CS425, Distributed Systems: Fall 2017

Machine Programming 4 – Sava

Tiancheng Wu netid: twu54

Haowen Jiang netid: haowenj2

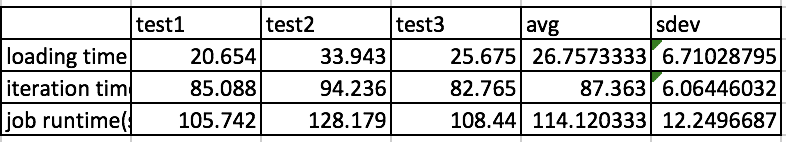
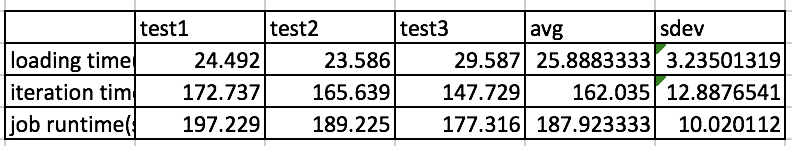
# Design:

The purpose of this project is to build a Sava system to realize simple Pagerank Algorithm and Single Source Shortest Path Algorithm. Through analyzing the command from client vm, accepting the application code and txt file, master will partition the graph with edge cut and assign several workers to do calculations. The portioning function used is random partitioning for simplicity. For each super step, the workers will save the current vertices info to HDFS if asked and compute for each vertex as specified in application. Whenever no more messages received and every vertex is set to halt by user, all the workers will finish the computation and send the results to the master. The master will combine all the values from workers and send the results to the user clients. During the super step, the calculation delivery is based on TCP protocol. Each worker will analyze where the next vertex is and do the calculation on the local if the next vertex and current vertex locate in the same vm. For the failure detection, if one of the workers down during the super step progress or calculation progress, the master will recover the vertices of this worker to the other alive workers. The executing value can be fetched from the HDFS so that the Sava can keep calculation rather than restart the whole progress. A standby machine will monitor the master all the time. Whenever the master illegal quit from the Sava system, the standby machine will replace the master and do the same tasks as the master. For Single Source Shortest Path, the vertex will save the step count on each super step. Whenever there are no new messages received, the vertex will halt itself and if all the vertexes finish the calculation, the workers will send the path count back to the master. The master will analysis the results and send the shortest paths to the client.

# Sava Data:

Number of Tasks (7 workers):

Pagerank: 10 steps Pagerank: 20 steps

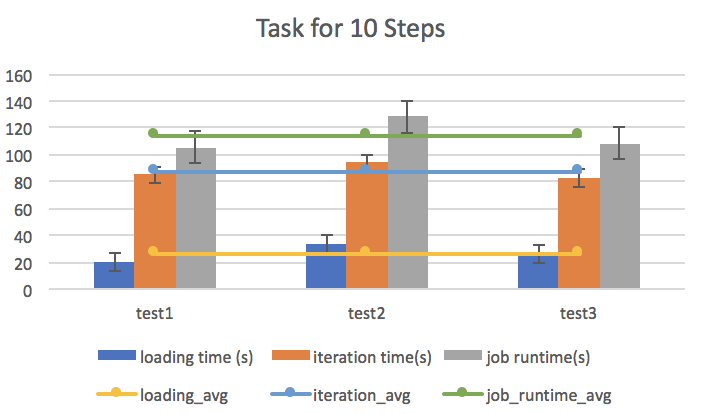
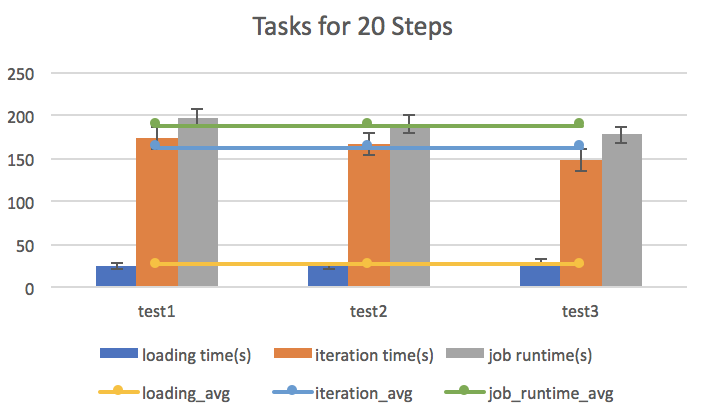
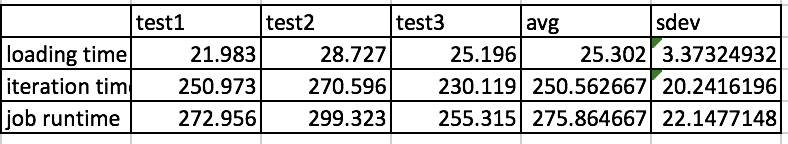
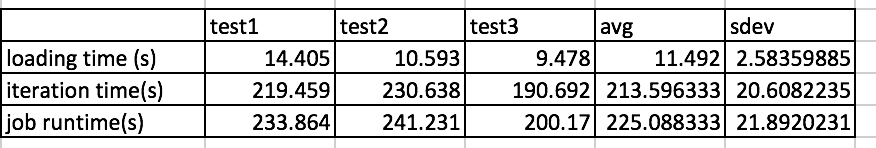
 

