Lei Huang; huan1397; 5330737
Yihan Zhou; zhou1298; 5547186

Testing Description

1) Performance Evaluation

We test the system performance based on different scheduling policies and different load-probability assigned to Compute Nodes.

Part 1: All compute nodes have the same load prabablies

We set up all 4 compute nodes with same load-probability. Time is in milliseconds.

Set 4 nodes load-probability to 0.0

random: 7310 7210 7295 Average: 7271.67
load-balancing: 7412 7201 7295 Average: 7271.67

Set 4 nodes load-probability to 0.2

random: 8538 8461 8317 Average: 8437.67
load-balancing: 8366 8420 8295 Average: 8360.3

Set 4 nodes load-probability to 0.5

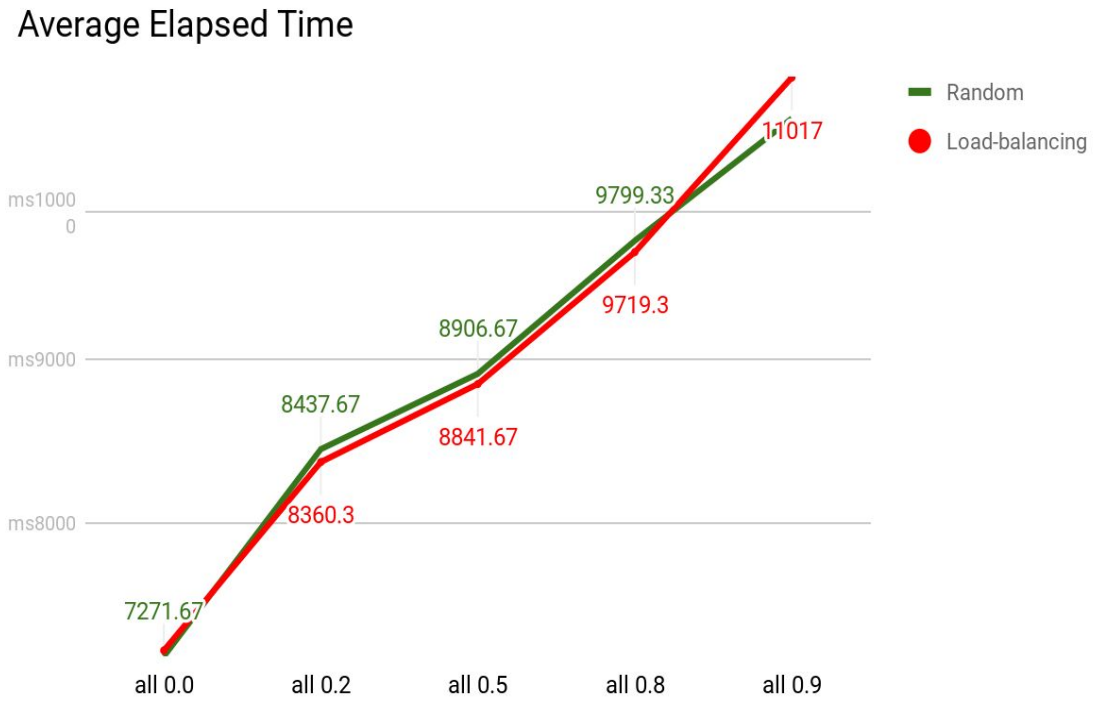
random: 8946 8910 8864 Average: 8906.67
load-balancing: 8820 8790 8915 Average: 8841.67

Set 4 nodes load-probability to 0.8

random: 9950 9820 9628 Average: 9799.33
load-balancing: 9816 9767 9575 Average: 9719.3

Set 4 nodes load-probability to 0.9

random: 11381 10353 10359 Average: 11017
load-balancing: 11343 10573 11135 Average: 11017



Apparently, with the load-probability increasing, the time injection possibility for each task also increases.

Part 2: All compute nodes have different load probabilities: 0.1 0.5 0.2 0.9.

We did tests with random policy and load-balancing policy. Each one for 6 times.

random:	8290	8263	8394	8440	8339	8421	Average: 8357
load-balancing:	8360	8347	8294	8283	8401	8361	Average: 8341

From the test results, we can see that, based on the different load probability assignments, the average elapsed time for load-balancing policy is very close to the average elapsed time for random policy. Apparently, our expectation is that load-balancing policy should have a better performance since it tends to schedule more tasks on low-load compute node which will have a low possibility to inject time delay. However, the experiment results show that two policies basically have the same performance.

There are two major factors which could lead to this result:

1) Multi-thread compute node

For a compute node, multiple tasks are running on it concurrently. No matter how many of the accepted tasks are injected with time delay, they will all get into sleep function and wait concurrently. In this case, more threads on the sleep status will not cause longer waiting time generally.

2) Network overhead

For the load-balancing policy, rejection of the task could happen on the compute node side, which means that many rejections and resending messages will be transmitting back and forth between the server and compute node. Ideally, the load-balancing policy should be completely implemented on the server side which will save the network overhead.

2) Negative test case

1) Start the server but not start the compute node. In this case, if a job is submitted, the system will stop with error messages reported on both the Server and Client sides.

2) In our project, we assume the 3 directories (input_dir, intermediate_dir and output_dir) are already there in the same directory with executable files. If one of the directories is missing, the system will stop with error messages.

3) If the params.txt file is not in the folder or the content has wrong format, the system will stop with error messages.

4) System will stop when the server and compute node addresses are not correct.