SORT ON SINGLE SHARED MEMORY NODE

Abstract

Implement external merge sort using multi-threading and compare it's performance with the Linux sort

Mohit Aggarwal
Siddhant Jain

AIM:

To gain experience programming with external data sort and multi-threading for various file sizes with a constraint on the memory of 8 GB:

1. 1 GB
2. 4 GB
3. 16 GB
4. 64 GB

Code Information:

The project was developed in Java. Following functions or libraries were used to perform the required operations:

1. Java.io
2. Java.util

Design Decisions:

The internal memory sort was implemented as merge sort. Even though merge sort requires extra O(N) space, it always gives O(NlogN) time complexity. While quicksort doesn't need the extra memory, it can give O(N*N) time complexity in worst case and the implementation to make sure it only gives O(NlogN) in worst case requires implementation of further algorithms such as median of medians to get a good partition which increases the complexity of the code much further and was thus not implemented.

Thus, if the size of the file is less than half of the memory available, then the data will be sorted in the memory or it will be sorted using the external merge sort.

The system on which the experiment was done is an SSD and to make sure to get the maximum utilization from the disk, we used Java's Runnable class to use multi-threading while performing the division of the large files into multiple chunks.

The program requires user to provide the input file name, output file name and the number of threads. No. of threads if left for the person running to choose because various systems can have varying performance for the different number of threads and it's thus best if the user has the flexibility on his/her understanding.

Further, the caches was cleared to make sure the file is not in the buffer cache by using:

sudo sysctl -w vm.drop_caches=3

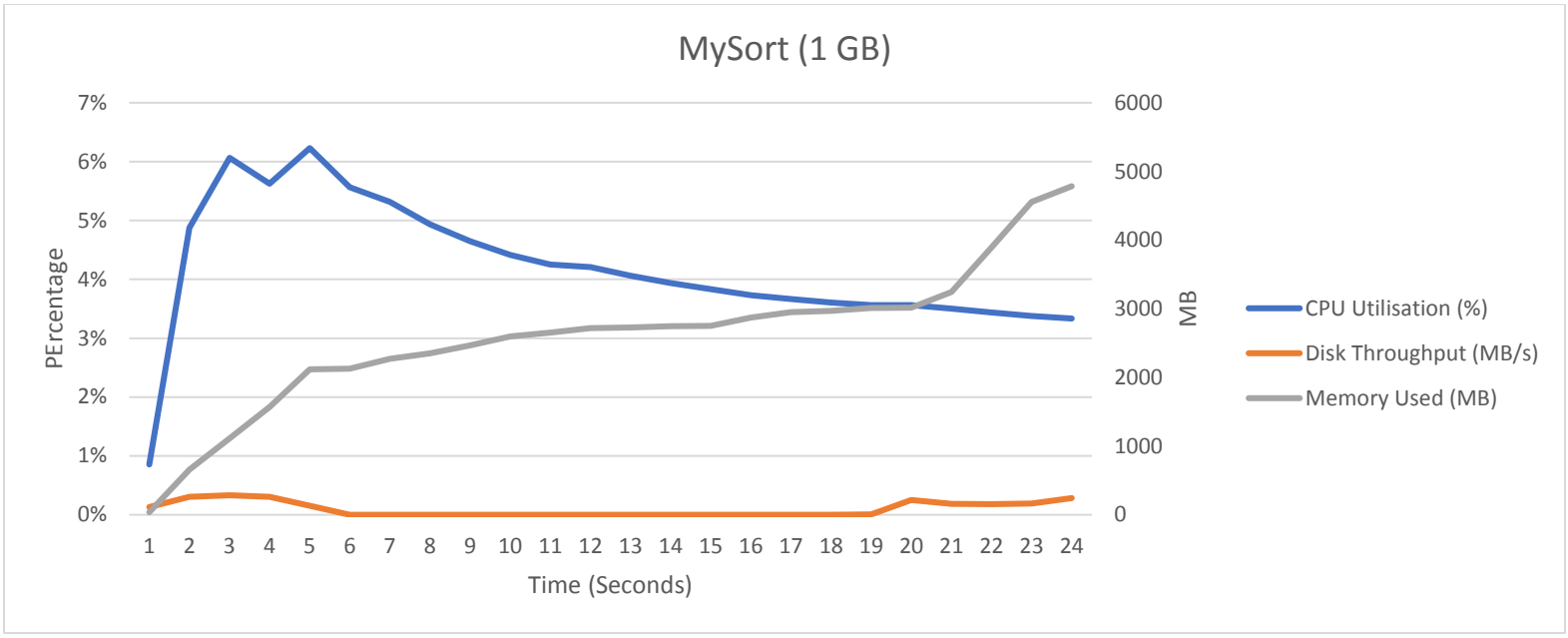
A tree map was used to store the input while merging the individual chunks. This is because we can store both the string as well the index to the file from where it is being taken plus tree map is a sorted structure which takes O(logN) for both insertion and removal and hence gives best performance for our requirements.

