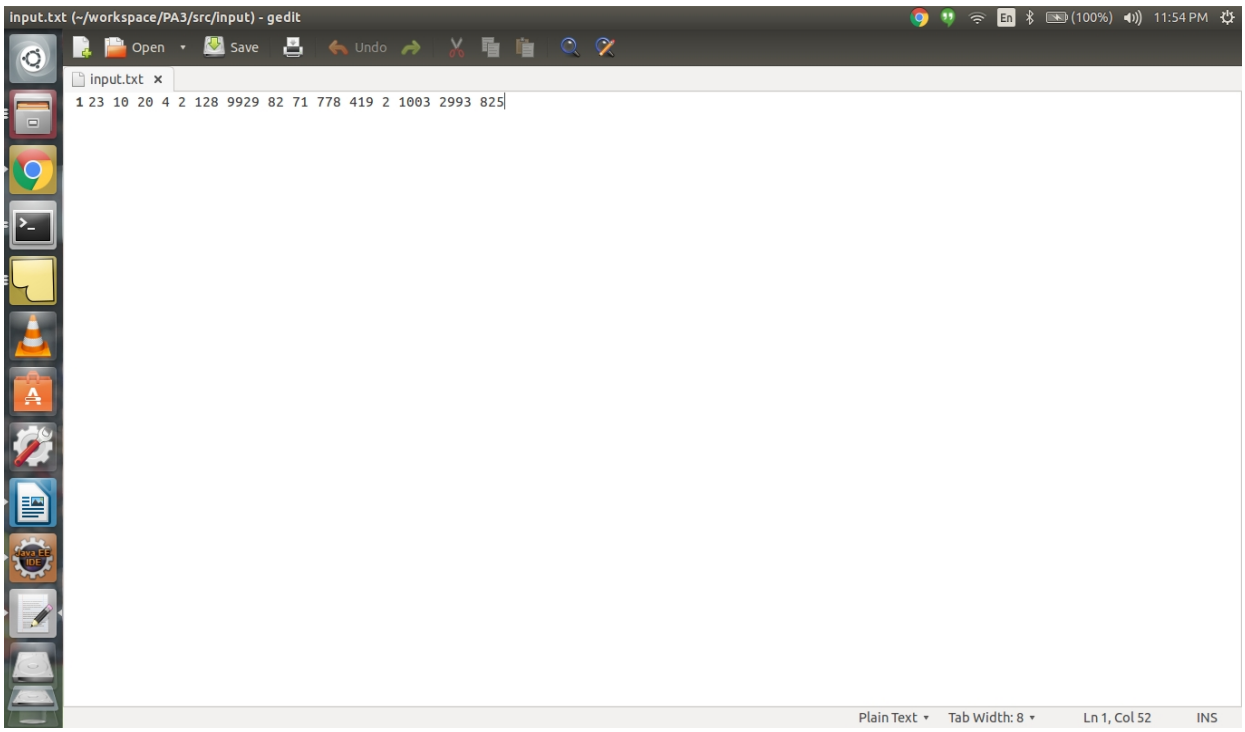
Test cases

Please have a look at the user document which details how to start all components of the system and how to use the interface. Before running these test cases, PLEASE ENSURE THAT YOU HAVE PLACED THE CODE IN A FOLDER WHERE YOU HAVE WRITE ACCESS, since input, intermediate and output files will be read and written to the file system.

