CSC 555: HW3

Name: Nachiketh Reddy
ID: 2117731

QUESTION 1:

Multi-node cluster setup Follow the instructions in the accompanying Word document to set up a 4 node cluster.

1. **You should verify that the cluster is running by pointing your browser to the link below. http://[insert-the-public-ip-of-master]:50070/ Make sure that the cluster is operational. You should see the 3 nodes under Datanodes tab. Submit a screenshot of your cluster status view.**

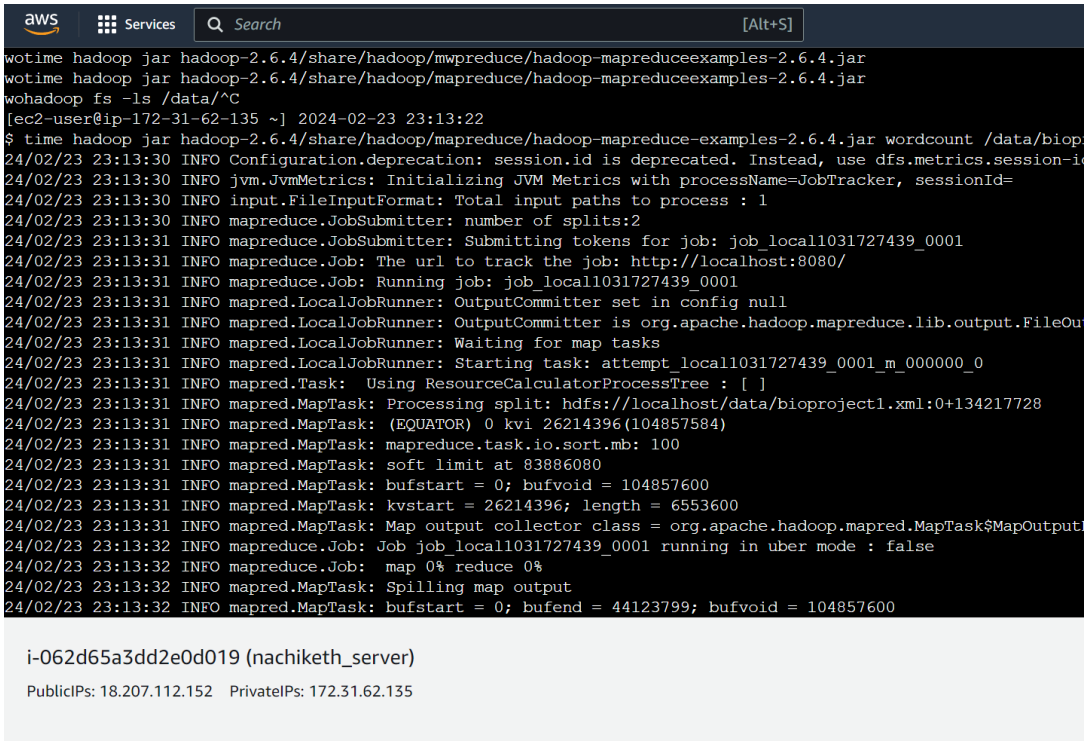
Datanode Information

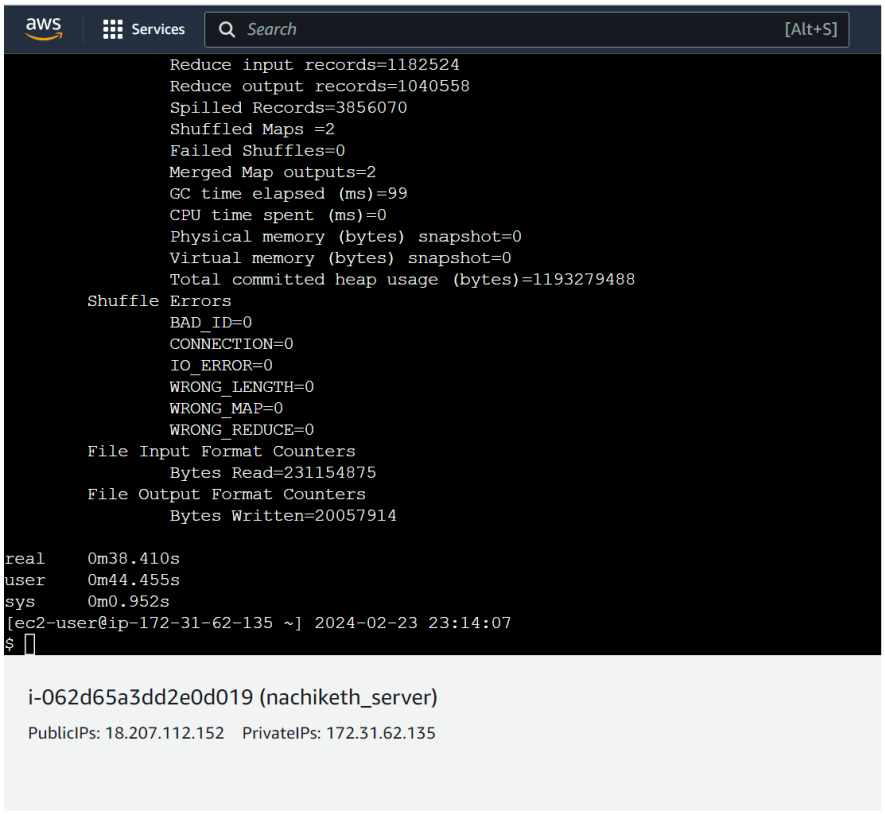
In operation

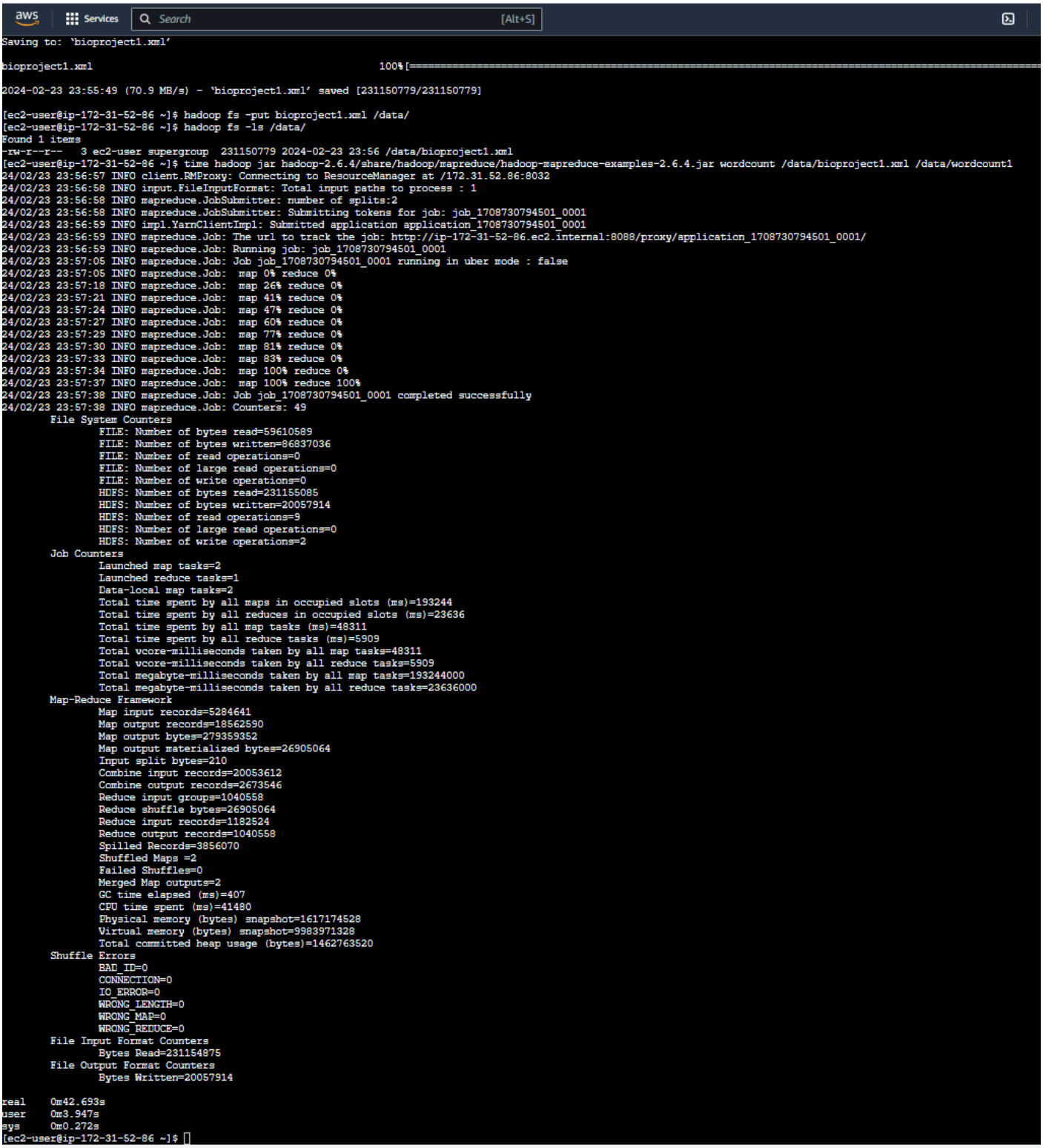
Node	Last contact	Admin State	Capacity	Used	Non DFS Used	Remaining	Blocks	Block pool used	Failed Volumes	Version
ip-172-31-88-154.ec2.internal (172.31.88.154:50010)	2	In Service	1.91 GB	512 KB	128 KB	1.91 GB	4	512 KB (0.03%)	0	2.6.4
ip-172-31-82-170.ec2.internal (172.31.82.170:50010)	1	In Service	1.91 GB	672 KB	112 KB	1.91 GB	6	672 KB (0.03%)	0	2.6.4
ip-172-31-52-86.ec2.internal (172.31.52.86:50010)	1	In Service	3.87 GB	768 KB	1.29 MB	3.87 GB	5	768 KB (0.02%)	0	2.6.4
ip-172-31-93-130.ec2.internal (172.31.93.130:50010)	2	In Service	1.91 GB	804 KB	144 KB	1.91 GB	6	804 KB (0.04%)	0	2.6.4

2. **Repeat the steps for the wordcount example in HW1 and submit the screenshots of running it. Write a short paragraph with a discussion about how the results compare. Are they faster or slower? How much faster or slower?**

time hadoop jar hadoop-2.6.4/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.6.4.jar wordcount /data/bioproject1.xml /data/wordcount1







SINGLE NODE	MULTI NODE
real 0m38.410s	real 0m42.629s
user 0m44.455s	user 0m3.947s
sys 0m0.952s	sys 0m0.272s

When the wordcount script was executed on the bioproject.xml file using single-node and multi-node setups, the single-node setup finished the operation in 38.410 seconds, with a user time of 44.455 seconds. The multi-node arrangement, on the other hand, finished in 42.629 seconds, although with a much shorter user time (3.947 seconds). The discrepancy implies that even though the multi-node configuration had a minor runtime overhead overall, it effectively used CPU resources in user mode, most likely as a result of parallel processing over several nodes. Regardless of the number of nodes deployed, both setups displayed low system time, showing constant overhead linked to system operations.

3. **Run all the queries implemented in HW2. Submit the timings for each query in the form of a table and compare it with the timing using a single node cluster. Here is a sample table:**

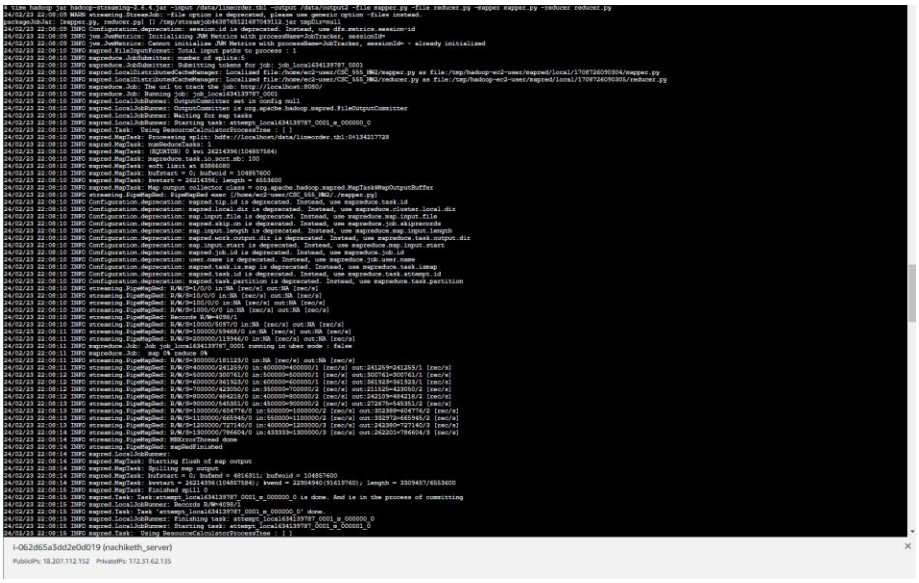
QUERY	SINGLE NODE	MULTI NODE
SELECT lo_quantity, lo_linenumber FROM lineorder WHERE lo discount < 10 AND lo tax > 2	real 0m26.453s user 0m33.889s sys 0m3.267s	real 0m30.124s user 0m3.884s sys 0m0.245s
SELECT p_category, COUNT(p_type) FROM part GROUP BY p_category	real 0m5.508s user 0m7.298s sys 0m0.483s	real 0m20.967s user 0m3.585s sys 0m0.330s
SELECT lo_discount, AVG(lo_extendedprice) FROM lineorder GROUP BY lo_discount	real 0m35.510s user 0m43.745s sys 0m3.915s	real 0m31.081s user 0m3.679s sys 0m0.343s
SELECT lo_custkey, SUM(lo_extendedprice) AS revenue FROM lineorder WHERE lo_quantity < 12 GROUP BY lo custkey	real 0m18.552s user 0m22.576s sys 0m2.064s	real 0m23.018s user 0m3.787s sys 0m0.232s
SELECT s_suppkey FROM supplier MINUS SELECT lo_suppkey FROM lineorder WHERE lo_discount < 10	real 0m30.550s user 0m38.458s sys 0m3.300s	real 0m28.030s user 0m3.865s sys 0m0.300s

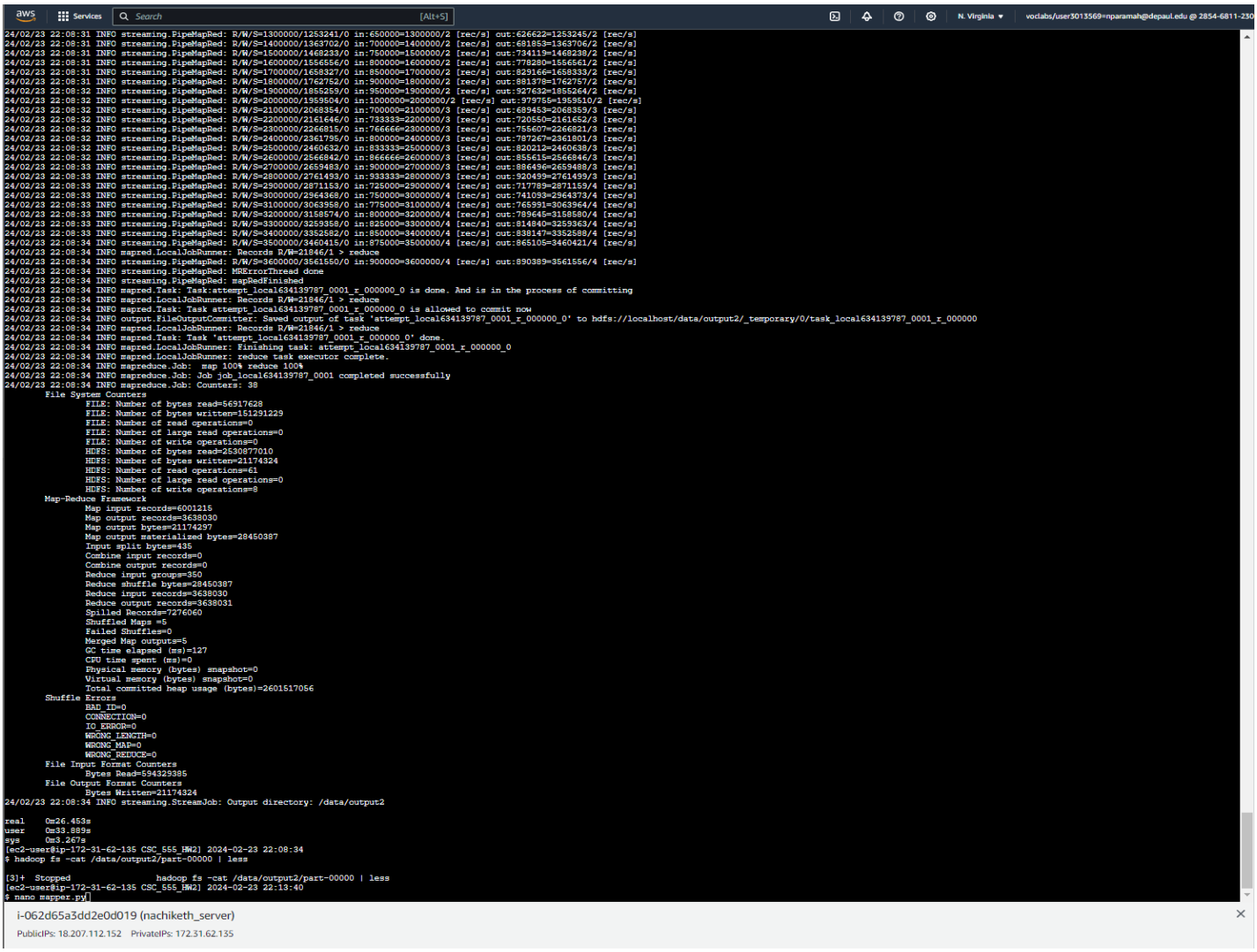
Several trends become apparent when comparing the query execution speeds of single-node and multi-node systems. The single-node arrangement performs better than the multi-node setup for queries containing complex filtering conditions such as tax and discount thresholds, however it has greater system and user times. However, even with slightly longer actual times, smaller queries requiring simple aggregations such as averaging or counting benefit from parallel processing in multi-node settings. Because there is less overhead involved in coordinating operations among nodes, the single-node arrangement generally exhibits faster actual timings. On the other hand, multi-node configurations show benefits in terms of user and system times, exhibiting better resource management and parallelism for certain kinds of queries.

