# Engineering Cloud Computing

## Assignment 1

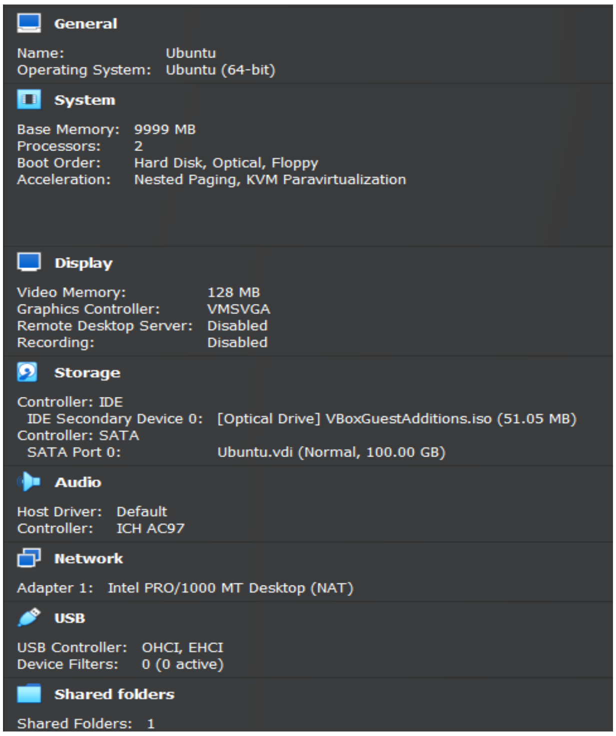
Dilip Nikhil Francies

dfrancie@iu.edu

2001274638

*Note : Due to limited availability of Jetstream2 instance, I have installed and configured Hadoop on my Windows laptop, following the tutorial uploaded on Canvas.*

Here are the specifications of my virtual machine:




## Part 1:

- Output the top-3 IP addresses with the granularity of an hour
    a) You should read the two examples.
    b) Develop your code based on examples. The program may take more than one round of MapReduce.

Part 1 – Mapper Function:

```
#Dilip Nikhil Francies
#Part 1 mapper function

import sys
import re
from datetime import datetime

# Regular expression pattern to match IP addresses and timestamps
log_pattern = re.compile(r'(?P<ip>\d+\.\d+\.\d+\.\d+) .*
\[(?P<timestamp>[^\]]+)\]')

for line in sys.stdin:
    match = log_pattern.search(line.strip()) #find matches in the input
    if match:
        ip_address = match.group('ip')
        timestamp_str = match.group('timestamp').split()[0]  # Extracting
date from timestamp
        try:
            timestamp = datetime.strptime(timestamp_str,
"%d/%b/%Y:%H:%M:%S") #strip the time into Hour, minutes and seconds
            hour = timestamp.strftime("%H")
            print(f"{hour}\t{ip_address}\t1")  # print the hour and the
associated IP address
        except ValueError:
            pass
```

Part 1 – Reducer Function:

```
#Author: Dilip Nikhil Francies
#reducer function

import sys
from operator import itemgetter
from collections import defaultdict
from datetime import datetime, timedelta

hourly_ip_count = defaultdict(lambda: defaultdict(int))
for line in sys.stdin: #input
    line = line.strip()
    hour, ip, count = line.split('\t', 2)
    count = int(count)
    hourly_ip_count[hour][ip] += count

# Iterate through each hour
for hour, ip_counts in hourly_ip_count.items():
    start_time = datetime.strptime(hour, "%H")
    end_time = start_time + timedelta(hours=1) - timedelta(seconds=1) #
from 00:00:00  to 00:00:59 every hour
    hour_range = f"{start_time.strftime('%H:%M:%S')} to
{end_time.strftime('%H:%M:%S')}"
    # Sort IP addresses based on their counts
    sorted_ips = sorted(ip_counts.items(), key=itemgetter(1), reverse=True)
    # Output the top 3 IP addresses for each hour
    top_3_ips = sorted_ips[:3] #once sorted, get the top three IPs
    for ip, count in top_3_ips:
        print(f"From {hour_range}, IP: {ip}, Count: {count}") #print
```

# Steps followed:

1. Copy log files and the source code from the localhost to the VM through a shared folder, and copy it to the "Hadoop" user from "Dilip".

```
dilip@Ubuntu:~/Desktop$ sudo cp -r ECC /home/hadoop/
dilip@Ubuntu:~/Desktop$ su - hadoop
Password:
hadoop@Ubuntu:~$ ls
ECC  hadoop  hadoop-3.3.6.tar.gz  hadoopdata  snap
hadoop@Ubuntu:~$ cd ECC
hadoop@Ubuntu:~/ECC$ ls
a1               logstat1_reduce.py  logstat2_reduce.py
logstat1_map.py  logstat2_map.py     sample.log
```

2. Run "Start-all.sh" to initiate the startup process of various Hadoop daemons.

```
hadoop@Ubuntu:~/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 [Ubuntu]
Starting resourcemanager
Starting nodemanagers
```