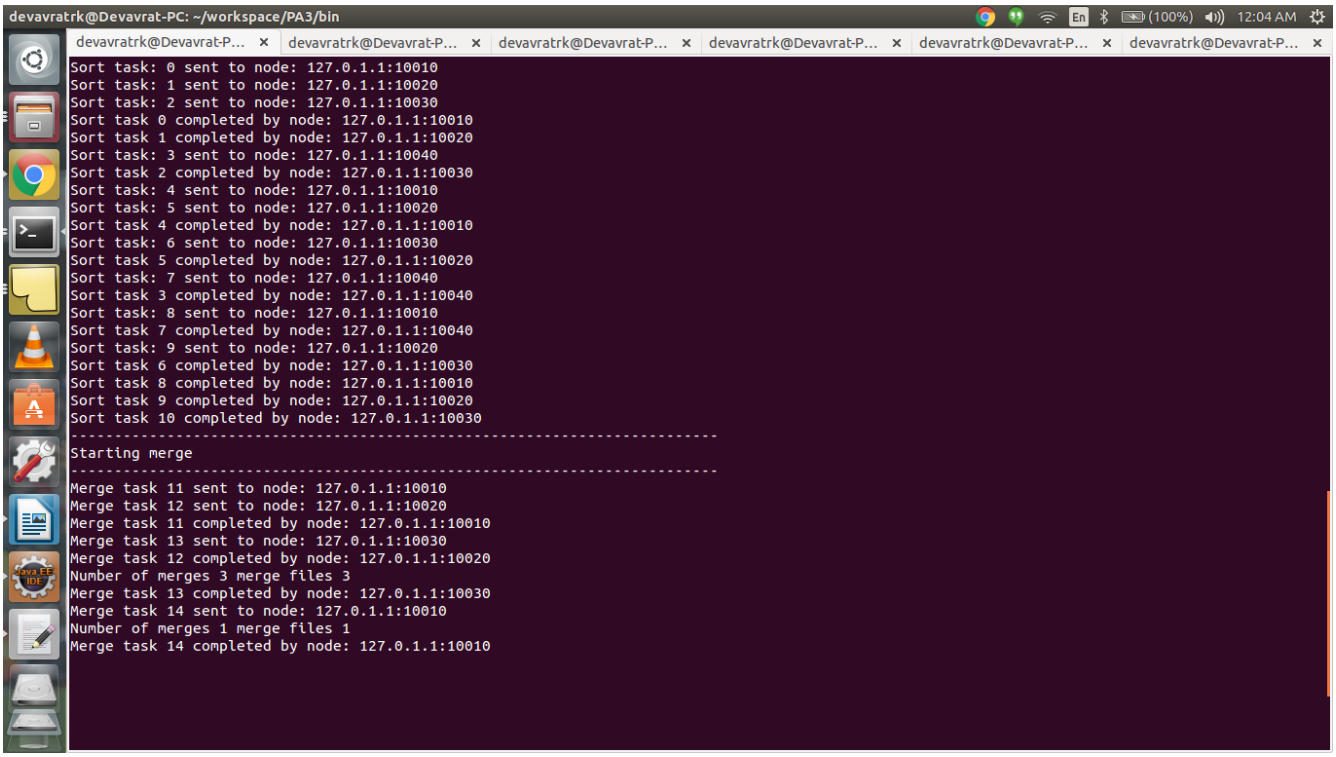
Before running any testcases, please do not forget to copy the input file to the input folder inside the source code folder.

All of these test cases have been run using four compute nodes.