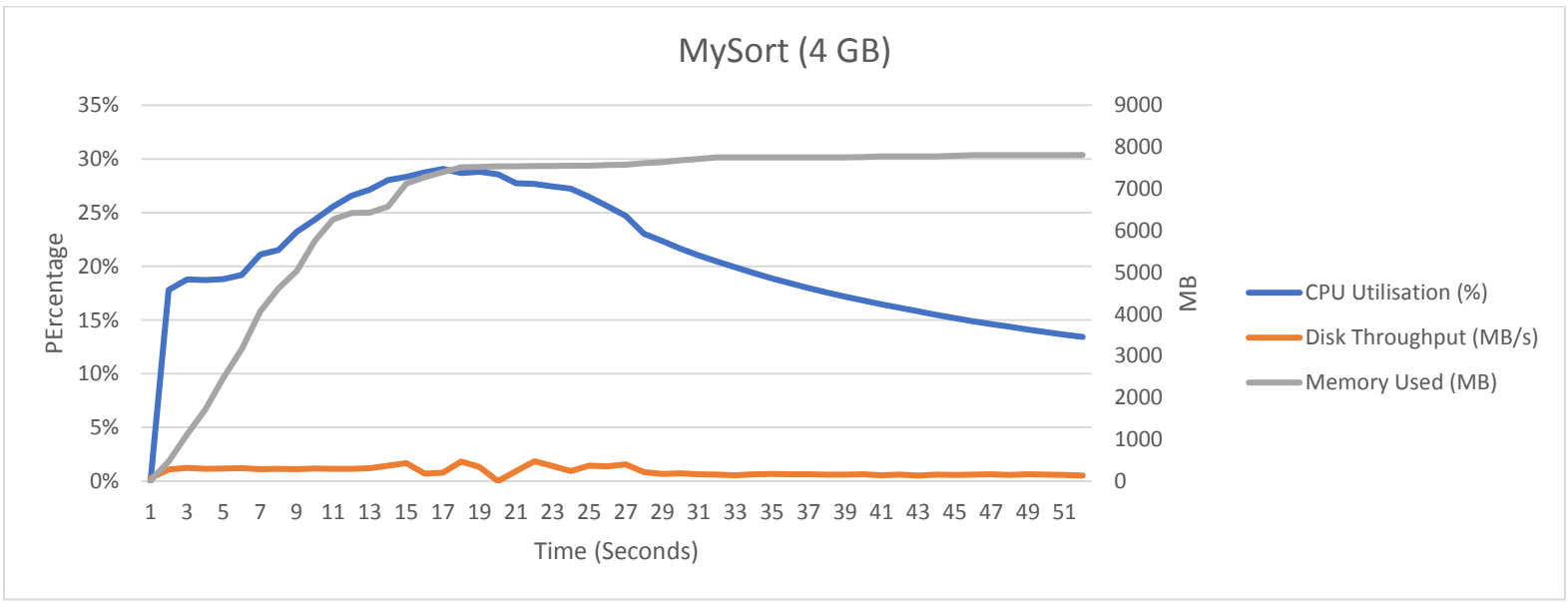
Results:

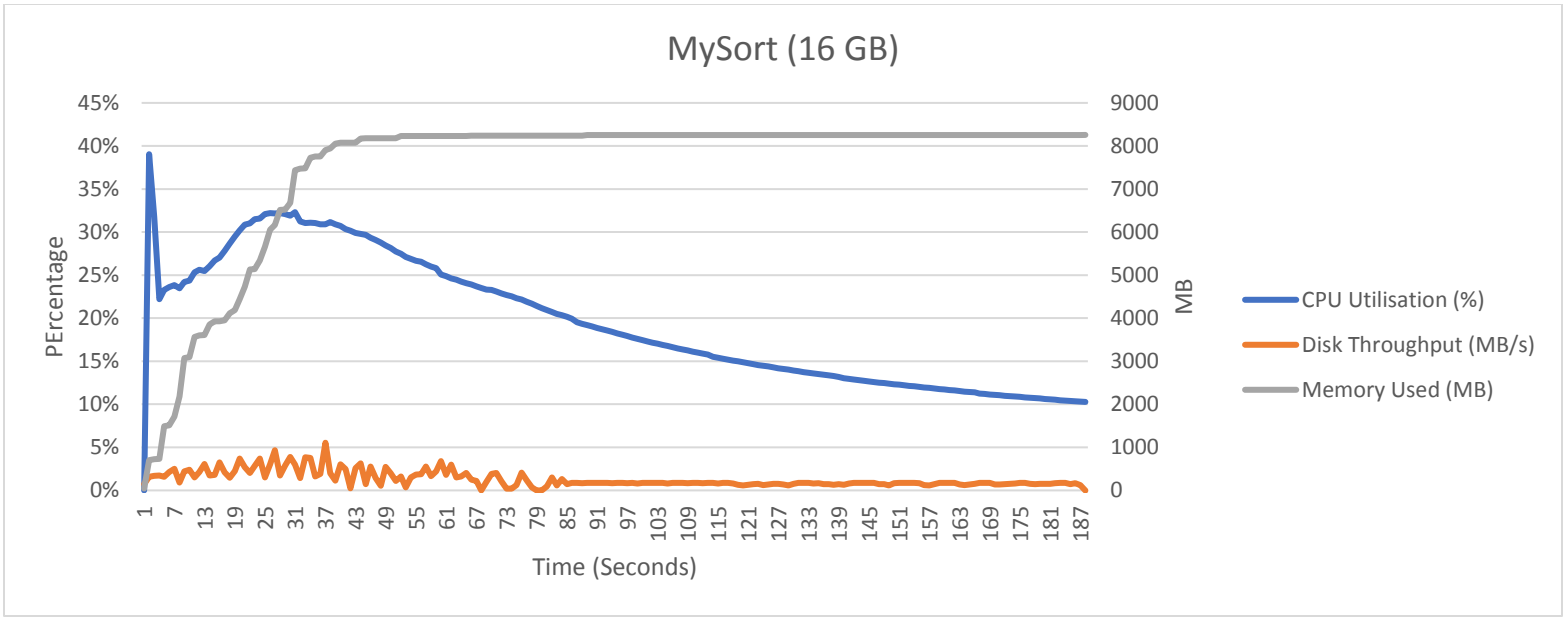
Experiment	Share Memory (1GB)	Linux Sort (1GB)	Share Memory (4GB)	Linux Sort (4GB)	Share Memory (16GB)	Linux Sort (16GB)	Share Memory (64GB)	Linux Sort (64GB)
No. of Threads	1	1	8	8	16	16	16	16
Sort Approach	In Memory	In Memory	External	External	External	External	External	External
Sort Algorithm	Merge Sort	Merge Sort	Merge Sort	Merge Sort	Merge Sort	Merge Sort	Merge Sort	Merge Sort
Data Read (GB)	1	1	8	8	32	32	64	64
Data Write(GB)	1	1	8	8	32	32	64	64
Sort Time(sec)	24	11	52	52	188	196	865	820
Overall I/O Throughput (MB/Sec)	83	186	233	236	258	251	226	301
Overall CPU Utilisation (%)	4%	5%	21%	8%	20%	8%	17%	8%
Average Memory Utilisation (MB)	2,592	1,074	6,639	4,729	7,499	6,155	7,364	7,498