3. Convert access.log to hdfs [Hadoop distributed file system] and start the map reduce job.
   MapReduce code:

```
mapred streaming
-input /access.log
-file logstat1_map.py
-mapper "python3 logstat1_map.py"
-file logstat1_reduce.py
-reducer "python3 logstat1_reduce.py"
-output /sample_output_access
```

```
hadoop@Ubuntu:~/ECC$ hdfs dfs -copyFromLocal access.log /
hadoop@Ubuntu:~/ECC$ mapred streaming -input /access.log -file logstat1_map.py -mapper "python3 logstat1_map.py" -file logstat1_reduce.py -reducer "python3
 logstat1_reduce.py" -output /sample_output_access
2024-03-05 17:04:47,972 WARN streaming.StreamJob: -file option is deprecated, please use generic option -files instead.
packageJobJar: [logstat1_map.py, logstat1_reduce.py] [/home/hadoop/hadoop/share/hadoop/tools/lib/hadoop-streaming-3.3.6.jar] /tmp/streamjob7045267247551393
445.jar tmpDir=null
2024-03-05 17:04:51,844 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceManager at /0.0.0.0:8032
2024-03-05 17:04:52,813 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceManager at /0.0.0.0:8032
2024-03-05 17:04:53,660 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/hadoop/.staging/job_1709660894826_0
007
2024-03-05 17:04:54,657 INFO mapred.FileInputFormat: Total input files to process : 1
2024-03-05 17:04:54,715 INFO net.NetworkTopology: Adding a new node: /default-rack/127.0.0.1:9866
2024-03-05 17:04:54,852 INFO mapreduce.JobSubmitter: number of splits:26
2024-03-05 17:04:55,407 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1709660894826_0007
2024-03-05 17:04:55,408 INFO mapreduce.JobSubmitter: Executing with tokens: []
2024-03-05 17:04:55,963 INFO conf.Configuration: resource-types.xml not found
2024-03-05 17:04:55,964 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2024-03-05 17:04:56,181 INFO impl.YarnClientImpl: Submitted application application_1709660894826_0007
2024-03-05 17:04:56,355 INFO mapreduce.Job: The url to track the job: http://Ubuntu.myguest.virtualbox.org:8088/proxy/application_1709660894826_0007/
2024-03-05 17:04:56,358 INFO mapreduce.Job: Running job: job_1709660894826_0007
2024-03-05 17:05:09,898 INFO mapreduce.Job: Job job_1709660894826_0007 running in uber mode : false
2024-03-05 17:05:09,900 INFO mapreduce.Job:  map 0% reduce 0%
2024-03-05 17:05:57,013 INFO mapreduce.Job:  map 1% reduce 0%
```

4. Once the MapReduce job is initiated, make sure it terminates with a successful output generation and check for the files written in the folder /sample_output_access.

```
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=3502543223
        File Output Format Counters
                Bytes Written=4268
2024-03-05 17:13:05,056 INFO streaming.StreamJob: Output directory: /sample_output_access
hadoop@Ubuntu:~/ECC$ hadoop fs -ls /sample_output_access
Found 2 items
-rw-r--r--   1 hadoop supergroup          0 2024-03-05 17:13 /sample_output_access/_SUCCESS
-rw-r--r--   1 hadoop supergroup       4268 2024-03-05 17:13 /sample_output_access/part-00000
```

5. View the output:
   Format: "From *Start time* to *End time*, IP: *XX.XXX.XX.XXX*, Count: *XXXX*"

```
hadoop@Ubuntu:~/ECC$ hdfs dfs -cat /sample_output_access/*
From 00:00:00 to 00:59:59, IP: 66.249.66.194, Count: 14298
From 00:00:00 to 00:59:59, IP: 66.249.66.91, Count: 12232
From 00:00:00 to 00:59:59, IP: 66.249.66.92, Count: 4291
From 01:00:00 to 01:59:59, IP: 66.249.66.91, Count: 13874
From 01:00:00 to 01:59:59, IP: 66.249.66.194, Count: 12485
From 01:00:00 to 01:59:59, IP: 66.249.66.92, Count: 2924
From 02:00:00 to 02:59:59, IP: 66.249.66.91, Count: 11697
From 02:00:00 to 02:59:59, IP: 66.249.66.194, Count: 10345
From 02:00:00 to 02:59:59, IP: 91.99.72.15, Count: 1448
From 03:00:00 to 03:59:59, IP: 66.249.66.194, Count: 8644
From 03:00:00 to 03:59:59, IP: 66.249.66.91, Count: 7914
From 03:00:00 to 03:59:59, IP: 91.99.72.15, Count: 1275
From 04:00:00 to 04:59:59, IP: 66.249.66.194, Count: 10805
From 04:00:00 to 04:59:59, IP: 66.249.66.91, Count: 7571
From 04:00:00 to 04:59:59, IP: 91.99.72.15, Count: 1511
From 05:00:00 to 05:59:59, IP: 66.249.66.194, Count: 10534
From 05:00:00 to 05:59:59, IP: 66.249.66.91, Count: 7035
From 05:00:00 to 05:59:59, IP: 91.99.72.15, Count: 1921
```

***Figure: Top three IPs for every Hour*.**

*Note: Full-length output is provided in the logs folder with the file named "part1_output.txt"*

