Problem Statement:

Your task is to run a distributed Merge Sort on the High-Performance Computing facility. You will need to create job submissions such that each job deals with a reasonable amount of sorting.

Message Passing Interface (MPI) Based Approach:

1. Merging at the root node only with one chunk per node.

In this approach, there is one master node that will create the original list and distribute the different chunks among different nodes. Each node will receive one chunk, sort it, and send it back to the master node. Fig.1 shows the structure of the system. This system has 25 nodes, one master (00), and 24 slave nodes.

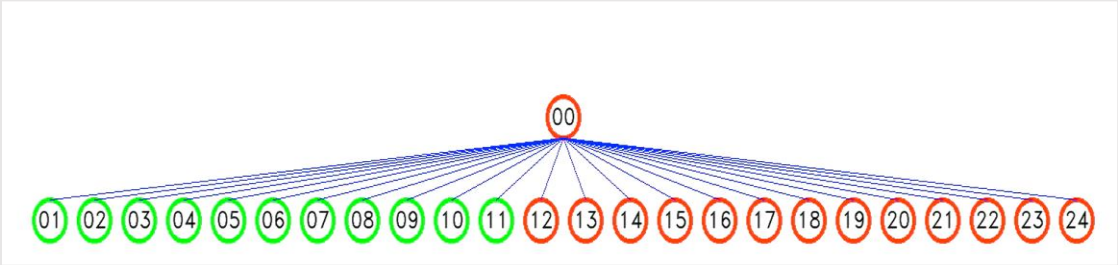


Figure 1: System architecture with one master, 24 slaves.

The results of this implementation with 4 slaves and 8 slaves are shown in Fig.2. As expected, the running time decreases with more working slaves being added to the system.

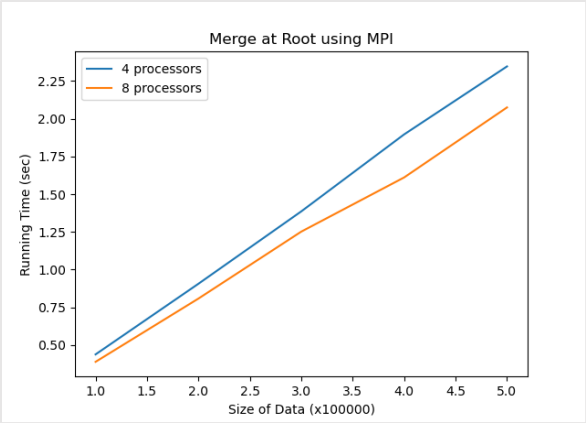


Figure 2: Running times for different data sizes with merging happing at root.

2. Merging at the root node only with multiple chunks per node.

The same system of one master, several slave, but with smaller size of data chunks is reimplemented again. Slave nodes will send a request to the master node to get data chunk, sort it, send it back, and ask for other chunks. The master node will keep track of received chunks and send back unsorted chunk. If all chunks are received, then, the master node will send a termination message to all slave nodes to terminate. The running times for different size of data are shown in Fig.3. As we increase the number of chunks, the running time will increase regardless of the number of slave nodes used. This is mainly because of the increase in the number of passed messages between slave nodes and master node. If the data was extremely large where sorting time is far more than passing sorted arrays time, then, we might see the opposite trend where the running time is decreased as we increase the number of slave nodes performing the sorting operation.

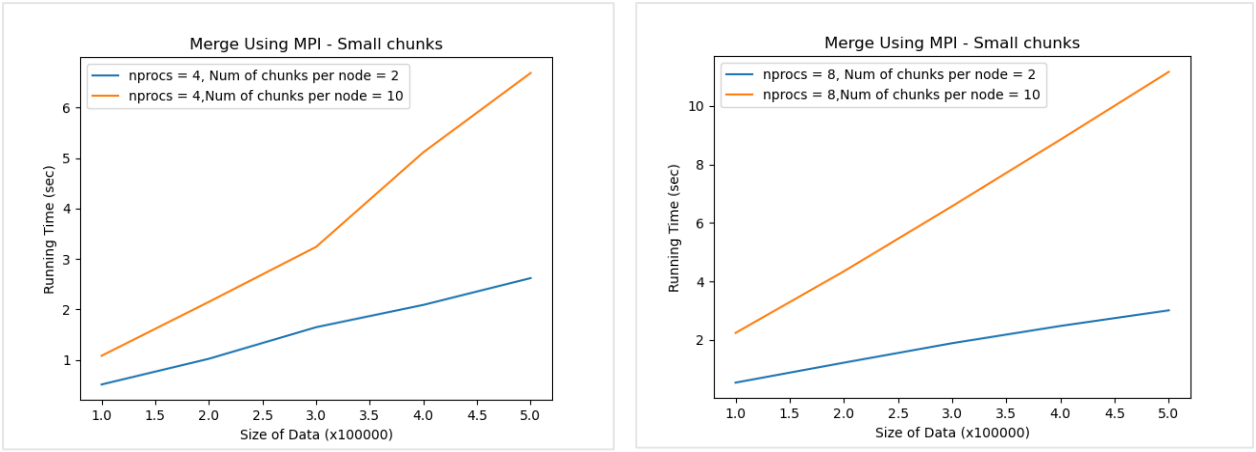


Figure 3: Running time for different data sizes as we increase the number of passed chunks.