SINGLE NODE:

QUERY 1:

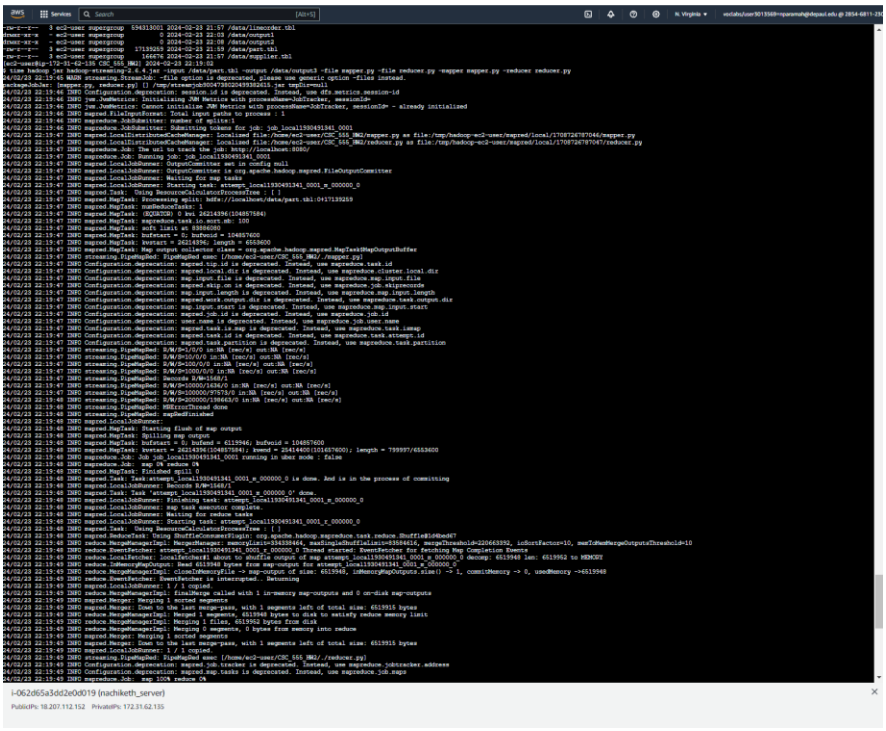
SELECT lo_quantity, lo_linenumber FROM lineorder WHERE lo discount < 10 AND lo tax > 2

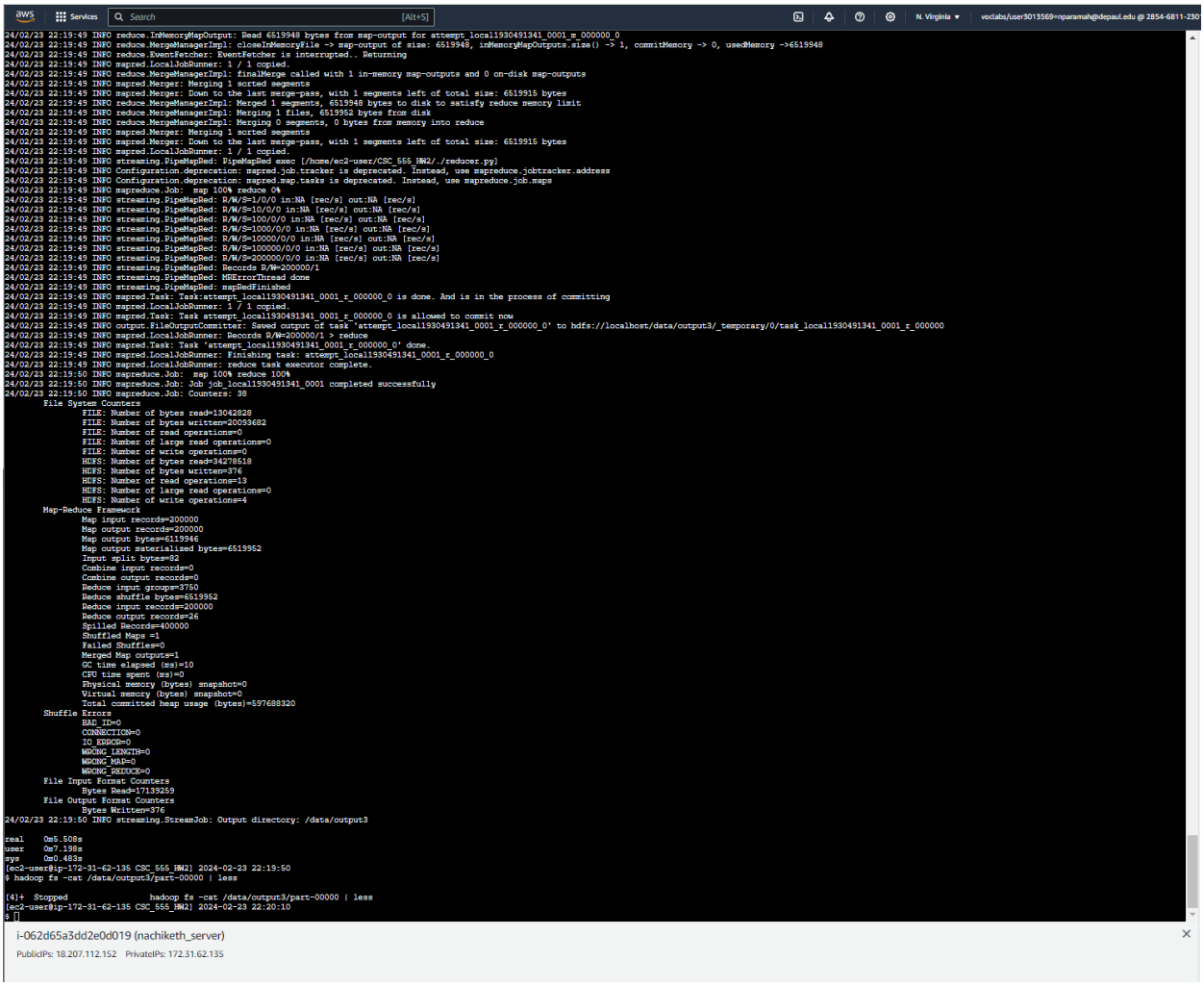




QUERY 2:

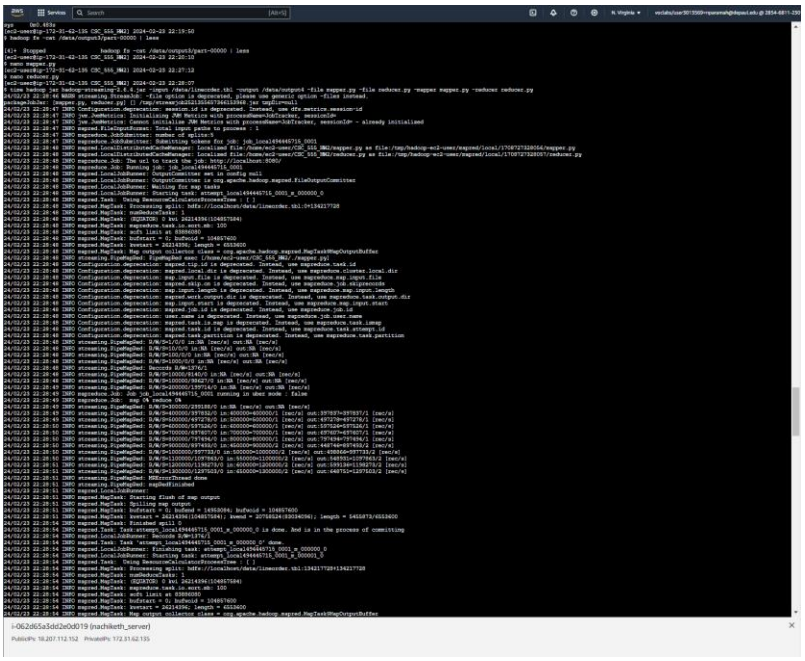
SELECT p_category, COUNT(p_type) FROM part GROUP BY p_category

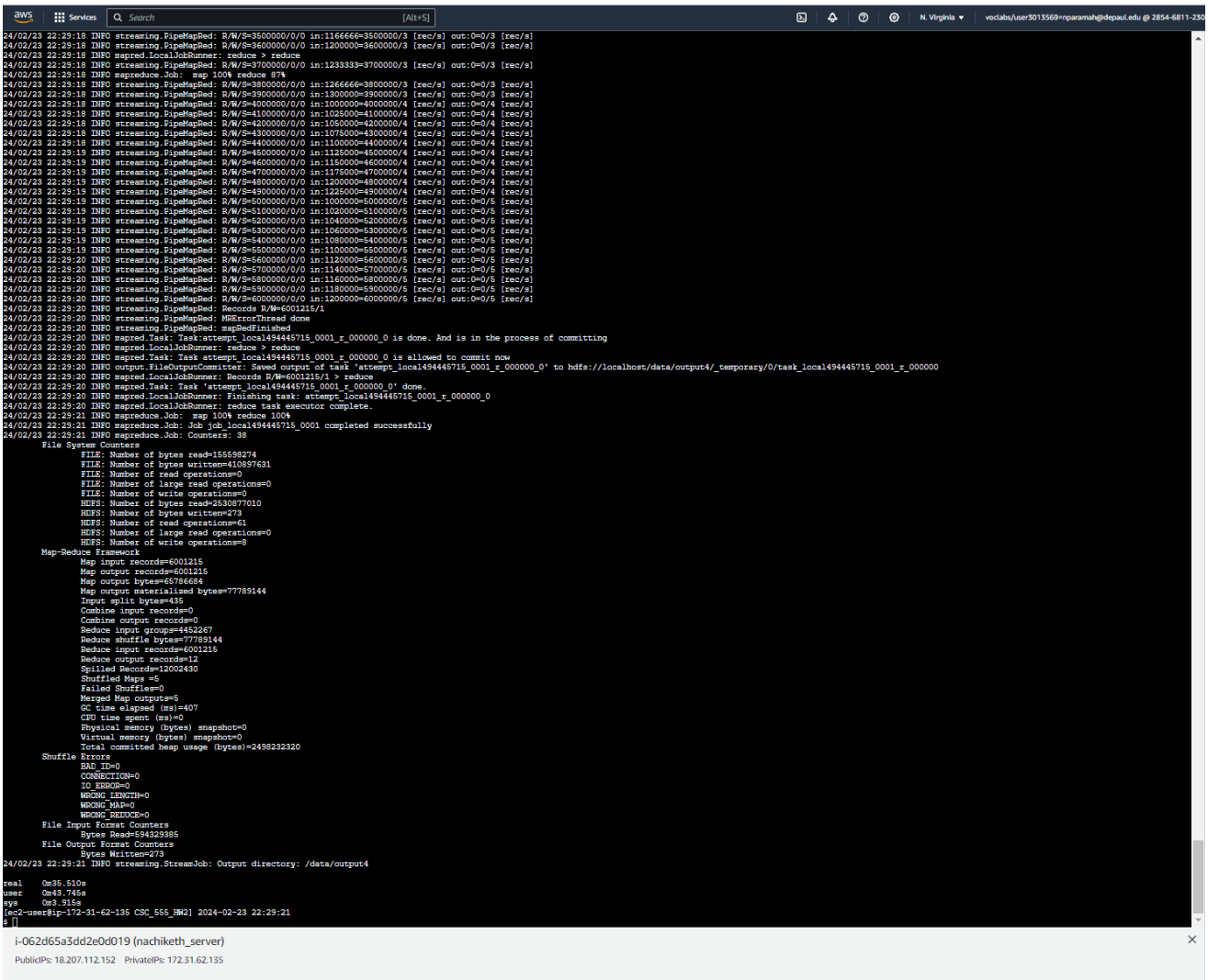




QUERY 3:

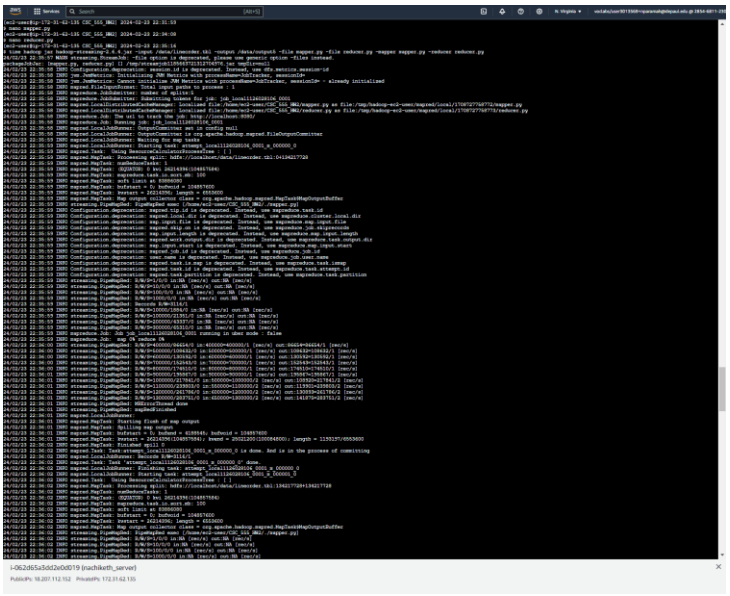
SELECT lo_discount, AVG(lo_extendedprice) FROM lineorder GROUP BY lo_discount

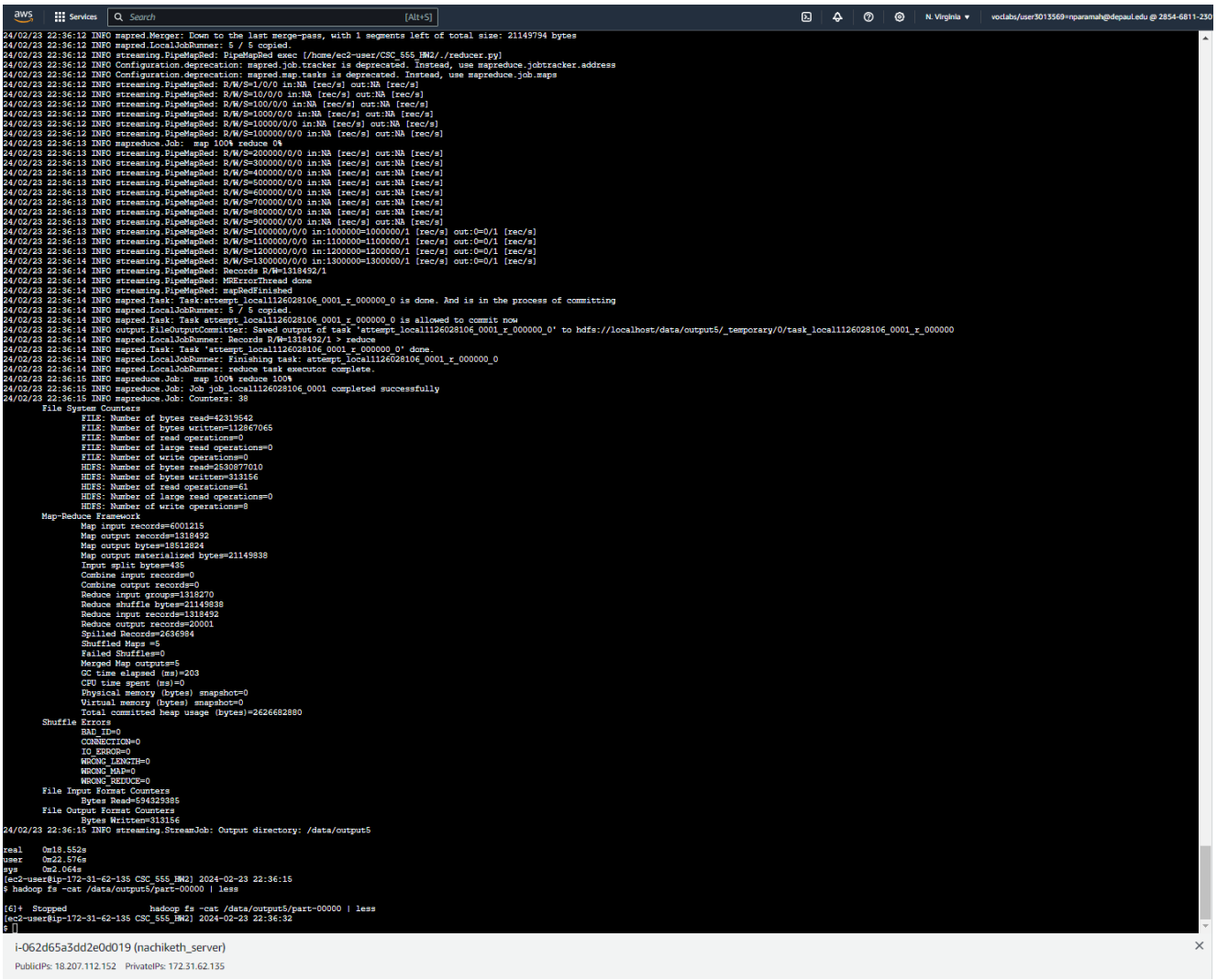




QUERY 4:

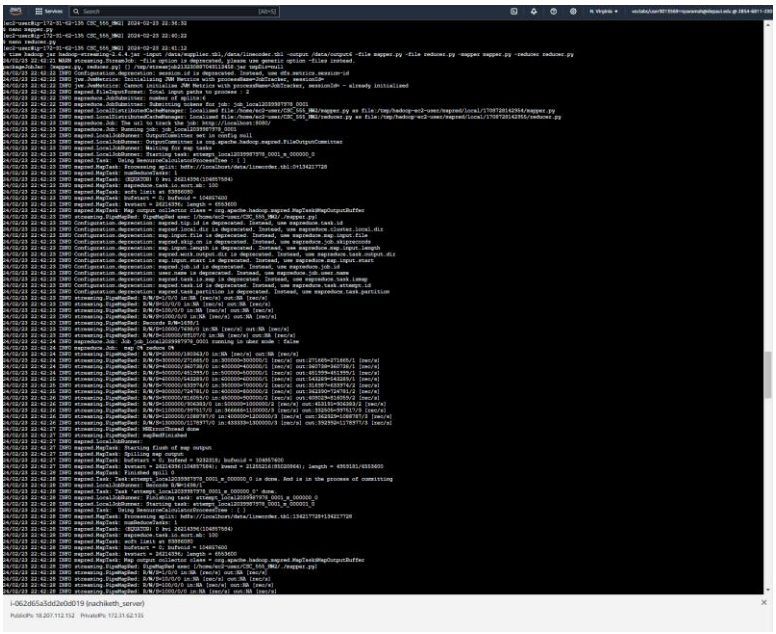
SELECT lo_custkey, SUM(lo_extendedprice) AS revenue FROM lineorder WHERE lo_quantity < 12 GROUP BY lo custkey

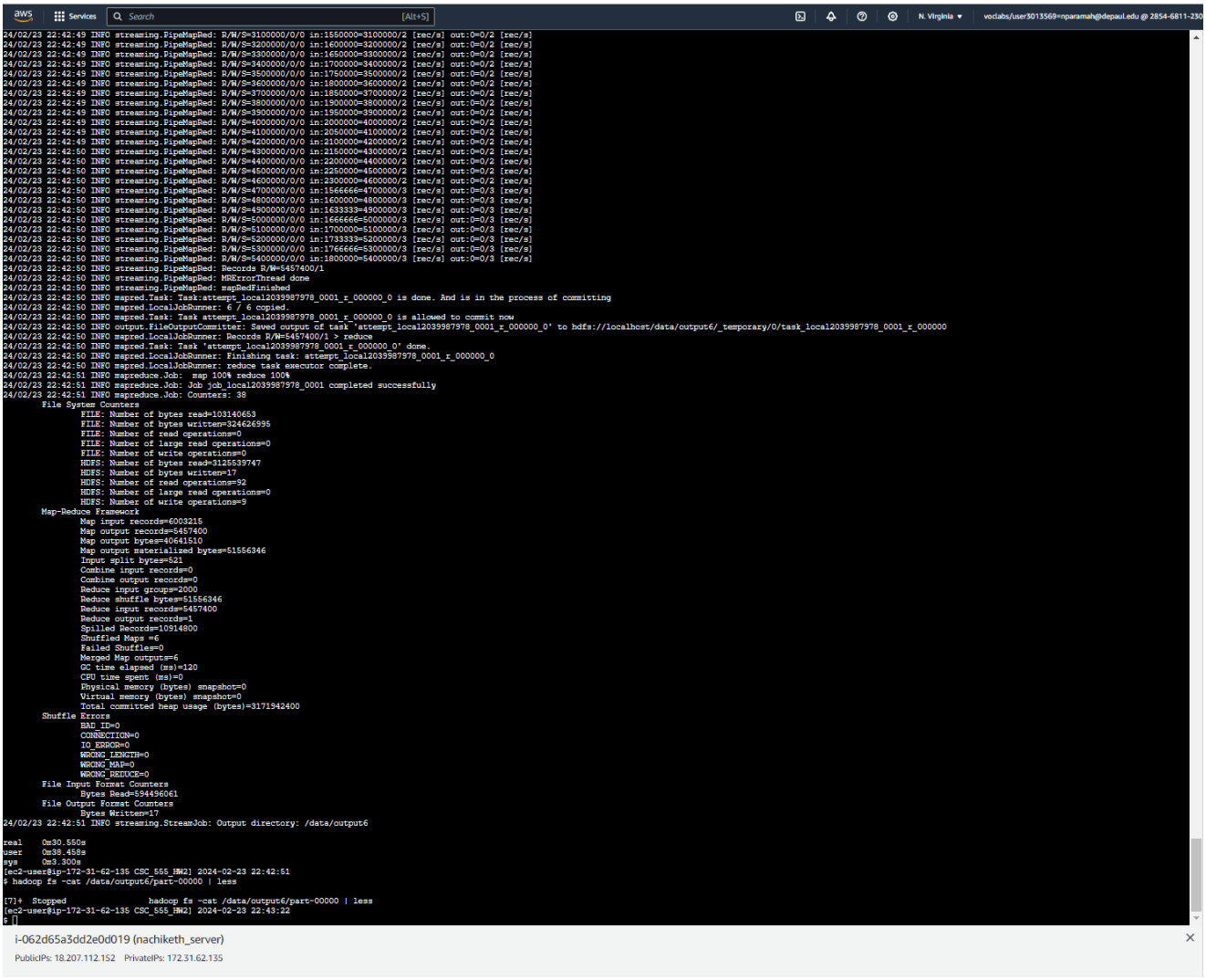




QUERY 5:

SELECT s_suppkey FROM supplier MINUS SELECT lo_suppkey FROM lineorder WHERE lo_discount < 10

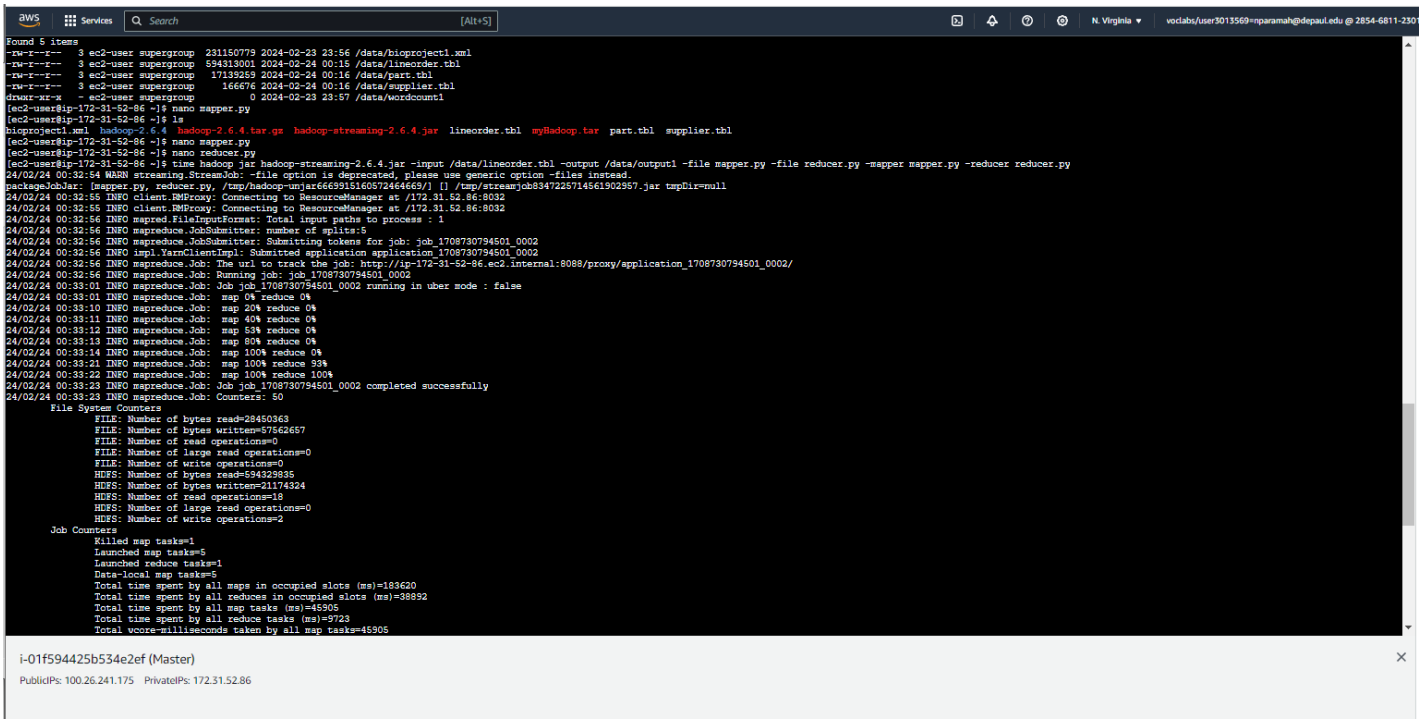


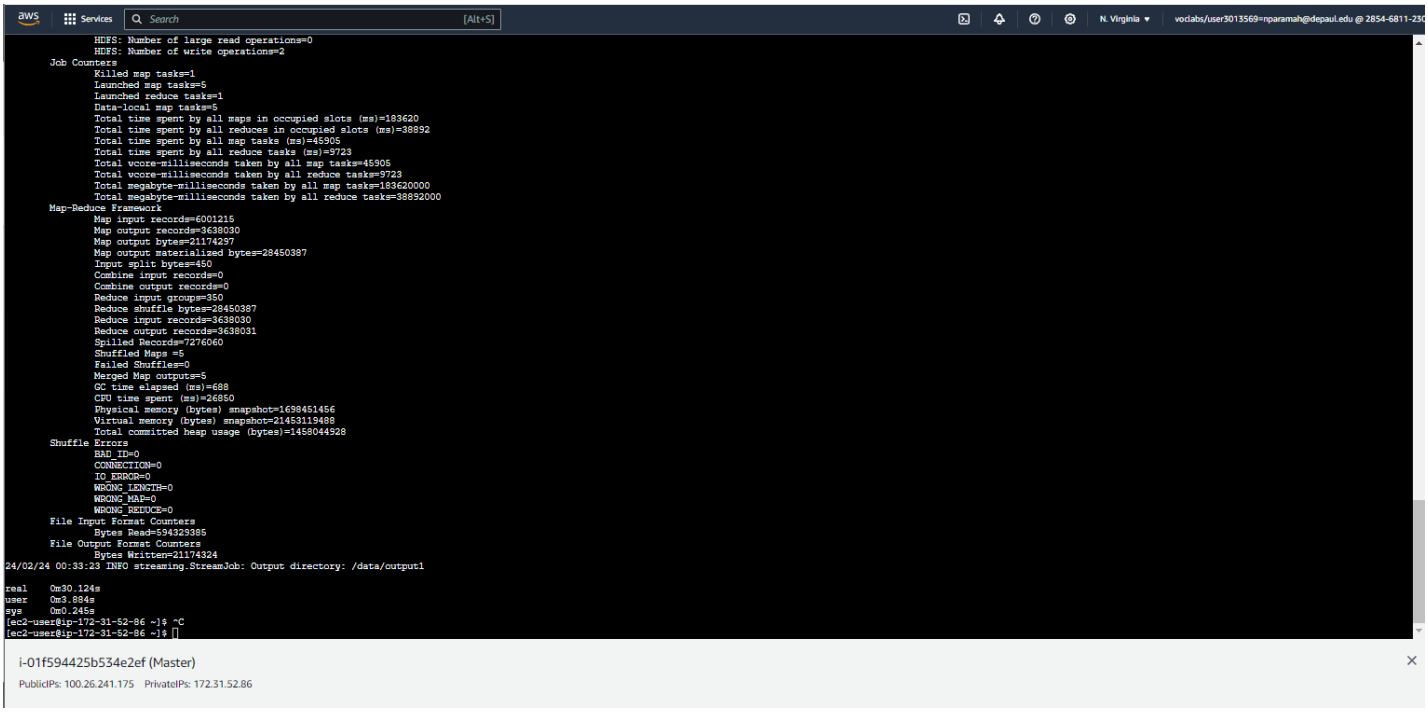


MULTI NODE:

QUERY 1:

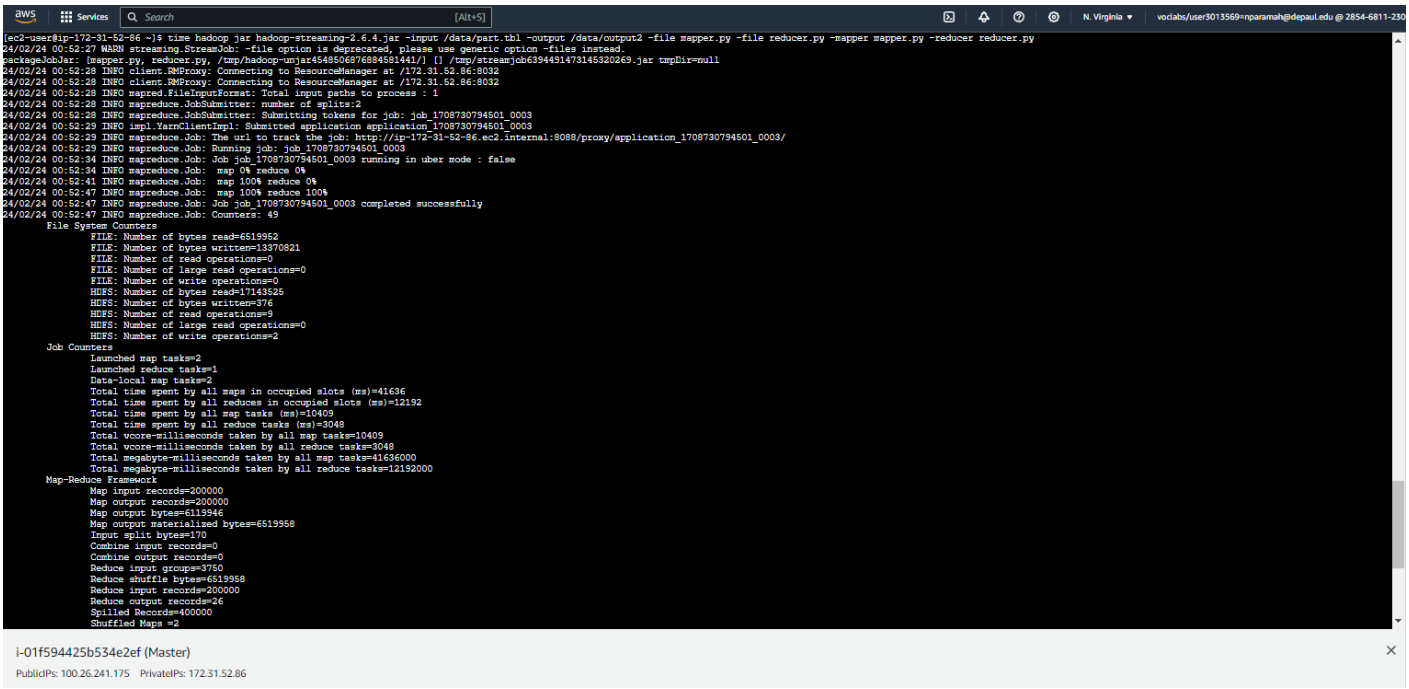
SELECT lo_quantity, lo_linenumber FROM lineorder WHERE lo discount < 10 AND lo tax > 2

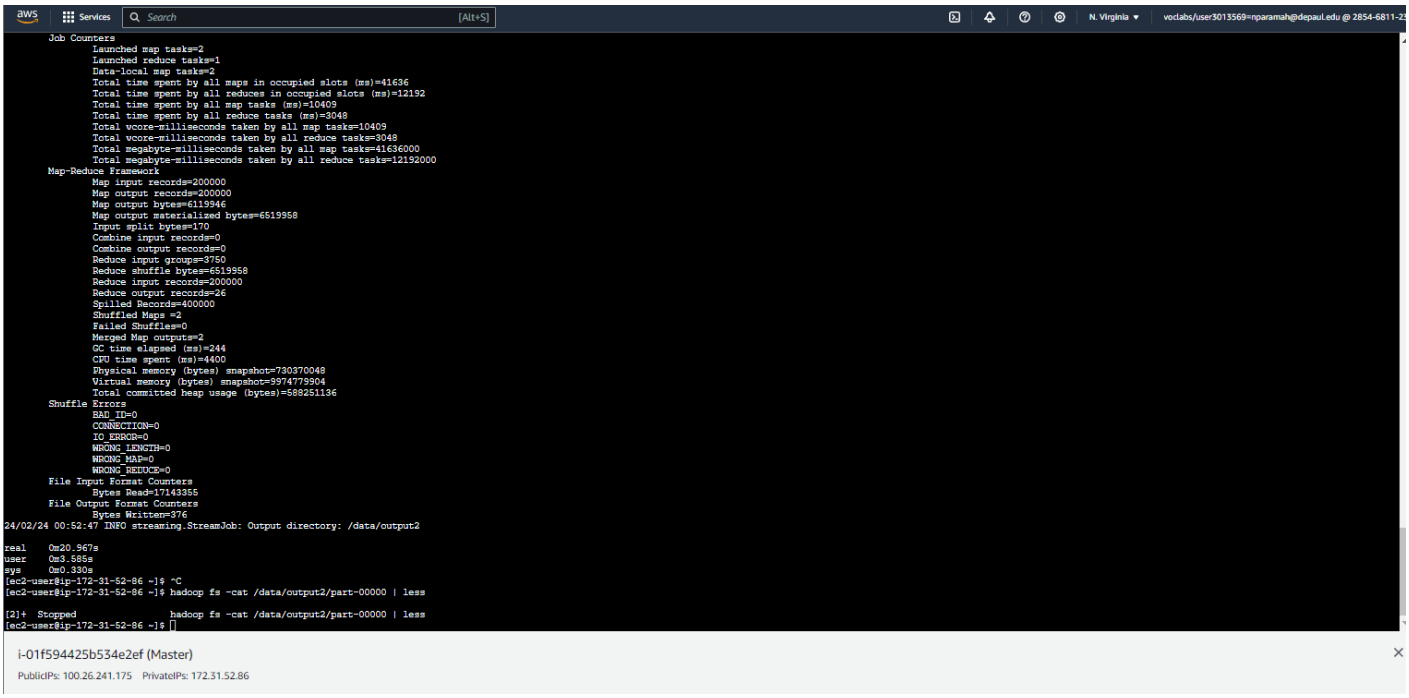




QUERY 2:

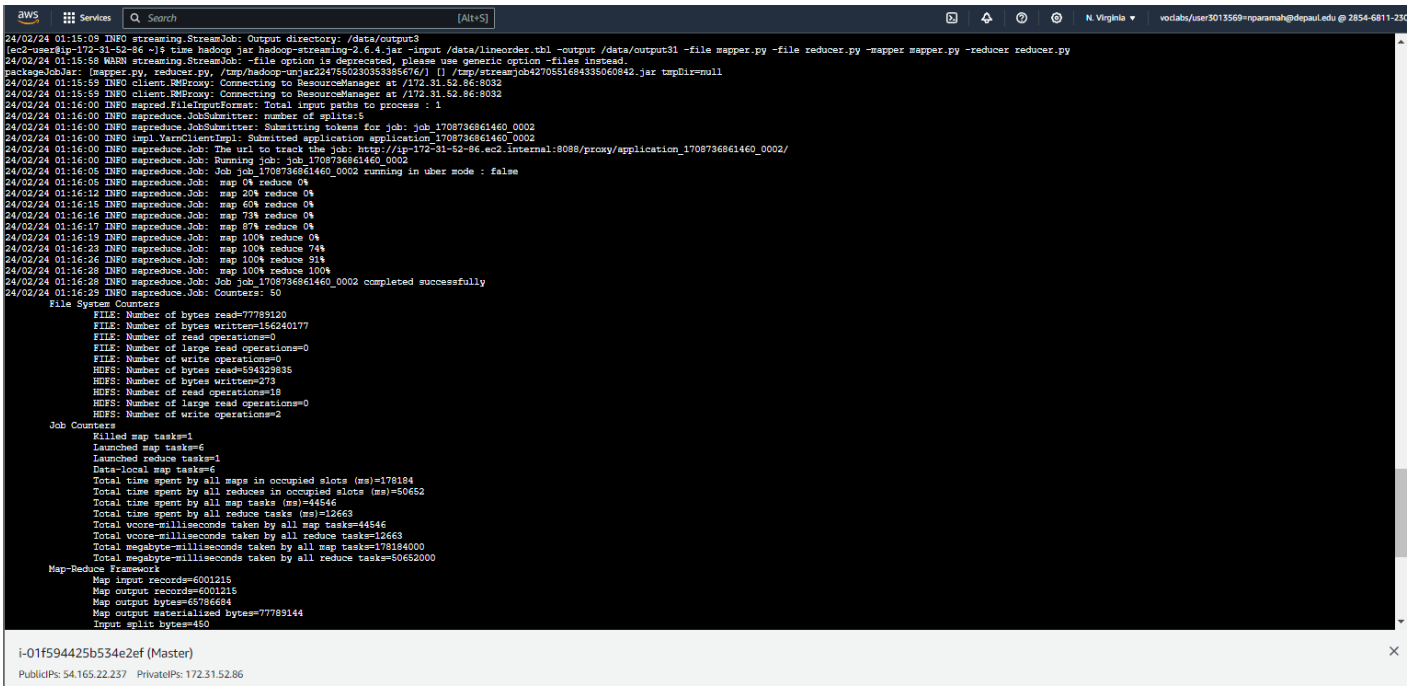
SELECT p_category, COUNT(p_type) FROM part GROUP BY p_category

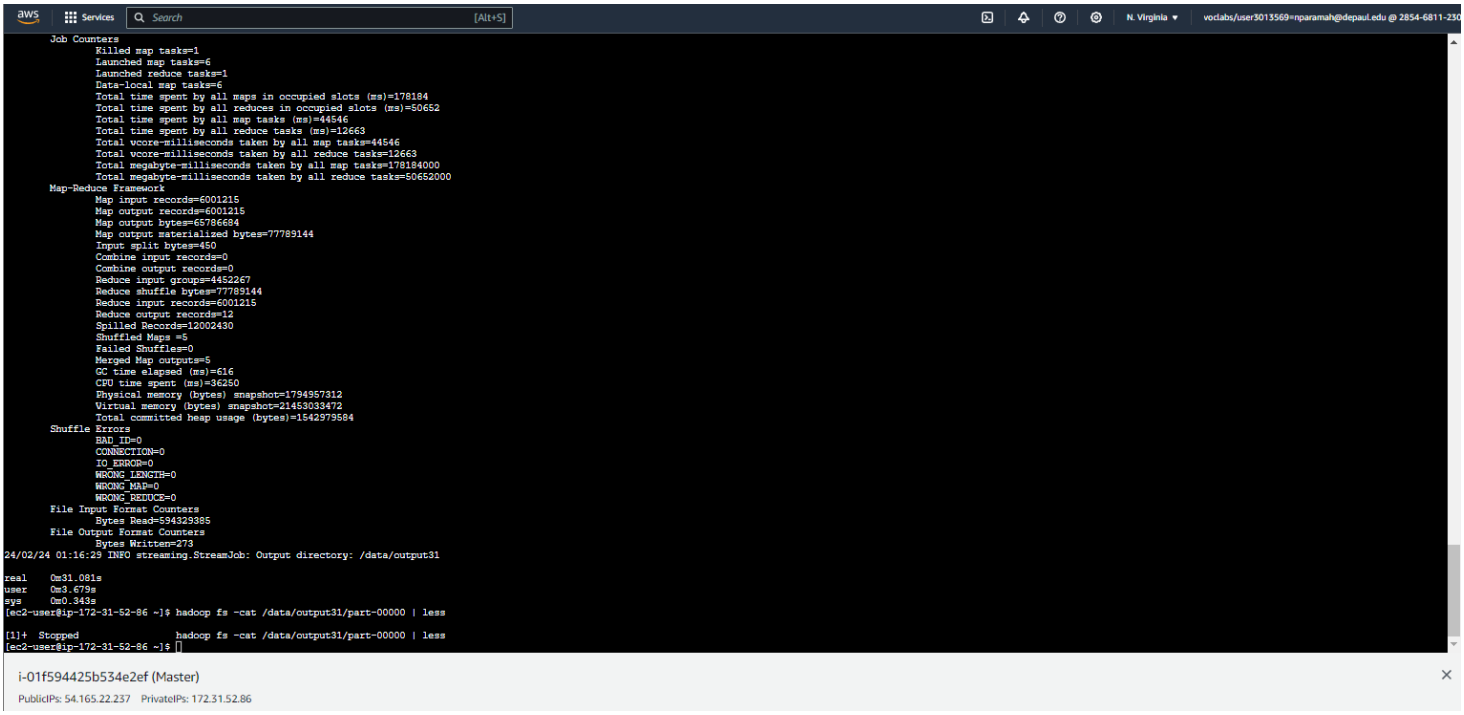




QUERY 3:

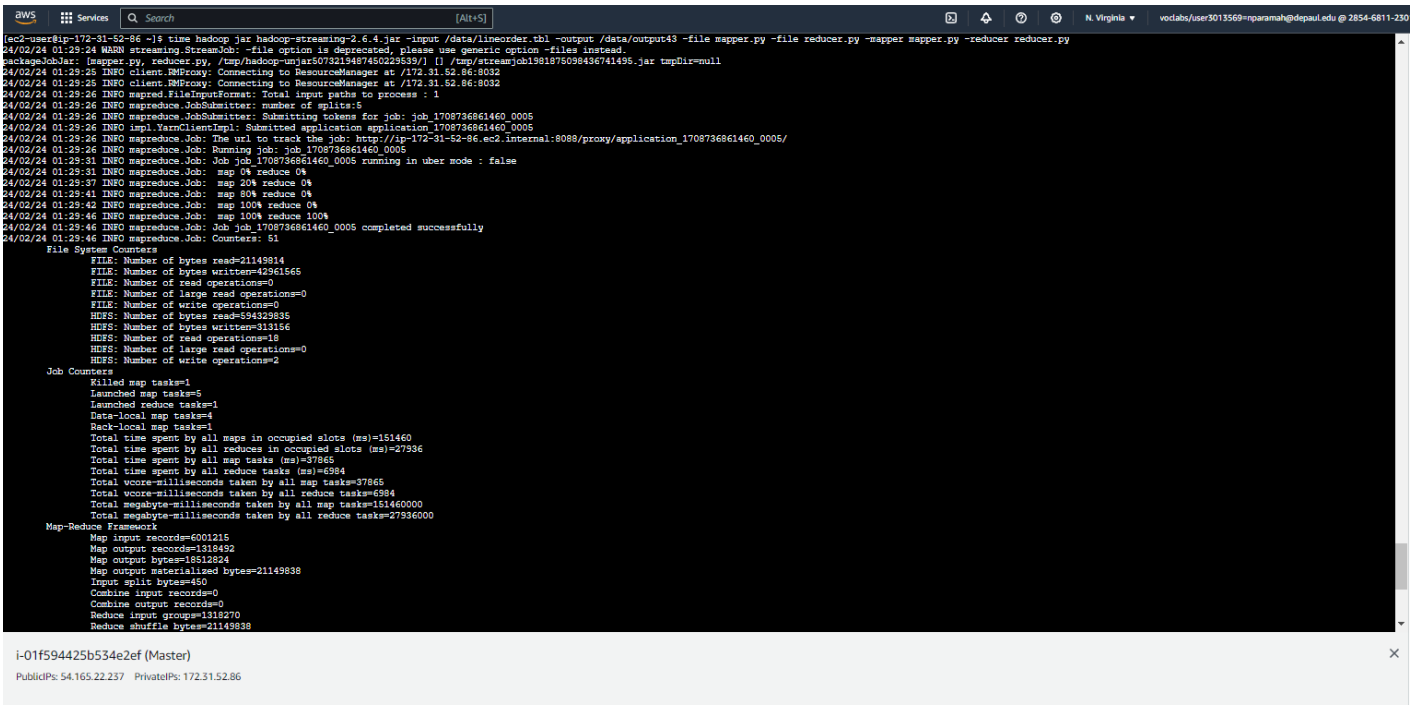
SELECT lo_discount, AVG(lo_extendedprice) FROM lineorder GROUP BY lo_discount

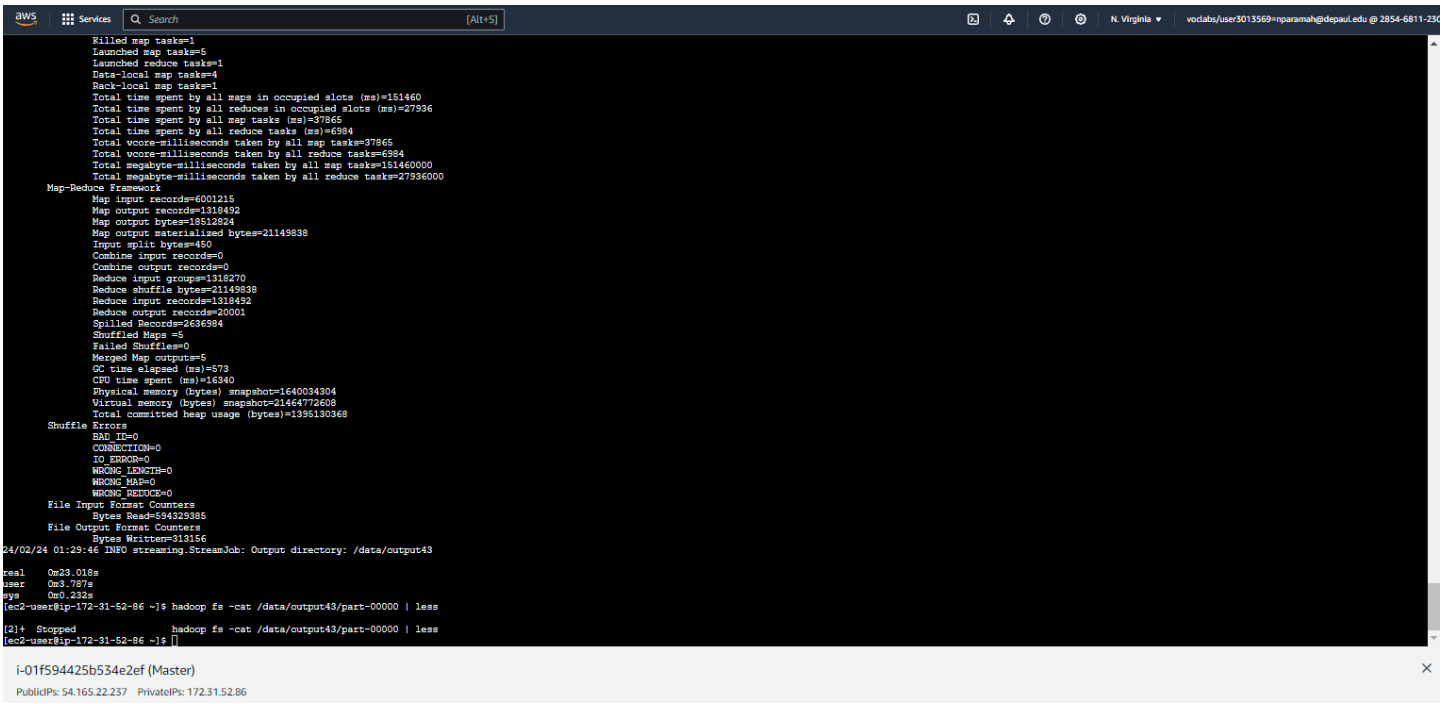




QUERY 4:

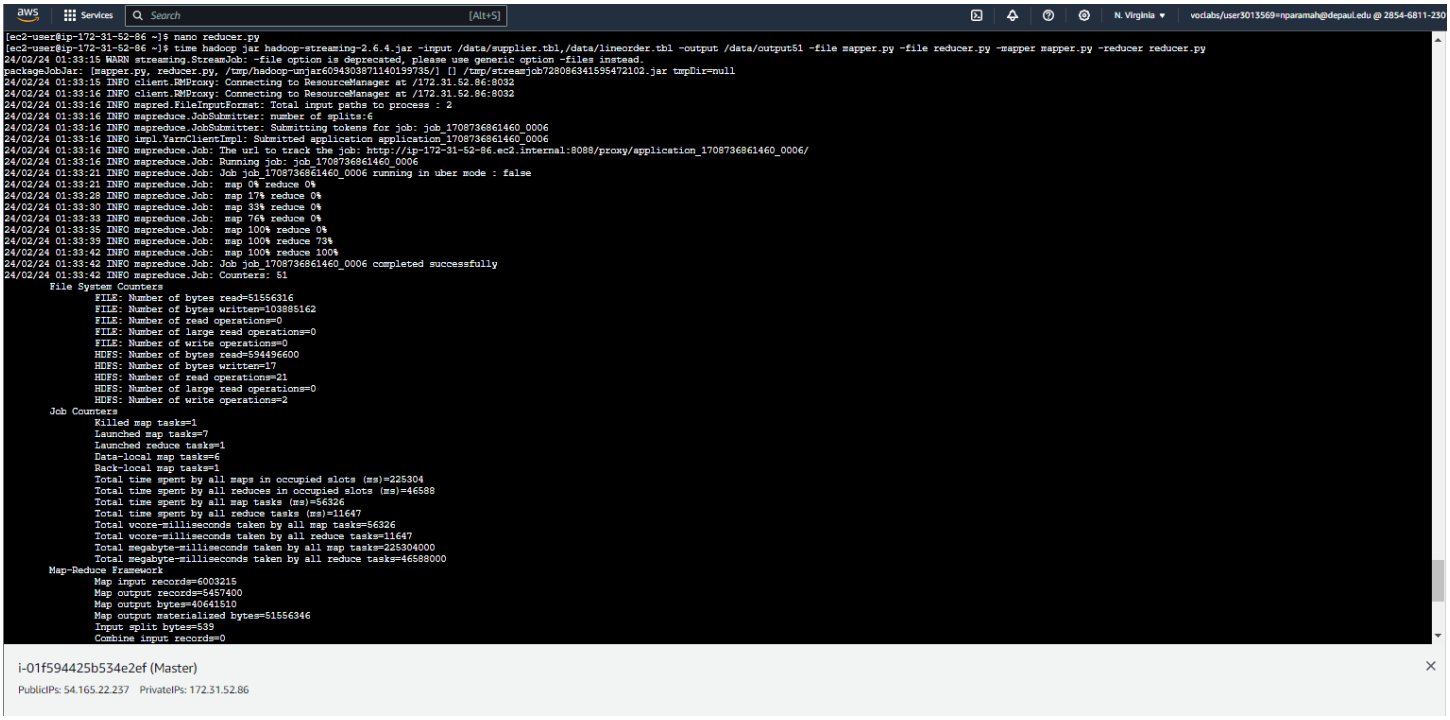
SELECT lo_custkey, SUM(lo_extendedprice) AS revenue FROM lineorder WHERE lo_quantity < 12 GROUP BY lo custkey

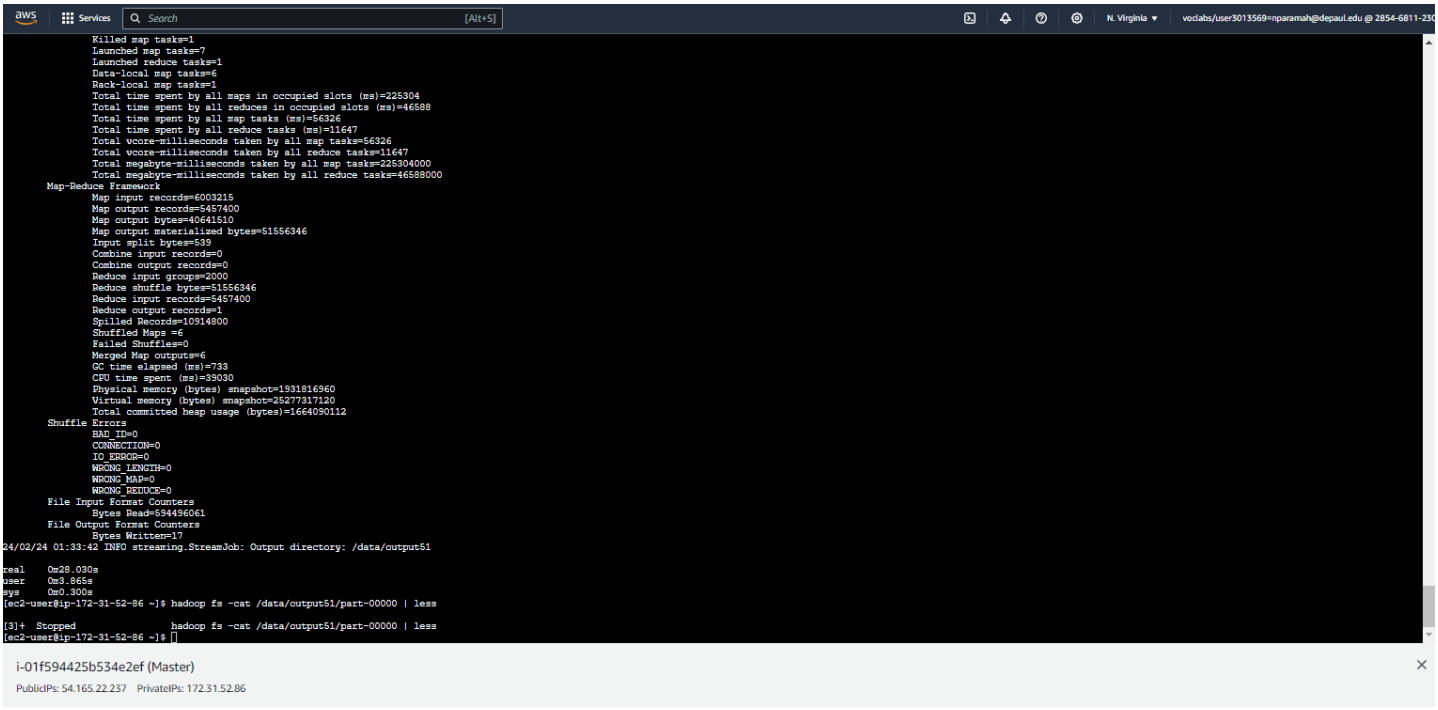




QUERY 5:

SELECT s_suppkey FROM supplier MINUS SELECT lo_suppkey FROM lineorder WHERE lo_discount < 10





QUESTION 2:

Implement, execute, and time the following query using Hadoop streaming with python.

SELECT lo_quantity, MAX(lo_revenue)

FROM (

SELECT lo_revenue, MAX(lo_quantity) as lo_quantity, MAX(lo_discount) as lo_discount

FROM lineorder

WHERE lo_orderpriority LIKE '%URGENT'

GROUP BY lo_revenue)

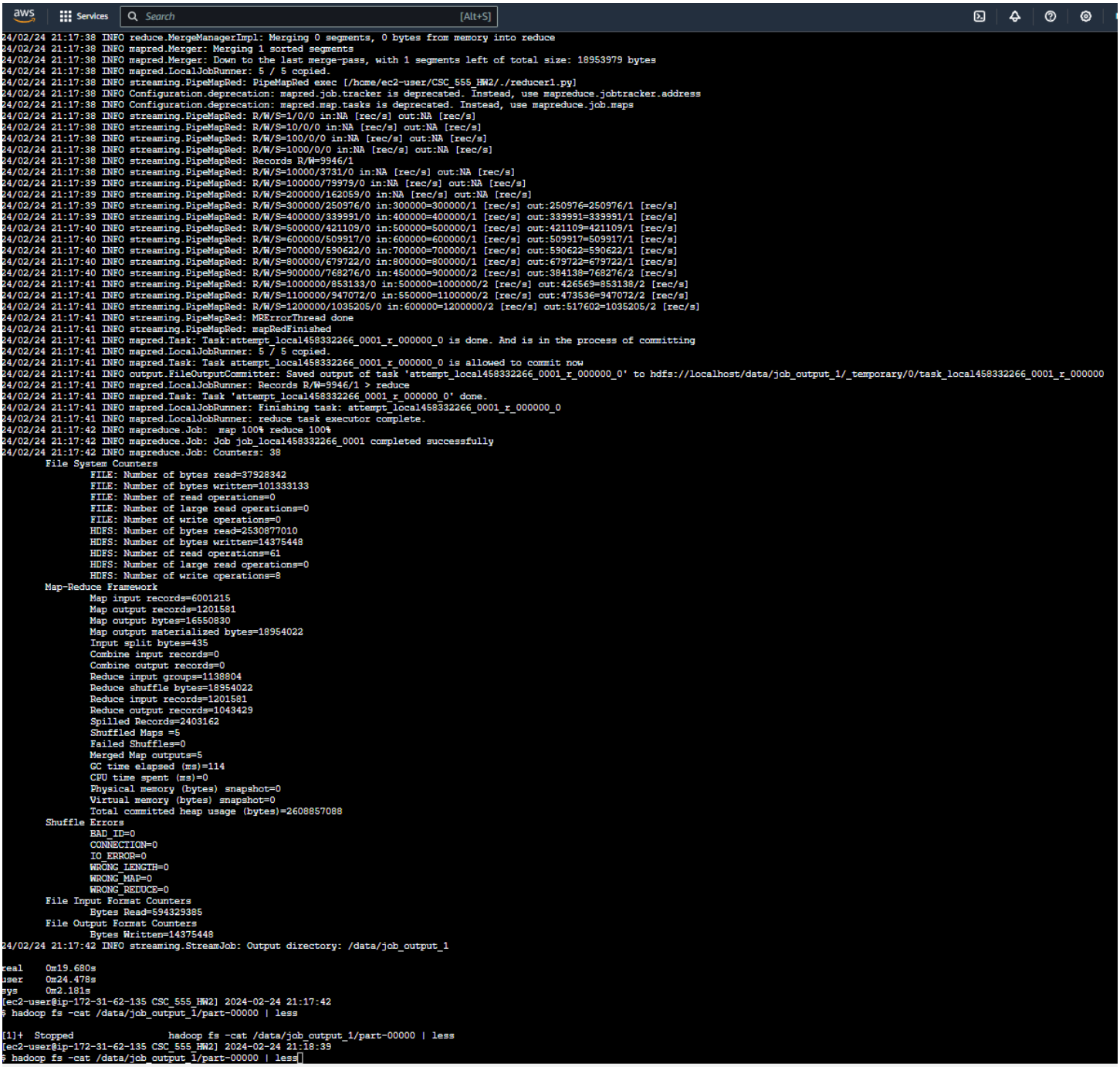
WHERE lo_discount BETWEEN 5 AND 8

GROUP BY lo_quantity;

SINGLE NODE:

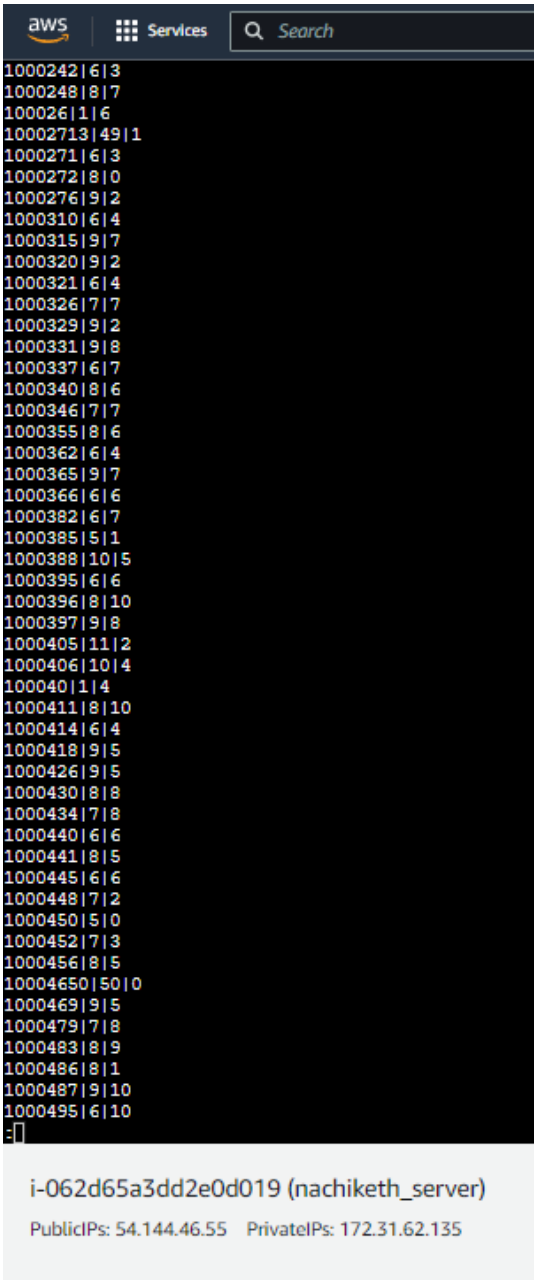
JOB 1:

time hadoop jar hadoop-streaming-2.6.4.jar -input /data/lineorder.tbl -output /data/job_output_1 -mapper mapper1.py -reducer reducer1.py -file mapper1.py -file reducer1.py



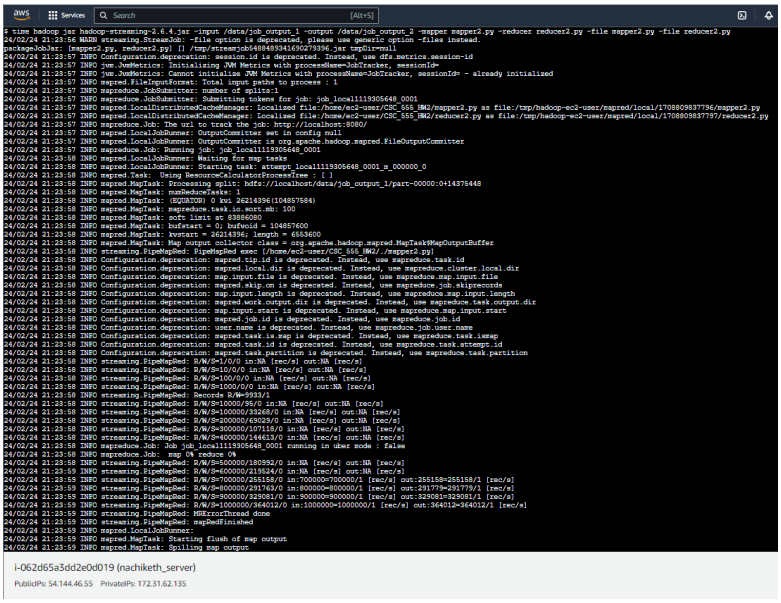
i-062d65a3dd2e0d019 (nachiketh_server)

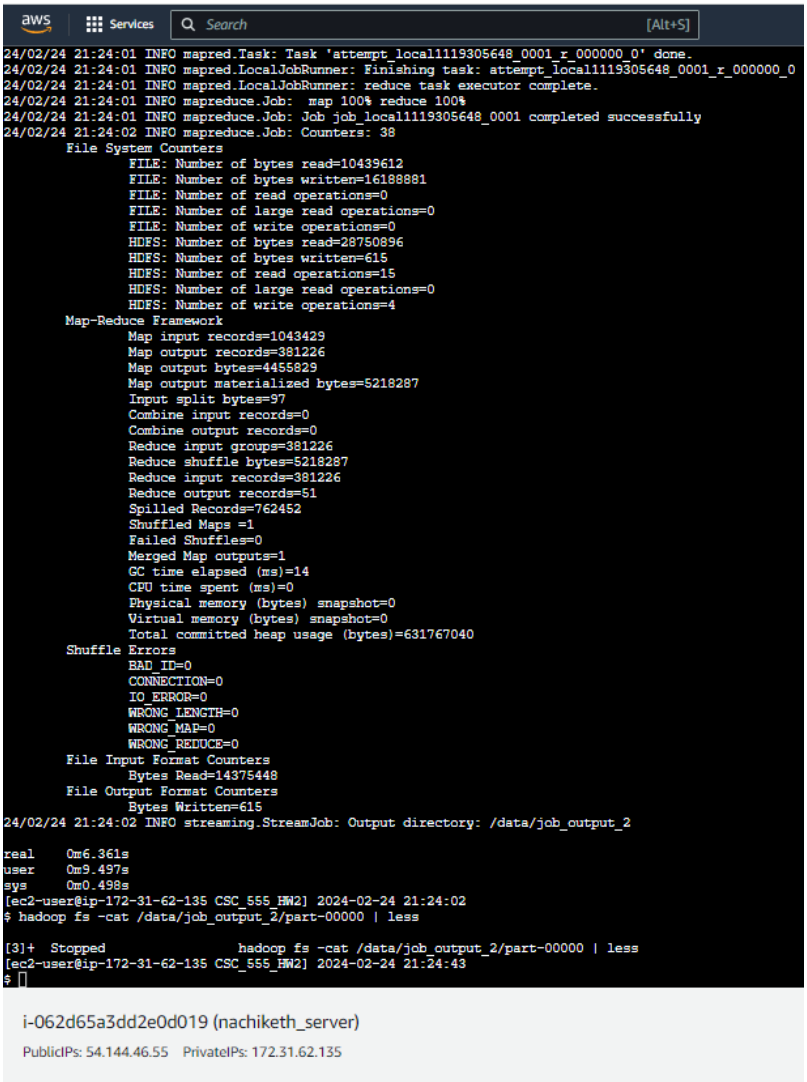
PublicIPs: 54.144.46.55 PrivateIPs: 172.31.62.135

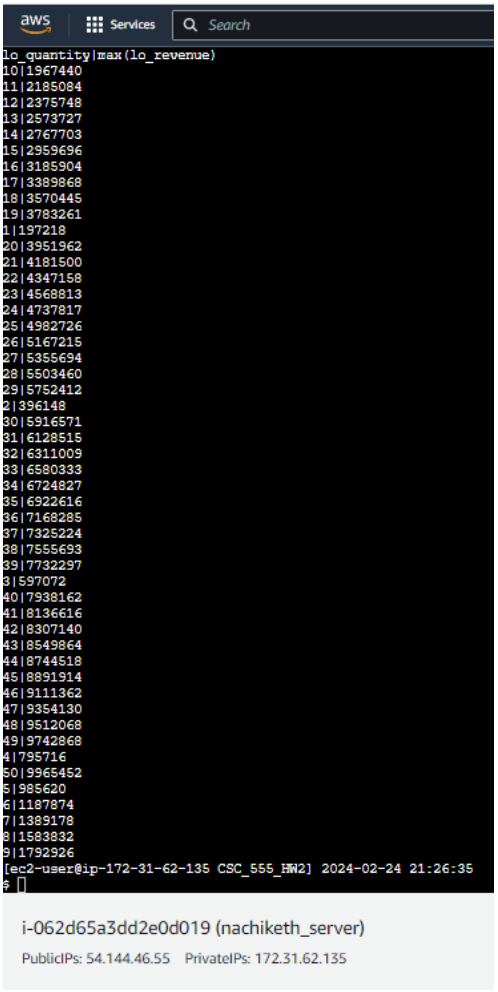

aws
Services
Search
1000242|6|3
1000248|8|7
100026|1|6
10002713|49|1
1000271|6|3
1000272|8|0
1000276|9|2
1000310|6|4
1000315|9|7
1000320|9|2
1000321|6|4
1000326|7|7
1000329|9|2
1000331|9|8
1000337|6|7
1000340|8|6
1000346|7|7
1000355|8|6
1000362|6|4
1000365|9|7
1000366|6|6
1000382|6|7
1000385|5|1
1000388|10|5
1000395|6|6
1000396|8|10
1000397|9|8
1000405|11|2
1000406|10|4
100040|1|4
1000411|8|10
1000414|6|4
1000418|9|5
1000426|9|5
1000430|8|8
1000434|7|8
1000440|6|6
1000441|8|5
1000445|6|6
1000448|7|2
1000450|5|0
1000452|7|3
1000456|8|5
10004650|50|0
1000469|9|5
1000479|7|8
1000483|8|9
1000486|8|1
1000487|9|10
1000495|6|10
:
i-062d65a3dd2e0d019 (nachiketh_server)
PublicIPs: 54.144.46.55 PrivateIPs: 172.31.62.135

JOB 2:

time hadoop jar hadoop-streaming-2.6.4.jar -input /data/job_output_1 -output /data/job_output_2 -mapper mapper2.py -reducer reducer2.py -file mapper2.py -file reducer2.py





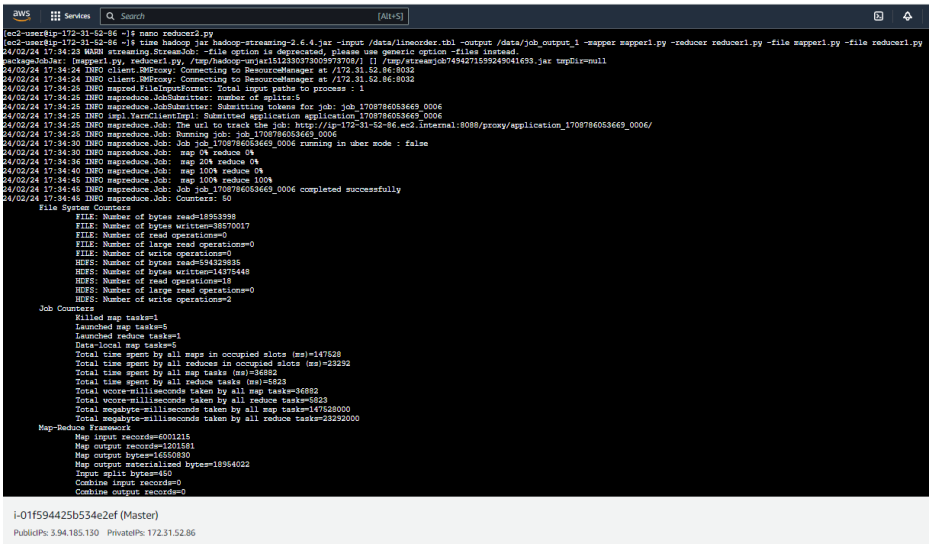


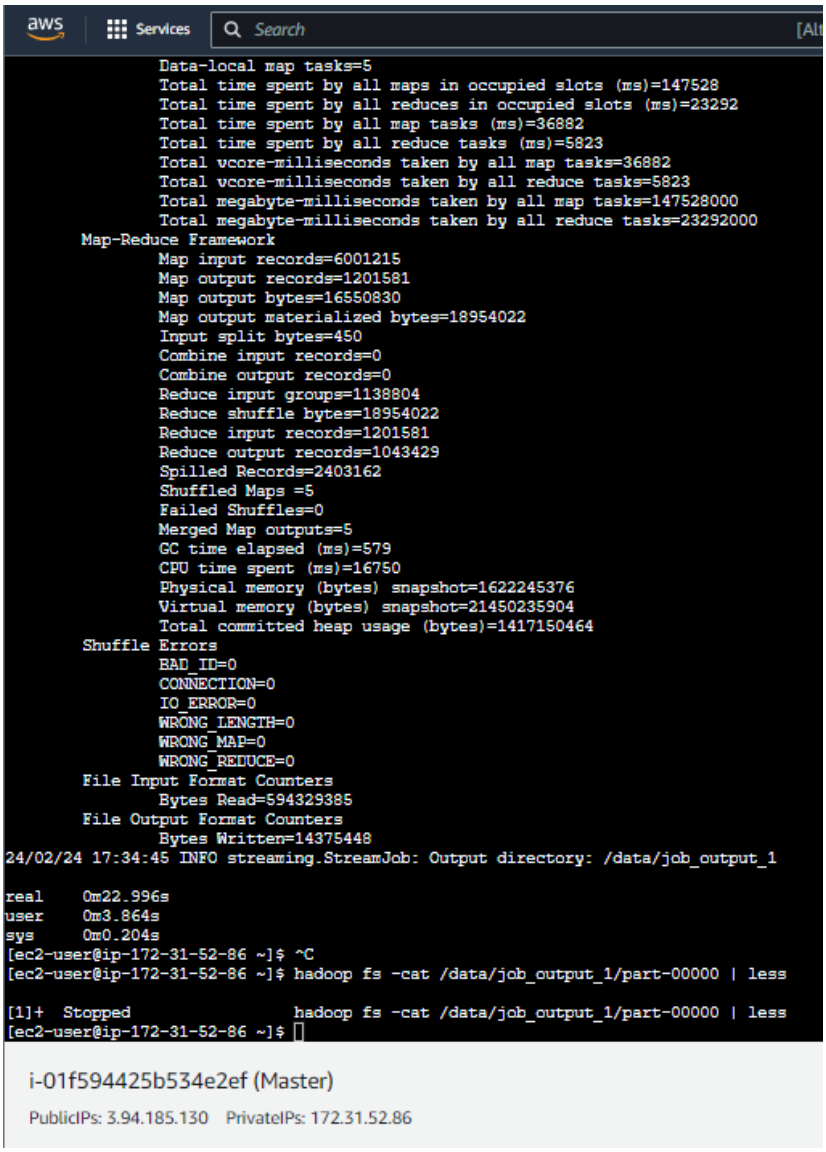
MULTI NODE RUNS:

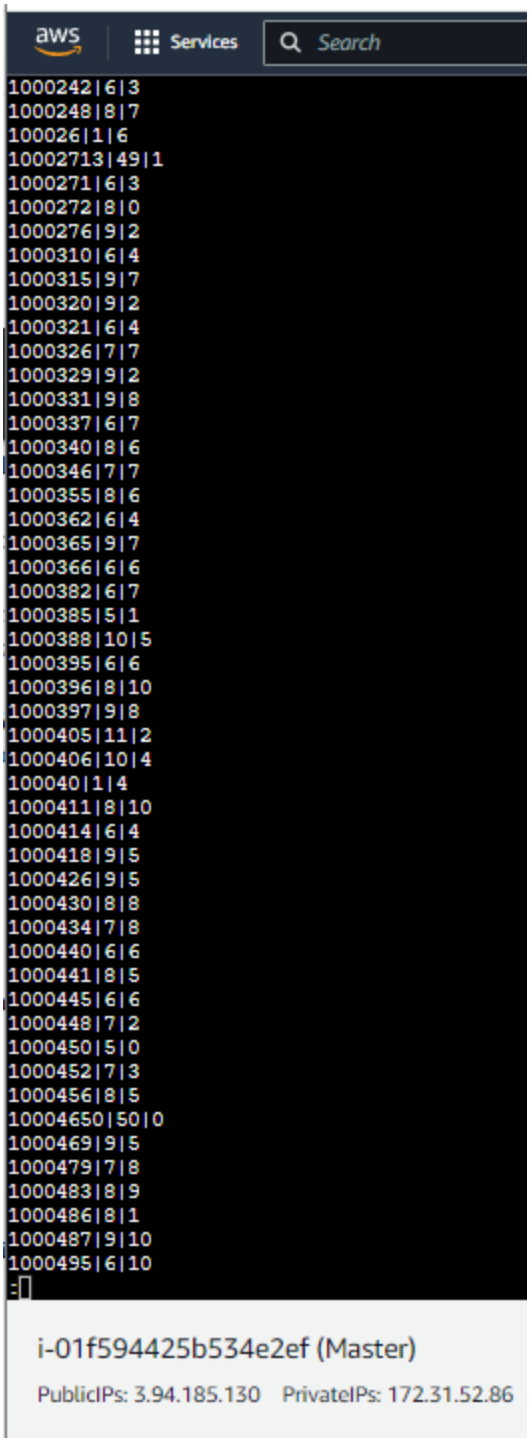
JOB 1:

time hadoop jar hadoop-streaming-2.6.4.jar -input /data/lineorder.tbl -output /data/job_output_1 -mapper mapper1.py -reducer reducer1.py -file mapper1.py -file reducer1.py

job 1 output:




aws
Services
Search
[Alt
Data-local map tasks=5
Total time spent by all maps in occupied slots (ms)=147528
Total time spent by all reduces in occupied slots (ms)=23292
Total time spent by all map tasks (ms)=36882
Total time spent by all reduce tasks (ms)=5823
Total vcore-milliseconds taken by all map tasks=36882
Total vcore-milliseconds taken by all reduce tasks=5823
Total megabyte-milliseconds taken by all map tasks=147528000
Total megabyte-milliseconds taken by all reduce tasks=23292000
Map-Reduce Framework
Map input records=6001215
Map output records=1201581
Map output bytes=16550830
Map output materialized bytes=18954022
Input split bytes=450
Combine input records=0
Combine output records=0
Reduce input groups=1138804
Reduce shuffle bytes=18954022
Reduce input records=1201581
Reduce output records=1043429
Spilled Records=2403162
Shuffled Maps =5
Failed Shuffles=0
Merged Map outputs=5
GC time elapsed (ms)=579
CPU time spent (ms)=16750
Physical memory (bytes) snapshot=1622245376
Virtual memory (bytes) snapshot=21450235904
Total committed heap usage (bytes)=1417150464
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=594329385
File Output Format Counters
Bytes Written=14375448
24/02/24 17:34:45 INFO streaming.StreamJob: Output directory: /data/job_output_1
real 0m22.996s
user 0m3.864s
sys 0m0.204s
[ec2-user@ip-172-31-52-86 ~]$ ^C
[ec2-user@ip-172-31-52-86 ~]$ hadoop fs -cat /data/job_output_1/part-00000 | less
[1]+ Stopped hadoop fs -cat /data/job_output_1/part-00000 | less
[ec2-user@ip-172-31-52-86 ~]$
i-01f594425b534e2ef (Master)
PublicIPs: 3.94.185.130 PrivateIPs: 172.31.52.86


aws
Services
Search
1000242|6|3
1000248|8|7
100026|1|6
10002713|49|1
1000271|6|3
1000272|8|0
1000276|9|2
1000310|6|4
1000315|9|7
1000320|9|2
1000321|6|4
1000326|7|7
1000329|9|2
1000331|9|8
1000337|6|7
1000340|8|6
1000346|7|7
1000355|8|6
1000362|6|4
1000365|9|7
1000366|6|6
1000382|6|7
1000385|5|1
1000388|10|5
1000395|6|6
1000396|8|10
1000397|9|8
1000405|11|2
1000406|10|4
100040|1|4
1000411|8|10
1000414|6|4
1000418|9|5
1000426|9|5
1000430|8|8
1000434|7|8
1000440|6|6
1000441|8|5
1000445|6|6
1000448|7|2
1000450|5|0
1000452|7|3
1000456|8|5
10004650|50|0
1000469|9|5
1000479|7|8
1000483|8|9
1000486|8|1
1000487|9|10
1000495|6|10
:
i-01f594425b534e2ef (Master)
PublicIPs: 3.94.185.130 PrivateIPs: 172.31.52.86