# Part 2:

1. Make your program like a database search.
   a. 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.

For Part 2 of the assignment, I developed a shell program that lets the user enter the time frame that he would like to query in the format "*start_time*[space]*end_time*". These values are then passed on to my MapReduce code as arguments to run the job.

```
#!/usr/bin/env bash
# Prompt the user to enter two numbers separated by a space
echo "Enter start and end time separated by a space:"
read start end

# Check if both inputs are valid integers
if ! [[ "$start" =~ ^[0-9]+$ ]] || ! [[ "$end" =~ ^[0-9]+$ ]]; then
    echo "Error: Input is not valid. Please enter two integers separated by a space."
    exit 1

fi
mapred streaming  -input /access.log  -output /part2_A -file logstat2_map.py -mapper "python3 logstat2_map.py $start $end" -file logstat2_reduce.py -reducer "python3 logstat2_reduce.py"
```

Part 2 Mapper Function:

```
#Author : Dilip Nikhil Francies
#mapper function


import re
import sys
# Compile the regex pattern outside the loop for better performance
ip_hour_pattern =
re.compile(r'(?P<ip>\d+\.\d+\.\d+\.\d+).*?\d{4}:(?P<hour>\d{2}):\d{2}')
# Retrieve command-line arguments directly

try:
    start_period, end_period = map(int, sys.argv[1:3]) # pars from
arguments
except (ValueError, IndexError):
    print("Invalid input. Please provide a valid range as command-line
arguments (e.g., '3 4')")
    sys.exit(1)


for line in sys.stdin: #read from acces.log
    line = line.strip()
    match = ip_hour_pattern.search(line)
    if match:
        ip_address = match.group('ip')
        hour = int(match.group('hour'))
        if start_period <= hour < end_period: # check if start time is less
than end time
            print(f"[{hour:02}:00] {ip_address} 1") #print ip and hour.
```

Part 2 Reducer function:

```
#author : Dilip Nikhil Francies
#reducer function for part 2

import sys
# Create an empty dictionary to store IP counts
ip_counts = {}

for line in sys.stdin:
    line = line.strip()
    if line.startswith("["):
        # Extract hour and IP address
        hour, ip = line.split()[0], line.split()[1]
        # Strip brackets from hour
        hour = hour.strip("[]")
        # Update IP count in the dictionary
        ip_counts[ip] = ip_counts.get(ip, 0) + 1

sorted_ip_counts = sorted(ip_counts.items(), key=lambda x: x[1],
reverse=True)
# Print the dictionary containing IP counts
for ip, count in sorted_ip_counts[:3]:  #sort top three
    print(f"[{hour}] {ip} {count}")
```

Running the shell file *"./run.sh"* first prompts the user to input a time frame, in this example it would be from 3AM to 6 AM. These inputs will be passed on to my mapper function with arguments as *$start $end.*

MapReduce code used in the shell:

```
mapred streaming
-input /access.log
-output /part2_A
-file logstat2_map.py
-mapper "python3 logstat2_map.py $start $end"
-file logstat2_reduce.py
-reducer "python3 logstat2_reduce.py"
```


Figure: Initiating the MapReduce program

Once the job is run, the output is saved in /part2_A folder which then can be viewed as shown below:

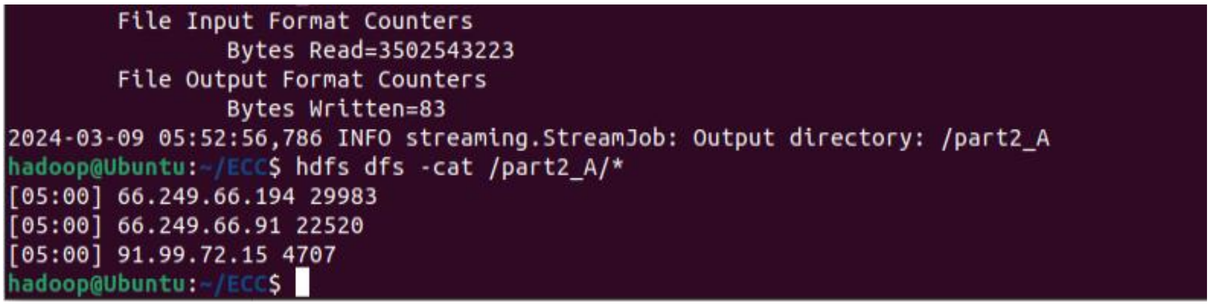



Figure: Part 2 Output of top three IPs in the given time frame

# Part 2:

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

## A Brief Introduction to Schedulers:

Scheduling mainly refers to the process of allocating and managing computing resources among multiple jobs and users in a distributed computing environment to ensure efficient resource utilization and fair sharing of resources among different applications running on a Hadoop cluster.

Scheduling is essential in Hadoop for proper resource management, efficient utilization by keeping all nodes busy while minimizing resource wastage, prioritizing different jobs based on importance, urgency, and finally failure tolerance to maintain jobs in progress and to reduce disruptions

Types of Schedulers:

Fair Scheduler - The Fair Scheduler in Hadoop is a resource scheduler designed to ensure fair and equitable allocation of computing resources among users and jobs in a Hadoop cluster. Overall, the Fair Scheduler promotes fairness, optimizes resource utilization, and ensures predictable performance for all users and jobs in a Hadoop cluster.

