CS553 Cloud Computing - Assignment 6

Sagar Shekhargouda Patil(A20501427)
Isa Muradli (A20474520)

This assignment aims to run Linux Sort, Hadoop Sort and Spark Sort on multiple Chameleon nodes. We run these sorts on 1 small (4-cores, 16G RAM, 38G disk), 1 large (24-cores, 96G RAM, 228G disk) and 4 small instances. File sizes of 3G, 6G, 12G, 24G are sorted and the results are recorded and analyzed. The goal is to access the performance of Hadoop and Spark on a single node to that of running it on four nodes. Similarly, comparing Linux sort on instances with different configurations.

We created and deployed bare metal instances on Chameleon at CHI@IIT sites and created containers in the specified configuration. We set up NFS already mounted on all containers. ASCII input files are generated and validated using gensort and valsort respectively.
The Linux time command is used to calculate the execution time for each type, and pidstat is used to log information about CPU usage, memory usage, and I/O speed.

Experiment	Linux(sec)	Hadoop Sort(sec)	Spark Sort(sec)
1 small.instance, 3GB dataset	228.499	322.198	290.37
1 small.instance, 6GB dataset	424.977	810.912	751.62
1 small.instance, 12GB dataset	809.613	Was not able to run	Was not able to run
1 large.instance, 3GB dataset	49.344	205.112	195.25
1 large.instance, 6GB dataset	296.675	353.742	444.95
1 large.instance, 12GB dataset	631.08	875.643	800.17
1 large.instance, 24GB dataset	1130.894	2525.942	2100.56
4 small.instances, 3GB dataset	N/A	234.200	190.36
4 small.instances, 6GB dataset	N/A	335.145	356.45
4 small.instances, 12GB dataset	N/A	486.276	480.24
4 small.instances, 24GB dataset	N/A	1119.678	1040.32

Linux Sort:

3GB

```
ubuntu@datanode1:/exports/projects/64$ time LC_ALL=C sort  --parallel=16 3GB.txt -o out3GB.txt

real    0m49.344s
user    0m49.364s
sys     0m6.674s
ubuntu@datanode1:/exports/projects/64$ ./valsort out3GB.txt
Records: 30000000
Checksum: e4d987b89b2004
Duplicate keys: 0
SUCCESS - all records are in order
```

24GB

```
ubuntu@datanode1:/exports/projects/64$ time LC_ALL=C sort  --buffer-size=8G  --parallel=48 24GB.txt  -o out24GB.txt

real    18m23.894s
user    8m11.962s
sys     3m17.919s
ubuntu@datanode1:/exports/projects/64$ ./valsort out24GB.txt
Records: 240000000
Checksum: 7270827eb008677
Duplicate keys: 0
SUCCESS - all records are in order
ubuntu@datanode1:/exports/projects/64$
```

12GB

```
ubuntu@datanode1:/exports/projects/64$ time LC_ALL=C sort  --buffer-size=8G  --parallel=48 12GB.txt  -o out12GB.txt

real    10m31.089s
user    4m5.115s
sys     1m51.220s
ubuntu@datanode1:/exports/projects/64$ ./valsort out12GB.txt
Records: 120000000
Checksum: 3938190eff9ed88
Duplicate keys: 0
SUCCESS - all records are in order
```

6GB

```
ubuntu@datanode1:/exports/projects/64$ time LC_ALL=C sort  --buffer-size=6G  --parallel=48 6GB.txt  -o out6GB.txt

real    4m56.675s
user    1m57.171s
sys     0m41.948s
ubuntu@datanode1:/exports/projects/64$ ./valsort out6GB.txt
Records: 60000000
Checksum: 1c9baf9f5e81489
Duplicate keys: 0
SUCCESS - all records are in order
```

