Test design and report

Test case 1:
Random mode
4 compute nodes with 3000ms inject delay
Same load probability

Random

Load prob	5 times				
0.1	9766	9422	8316	8646	8306
0.2	9817	8369	7844	8137	8135
0.3	9352	8005	7830	7884	8096
0.4	9062	7782	6841	6546	7171
0.5	9262	6835	6642	6564	6487
0.6	7668	6627	6811	6868	7251
0.7	8029	7567	7320	7301	7297
0.8	8663	7726	8293	7640	7693
0.9	9653	8271	8505	8374	8444

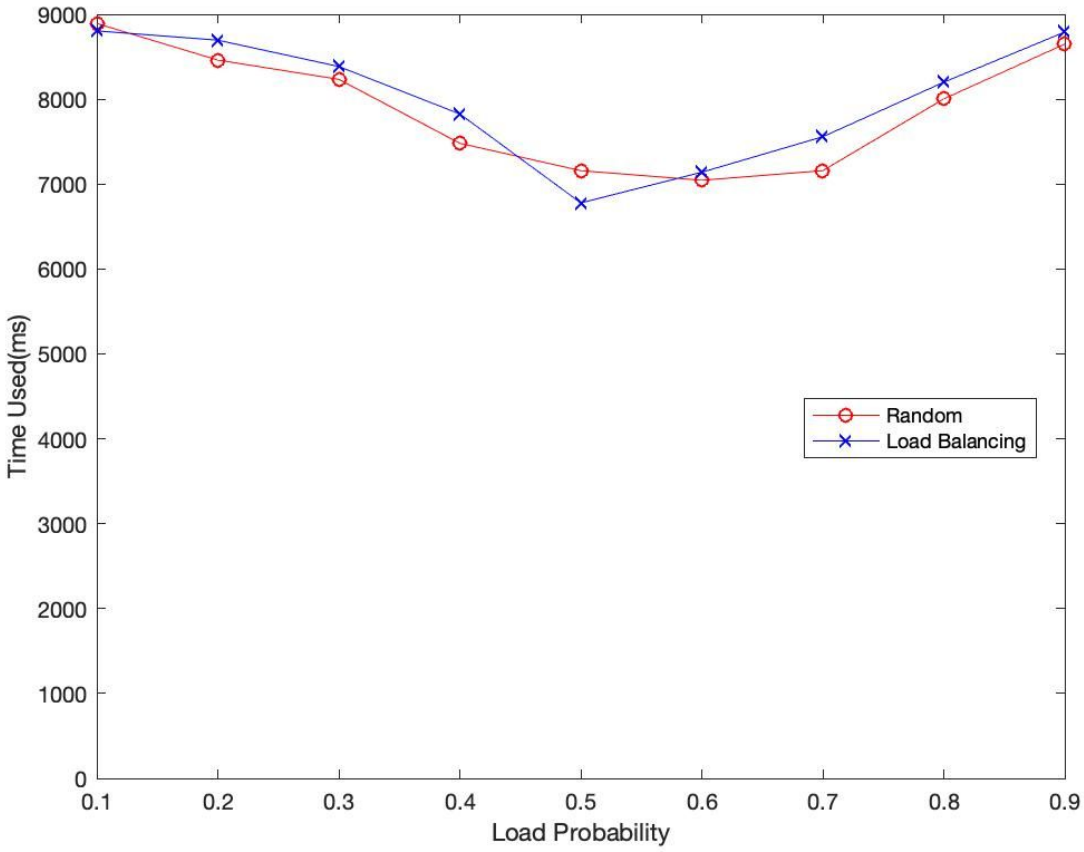
Test case 2:
Load-Balancing Mode
4 compute nodes with 3000ms inject delay
Same load probability

Load balancing

Load prob	5 times				
0.1	9887	8682	8574	8498	8380
0.2	10121	8664	8203	8344	8142
0.3	9593	7952	7842	8080	8456
0.4	9241	7769	7754	7024	7340
0.5	7873	6368	6791	6417	6443

0.6	8048	7086	7003	6764	6796
0.7	8558	7302	7474	7364	7090
0.8	8930	8049	8164	8003	7853
0.9	9465	8810	8489	8332	8860

Plot using average time:



Analysis of test case 1 and 2:
When load probability of each node is the same, random scheduling and load balancing both are likely to distribute tasks equally to each node. Thus the performance of random and load balancing is more likely to be similar. And with the increase of load probability, the time used should increase at the same time, because each node is more likely to be injected, which will slow down the speed of processing our tasks.
While in the test, we find our result goes down first then goes up. We guess the reason is that the number of files in memory on each compute node is increasing and next time when a test is running some file is still in the memory, so it does not need to be loaded again.

Test case 3:

All 4 compute nodes have different load probability, i.e., (0.1, 0.5, 0.2, 0.9), in two cases(with 3000ms inject delay and 5000ms inject delay)

Load Prob of (0.1, 0.5, 0.2, 0.8)

scheduling	5 times				
Random w/ 3000ms & 5000ms delay	9166 10086	7600 9057	7703 8489	6848 8949	6824 9026
Load Balancing w/ 3000ms & 5000ms delay	9620 11237	8002 9669	7813 9689	8015 9285	7732 9561

Analysis of test case 3:
Theoretically, for nodes with different load probability, random scheduling are more likely to distribute tasks equally and load balancing will distribute fewer tasks to nodes with high load probability because they will inject other tasks, which means random scheduling are more likely to distribute more tasks to the slow nodes, i.e. nodes with high load probabilities, so the random scheduling performance should perform worse than Load balancing.
But in practical, random scheduling performs a little bit better than Load Balancing in both injecting delay cases. This may be caused by the time consumed in redistributing procedure when tasks are rejected by nodes, or lack of testing nodes.

Test case 4(Negative case):

When we input an error folder parameter to the client, an exception would throw: “Exception in thread "main" java.lang.NullPointerException at Client.main(Client.java:25)”

When we input a correct folder to the client, but some file is deleted before the server sends tasks to compute node, an error would be return and the whole task stop: "Error, some file does not exist."

When compute node cannot find a file or other exception, the compute node would print exception info and the server would show “Map job met with some error, check if file exists.” But the whole task continues without processing this file.