Capacity Scheduler- The Capacity Scheduler in Hadoop is a resource scheduler that facilitates efficient and predictable resource allocation. Key features include flexible and efficient mechanisms for managing resources in shared Hadoop clusters, ensuring fair access, predictable performance, and effective utilization.

As for part 2, question B, the following steps that I performed to test different schedulers and their effect on resource allocation to Hadoop jobs.

1. Configure the “yarn-site.xml” to indicate the preferred scheduler to be used.

```
<configuration>
        <property>
                <name>yarn.resourcemanager.scheduler.class</name>
                <value>org.apache.hadoop.yarn.server.resourcemanager.scheduler.capacity.CapacityScheduler</value>
        </property>
        <property>
               <name>yarn.nodemanager.aux-services</name>
               <value>mapreduce_shuffle</value>
        </property>
</configuration>
```

*Fig: Capacity scheduler configuration*

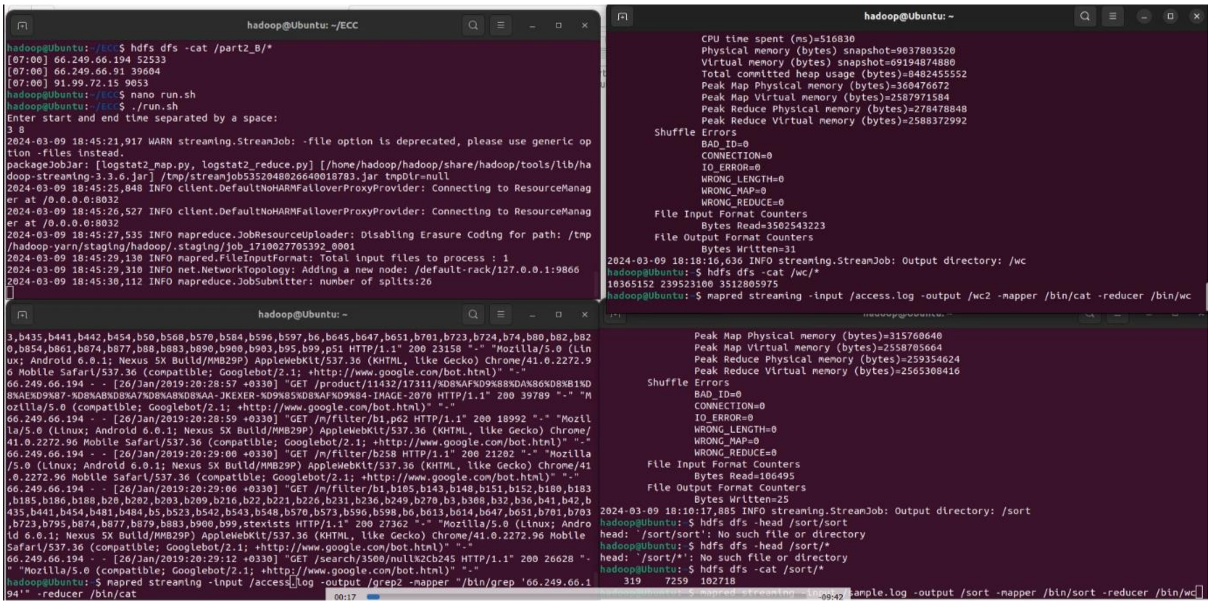
```
<configuration>
        <property>
                <name>yarn.resourcemanager.scheduler.class</name>
                <value>org.apache.hadoop.yarn.server.resourcemanager.scheduler.fair.FairScheduler</value>
        </property>
        <property>
               <name>yarn.nodemanager.aux-services</name>
               <value>mapreduce_shuffle</value>
        </property>
</configuration>
```

*Fig: Fair scheduler configuration*

2. Once the configuration is complete, stop and restart the resource manager daemon with the following command

```
hadoop@Ubuntu:~/hadoop/etc/hadoop$ yarn --daemon stop resourcemanager
hadoop@Ubuntu:~/hadoop/etc/hadoop$ yarn --daemon start resourcemanager
```

3. Open 4 different terminals and execute 4 MapReduce programs parallelly
    1. Top 3 IPs,
    2. WordCount,
    3. Grep
    4. Sort



These are the following commands that I used:

1. Top 3 IP:

```
mapred streaming
-input /access.log
-output /part2_B_B
-file logstat2_map.py
-mapper "python3 logstat2_map.py $start $end"
-file logstat2_reduce.py
-reducer "python3 logstat2_reduce.py"
```

2. WordCount:

```
mapred streaming
-input /access.log
 -output /wc2
 -mapper /bin/cat
 -reducer /bin/wc
```

3. Sort

```
mapred streaming -input /sample.log
-output /sort
-mapper /bin/sort
-reducer /bin/cat
```

4. Grep

```
mapred streaming
-input /access.log
-output /grep2
-mapper "/bin/grep '66.249.66.194'"
-reducer /bin/cat
```

# Discussion:

Capacity Scheduler:

When the 4 jobs are run, all the resources are allocated to the first process in the queue and all the users can use the maximum resources available in the cluster. Initially, the first program is allocated with 7 CPU vcores and 8192 of memory, which is the entire resource available in the cluster.