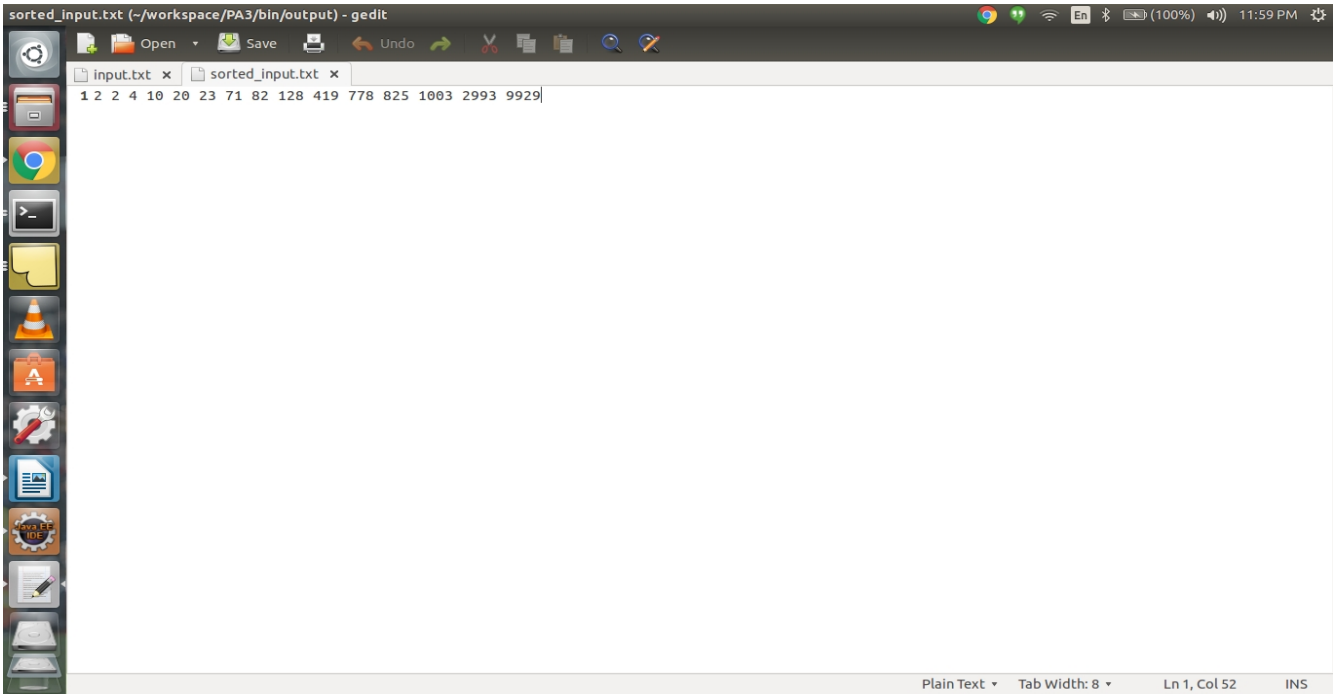
1) **Sorting a small file:** Please have a look at the input.txt file which has been provided in the input folder inside the source code folder. It contains 15 numbers in unsorted order. The size of the file is 52 bytes. The following testcase uses the chunk size parameter as 5 bytes and the number of files to be merged parameter as 4. The failure probability has been set to 0 for simplicity at all compute nodes. Further testcases will have a non zero failure probability at some nodes. Here is how the input file looks like:



The system is then started as mentioned in the user document and the filename input.txt is entered at the client UI. The system starts working on sorting the input file. Here is how the server terminal window looks during execution:

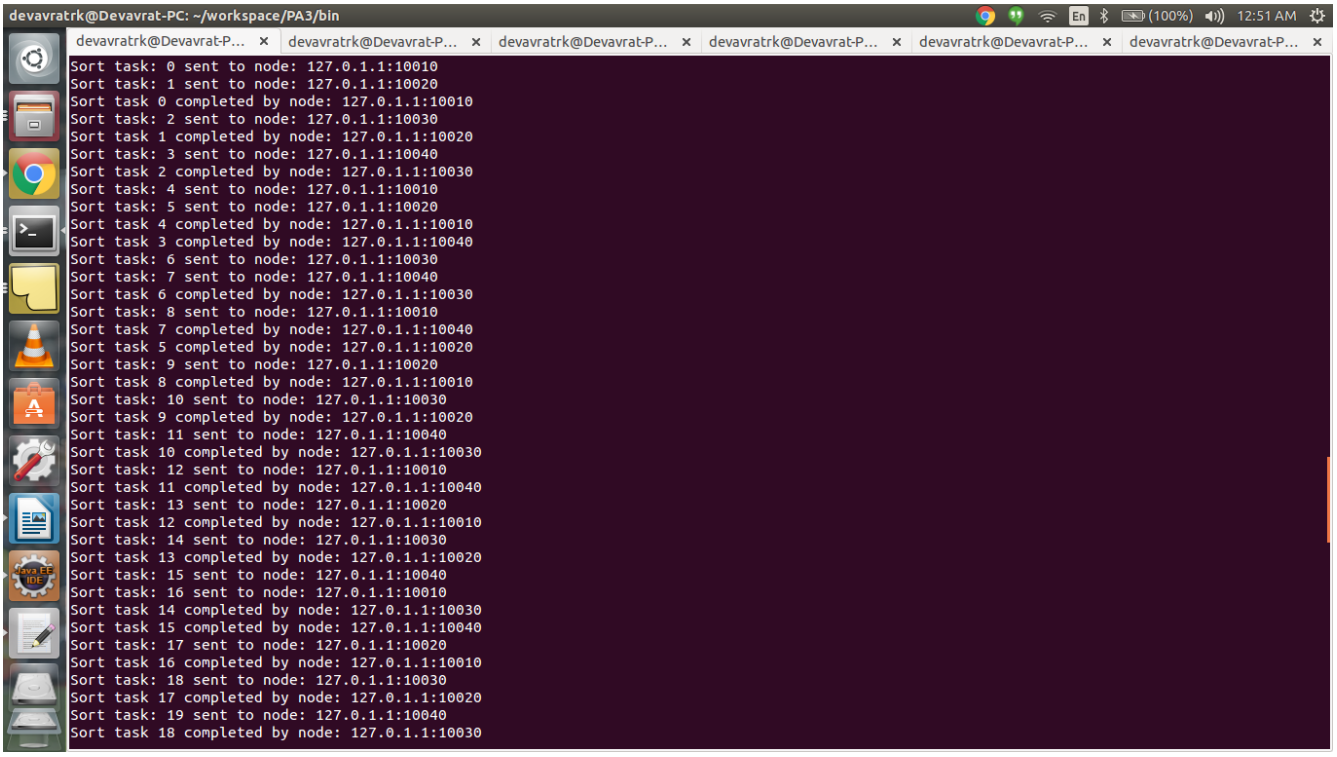


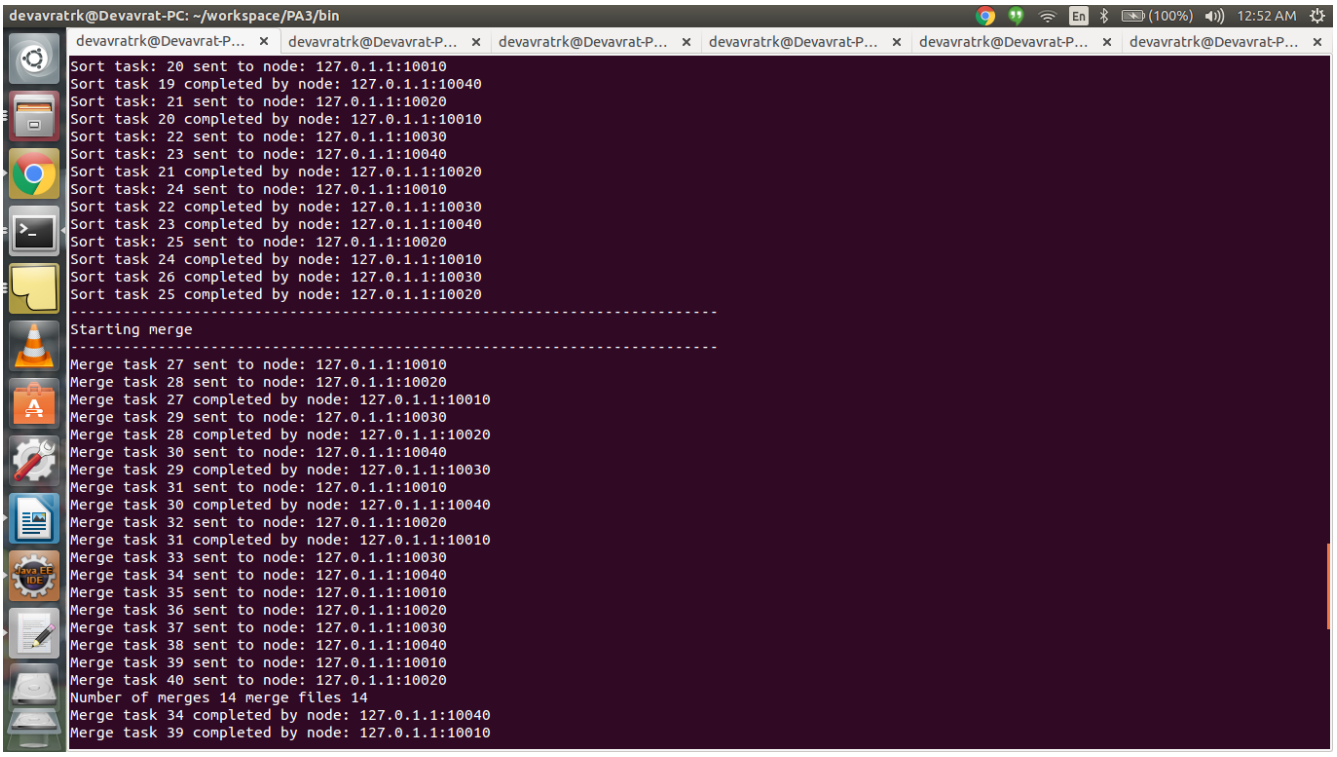
We see from the above output that there are 11 sort tasks as expected (52/5). There are 3 merge tasks in the first pass (11/4) and these 3 merge tasks produce 3 intermediate files, which are then merged in the second pass to produce the final sorted file, which is then moved to the output folder. Here is how the output file looks like:

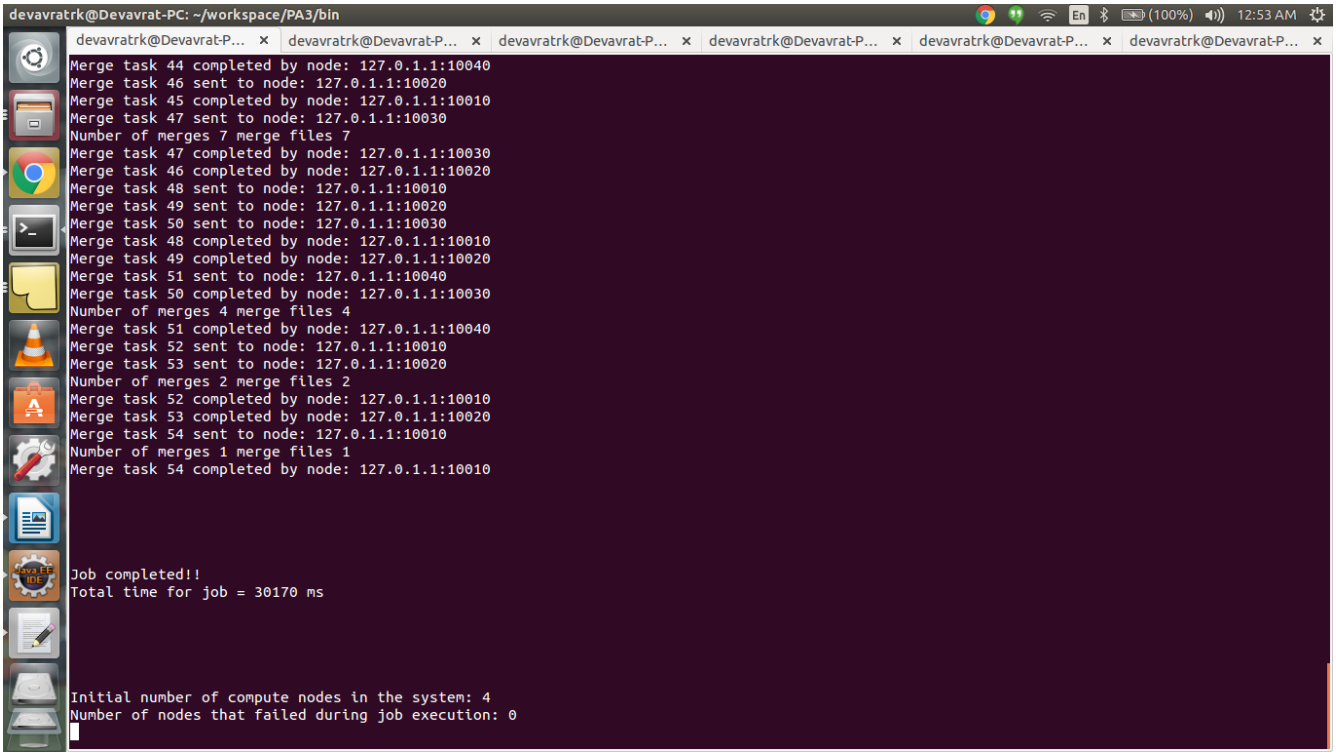


We can see that the file is sorted.

2) **Sorting the same file with different parameters:** We start the server and compute nodes once again. The parameters now are 2 bytes as chunk size and 2 files to be merged per merge task. The client is started once again and given the same input.txt file as input to the system. Here is how the server terminal window looks like after execution:



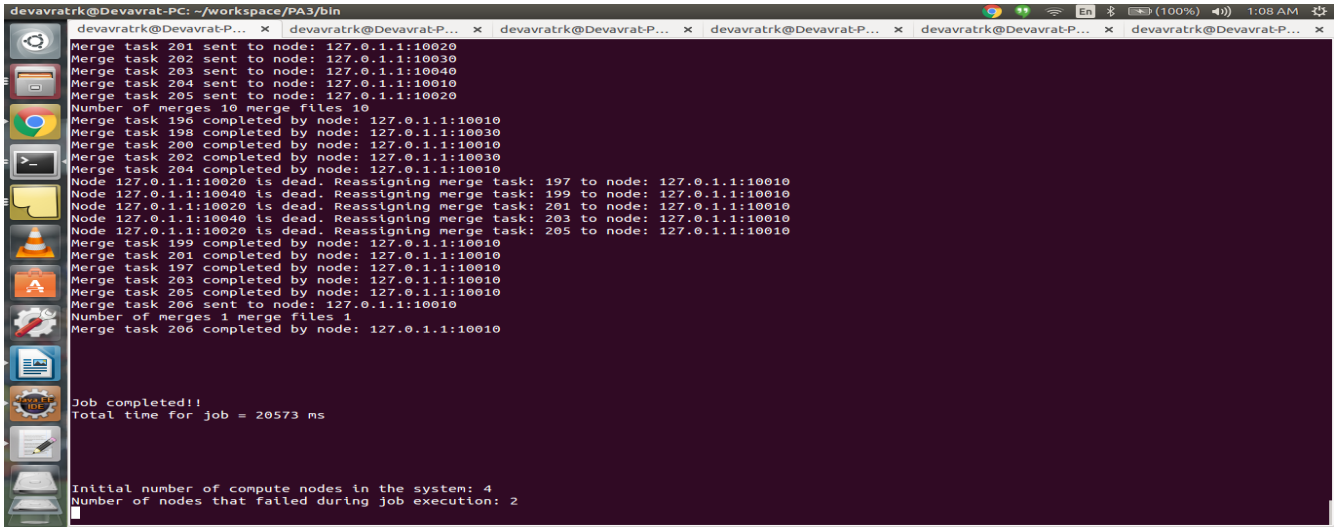




Some merges have been skipped from the terminal window output in the last screenshot to show the final few lines of the output. The final lines say that the whole job took about 30 seconds to complete and there were no compute node failures in the system, since the probability of failure at each node was 0. The large time can be attributed to having very small tasks in the system and now the processes of communication and thread creation and the reassign loop dominate to give a larger runtime than expected.

