_sort.py,Reduce_sort.py -mapper 'python3 Map_sort.py' -reducer 'python3 Reduce_sort.py' -input /inputFolder/sample.log -output /outputFolder_sort_parallel &

# Grep Job

hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-*.jar -D mapreduce.job.queuename=Grep  -files Map_grep.py,Reduce_grep.py -mapper 'python3 Map_grep.py' -reducer 'python3 Reduce_grep.py' -input /inputFolder/sample.log -output /outputFolder_grep_parallel -cmdenv search_pattern="Dual" &

wait

echo "------------- All task completeted ------------"
~
```

Fig: runAll.sh

Sort:

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.

```
import sys


for line in sys.stdin:
    line = line.strip()
    words = line.split()
    if words:
        IPAddress, *rest = words
        print(f"{IPAddress}\t{' '.join(rest)}")
~
~
~
```

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.

```python
import sys

# Created a dictionary to store log entries by IP address
logEntries = {}

for line in sys.stdin:
    line = line.strip()
    ipAddress, logEntry = line.split('\t', 1)
# Check if IP address is already in the dictionary and if not add it with empty list
    if ipAddress not in logEntries:
        logEntries[ipAddress] = []
# Adding log entry to list with corresponding IP Address
    logEntries[ipAddress].append(logEntry)
sortedIps = sorted(logEntries.keys())
# Iterating through IP Address and printing the IP Address and their Log Entries
for ipAddress in sortedIps:
    for logEntry in logEntries[ipAddress]:
        print(f"{ipAddress} {logEntry}")
        print("------------------------------------------------------------------------------------------
~
~
```

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.

Fig: Final Output

Word Count:

Mapper Word Count (Map_wc.py):

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

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

import sys
import re

# Initialize a dictionary to store word counts
wordCount = {}

for line in sys.stdin:
    line = line.strip()
    words = line.split()
    words = re.findall(r'\b\w+\b|\.\w+\b', line)
    for w in words:
# Increment the word count in the dictionary
        wordCount[w] = wordCount.get(w, 0) + 1
# Output word counts to STDOUT
for w, c in wordCount.items():
     print(f'{w}\t{c}')
```

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.

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

import sys

currentWord = None
currentCount = 0

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

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

    if currentWord == word:
        currentCount += count
    else:
        if currentWord:
            print(f'{currentWord}\t{currentCount}')
            print("--------------------------------")
        currentWord = word
        currentCount = count
if currentWord:
    print(f'{currentWord}\t{currentCount}')
```

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.

```
sim     6
-------------------------------
site    1
-------------------------------
sport   1
-------------------------------
sports  1
-------------------------------
static  30
-------------------------------
stexists        8
-------------------------------
stove   1
-------------------------------
support 2
-------------------------------
t445    1
-------------------------------
t51     1
-------------------------------
tag     3
-------------------------------
telegram        4
-------------------------------
third   3
-------------------------------
tools   2
-------------------------------
torob   1
-------------------------------
updateVariation 8
-------------------------------
usqp    3
-------------------------------
v       3
-------------------------------
v1      5
-------------------------------
```

Fig: Final Output

Grep:

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.

```python
import sys
import re
import os

# Check if the 'search_pattern' environment variable is set
search_pattern = os.environ.get('search_pattern')
if not search_pattern:
    print("The 'search_pattern' environment variable is not defined")
    sys.exit(1)
patternFound = re.compile(search_pattern)
for line in sys.stdin:
    line = line.strip()
    if patternFound.search(line):
        print(line)
~
~
~
```

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.