Show 20 entries

| ID | User | Name | Application Type | Application Tags | Queue | Application Priority | StartTime | LaunchTime | FinishTime | State | FinalStatus | Running Containers | Allocated CPU VCores | Allocated Memory MB | Allocated GPUs |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| application_1710027705392_0003 | hadoop | streamjob4556474596164970945.jar | MAPREDUCE | | default | 0 | Sat Mar 9 18:46:02 -0500 2024 | N/A | N/A | ACCEPTED | UNDEFINED | 0 | 0 | 0 | -1 |
| application_1710027705392_0002 | hadoop | streamjob5807326587261156015.jar | MAPREDUCE | | default | 0 | Sat Mar 9 18:45:55 -0500 2024 | N/A | N/A | ACCEPTED | UNDEFINED | 0 | 0 | 0 | -1 |
| application_1710027705392_0001 | hadoop | streamjob5352048026640018783.jar | MAPREDUCE | | default | 0 | Sat Mar 9 18:45:33 -0500 2024 | Sat Mar 9 18:45:38 -0500 2024 | N/A | RUNNING | UNDEFINED | 7 | 7 | 8192 | -1 |

Once the 1st program is successfully executed complete, the resource utilization drops.

Application Queues

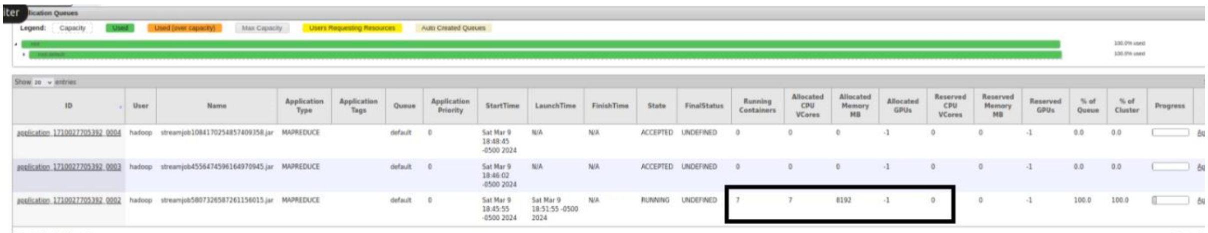
Legend: Capacity Used Used (over capacity) Max Capacity Users Requesting Resources Auto Created Queues

25.0% used
25.0% used

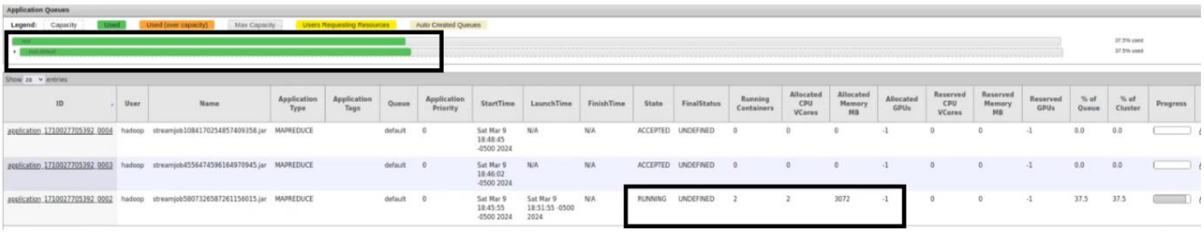
Show 20 entries

| ID | User | Name | Application Type | Application Tags | Queue | Application Priority | StartTime | LaunchTime | FinishTime | State | FinalStatus | Running Containers | Allocated CPU VCores | Allocated Memory MB | Allocated GPUs | Reserved CPU VCores | Reserved Memory MB | Reserved GPUs | % of Queue | % of Cluster | Progress |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| application_1710027705392_0004 | hadoop | streamjob1084170254857409358.jar | MAPREDUCE | | default | 0 | Sat Mar 9 18:48:45 -0500 2024 | N/A | N/A | ACCEPTED | UNDEFINED | 0 | 0 | 0 | -1 | 0 | 0 | -1 | 0.0 | 0.0 | |
| application_1710027705392_0003 | hadoop | streamjob4556474596164970945.jar | MAPREDUCE | | default | 0 | Sat Mar 9 18:46:02 -0500 2024 | N/A | N/A | ACCEPTED | UNDEFINED | 0 | 0 | 0 | -1 | 0 | 0 | -1 | 0.0 | 0.0 | |
| application_1710027705392_0002 | hadoop | streamjob5807326587261156015.jar | MAPREDUCE | | default | 0 | Sat Mar 9 18:45:55 -0500 2024 | Sat Mar 9 18:51:55 -0500 2024 | N/A | ACCEPTED | UNDEFINED | 1 | 1 | 2048 | -1 | 0 | 0 | -1 | 25.0 | 25.0 | |

At this point the second process in queued and the resource allocations shoots up again, to all the available resources.