Small Instance
Linux Sort
12 GB

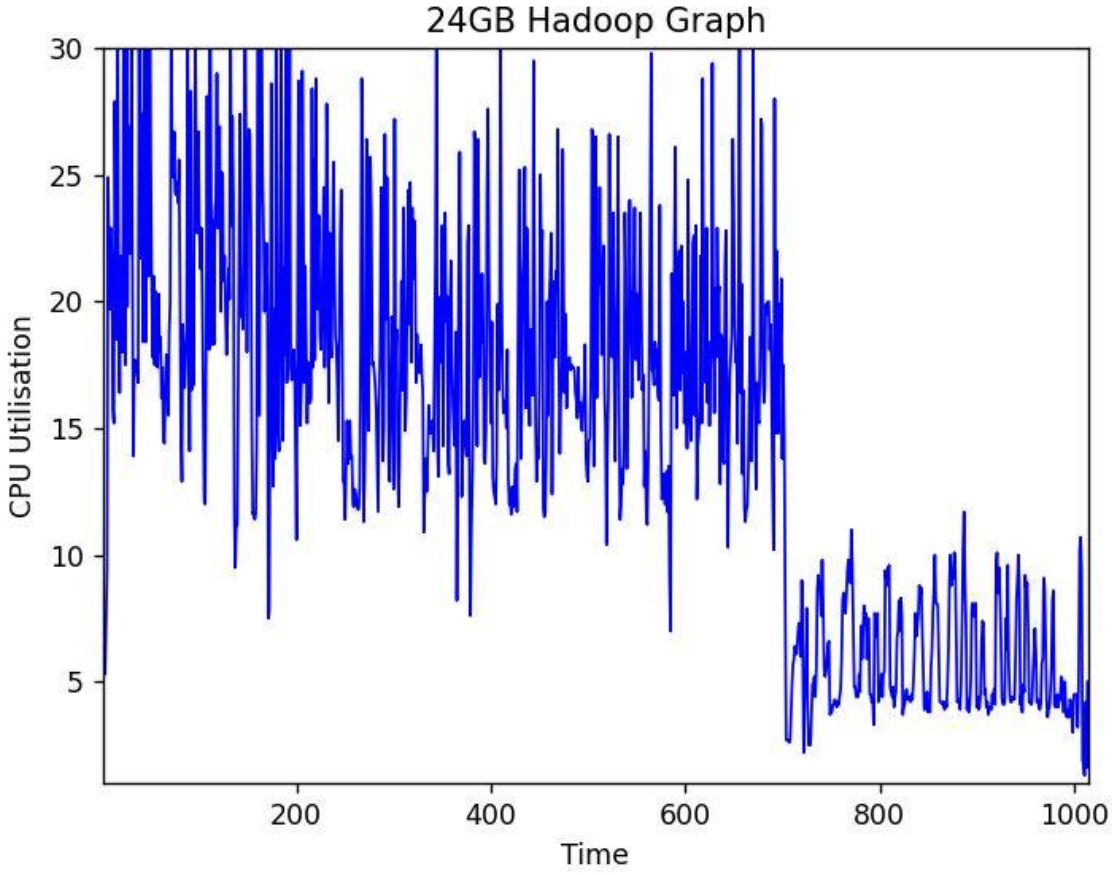
```
ubuntu@datanode1:/exports/projects/64$ time LC_ALL=C sort --buffer-size=6g --parallel=6 12GB_2.txt -o out12GB_2.txt

real    13m29.613s
user    3m6.428s
sys     2m24.845s
ubuntu@datanode1:/exports/projects/64$ ./valsort out12GB_2.txt
Records: 120000000
Checksum: 3938190eff9ed88
Duplicate keys: 0
SUCCESS - all records are in order
ubuntu@datanode1:/exports/projects/64$
```