```python
import sys

for line in sys.stdin:
    line = line.strip()
    print(line)
    print("-----------------------------------------------------------------
~
~
~
```

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:

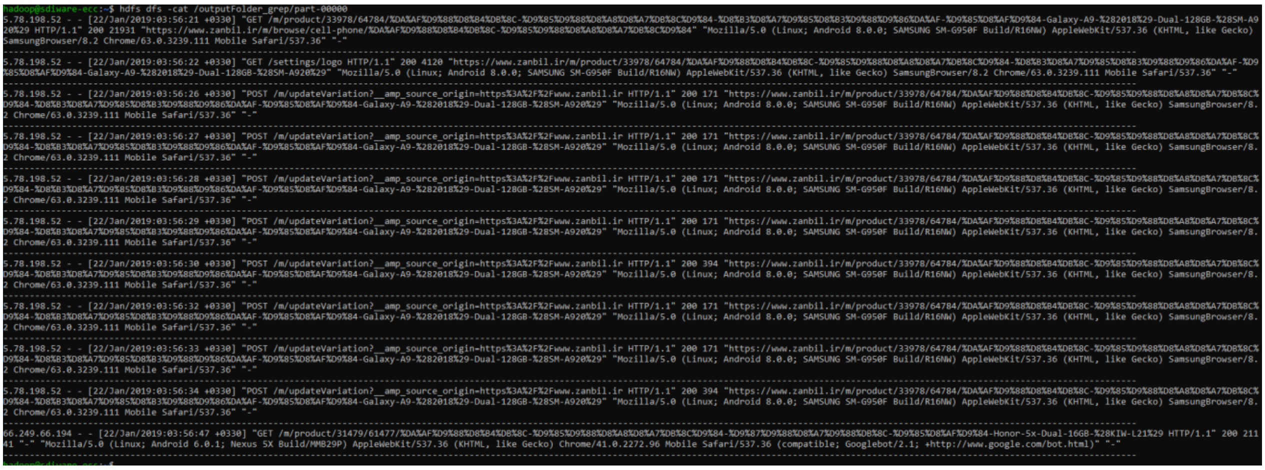


Fig: Final Output

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

<u>Testing the Fair Scheduler:</u>

→ 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.

```
<!-- YARN configurations for fair scheduling-->

<property>
<name>yarn.resourcemanager.scheduler.class</name>
<value>org.apache.hadoop.yarn.server.resourcemanager.scheduler.fair.FairScheduler</value>
</property>
<property>
<name>yarn.scheduler.fair.allocation.file</name>
<value>/home/hadoop/hadoop/etc/hadoop/fair-scheduler.xml</value>
</property>
</configuration>
```

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