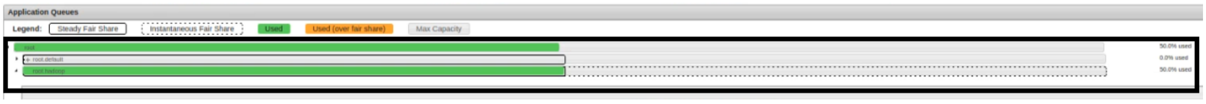
The same process continues as long as all the jobs are complete in sequential manner, providing all the available resources to jobs that are executed in priority.



Note: The output of each program is available in the folder attached.

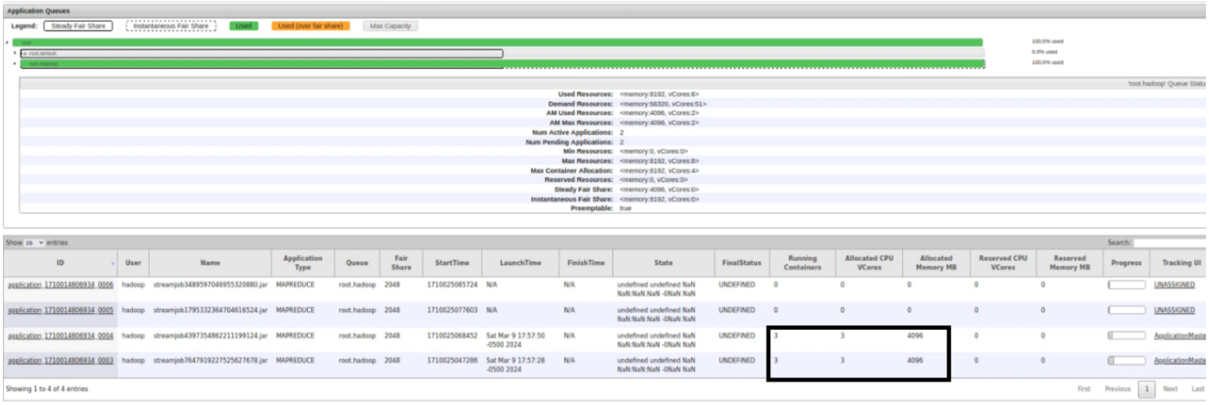
## Fair scheduler:

Fair scheduler on the other hand distributes the resources equally to the jobs that demand resources.Users get equal amount of resources [Dilip 50% Hadoop 50% [default value] as displayed below

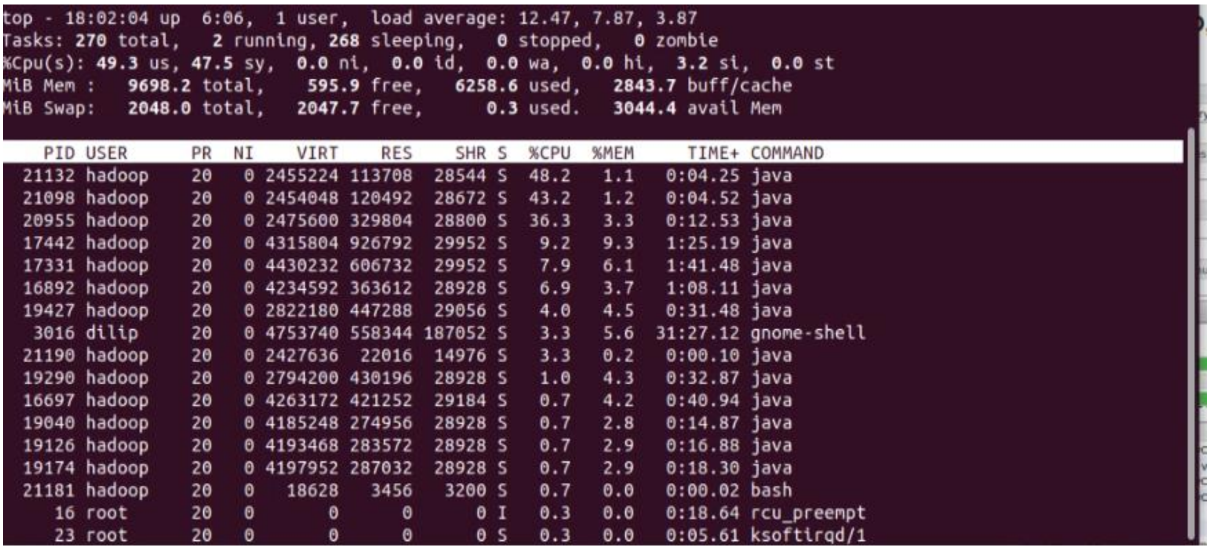


Based on available resource, it is distributed to two jobs received.

Because, the other user "Dilip" has no jobs running, and the resources are available, the "Hadoop" user makes use of it and distributes available resources to jobs 1 and 2.