figure 1 figure 2

Pagerank: 30 steps Pagerank: 3 servers (20 steps)

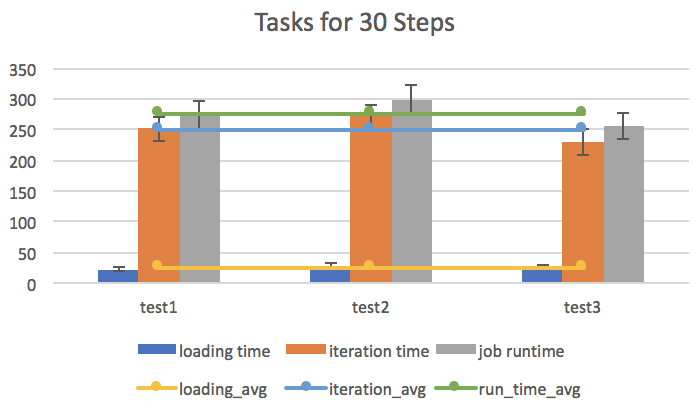
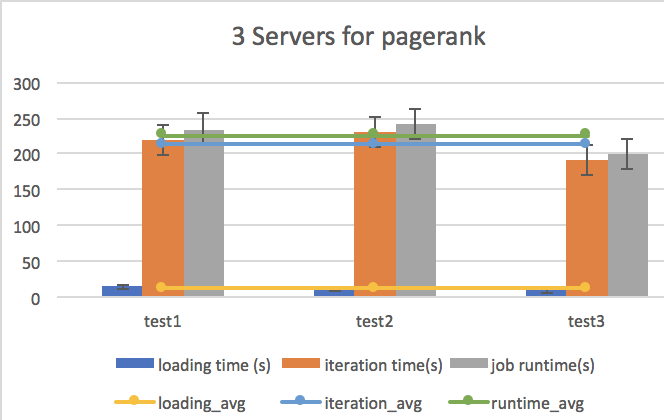
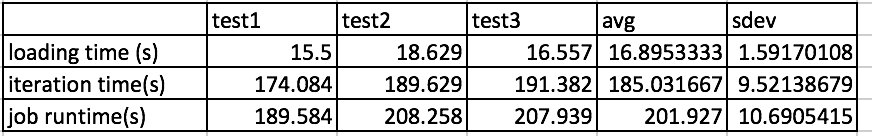
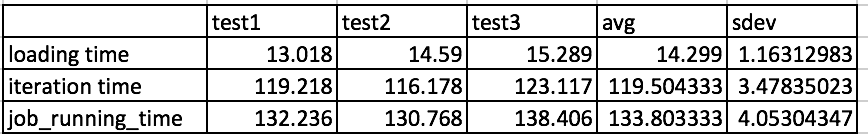
 

figure 3 figure 4

Pagerank: 7 servers (20 steps) 🡪See figure 2 (Same)

Pagerank: 5 servers (20 steps) SSSP 3 servers:

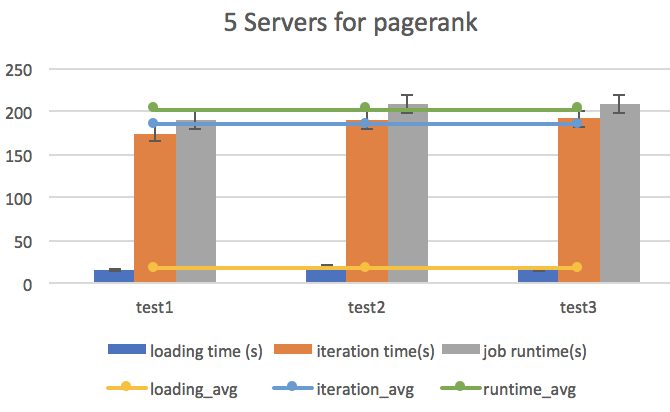
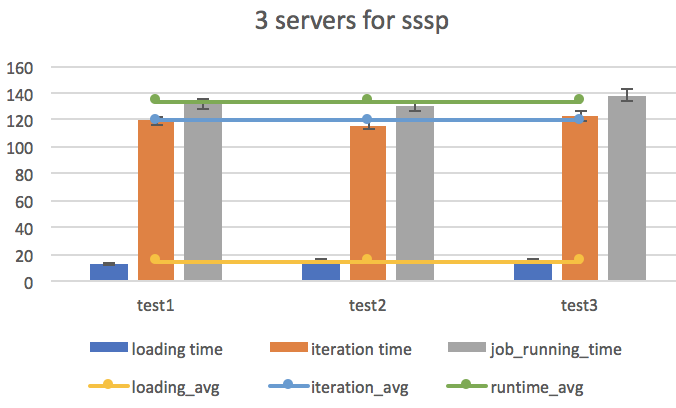
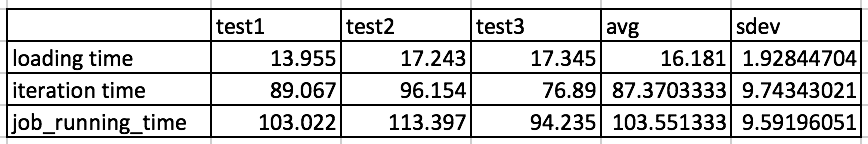
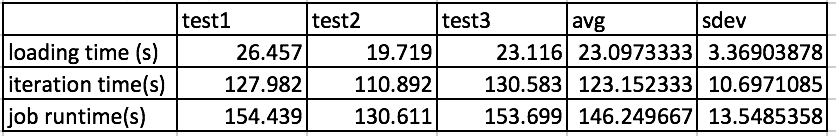
 

figure 5 figure 6

SSSP 5 servers: SSSP 7 servers:

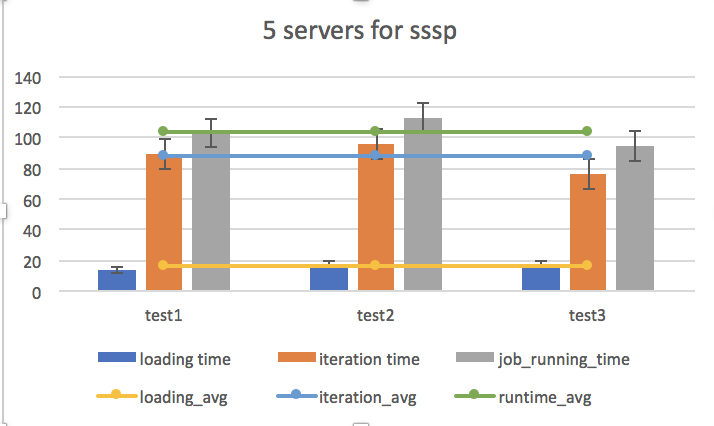
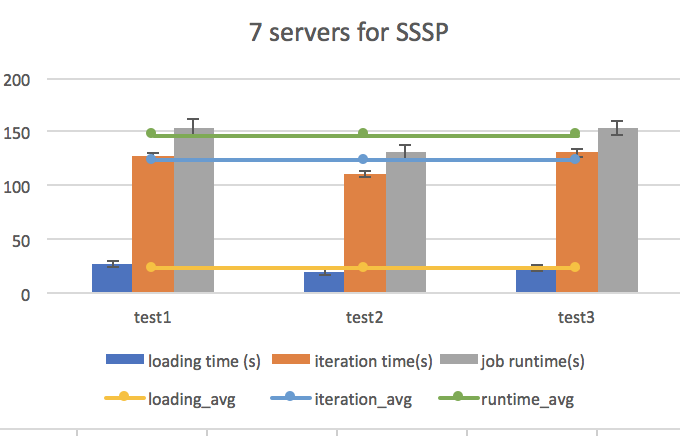
 

figure 7 figure 8

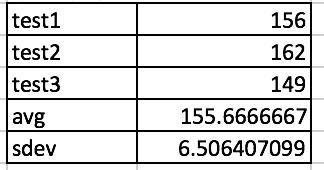
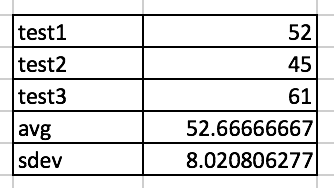
From the plot above, when servers number is given, the execution time will increase with step number. The reason is that our system is not a converge system. In other words, when the superstep is achieved the setting number, the system will force to stop and all workers will return the results to the master rather than calculate to the end. Meanwhile, the execution time should decrease with the server numbers when the step number is fixed. However, from figure 8 we can know that the runtime increases when the servers number arrived at 7. The reason why this happened might be the great communication overhead between machines and the waiting time between iterations.