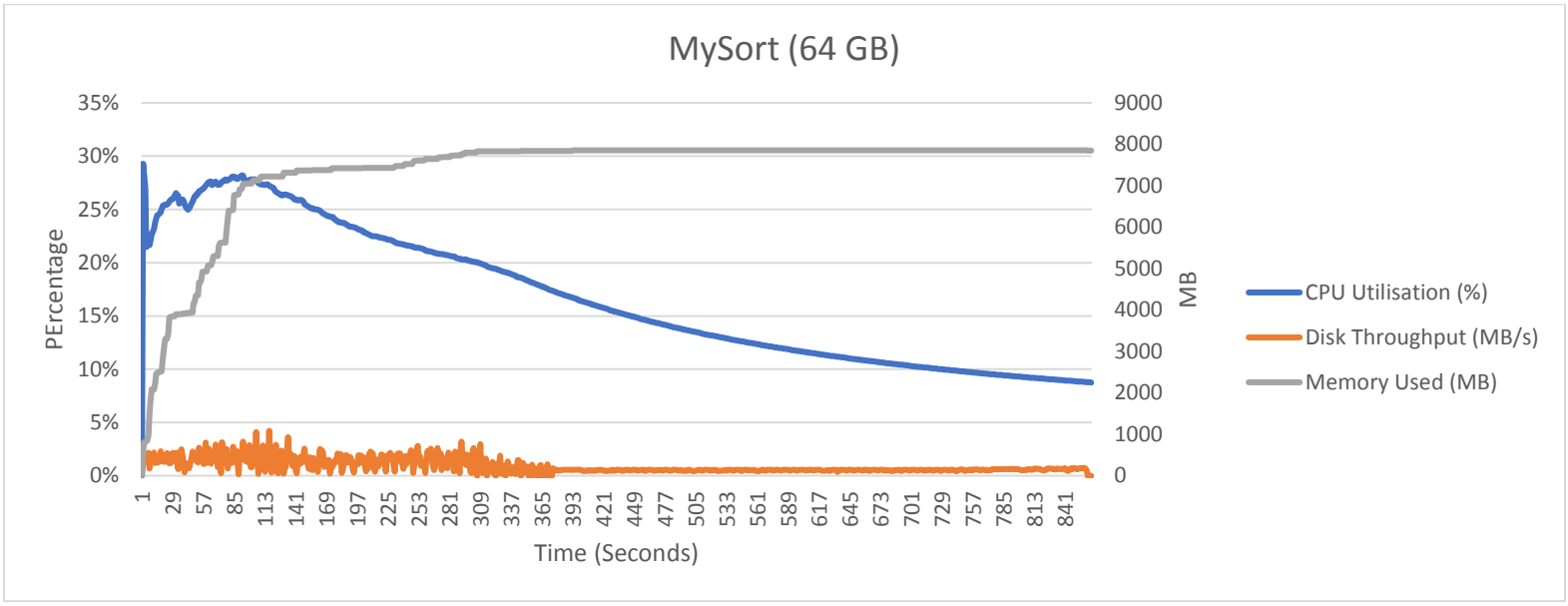
The linux sort is able to outperform MySort for 1 & 64 GB files while for 4 & 16 GB files that's not the case in terms of timing as well. Also, it seems that the Linux sort is able to better utilise the CPU and memory and gives and have lower overhead.

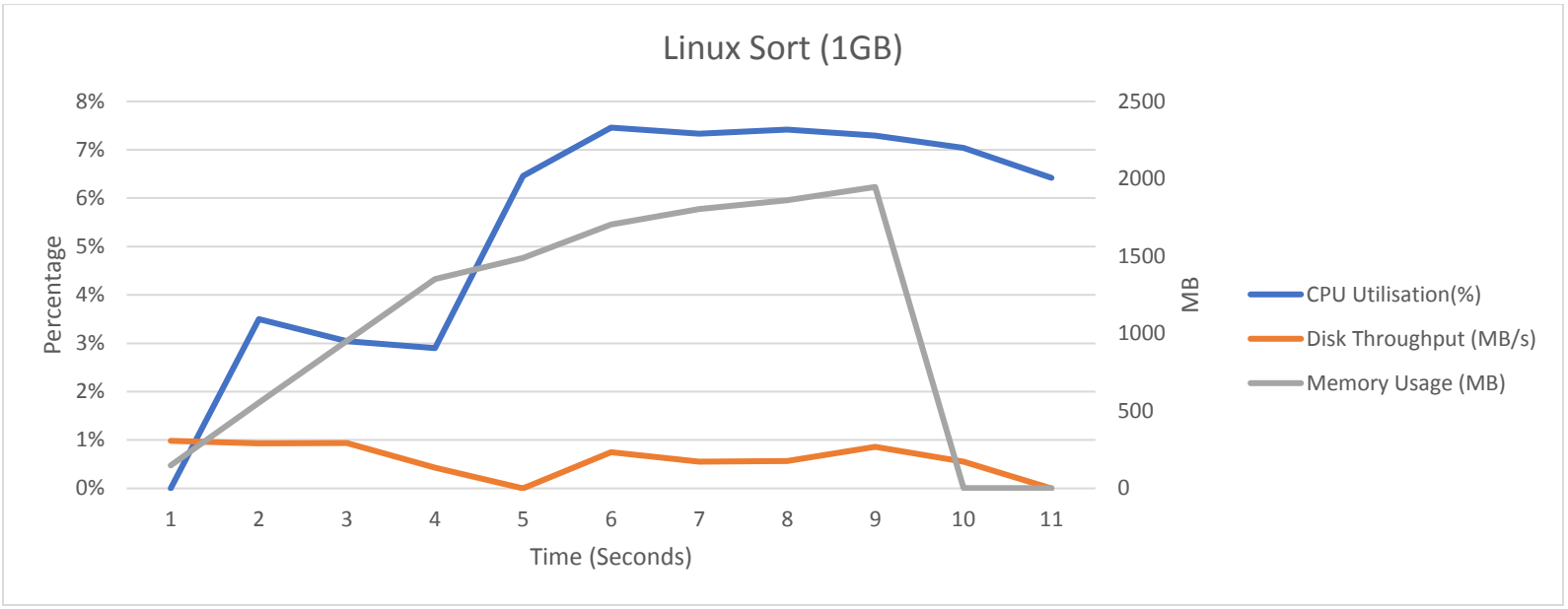
The system on which the experiment was done contained 48 cores while ps gives CPU utilisation as 100% for 1 full core utilisation, 200% for 2 full cores utilisation and so on. Thus to give a better picture on the CPU utilisation, the values were divided by 4800 to give utilisation as of the total cores available