```
<?xml version="1.0"?>
<allocations>
        <defaultQueueSchedulingPolicy>fair</defaultQueueSchedulingPolicy>
         <queue name="root">
                <queue name="DatabaseSearch">
                        <schedulingPolicy>fair</schedulingPolicy>
                        <weight>0.75</weight>
                </queue>
                <queue name="WordCount">
                        <schedulingPolicy>fair</schedulingPolicy>
                        <weight>0.50</weight>
                </queue>
                <queue name="Sort">
                        <schedulingPolicy>fair</schedulingPolicy>
                        <weight>0.25</weight>
                </queue>
                <queue name="Grep">
                        <schedulingPolicy>fair</schedulingPolicy>
                        <weight>0.25</weight>
                </queue>
         </queue>
 </allocations>
```

Fig: fair-scheduler.xml

→ Below are the queues which are created in Hadoop.

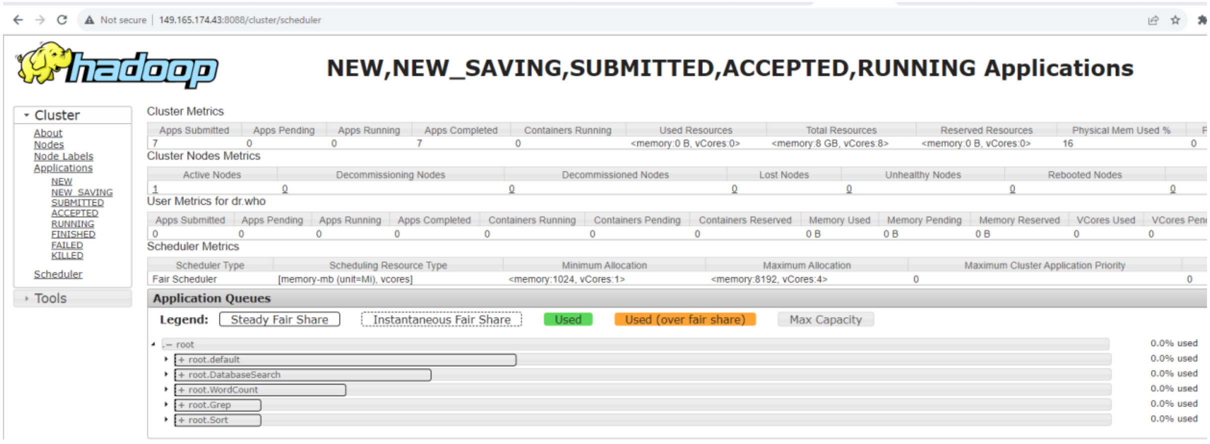


Fig: Initial Queues in Hadoop

→ 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.

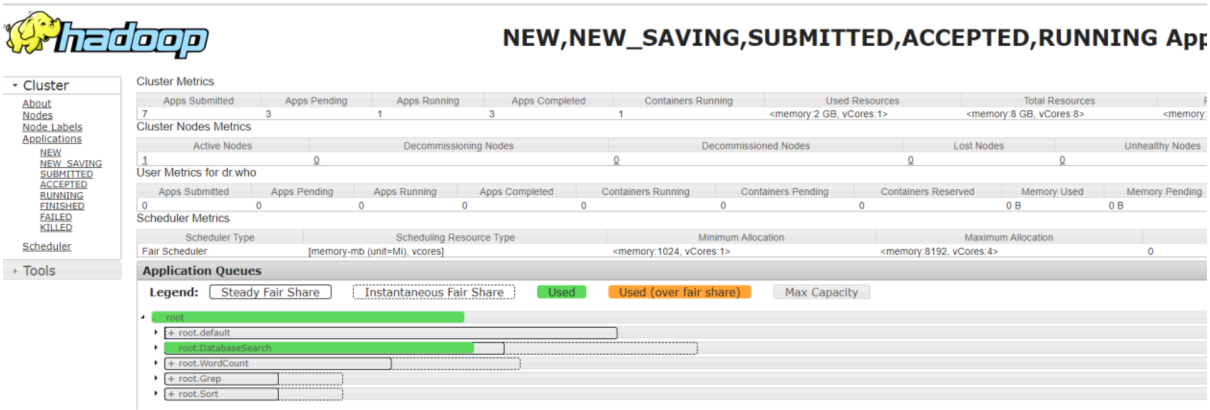


Fig: DatabaseSearch Queue running first

→ 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.

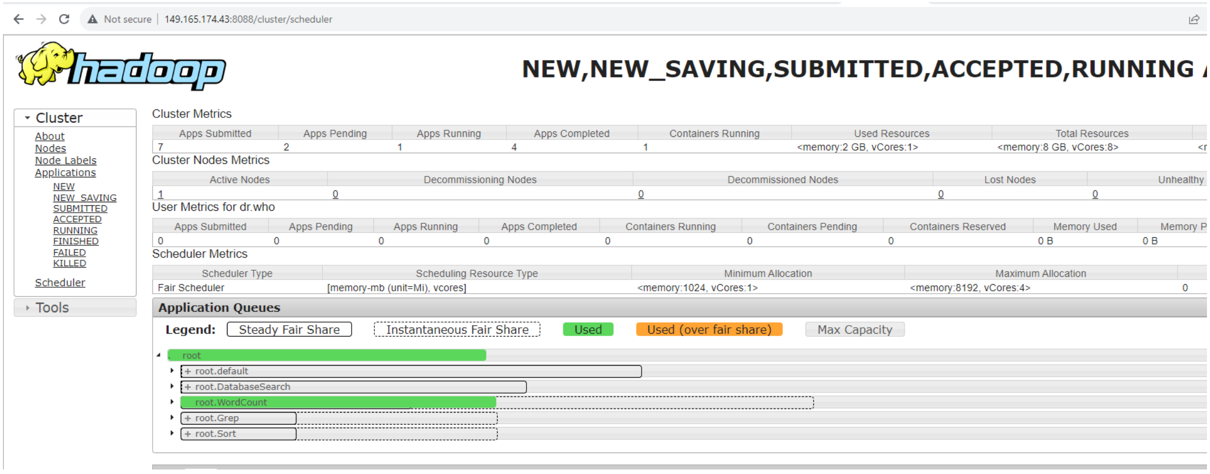


Fig: WordCount queue running second

→ 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.

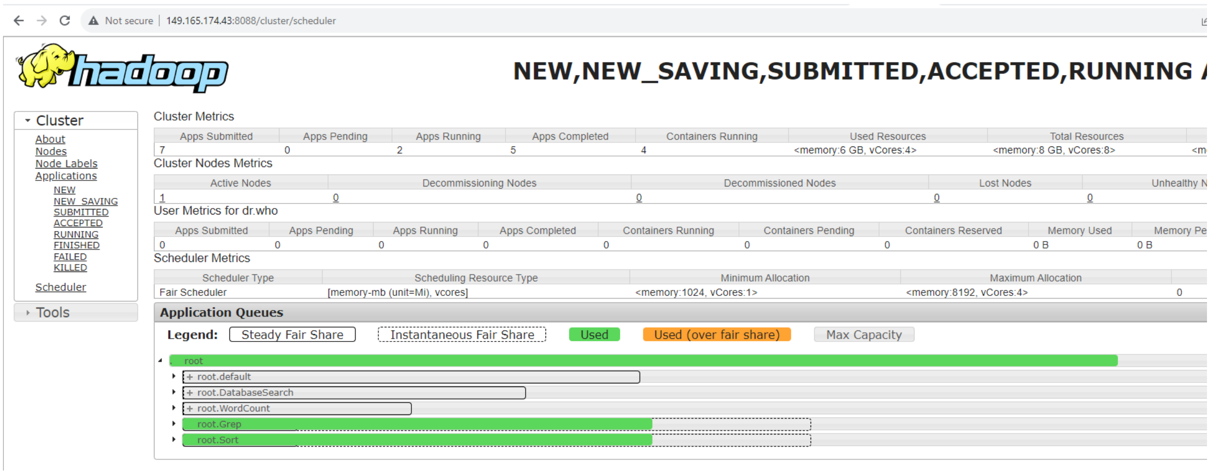


Fig: Grep and Sort queue running concurrently

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.