# Sava vs. Spark Graphx

Sava pagerank performance: Sava SSSP performance:

see figure 2 see figure 8

Spark pagerank performance: Spark SSSP performance:

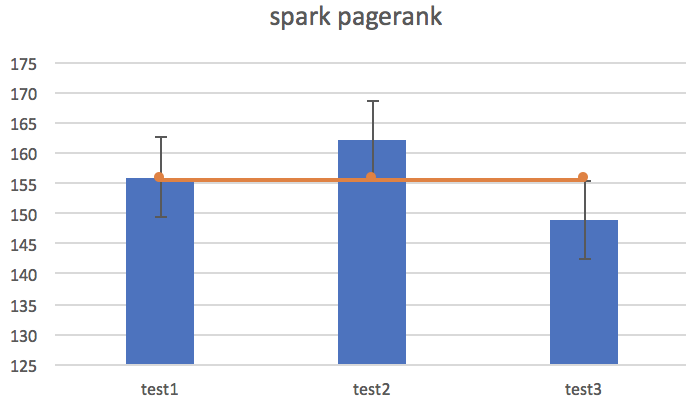
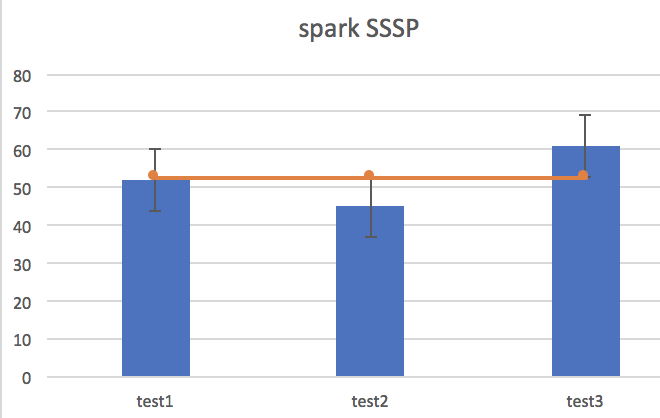
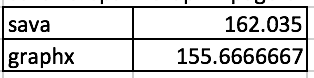
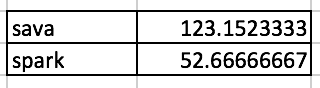
 

figure 9 figure 10

sava vs graphx: pagerank sava vs. graphx sssp

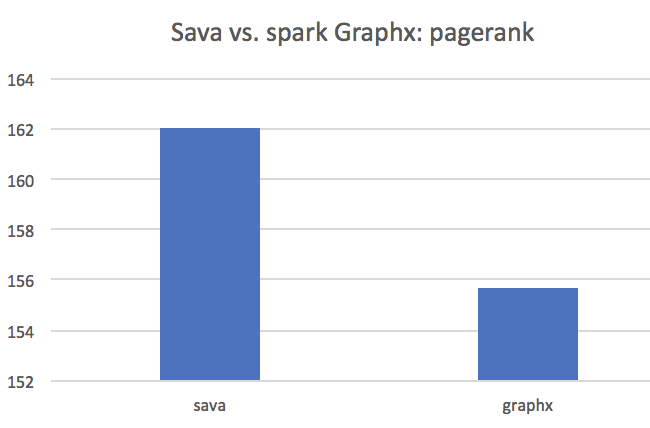
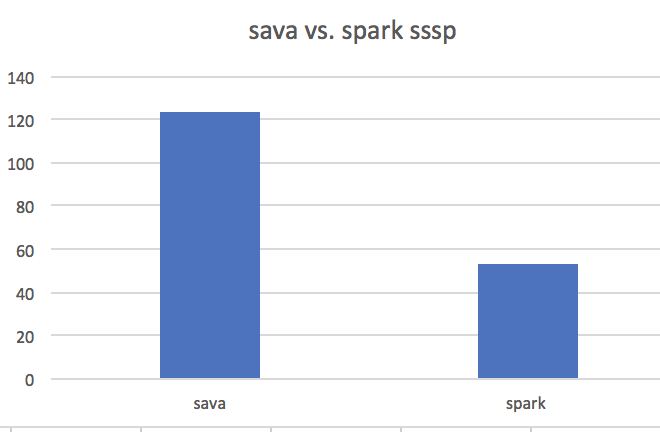
 

figure 11 figure 12

From the plot showing above, the spark graphx is still a little bit faster than sava, even though the executed time is close. Some solutions we can think about are: improving parallelism, reducing the waiting time among supersteps and improving the partitioning function efficiency.