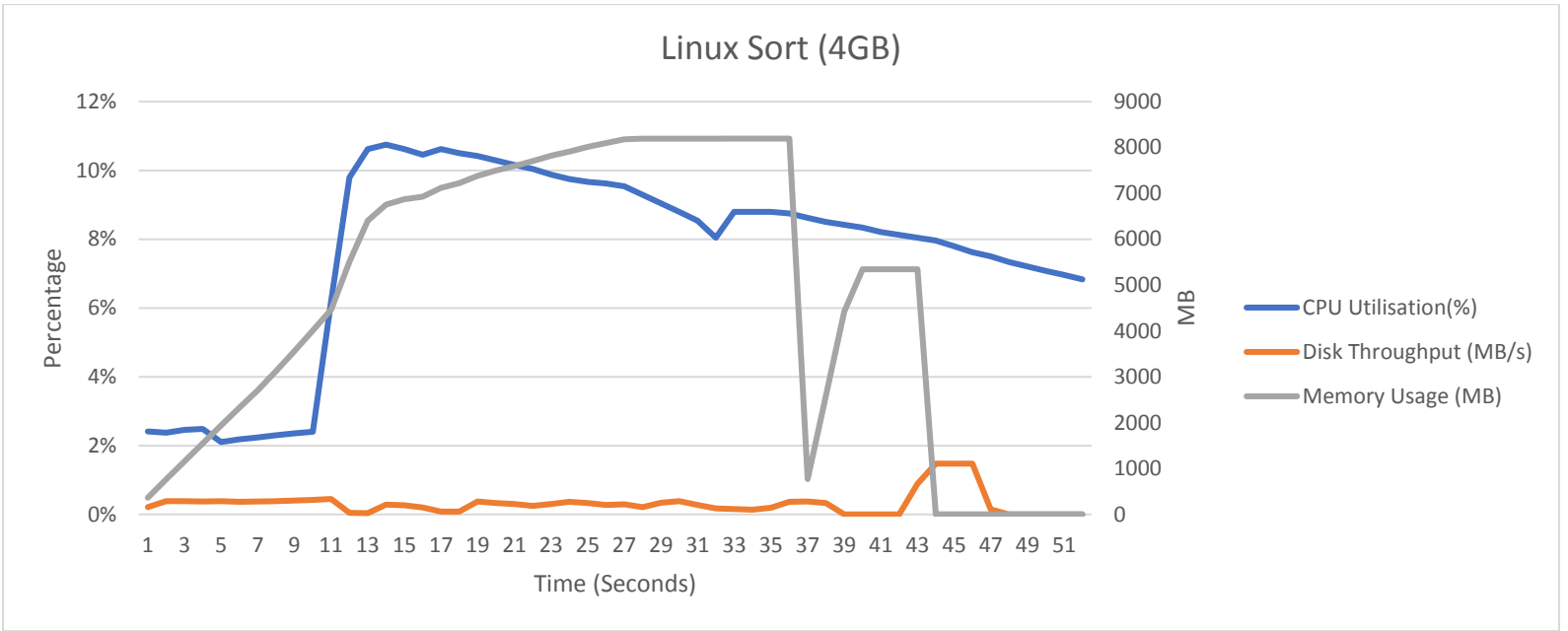

MySort (1 GB)
PErcentage
7%
6%
5%
4%
3%
2%
1%
0%
6000
5000
4000
3000
2000
1000
0
MB
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Time (Seconds)
CPU Utilisation (%)
Disk Throughput (MB/s)
Memory Used (MB)