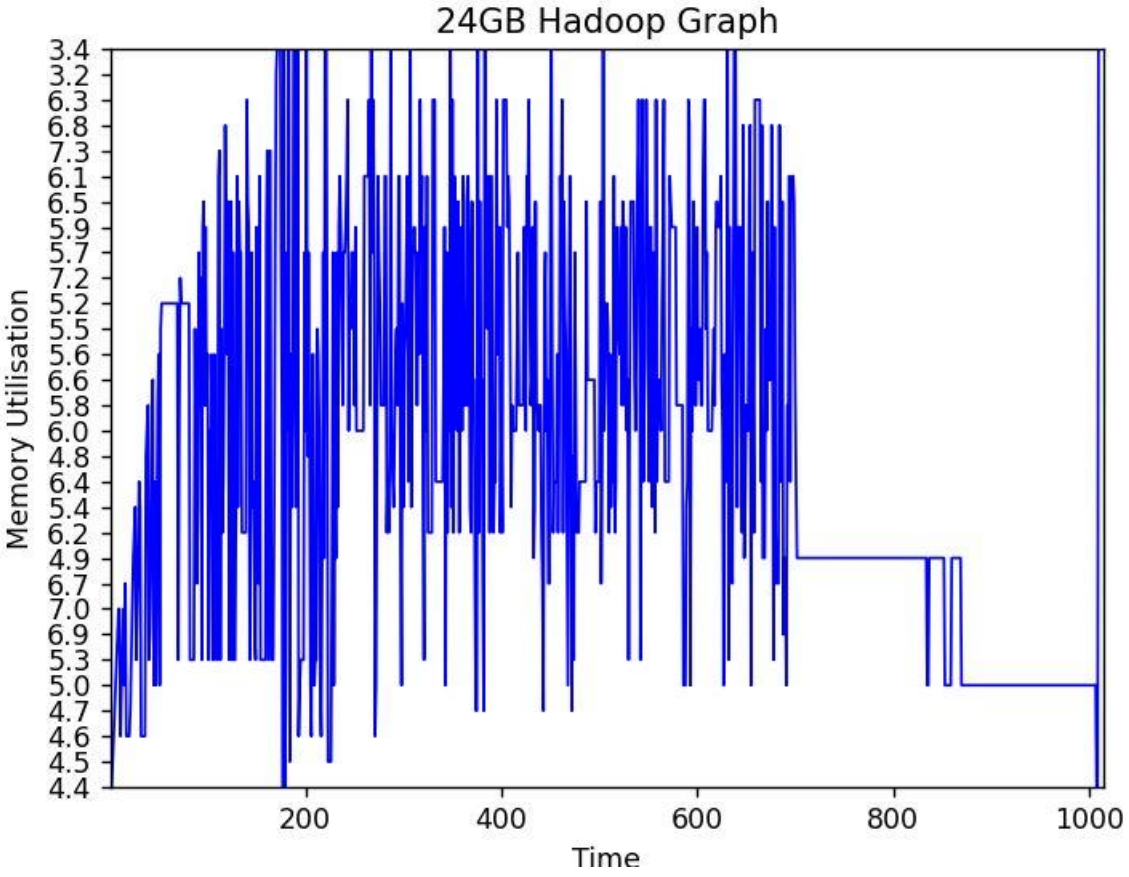
6GB

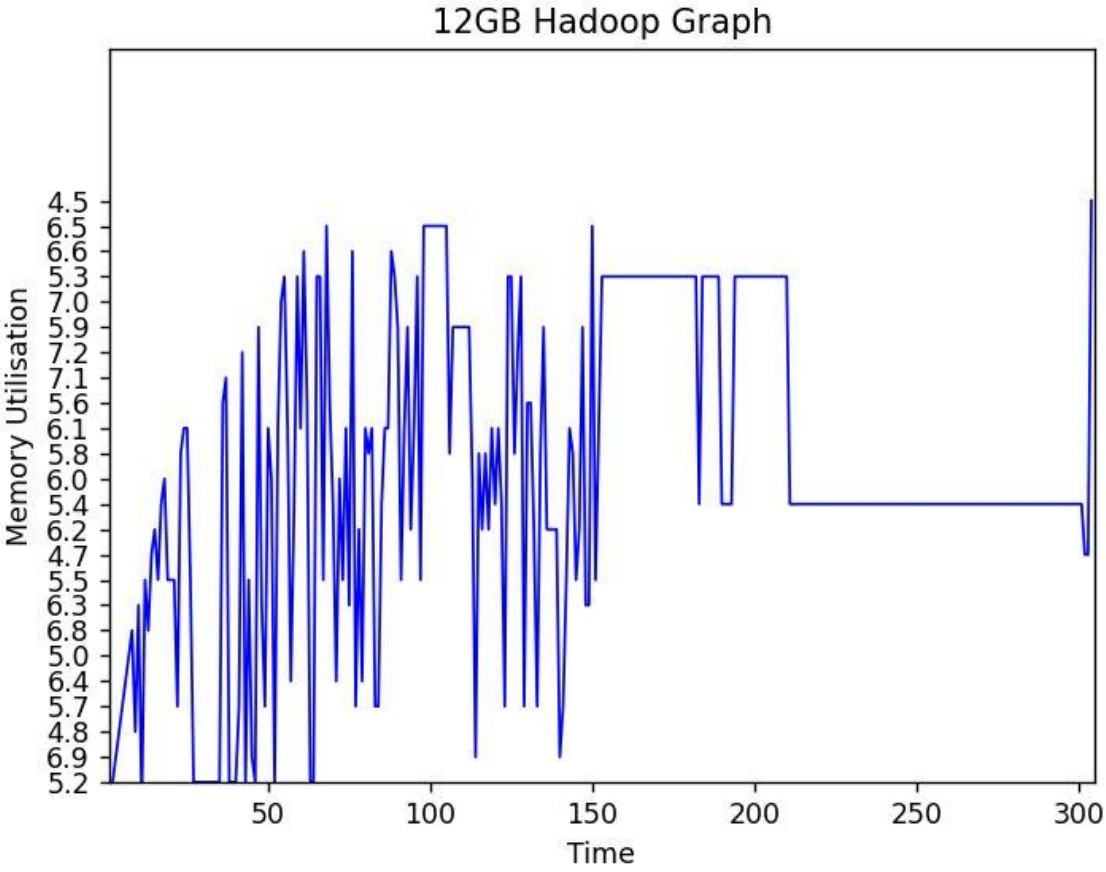
```
ubuntu@datanode1:/exports/projects/64$ time LC_ALL=C sort --buffer-size=6g --parallel=6 6GB_2.txt -o out6GB_2.txt

real    7m4.977s
user    1m31.285s
sys     1m8.728s
ubuntu@datanode1:/exports/projects/64$ ./valsort out6GB_2.txt
Records: 60000000
Checksum: 1c9baf9f5e81489
Duplicate keys: 0
SUCCESS - all records are in order
```