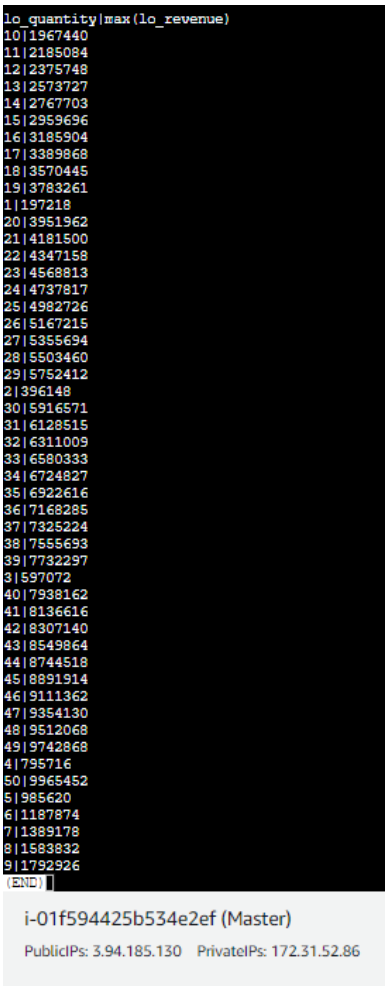
JOB 2:

time hadoop jar hadoop-streaming-2.6.4.jar -input /data/job_output_1 -output /data/job2_output -mapper mapper2.py -reducer reducer2.py -file mapper2.py -file reducer2.py

```
[ec2-user@ip-172-31-52-86 ~]$ time hadoop jar hadoop-streaming-2.6.4.jar -input /data/job_output_1 -output /data/job_output2 -mapper mapper2.py -reducer reducer2.py -file mapper2.py -file reducer2.py
24/02/24 17:47:21 WARN streaming.StreamJob: -file option is deprecated, please use generic option -files instead.
packageJobJar: [mapper2.py, reducer2.py, /tmp/hadoop-unjar2015844659589727637/] [] /tmp/streamjob4232451981533053644.jar tmpDir=null
24/02/24 17:47:22 INFO client.RMProxy: Connecting to ResourceManager at /172.31.52.86:8032
24/02/24 17:47:22 INFO client.RMProxy: Connecting to ResourceManager at /172.31.52.86:8032
24/02/24 17:47:22 INFO mapred.FileInputFormat: Total input paths to process : 1
24/02/24 17:47:22 INFO mapreduce.JobSubmitter: number of splits:2
24/02/24 17:47:22 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1708786053669_0009
24/02/24 17:47:23 INFO impl.YarnClientImpl: Submitted application application_1708786053669_0009
24/02/24 17:47:23 INFO mapreduce.Job: The url to track the job: http://ip-172-31-52-86.ec2.internal:8088/proxy/application_1708786053669_0009/
24/02/24 17:47:23 INFO mapreduce.Job: Running job: job_1708786053669_0009
24/02/24 17:47:28 INFO mapreduce.Job: Job job_1708786053669_0009 running in uber mode : false
24/02/24 17:47:28 INFO mapreduce.Job:  map 0% reduce 0%
24/02/24 17:47:35 INFO mapreduce.Job:  map 50% reduce 0%
24/02/24 17:47:36 INFO mapreduce.Job:  map 100% reduce 0%
24/02/24 17:47:41 INFO mapreduce.Job:  map 100% reduce 100%
24/02/24 17:47:41 INFO mapreduce.Job: Job job_1708786053669_0009 completed successfully
24/02/24 17:47:41 INFO mapreduce.Job: Counters: 50
        File System Counters
                FILE: Number of bytes read=5218287
                FILE: Number of bytes written=10767545
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=14379744
                HDFS: Number of bytes written=615
                HDFS: Number of read operations=9
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
        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)=45372
                Total time spent by all reduces in occupied slots (ms)=13000
                Total time spent by all map tasks (ms)=11343
                Total time spent by all reduce tasks (ms)=3250
                Total vcore-milliseconds taken by all map tasks=11343
                Total vcore-milliseconds taken by all reduce tasks=3250
                Total megabyte-milliseconds taken by all map tasks=45372000
                Total megabyte-milliseconds taken by all reduce tasks=13000000
        Map-Reduce Framework
                Map input records=1043429
                Map output records=381226
                Map output bytes=4455829
                Map output materialized bytes=5218293
                Input split bytes=200
                Combine input records=0
                Combine output records=0
                Reduce input groups=381226
                Reduce shuffle bytes=5218293
                Reduce input records=381226
                Reduce output records=51
                Spilled Records=762452
                Shuffled Maps =2
                Failed Shuffles=0
                Merged Map outputs=2
                GC time elapsed (ms)=281
                CPU time spent (ms)=5410
                Physical memory (bytes) snapshot=727285760
                Virtual memory (bytes) snapshot=9971367936
                Total committed heap usage (bytes)=579338240
        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=14379544
        File Output Format Counters
                Bytes Written=615
24/02/24 17:47:41 INFO streaming.StreamJob: Output directory: /data/job_output2

real    0m20.974s
user    0m3.746s
sys     0m0.276s
[ec2-user@ip-172-31-52-86 ~]$ hadoop fs -cat /data/job_output2/part-00000 | less

[4]+  Stopped                 hadoop fs -cat /data/job_output2/part-00000 | less
[ec2-user@ip-172-31-52-86 ~]$ hadoop fs -cat /data/job_output2/part-00000 | less

[5]+  Stopped                 hadoop fs -cat /data/job_output2/part-00000 | less
[ec2-user@ip-172-31-52-86 ~]$
```

i-01f594425b534e2ef (Master)

PublicIPs: 3.94.185.130 PrivateIPs: 172.31.52.86



JOB TYPE	SINGLE NODE	MULTI NODE
JOB 1	real 0m19.680s user 0m24.478s sys 0m2.181s	real 0m22.996s user 0m3.864s sys 0m0.204s
JOB 2	real 0m6.361s user 0m9.497s sys 0m0.498s	real 0m20.974s user 0m3.746s sys 0m0.276s

The single-node arrangement outperforms the multi-node system in terms of Job 1 execution times. When using a single node, Job 1's real time is 19.680 seconds; however, when using many nodes, it rises to 22.996 seconds. In a similar vein, the multi-node architecture also has longer system and user times. This suggests that there are more overhead costs involved in spreading the workload across several nodes than there are possible performance benefits, which causes Job 1 in the multi-node system to execute Job 1 more slowly. As an alternative, Job 2 shows different outcomes. Job 2 takes 6.361 real seconds to finish in the single-node setup, and 20.974 real seconds in the multi-node configuration. This significant increase in Job 2's runtime in the multi-node configuration points to inefficiencies or bottlenecks in the map's parallel processing and the distribution of tasks among several nodes.

CODE:

JOB 1:

MAPPER 1:

```
#!/usr/bin/python3
import sys

for line in sys.stdin:
    fields = line.strip().split('|')

    # Extract relevant fields
    lo_revenue = fields[12]
    lo_quantity = fields[8]
    lo_discount = fields[11]
    lo_orderpriority = fields[6]

    # Check if lo_orderpriority ends with 'URGENT'
    if lo_orderpriority.endswith('URGENT'):
        # Emit key-value pair (revenue, quantity, discount)
        print(f"{lo_revenue}|{lo_quantity}|{lo_discount}")
```

REDUCER 1:

```
#!/usr/bin/python3
import sys

curr_rev = None
max_qty = None
max_disc = None

for line in sys.stdin:
    # Split the input lines into fields
    lo_revenue, lo_quantity, lo_discount = line.strip().split('|')

    # Check if the revenue is the same as the current revenue
    if curr_rev == lo_revenue:
        # Update max quantity and max discount if necessary
        if int(lo_quantity) > max_qty:
            max_qty = int(lo_quantity)
            max_disc = int(lo_discount)
    else:
        # Output the result for the previous revenue if applicable
        if curr_rev:
            print(f"{curr_rev}|{max_qty}|{max_disc}")

        # Update curr_rev and reset max_qty and max_disc
        curr_rev = lo_revenue
        max_qty = int(lo_quantity)
        max_disc = int(lo_discount)
```

```
if curr_rev:
   print(f"{curr_rev}|{max_qty}|{max_disc}")
```

JOB 2:

MAPPER 2:

```
#!/usr/bin/python3
import sys
for line in sys.stdin:
   # Split the input line into fields
   lo_revenue, lo_quantity, lo_discount = line.strip().split('|')

   # Check if discount falls within the range 5 to 8
   if 5 <= int(lo_discount) <= 8:
      # Emit key-value pair (quantity, revenue)
      print(f"{lo_quantity}|{lo_revenue}")
```

REDUCER 2:

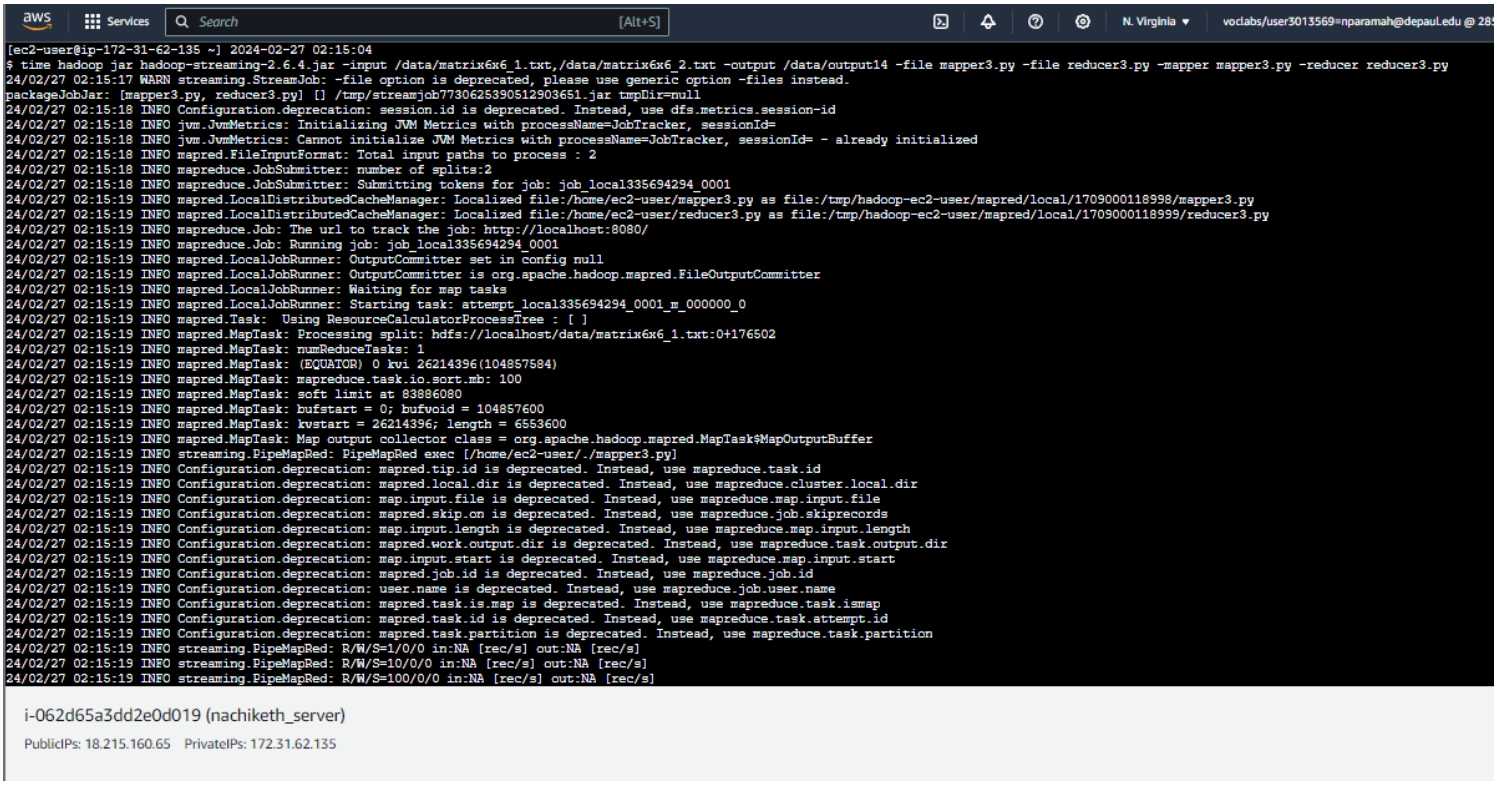
```
#!/usr/bin/python3
import sys
curr_qty = None
max_rev = None

# Print header
print("lo_quantity|max(lo_revenue)")

for line in sys.stdin:
   # Split the input line into fields
   lo_quantity, lo_revenue = line.strip().split('|')

   # Check if quantity is the same as the current quantity
   if curr_qty == lo_quantity:
      # Update max revenue if necessary
      max_rev = max(max_rev, int(lo_revenue))
   else:
      # Output the result for the previous quantity if applicable
      if curr_qty:
         print(f"{curr_qty}|{max_rev}")

      # Update curr_qty and reset max_rev
      curr_qty = lo_quantity
      max_rev = int(lo_revenue)

#final rev
if curr_qty:
   print(f"{curr_qty}|{max_rev}")
```

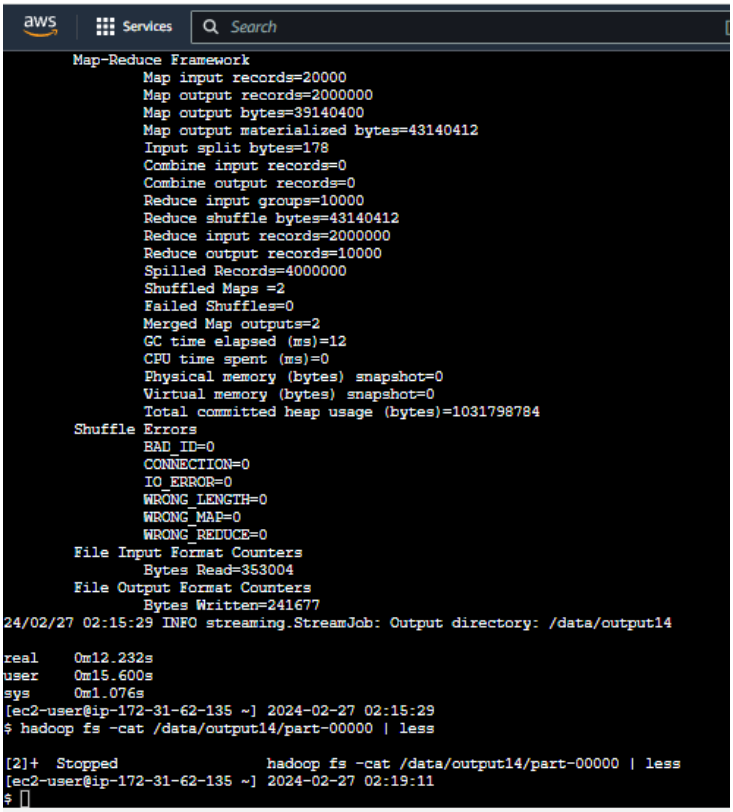
QUESTION 3:

Implement 1-step matrix multiplication using two matrix files matrix1.600 and matrix2.600 (posted in D2L). These matrix files both consist of a 600x600 matrix. Each row in the file matrix1.600 is of the form: i, j, A[i, j]. Similarly, each row in the file matrix2.600 is of the form: j, k, B[i, j] If a specific i, j (or j, k) combination is missing in matrix1.600 and matrix2.600 then its corresponding value is 0. You must implement 1-step matrix multiplication and write the corresponding mapper.py and reducer.py. You must run the matrix multiplication on a single-node and then on your multinode cluster and time its performance. Report the difference observed.

SINGLE NODE:

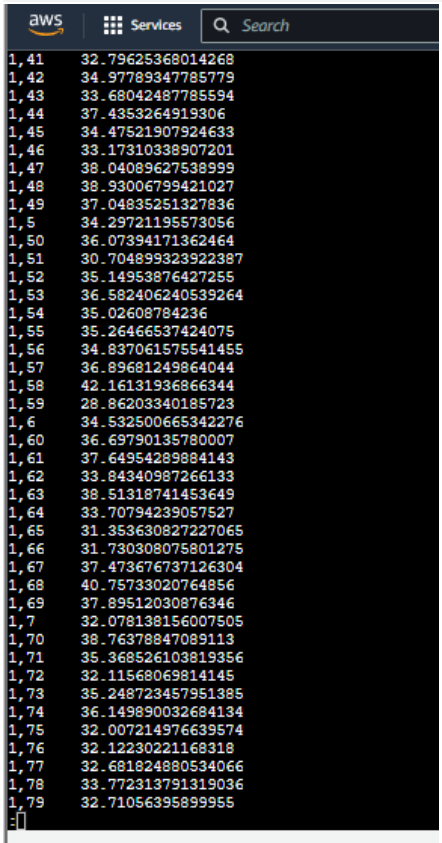
time hadoop jar hadoop-streaming-2.6.4.jar -input /data/matrix6x6_1.txt,/data/matrix6x6_2.txt -output /data/output14 -file mapper3.py -file reducer3.py -mapper mapper3.py -reducer reducer3.py





i-062d65a3dd2e0d019 (nachiketh_server)

PublicIPs: 18.215.160.65 PrivateIPs: 172.31.62.135

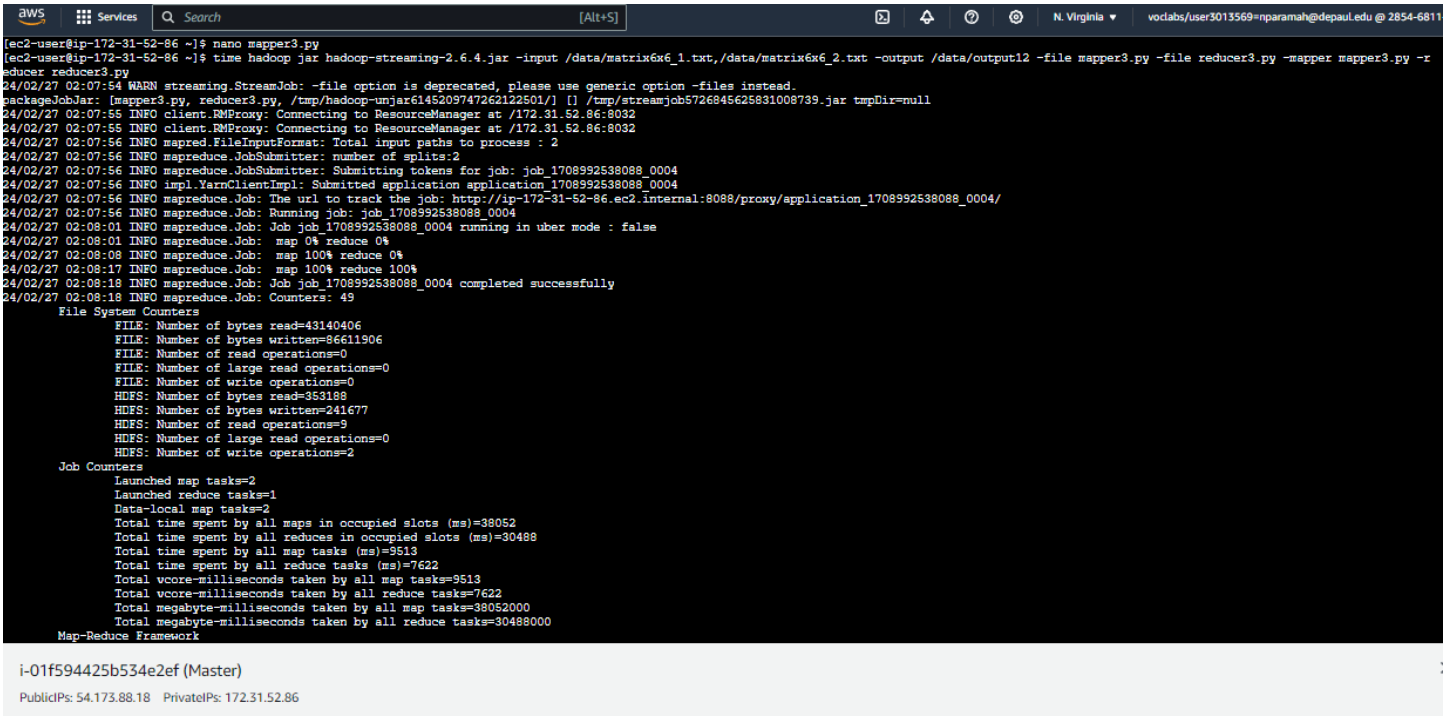


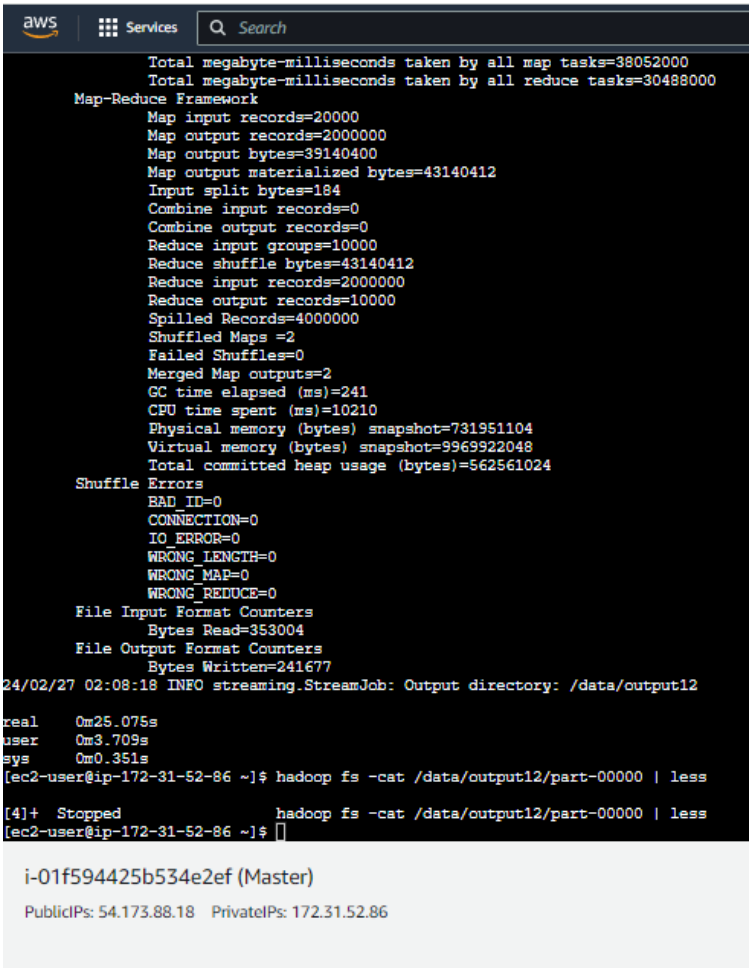
i-062d65a3dd2e0d019 (nachiketh_server)