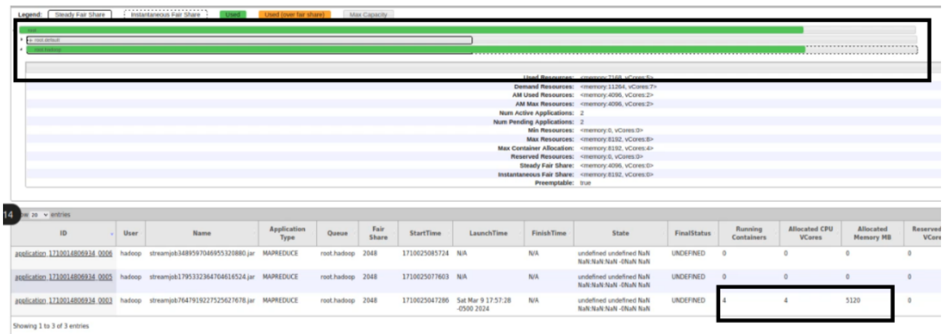
As suggested by TA Juliette, I used “top” command to see resources utilization of the VM.



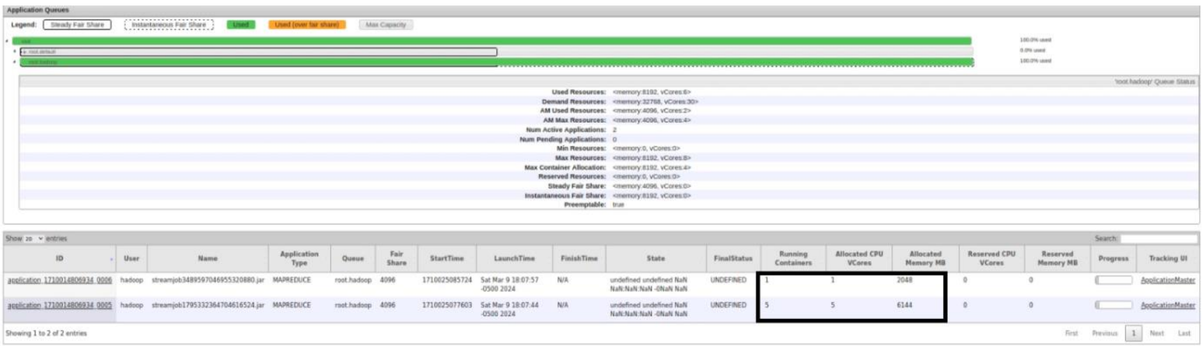
```
top - 18:02:04 up  6:06,  1 user,  load average: 12.47, 7.87, 3.87
Tasks: 270 total,   2 running, 268 sleeping,   0 stopped,   0 zombie
%Cpu(s): 49.3 us, 47.5 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  3.2 si,  0.0 st
MiB Mem :   9698.2 total,    595.9 free,   6258.6 used,   2843.7 buff/cache
MiB Swap:   2048.0 total,   2047.7 free,      0.3 used.   3044.4 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
  21132 hadoop    20   0 2455224 113708  28544 S  48.2   1.1   0:04.25 java
  21098 hadoop    20   0 2454048 120492  28672 S  43.2   1.2   0:04.52 java
  20955 hadoop    20   0 2475600 329804  28800 S  36.3   3.3   0:12.53 java
  17442 hadoop    20   0 4315804 926792  29952 S   9.2   9.3   1:25.19 java
  17331 hadoop    20   0 4430232 606732  29952 S   7.9   6.1   1:41.48 java
  16892 hadoop    20   0 4234592 363612  28928 S   6.9   3.7   1:08.11 java
  19427 hadoop    20   0 2822180 447288  29056 S   4.0   4.5   0:31.48 java
   3016 dilip     20   0 4753740 558344 187052 S   3.3   5.6  31:27.12 gnome-shell
  21190 hadoop    20   0 2427636  22016  14976 S   3.3   0.2   0:00.10 java
  19290 hadoop    20   0 2794200 430196  28928 S   1.0   4.3   0:32.87 java
  16697 hadoop    20   0 4263172 421252  29184 S   0.7   4.2   0:40.94 java
  19040 hadoop    20   0 4185248 274956  28928 S   0.7   2.8   0:14.87 java
  19126 hadoop    20   0 4193468 283572  28928 S   0.7   2.9   0:16.88 java
  19174 hadoop    20   0 4197952 287032  28928 S   0.7   2.9   0:18.30 java
  21181 hadoop    20   0   18628   3456   3200 S   0.7   0.0   0:00.02 bash
     16 root      20   0       0      0      0 I   0.3   0.0   0:18.64 rcu_preempt
     23 root      20   0       0      0      0 S   0.3   0.0   0:05.61 ksoftirqd/1
```