Even for the same number of chunks, as we increase the number of slave nodes, the running time increases as the number of passed message is also increasing as shown in Fig.4.

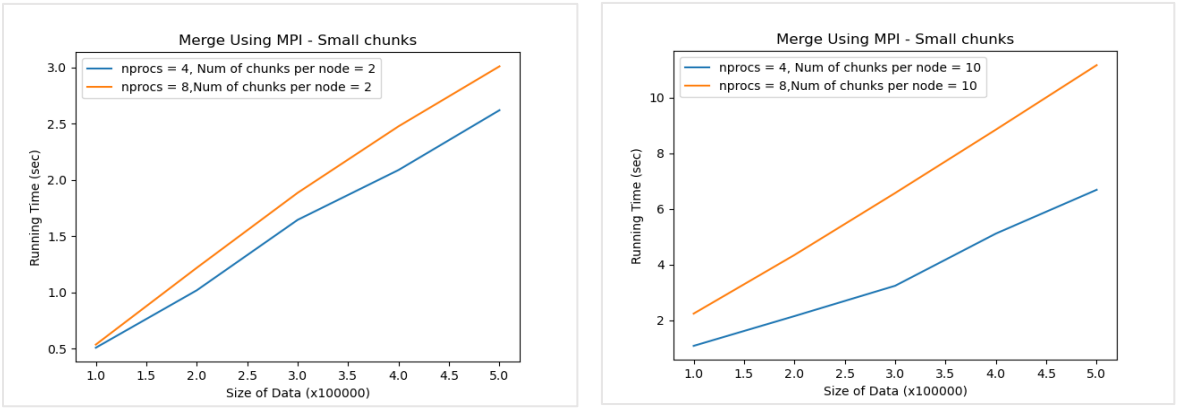


Figure 4: Running time for different data sizes. Same number of chunks, but with different number of slaves.

3. Merging at different nodes using dependency tree.

In the third implementation, MPI have been used to pass messages between nodes. A binary tree indicates the dependencies between nodes, where each node must send its sorted data in each iteration.

The final node having all sorted data is Node 01, this node will report the full data to the master node. A dependency tree of a system of one master, and 24 slave nodes is drawn in Fig.5.

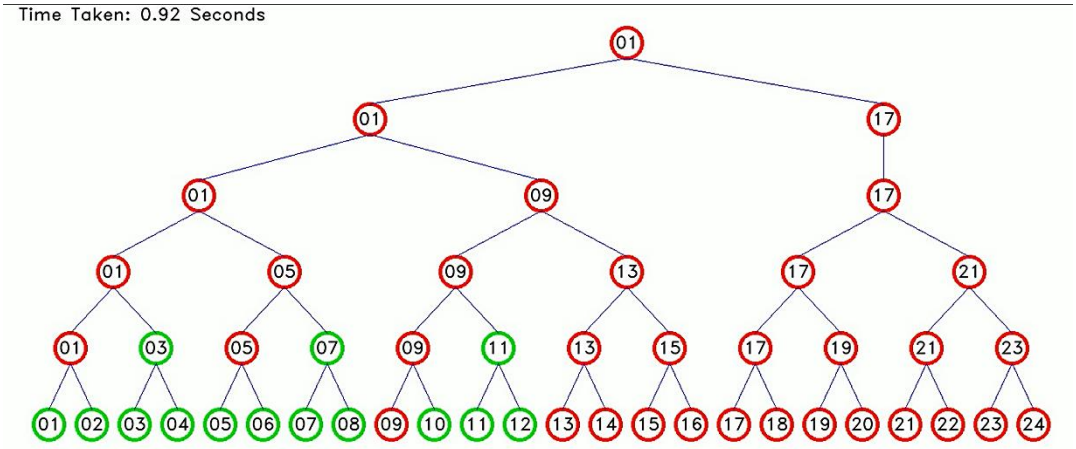
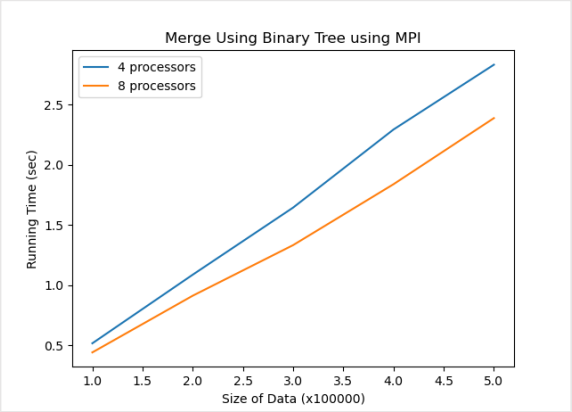


Figure 5: A dependency tree of a system of one master, and 24 slave nodes

When this approach is compared with the previous approach where all nodes are must send their sorted arrays to the master node, the dependency tree-based approach increases the performance with a greater number of slave nodes as shown in the left image in Fig.6. This is mainly because the messages are sent to several nodes in each iteration, this is better as the system will not have communication bottleneck at the root (master node).