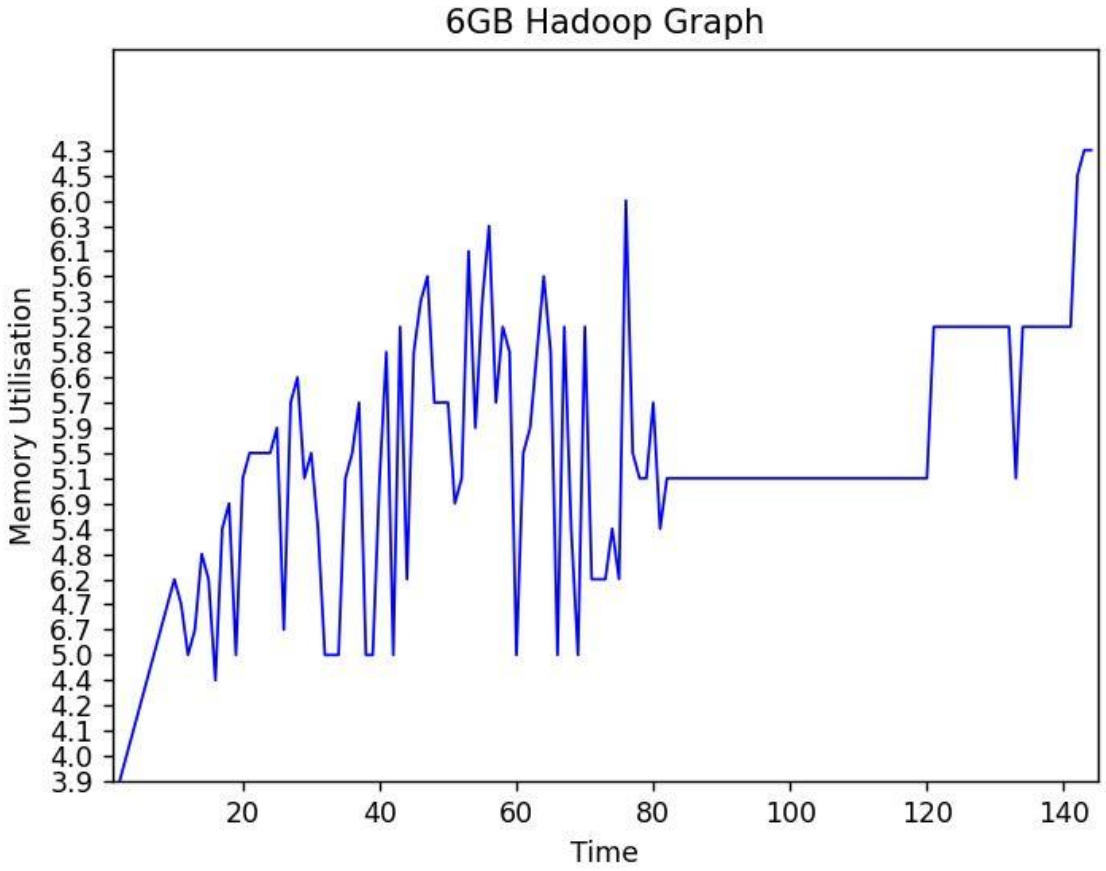
3GB

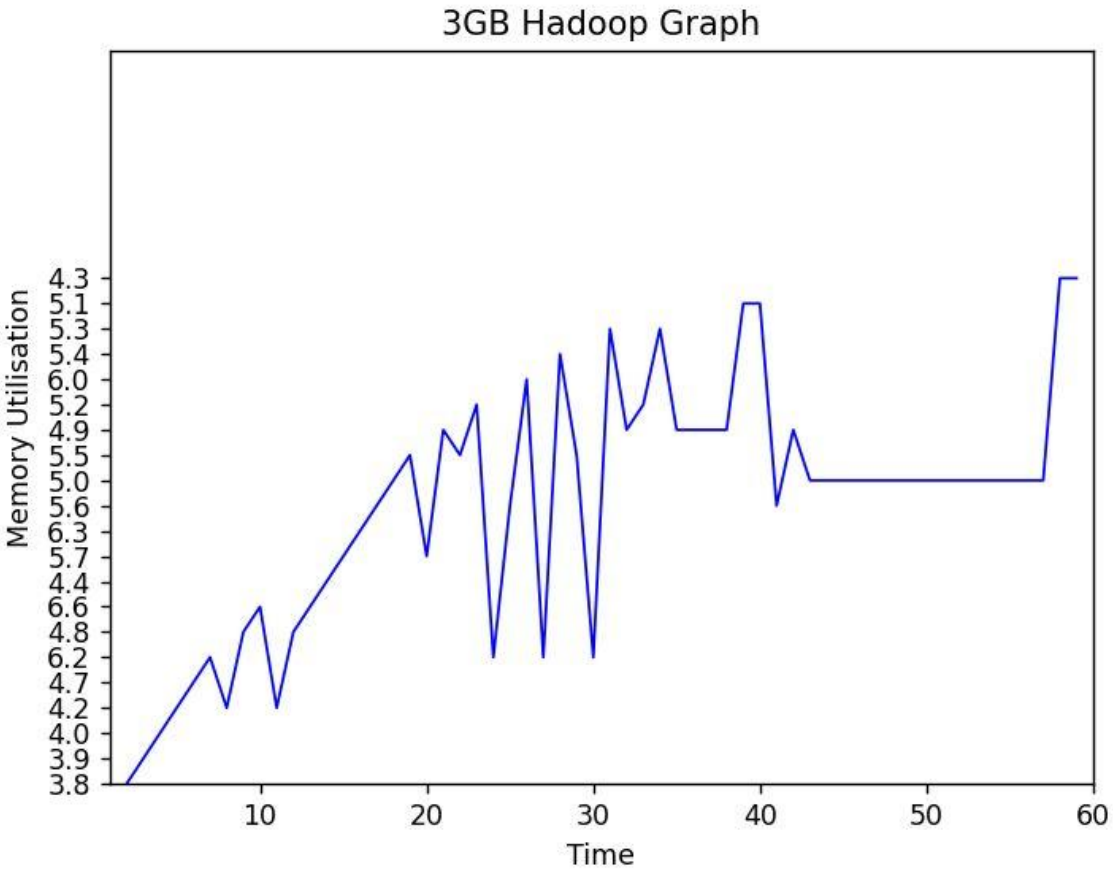
```
ubuntu@datanode1:/exports/projects/64$ time LC_ALL=C sort --buffer-size=6g --parallel=6 3GB_2.txt -o out3GB_2.txt

real    3m48.499s
user    0m44.850s
sys     0m35.771s
ubuntu@datanode1:/exports/projects/64$ ./valsort out3GB_2.txt
Records: 30000000
Checksum: e4d987b89b2004
Duplicate keys: 0
SUCCESS - all records are in order
ubuntu@datanode1:/exports/projects/64$
```