3) **Sorting a larger file (200,000) with a probability of failure:**

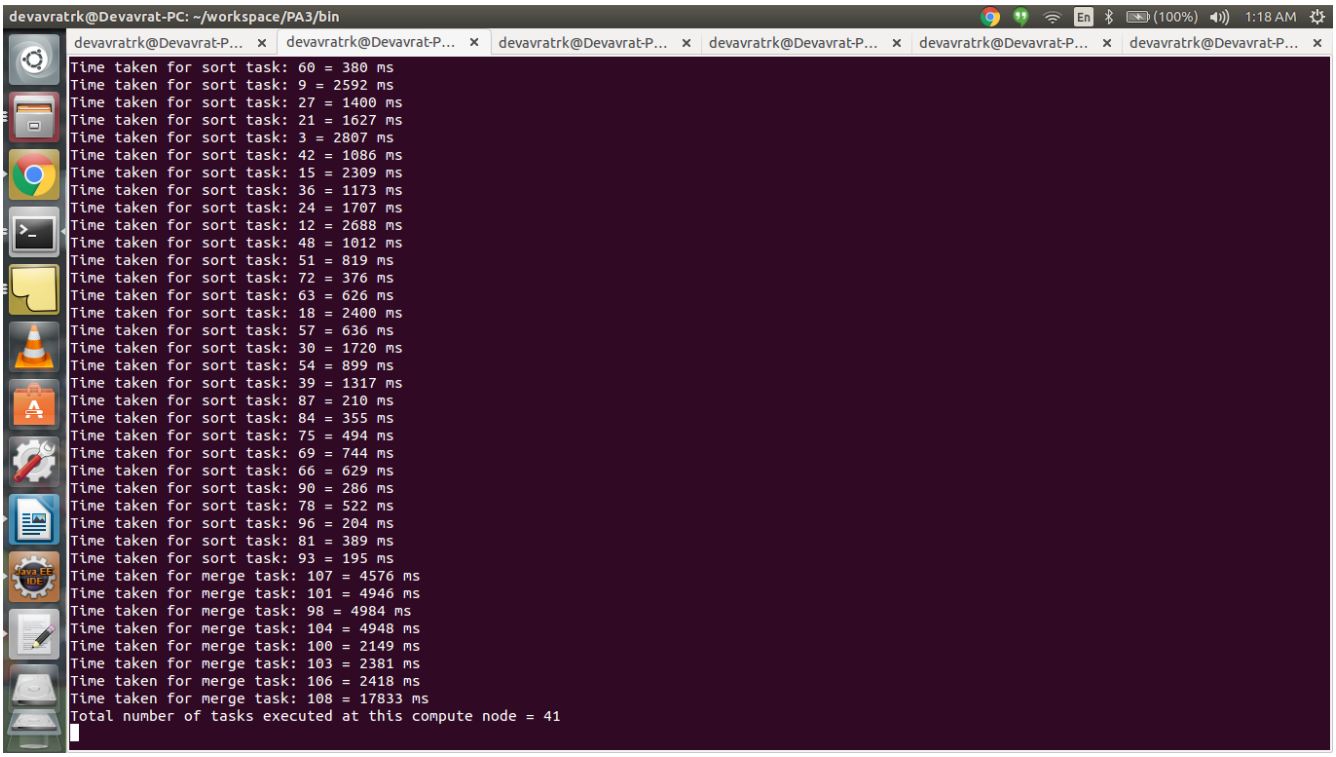
We now sort the smallest file out of the given input files, which has a size of 977,556 bytes, with chunk size parameter 5000 (5 kB) and merge list size parameter 20. Three nodes now have a failure probability of 0.5 and one node has a failure probability of 0. For more details on the failure injection logic, please refer to the design document. We now start the server and the compute node again with these parameters. The interesting parts of the output have been shown:



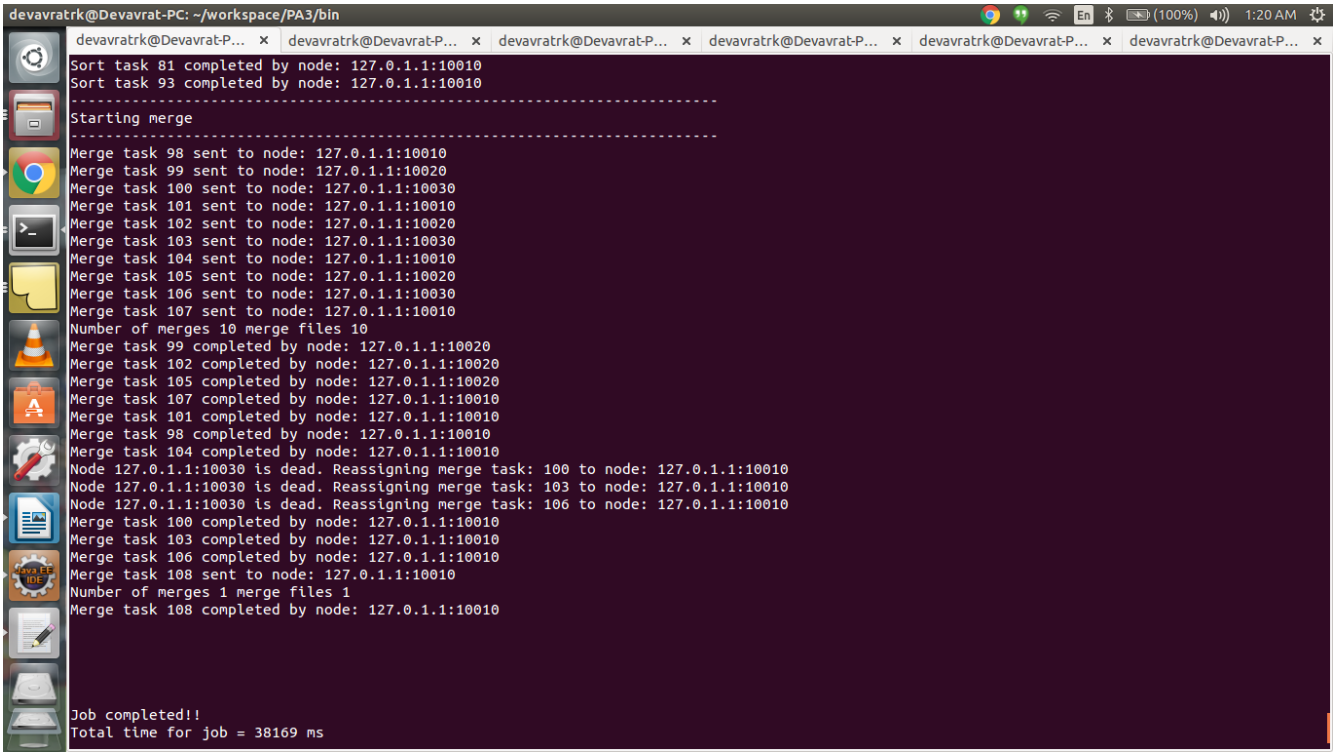
We see from the above screenshot that nodes 2 and 4 died while the merge process was going on and their merge tasks were reassigned. The job took 20 seconds to run and 2 nodes failed during execution.

4) **Sorting the largest file (20,000,000) with a probability of failure:**

We now sort the largest file out of the given input files, which has a size of 97,777,906 bytes, with chunk size parameter 1000000 (1 MB) and merge list size parameter 10. Three nodes now have a failure probability of 0.5 and one node has a failure probability of 0. For more details on the failure injection logic, please refer to the design document. We now start the server and the compute node again with these parameters. The interesting parts of the output have been shown:



The above screenshot is from a compute node. It shows the time each sort and merge task took and the total number of tasks it executed.



We see from the above screenshot that node 3 died while the merge process was going on and merge tasks were reassigned. The job took 38 seconds to run. We verified the files in the output folder and they had been sorted.