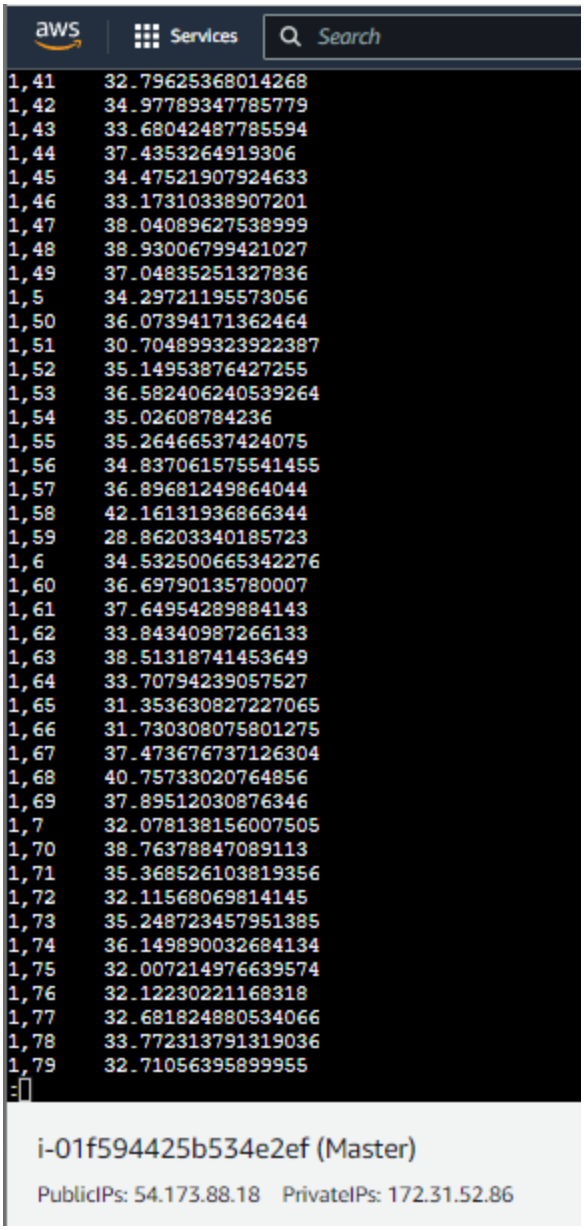
PublicIPs: 18.215.160.65 PrivateIPs: 172.31.62.135

MULTI NODE:

time hadoop jar hadoop-streaming-2.6.4.jar -input /data/matrix6x6_1.txt,/data/matrix6x6_2.txt -output /data/output12 -file mapper3.py -file reducer3.py -mapper mapper3.py -reducer reducer3.py



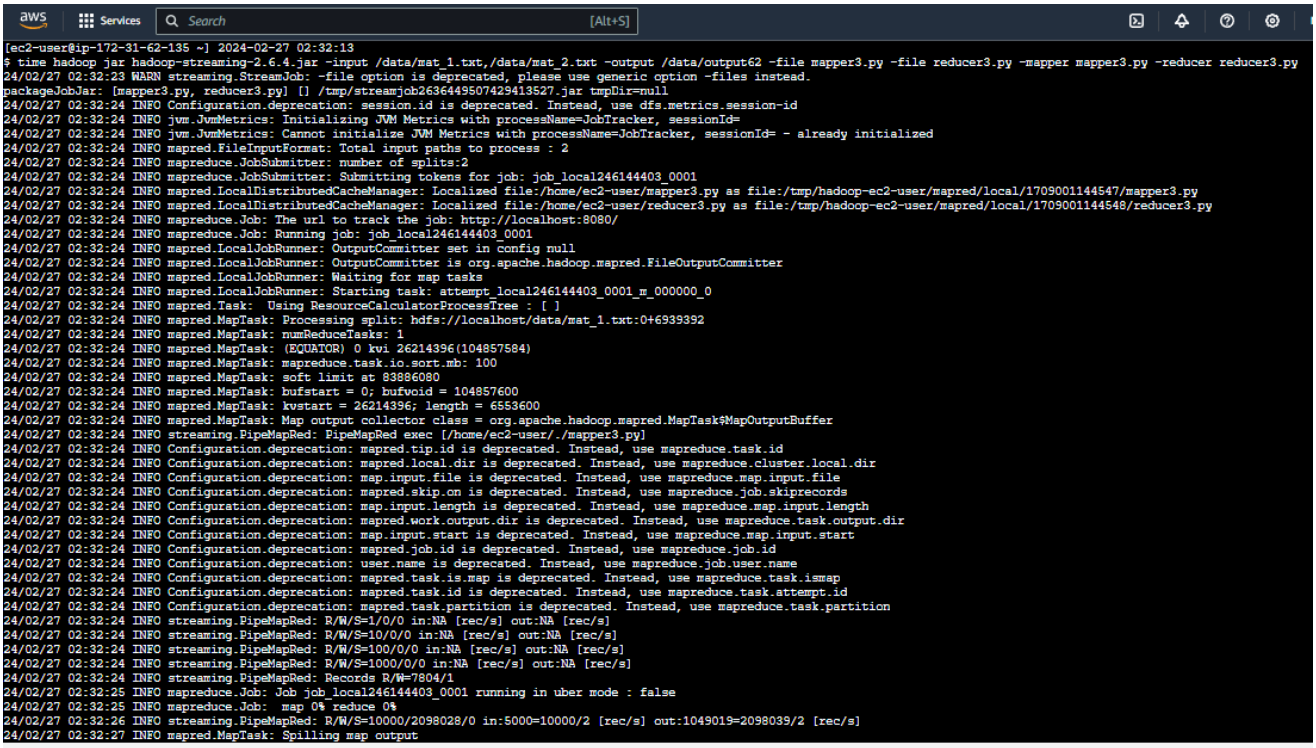


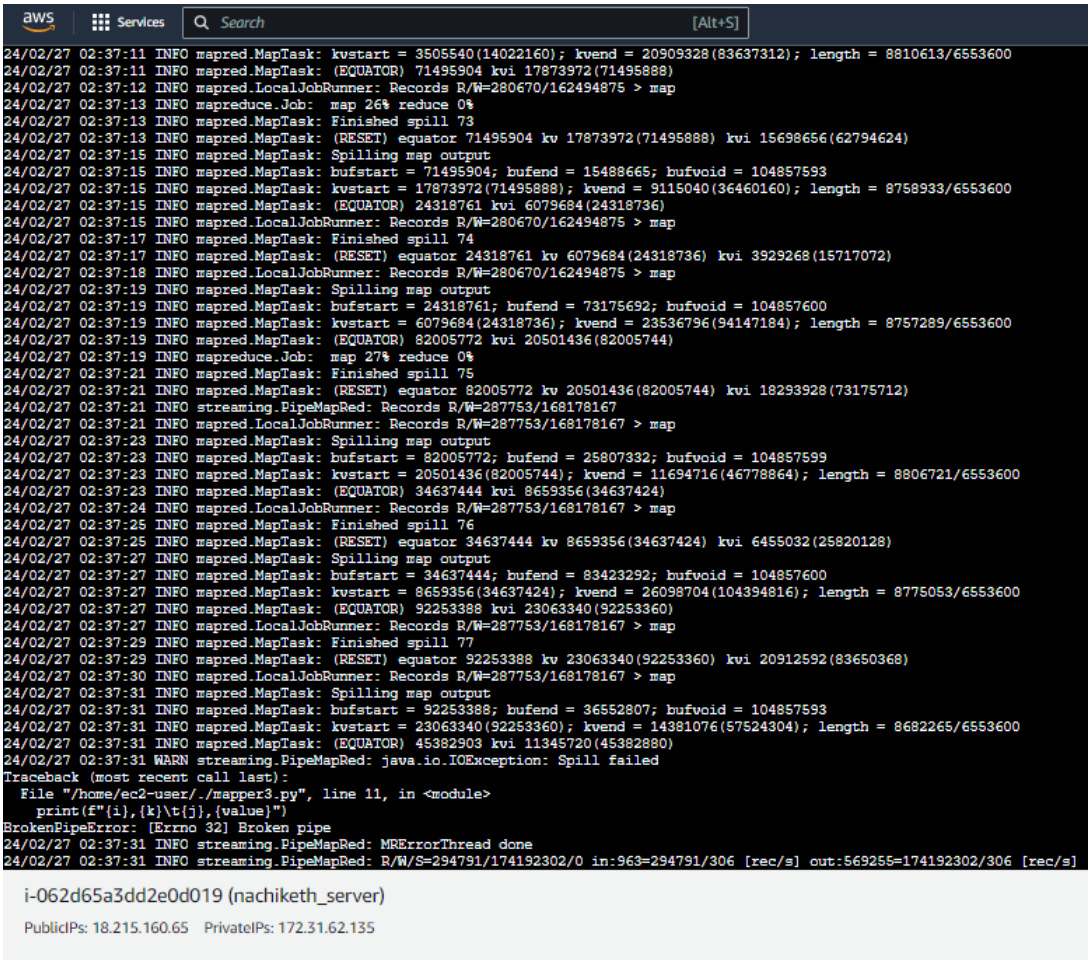


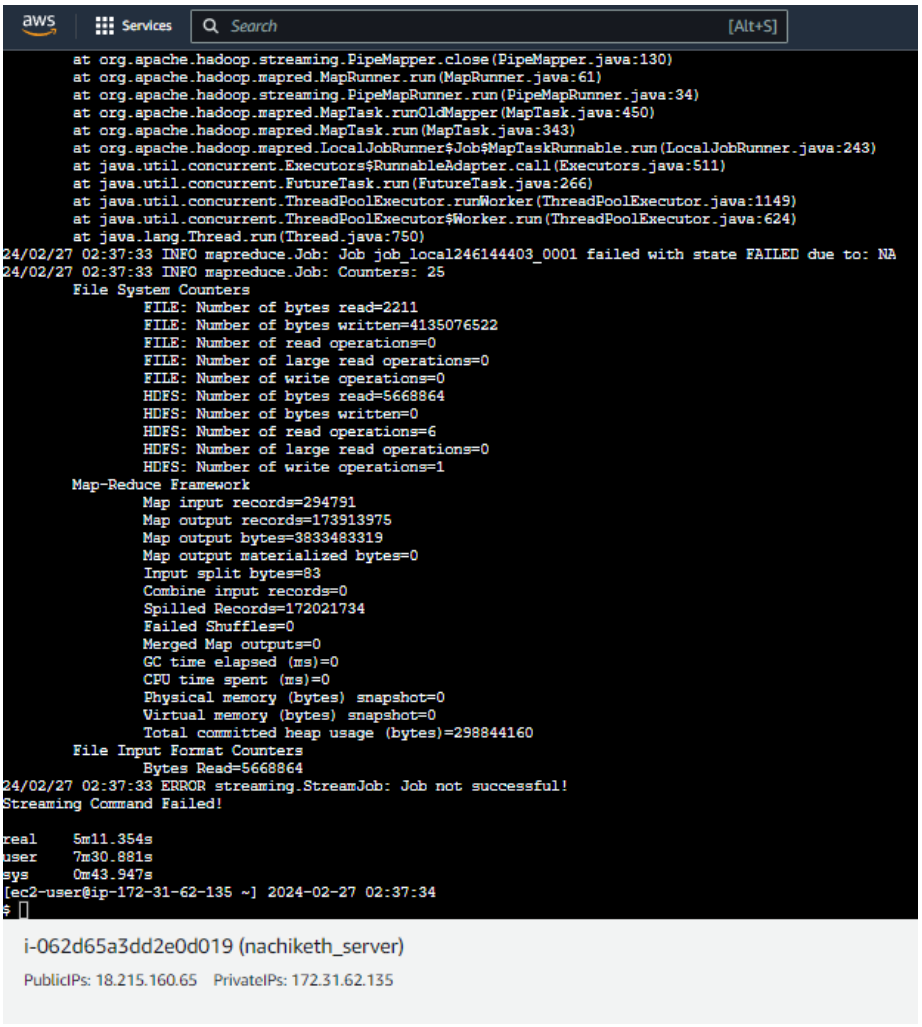
MATRIX SIZE	SINGLE NODE	MULTI NODE
100x100	real 0m12.232s user 0m15.600s sys 0m1.076s	real 0m25.075s user 0m3.709s sys 0m0.351s
600x600 (did not run completely)	real 5m11.354s user 7m30.881s sys 0m43.947s	real 8m12.192s user 0m5.199s sys 0m0.501s

Based on dataset size, the runtime comparisons for matrix multiplication show clear disparities between single-node and multi-node systems. The single-node solution shows shorter total processing times for the smaller 100x100 matrix, suggesting that the expense associated with dividing jobs across several nodes outweighs the benefits of parallelization for smaller datasets. On the other hand, when the size of the matrix reaches 600x600, the computational load on a single node causes a significant increase in runtime; in contrast, multi-node configurations show reduced user and system times but greater actual time, highlighting the efficiency gains from distributed processing. Nevertheless, mapper failures at about 28% completion for both configurations point to possible problems with resource allocation, requiring memory management adjustments.

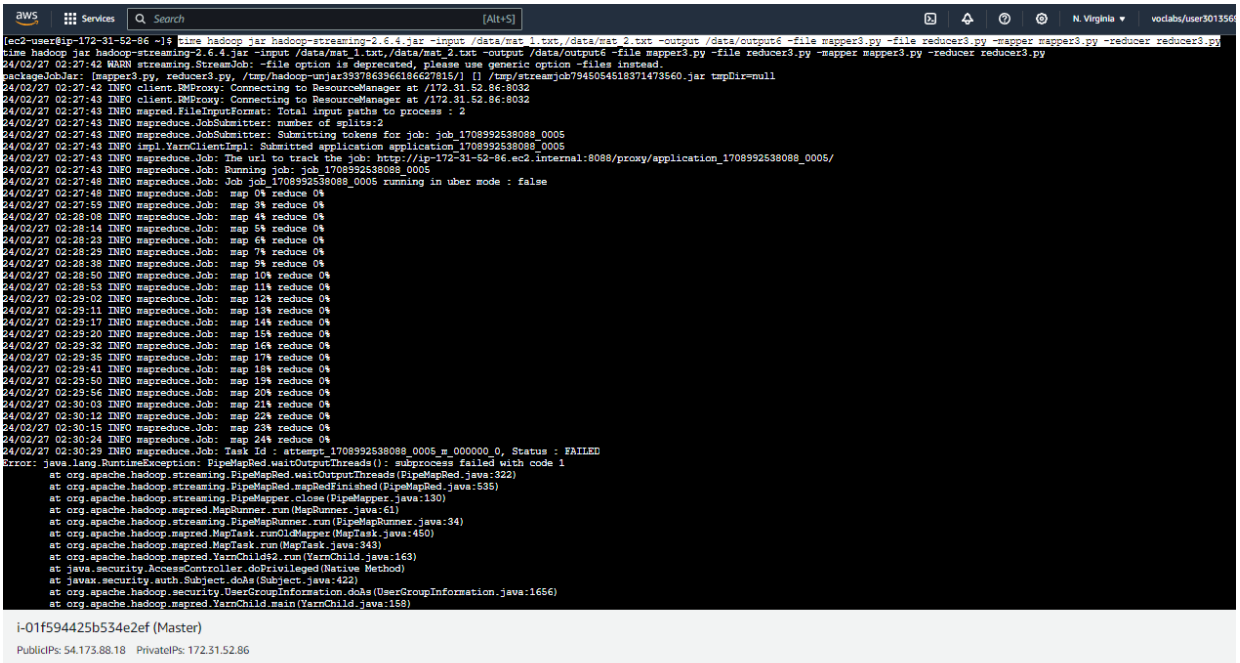
600x600 single node:

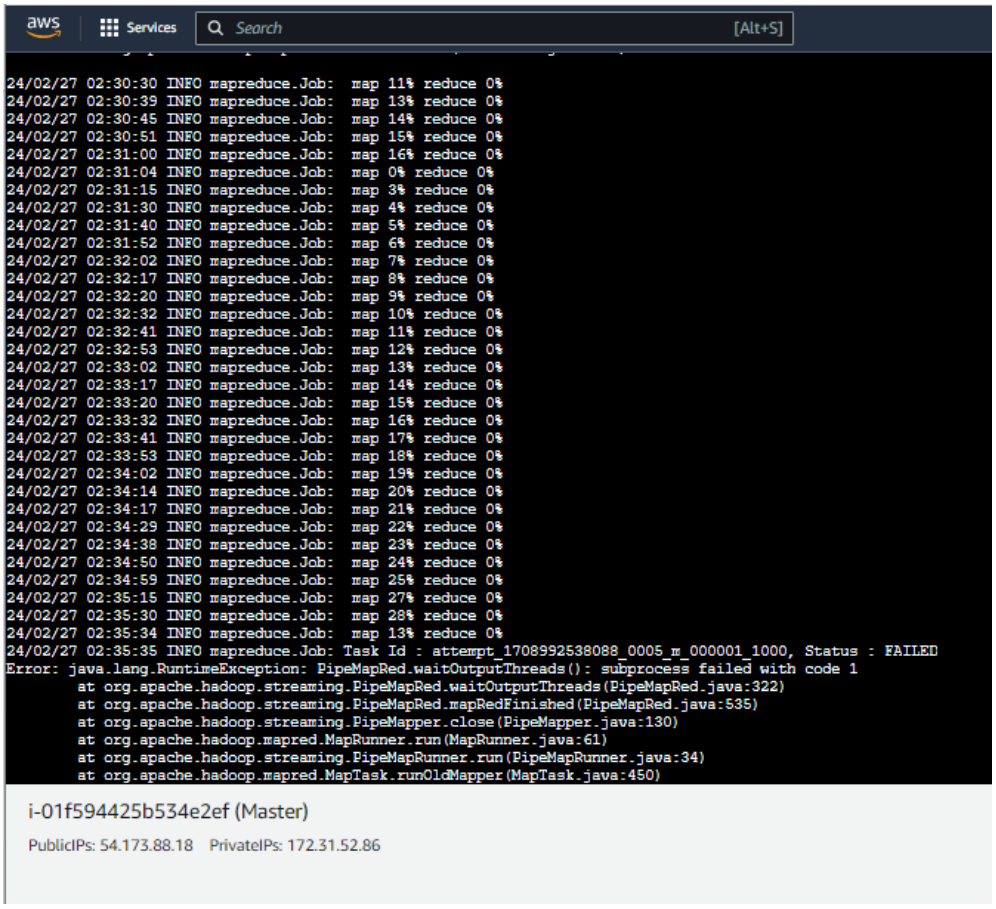


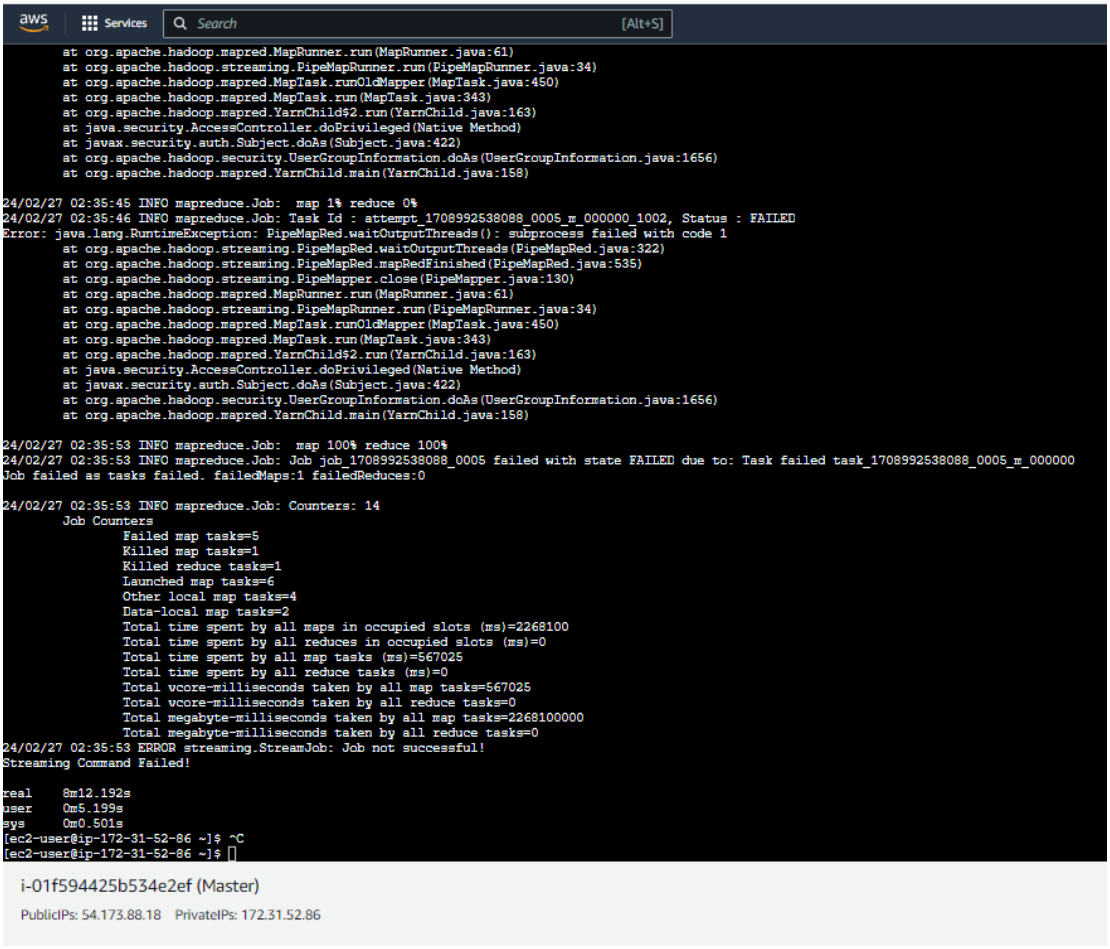




600x600 multi node:




aws
Services
Search
[Alt+S]
24/02/27 02:30:30 INFO mapreduce.Job: map 11% reduce 0%
24/02/27 02:30:39 INFO mapreduce.Job: map 13% reduce 0%
24/02/27 02:30:45 INFO mapreduce.Job: map 14% reduce 0%
24/02/27 02:30:51 INFO mapreduce.Job: map 15% reduce 0%
24/02/27 02:31:00 INFO mapreduce.Job: map 16% reduce 0%
24/02/27 02:31:04 INFO mapreduce.Job: map 0% reduce 0%
24/02/27 02:31:15 INFO mapreduce.Job: map 3% reduce 0%
24/02/27 02:31:30 INFO mapreduce.Job: map 4% reduce 0%
24/02/27 02:31:40 INFO mapreduce.Job: map 5% reduce 0%
24/02/27 02:31:52 INFO mapreduce.Job: map 6% reduce 0%
24/02/27 02:32:02 INFO mapreduce.Job: map 7% reduce 0%
24/02/27 02:32:17 INFO mapreduce.Job: map 8% reduce 0%
24/02/27 02:32:20 INFO mapreduce.Job: map 9% reduce 0%
24/02/27 02:32:32 INFO mapreduce.Job: map 10% reduce 0%
24/02/27 02:32:41 INFO mapreduce.Job: map 11% reduce 0%
24/02/27 02:32:53 INFO mapreduce.Job: map 12% reduce 0%
24/02/27 02:33:02 INFO mapreduce.Job: map 13% reduce 0%
24/02/27 02:33:17 INFO mapreduce.Job: map 14% reduce 0%
24/02/27 02:33:20 INFO mapreduce.Job: map 15% reduce 0%
24/02/27 02:33:32 INFO mapreduce.Job: map 16% reduce 0%
24/02/27 02:33:41 INFO mapreduce.Job: map 17% reduce 0%
24/02/27 02:33:53 INFO mapreduce.Job: map 18% reduce 0%
24/02/27 02:34:02 INFO mapreduce.Job: map 19% reduce 0%
24/02/27 02:34:14 INFO mapreduce.Job: map 20% reduce 0%
24/02/27 02:34:17 INFO mapreduce.Job: map 21% reduce 0%
24/02/27 02:34:29 INFO mapreduce.Job: map 22% reduce 0%
24/02/27 02:34:38 INFO mapreduce.Job: map 23% reduce 0%
24/02/27 02:34:50 INFO mapreduce.Job: map 24% reduce 0%
24/02/27 02:34:59 INFO mapreduce.Job: map 25% reduce 0%
24/02/27 02:35:15 INFO mapreduce.Job: map 27% reduce 0%
24/02/27 02:35:30 INFO mapreduce.Job: map 28% reduce 0%
24/02/27 02:35:34 INFO mapreduce.Job: map 13% reduce 0%
24/02/27 02:35:35 INFO mapreduce.Job: Task Id : attempt_1708992538088_0005_m_000001_1000, Status : FAILED
Error: java.lang.RuntimeException: PipeMapRed.waitOutputThreads(): subprocess failed with code 1
at org.apache.hadoop.streaming.PipeMapRed.waitOutputThreads(PipeMapRed.java:322)
at org.apache.hadoop.streaming.PipeMapRed.mapRedFinished(PipeMapRed.java:535)
at org.apache.hadoop.streaming.PipeMapper.close(PipeMapper.java:130)
at org.apache.hadoop.mapred.MapRunner.run(MapRunner.java:61)
at org.apache.hadoop.streaming.PipeMapRunner.run(PipeMapRunner.java:34)
at org.apache.hadoop.mapred.MapTask.runOldMapper(MapTask.java:450)
i-01f594425b534e2ef (Master)
PublicIPs: 54.173.88.18 PrivateIPs: 172.31.52.86


aws
Services
Search
[Alt+S]
at org.apache.hadoop.mapred.MapRunner.run(MapRunner.java:61)
at org.apache.hadoop.streaming.PipeMapRunner.run(PipeMapRunner.java:34)
at org.apache.hadoop.mapred.MapTask.runOldMapper(MapTask.java:450)
at org.apache.hadoop.mapred.MapTask.run(MapTask.java:343)
at org.apache.hadoop.mapred.YarnChild$2.run(YarnChild.java:163)
at java.security.AccessController.doPrivileged(Native Method)
at javax.security.auth.Subject.doAs(Subject.java:422)
at org.apache.hadoop.security.UserGroupInformation.doAs(UserGroupInformation.java:1656)
at org.apache.hadoop.mapred.YarnChild.main(YarnChild.java:158)
24/02/27 02:35:45 INFO mapreduce.Job: map 1% reduce 0%
24/02/27 02:35:46 INFO mapreduce.Job: Task Id : attempt_1708992538088_0005_m_000000_1002, Status : FAILED
Error: java.lang.RuntimeException: PipeMapRed.waitOutputThreads(): subprocess failed with code 1
at org.apache.hadoop.streaming.PipeMapRed.waitOutputThreads(PipeMapRed.java:322)
at org.apache.hadoop.streaming.PipeMapRed.mapRedFinished(PipeMapRed.java:535)
at org.apache.hadoop.streaming.PipeMapper.close(PipeMapper.java:130)
at org.apache.hadoop.mapred.MapRunner.run(MapRunner.java:61)
at org.apache.hadoop.streaming.PipeMapRunner.run(PipeMapRunner.java:34)
at org.apache.hadoop.mapred.MapTask.runOldMapper(MapTask.java:450)
at org.apache.hadoop.mapred.MapTask.run(MapTask.java:343)
at org.apache.hadoop.mapred.YarnChild$2.run(YarnChild.java:163)
at java.security.AccessController.doPrivileged(Native Method)
at javax.security.auth.Subject.doAs(Subject.java:422)
at org.apache.hadoop.security.UserGroupInformation.doAs(UserGroupInformation.java:1656)
at org.apache.hadoop.mapred.YarnChild.main(YarnChild.java:158)
24/02/27 02:35:53 INFO mapreduce.Job: map 100% reduce 100%
24/02/27 02:35:53 INFO mapreduce.Job: Job job_1708992538088_0005 failed with state FAILED due to: Task failed task_1708992538088_0005_m_000000
Job failed as tasks failed. failedMaps:1 failedReduces:0
24/02/27 02:35:53 INFO mapreduce.Job: Counters: 14
Job Counters
Failed map tasks=5
Killed map tasks=1
Killed reduce tasks=1
Launched map tasks=6
Other local map tasks=4
Data-local map tasks=2
Total time spent by all maps in occupied slots (ms)=2268100
Total time spent by all reduces in occupied slots (ms)=0
Total time spent by all map tasks (ms)=567025
Total time spent by all reduce tasks (ms)=0
Total vcore-milliseconds taken by all map tasks=567025
Total vcore-milliseconds taken by all reduce tasks=0
Total megabyte-milliseconds taken by all map tasks=2268100000
Total megabyte-milliseconds taken by all reduce tasks=0
24/02/27 02:35:53 ERROR streaming.StreamJob: Job not successful!
Streaming Command Failed!
real 8m12.192s
user 0m5.199s
sys 0m0.501s
[ec2-user@ip-172-31-52-86 ~]$ ^C
[ec2-user@ip-172-31-52-86 ~]$
i-01f594425b534e2ef (Master)
PublicIPs: 54.173.88.18 PrivateIPs: 172.31.52.86

MAPPER:

```
#!/usr/bin/python3
import sys
import os

input_file = os.environ['map_input_file']
if input_file.endswith("matrix_1.txt"):
    for line in sys.stdin:
        i, j, value = line.strip().split(',')
        for k in range(1, 101):  # starting range as 1.
            print(f"{i},{k}\t{j},{value}")
elif input_file.endswith("matrix_2.txt"):
    for line in sys.stdin:
        j, k, value = line.strip().split(',')
        for i in range(0, 100):  # Start from 0 to include the first row
            print(f"{i+1},{k}\t{j},{value}")  # Increment i by 1 to start from 1
```

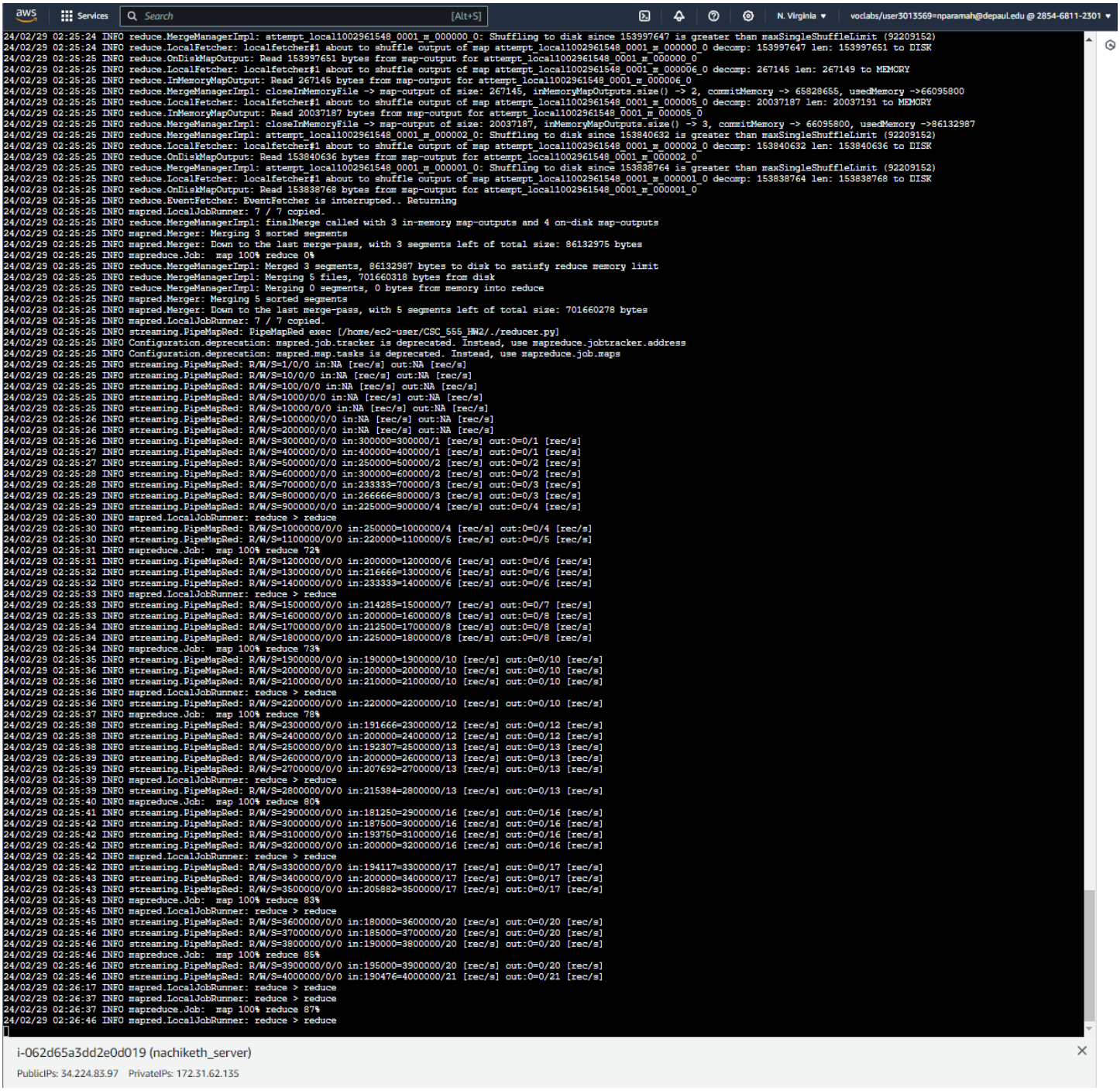
REDUCER:

```
#!/usr/bin/python3
import sys

curr_key = None
curr_values = []

# Processing input from mapper
for line in sys.stdin:
    key, value = line.strip().split('\t')
    pair, val = value.split(',')
    if key != curr_key:
        if curr_key is not None:
            result = sum(v1 * v2 for v1, v2 in curr_values)
            if result != 0:
                print(f"{curr_key}\t{result}")
        curr_key = key
        curr_values = []
    curr_values.append((float(val), float(val)))
#for last row values
if curr_key is not None:
    result = sum(v1 * v2 for v1, v2 in curr_values)
    if result != 0:
        print(f"{curr_key}\t{result}")
```

QUESTION 4:

There seems to be memory allocation error I tried different methods but ended up getting blank output (for one step), also cause the termination of master node a couple of times. The Reducer runs upto 85% before it fails.

SINGLE NODE:



MULTI NODE:

MAPPER 1:

```
#!/usr/bin/python3
import sys

def hash_function(key, num_buckets):
    return hash(key) % num_buckets

b = 4
c = 5
k = b * c

for line in sys.stdin:
    fields = line.strip().split('|')
```

```
    lo_orderdate = fields[5]  # Extracting lo_orderdate
    hashed_bucket = hash_function(lo_orderdate, k)
    print(f"{hashed_bucket}\tlineorder\t{'|'.join(fields)}")
```

MAPPER 2:

```
#!/usr/bin/python3
import sys

def hash_function(key, num_buckets):
    return hash(key) % num_buckets

b = 4
c = 5
k = b * c

for line in sys.stdin:
    fields = line.strip().split('|')
    d_datekey = fields[0]  # Extracting d_datekey
    hashed_bucket = hash_function(d_datekey, k)
    print(f"{hashed_bucket}\tdwdate\t{'|'.join(fields)}")
```

MAPPER 3:

```
#!/usr/bin/python3
import sys

def hash_function(key, num_buckets):
    return hash(key) % num_buckets

b = 4
c = 5
k = b * c

for line in sys.stdin:
    fields = line.strip().split('|')
    p_partkey = fields[0]  # Extracting p_partkey
    hashed_bucket = hash_function(p_partkey, k)
    print(f"{hashed_bucket}\tpart\t{'|'.join(fields)}")
```

REDUCER:

```
#!/usr/bin/python3
import sys

def hash_function(key, num_buckets):
    return hash(key) % num_buckets

b = 4
c = 5
k = b * c
```

```
lineorder_data = {}
dwdate_data = {}
part_data = {}
for line in sys.stdin:
   hashed_bucket, table_name, record = line.strip().split('\t')
   if table_name == 'lineorder':
      lineorder_data.setdefault(hashed_bucket, []).append(record.split('|'))
   elif table_name == 'dwdate':
      dwdate_data.setdefault(hashed_bucket, []).append(record.split('|'))
   elif table_name == 'part':
      part_data.setdefault(hashed_bucket, []).append(record.split('|'))

for bucket in range(k):
   if bucket in lineorder_data and bucket in dwdate_data and bucket in part_data:
      for lineorder_record in lineorder_data[bucket]:
         for dwdate_record in dwdate_data[bucket]:
            for part_record in part_data[bucket]:
               if dwdate_record[12] == 'Fall' and part_record[4] == 'MFGR#2123' and lineorder_record[5] ==
dwdate_record[0] and lineorder_record[3] == part_record[0]:
                  print('\t'.join(lineorder_record + dwdate_record + part_record))
```