The image to the right of Fig.6 shows the increase running time if we add the time of node 1 reporting the fully sorted list to the master node (Node 0). This is a great evidence showing that MPI might not be the best solution to use if the sorting time is not that large.



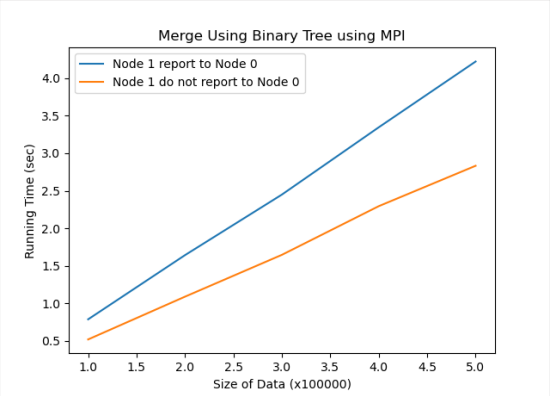


Figure 6: Dependency Tree based approach using MPI.

Network File Server Based Approach:

The fourth approach uses the network file server to distribute the tasks on different slave nodes. The master will create different data chunks, store each chunk in "**Unsorted_Files**" directory with a name of each node. Each node will wait until the master finishes the creation of the data, read the unsorted list, sort it, and save it in "**Sorted_Files**" directory with a name of the node. The master will merge the sorted files as they appear in the "**Sorted_Files**" directory. Based on the running time shown below in Fig.7, the NFS based approach becomes more stable as we increase the data size. For larger data, the running time is lower for larger number of slave nodes.

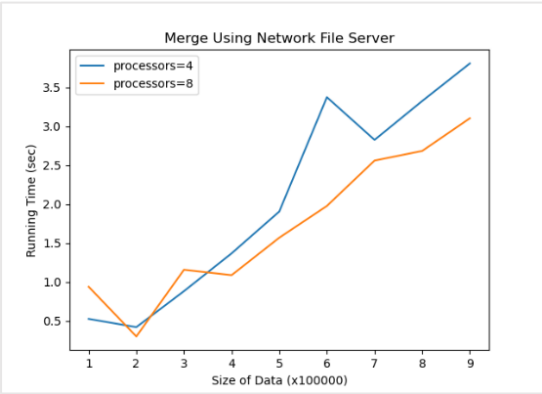


Figure 7: Merging using Network File Server.

Comparison between the four methods:

The NFS based approach is the best approach when compared with other MPI based approaches. This is mainly because there are no messages containing large arrays passed between nodes. For MPI approaches, the approach of merging all sorted lists at the root with fewer number of chunks is the best, again this is because it contains the least amount of message passing. Merging using dependency tree is the worst as it has the largest number of messages passed. Results are shown in Fig.8.

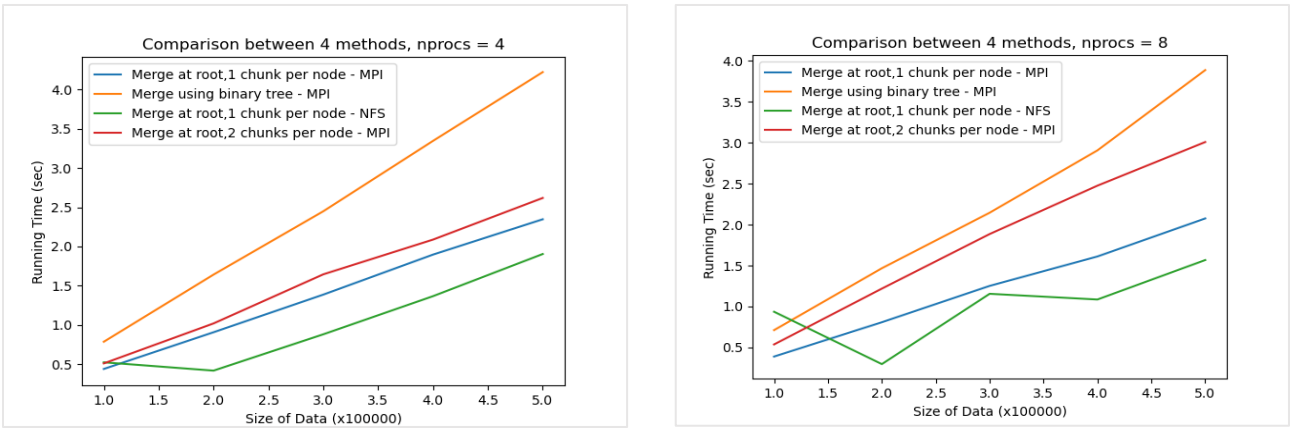


Figure 8: Comparison between the different approaches.