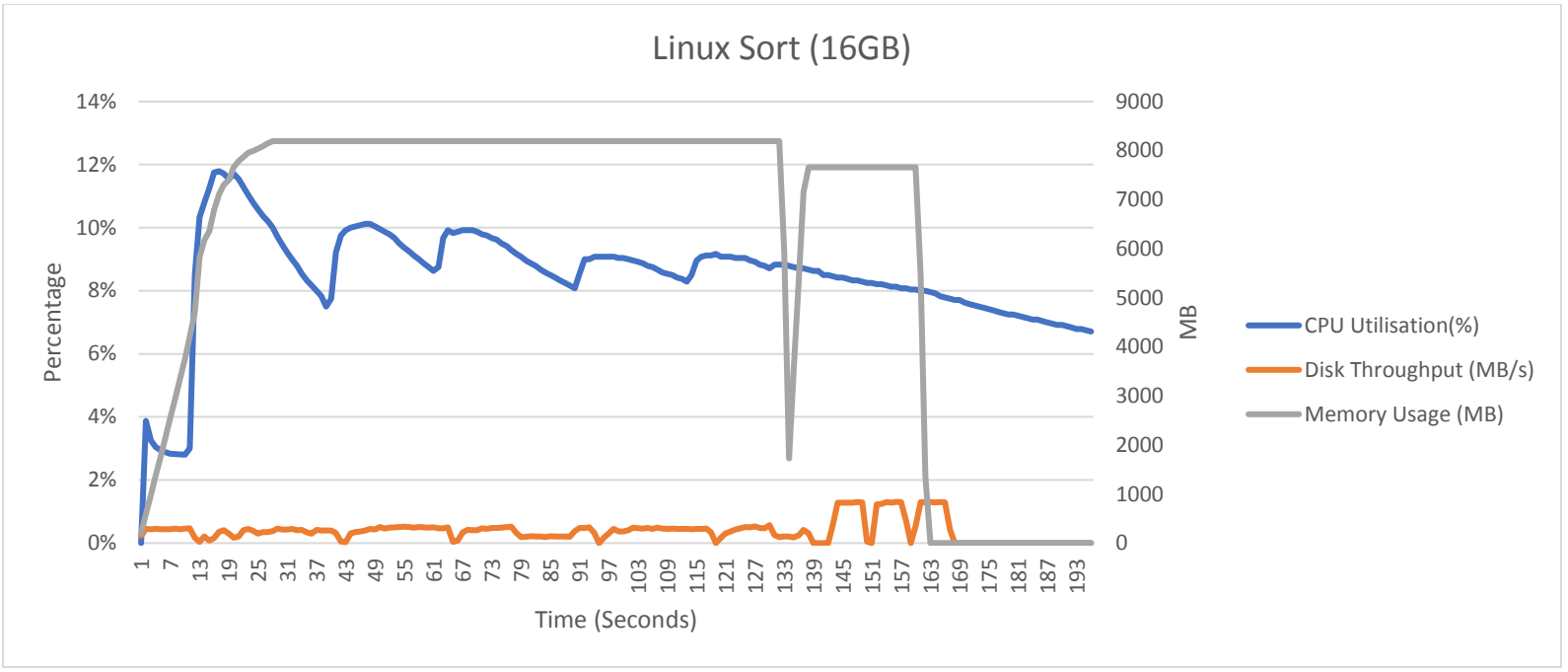

MySort (4 GB)
PErcentage
35%
30%
25%
20%
15%
10%
5%
0%
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
MB
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51
Time (Seconds)
CPU Utilisation (%)
Disk Throughput (MB/s)
Memory Used (MB)