24GB Hadoop Graph
CPU Utilisation
Time
30
25
20
15
10
5
200
400
600
800
1000


24GB Hadoop Graph
Memory Utilisation
3.4
3.2
6.3
6.8
7.3
6.1
6.5
5.9
5.7
7.2
5.2
5.5
5.6
6.6
5.8
6.0
4.8
6.4
5.4
6.2
4.9
6.7
7.0
6.9
5.3
5.0
4.7
4.6
4.5
4.4
200
400
600
800
1000
Time


12GB Hadoop Graph
Memory Utilisation
4.5
6.5
6.6
5.3
7.0
5.9
7.2
7.1
5.6
6.1
5.8
6.0
5.4
6.2
4.7
5.5
6.3
6.8
5.0
6.4
5.7
4.8
6.9
5.2
50
100
150
200
250
300
Time


6GB Hadoop Graph
Memory Utilisation
4.3
4.5
6.0
6.3
6.1
5.6
5.3
5.2
5.8
6.6
5.7
5.9
5.5
5.1
6.9
5.4
4.8
6.2
4.7
6.7
5.0
4.4
4.2
4.1
4.0
3.9
20
40
60
80
100
120
140
Time



How many threads, mappers, reducers, you used in each experiment?
We used 6 threads for small instances and 48 threads for large instances for Linux sort, 1 mapper/reducer for Hadoop Sort.

How many times did you have to read and write the dataset for each experiment?

Linux sort used gensort to generate the dataset and output of the sorted dataset in the system. So there will be a total of two writes as we are performing all the experiments in-memory. For reading the dataset before sorting,for validation reading is also required. So a total of two reads of the dataset will be required for Linux sort.

Spark and Hadoop used Gensort to create the dataset, move the data into the Hadoop file system, export the sorted dataset, and move the sorted output to /export/project. Thus, a total of 4 writes are performed for Spark & Hadoop sort. To validate the sorted data and sort the dataset, we need to read in both hadoop and Spark sort, so 2 times in total, we need to read the dataset here.

What speedup and efficiency did you achieve?

What conclusions can you draw?

For the large instance, the average CPU usage is much higher for Linux Sort. Hadoop classification shows consistently high CPU usage. Hadoop and Spark types show high memory usage while for Linux it gradually increases. Using 4 nodes, Hadoop still performs better than Spark sort.

Which seems to be best at 1 node scale (1 large.instance)?

Linux sort is best for 1 large instance while Hadoop and Spark perform poorly .

Is there a difference between 1 small.instance and 1 large.instance?

Small.instance shows a longer sort time for Linux sort in most cases. This shows that having more cores increases the performance. Hadoop and Spark show almost the same performance for both cases. However, with more resources, Large.instance performs a bit better here.

How about 4 nodes (4 small.instance)?What speedup do you achieve with strong scaling between 1 to 4 nodes?

Strong scaling is defined as how the solution time varies with the number of processors for a fixed total problem size.

[1] Comparing a 12GB dataset on 1 largeinstance to a 12GB dataset on 4 small instances for Hadoop Sort, we achieve a speedup efficiency of 44%

Comparing a 12GB dataset on 1 large instance to a 12GB dataset on 4 small instances for Spark Sort, we achieve a speedup efficiency of 24.98%

What speedup do you achieve with weak scaling between 1 to 4 nodes?

Weak scaling is defined as how the solution time varies with the number of processors for a fixed problem size per processor.

Comparing a 6GB dataset on 1 small instance to a 6GB dataset on 4 small instances for Hadoop Sort, we achieve a speedup efficiency of 51%

Comparing a 6GB dataset on 1 small instance to a 6GB dataset on 4 small instances for Spark Sort, we achieve a speedup efficiency of 54%