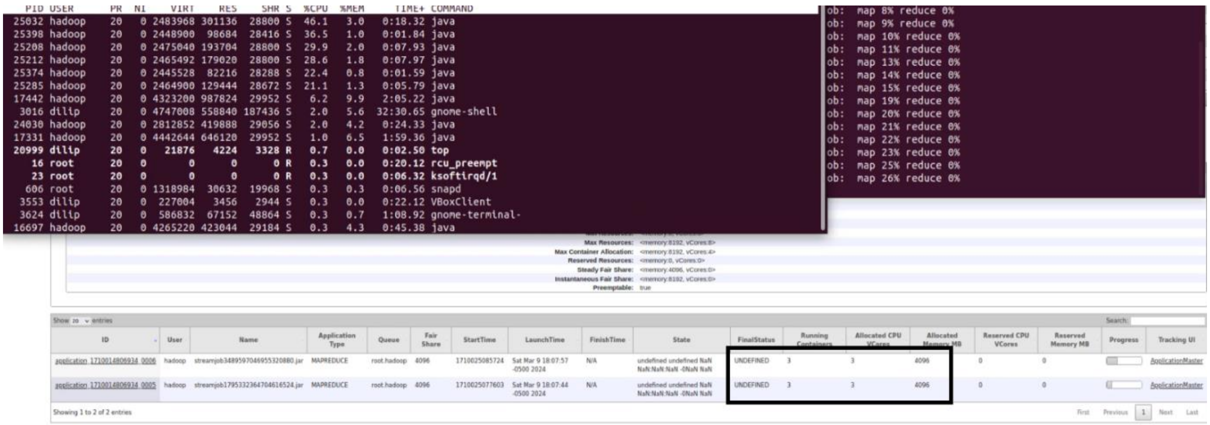
Once the first job in complete the remaining resources are allocated to the jobs that were running earlier, ensuring that all the jobs have some number of resources allocated. At the same time, once the previous job is completed, the remaining is distributed equally among other processes.



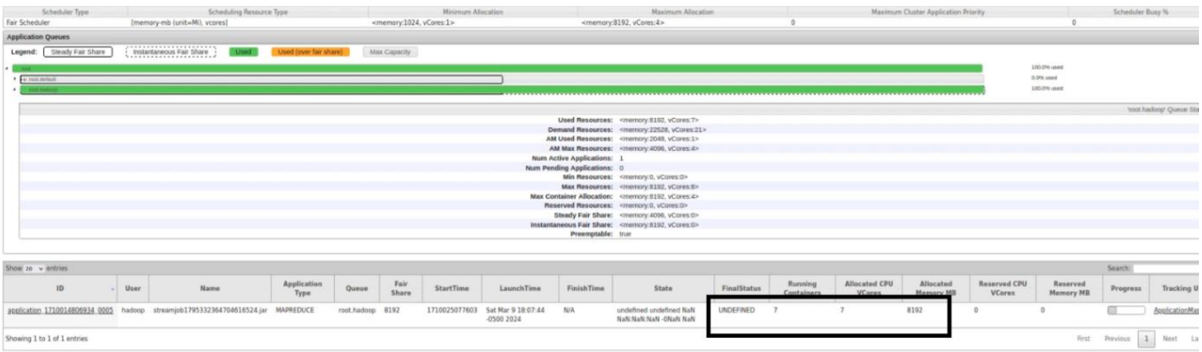
After the second job is complete, the remaining resources are allocated to 3rd and 4th job.



Now the jobs are distributed equally:




Final job process gets all the available resources:



One thing that I have observed is, the number of resources allocated reduces when most of the mapping part is complete, which kind of makes sense assuming the mapper is usually process heavy in some of these programs, and only the required amount of resources allocated to that particular job

## Comparison of different performance metrics for different types of schedulers:

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Capacity | | | | | | | | | | |
| | Program | total time -map tasks | total time - reduce tasks | MB ms map tasks | MB ms - reduce tasks | GC time elapsed | CPU time spent | Time Elapsed | Agg resource alloc | Agg vcore (s) |
| | ip | 1708570 | 181746 | 1749575680 | 186107904 | 20604 | 211820 | 6:14 | 2769186 | 2311 |
| | word_count | 2302033 | 515957 | 2357281792 | 528339968 | 22598 | 545240 | 16:30 | 4251706 | 3499 |
| | grep | 1166133 | 134193 | 1194120192 | 137413632 | 17657 | 153520 | 20:43 | 1912387 | 1593 |
| | sort | 31123 | 6758 | 31869952 | 6920192 | 635 | 4660 | 6:14 | 2769186 | 2311 |
| Fair | | | | | | | | | | |
| | Program | total time -map tasks | total time - reduce tasks | MB ms map tasks | MB ms - reduce tasks | GC time elapsed | CPU time spent | Time Elapsed | Agg resource alloc | Agg vcore (s) |
| | ip | 636525 | 239254 | 651801600 | 244996096 | 8565 | 97410 | 6:14 | 2769186 | 2311 |
| | word_count | 834627 | 520935 | 854658048 | 533437440 | 9501 | 392300 | 13:29 | 3091284 | 2187 |
| | grep | 799809 | 185287 | 819004416 | 189733888 | 13820 | 135190 | 14:26 | 1812821 | 1384 |
| | sort | 20848 | 7912 | 21348352 | 8101888 | 423 | 3890 | 14:51 | 159532 | 92 |

*Fig: Various metrics compared between Fair and Capacity Schedulers*

## Graphs Plots:


Fig: Plots displaying performance metrics of Fair and Capacity Schedulers

Conclusion:

On the surface, fair scheduler definitely looks better in terms of overall performance, but I believe other parameters define what type of scheduler algorithms to use. Perhaps I should dive deeper to understand better and derive conclusions.