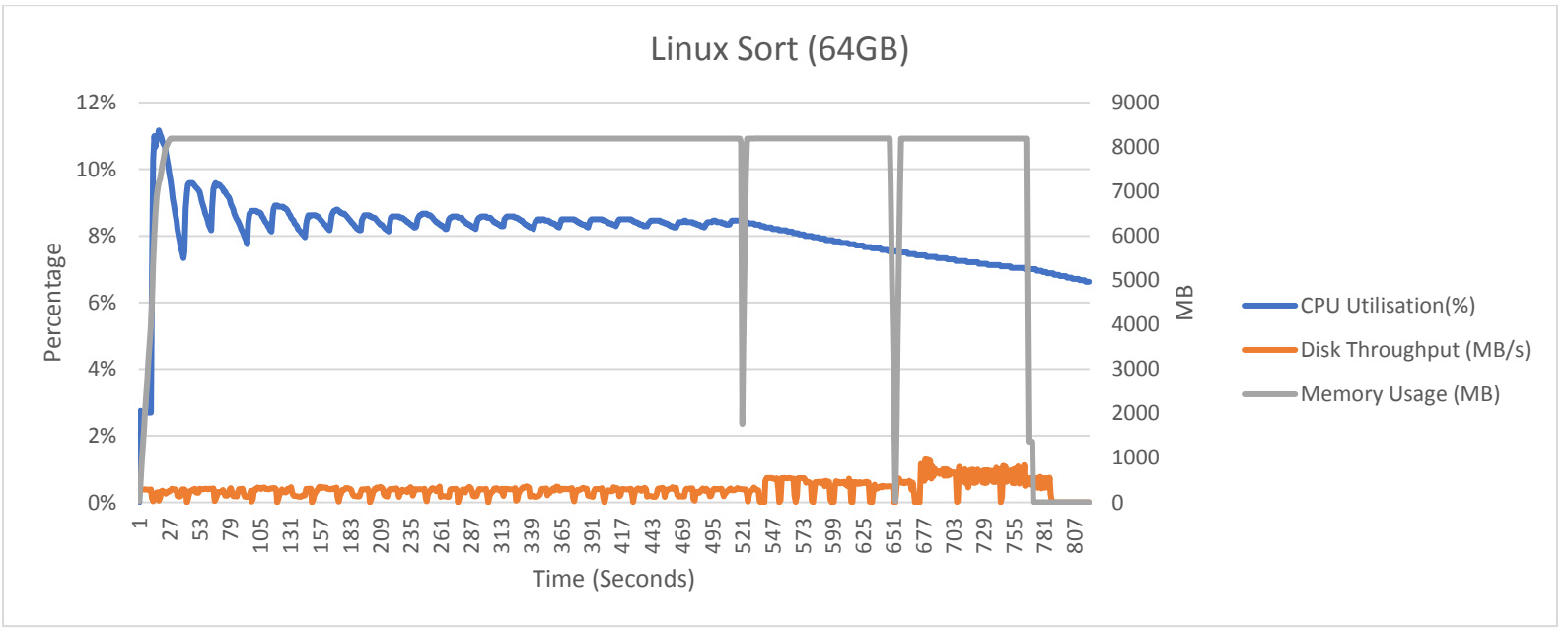

MySort (16 GB)
PErcentage
MB
Time (Seconds)
CPU Utilisation (%)
Disk Throughput (MB/s)
Memory Used (MB)


MySort (64 GB)
PErcentage
MB
Time (Seconds)
CPU Utilisation (%)
Disk Throughput (MB/s)
Memory Used (MB)


Linux Sort (1GB)
8%
7%
6%
5%
4%
3%
2%
1%
0%
Percentage
2500
2000
1500
1000
500
0
MB
1
2
3
4
5
6
7
8
9
10
11
Time (Seconds)
CPU Utilisation(%)
Disk Throughput (MB/s)
Memory Usage (MB)


Linux Sort (4GB)
12%
10%
8%
6%
4%
2%
0%
Percentage
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
MB
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51
Time (Seconds)
CPU Utilisation(%)
Disk Throughput (MB/s)
Memory Usage (MB)


Linux Sort (16GB)
Percentage
MB
Time (Seconds)
CPU Utilisation(%)
Disk Throughput (MB/s)
Memory Usage (MB)


Linux Sort (64GB)
Percentage
MB
Time (Seconds)
CPU Utilisation(%)
Disk Throughput (MB/s)
Memory Usage (MB)