```
 <property>
   <name>yarn.scheduler.capacity.root.DatabaseSearch.capacity</name>
   <value>35</value>
   <description>DatabaseSearch queue target capacity.</description>
 </property>

<property>
<name>yarn.scheduler.capacity.root.WordCount.capacity</name>
<value>25</value>
<description>WordCount queue target capacity.</description>
</property>


<property>
            <name>yarn.scheduler.capacity.root.Sort.capacity</name>
            <value>25</value>
<description>Sort queue target capacity.</description>
</property>


<property>
            <name>yarn.scheduler.capacity.root.Grep.capacity</name>
                <value>15</value>
                    <description>Grep queue target capacity.</description>
                      </property>
```

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

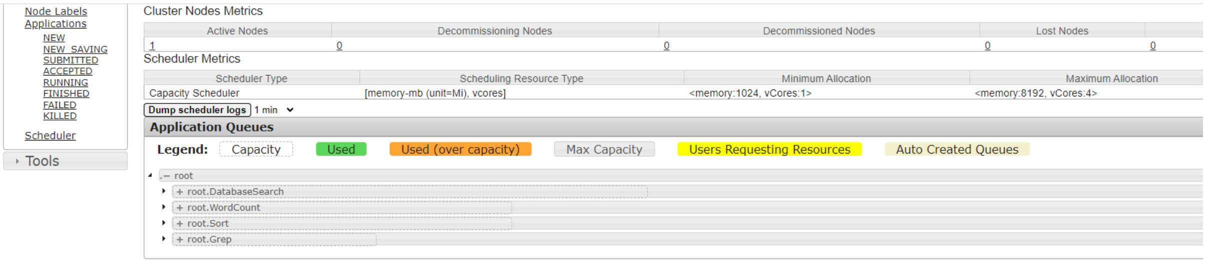
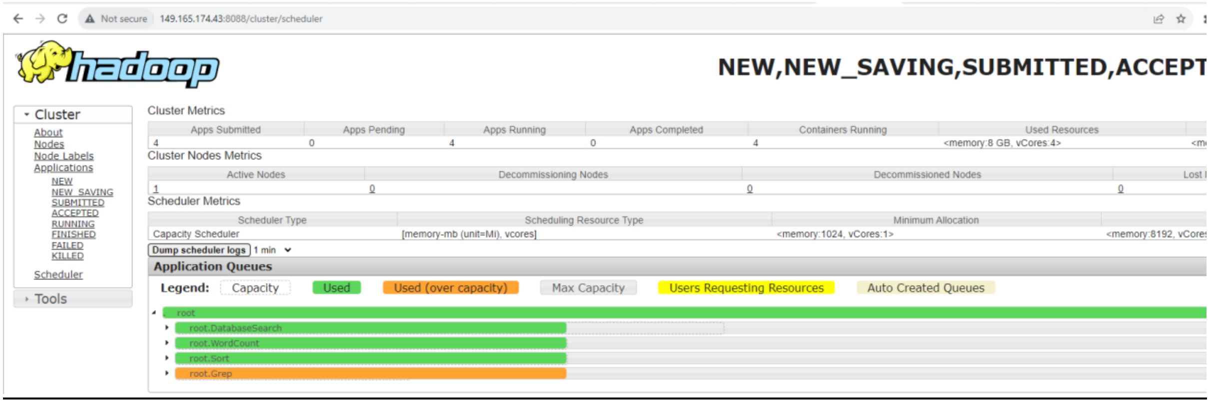


Fig: Capacity Scheduler with sort, grep, wordcount and DatabaseSearch Queue

→ 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.



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]