CS425, Distributed Systems: Fall 2017
Machine Programming 4 – Sava

Tiancheng Wu netid: twu54
Haowen Jiang netid: haowenj2

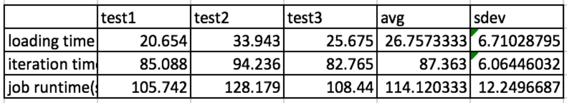
Design:

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:

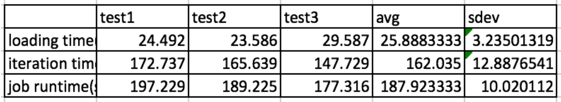
Number of Tasks (7 workers):

Pagerank: 10 steps



	test1	test2	test3	avg	sdev
loading time	20.654	33.943	25.675	26.7573333	6.71028795
iteration tim	85.088	94.236	82.765	87.363	6.06446032
job runtime(	105.742	128.179	108.44	114.120333	12.2496687

Pagerank: 20 steps



	test1	test2	test3	avg	sdev
loading time	24.492	23.586	29.587	25.8883333	3.23501319
iteration tim	172.737	165.639	147.729	162.035	12.8876541
job runtime(	197.229	189.225	177.316	187.923333	10.020112

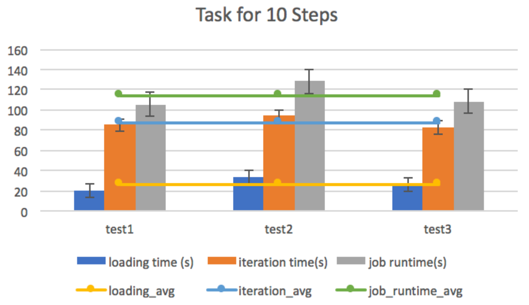


figure 1

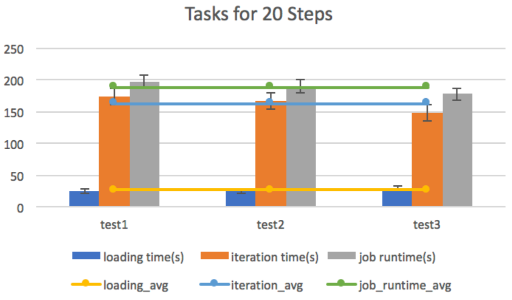
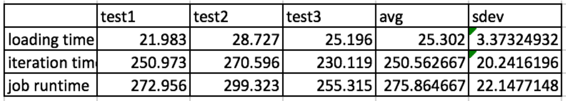


figure 2

Pagerank: 30 steps



	test1	test2	test3	avg	sdev
loading time	21.983	28.727	25.196	25.302	3.37324932
iteration tim	250.973	270.596	230.119	250.562667	20.2416196
job runtime	272.956	299.323	255.315	275.864667	22.1477148

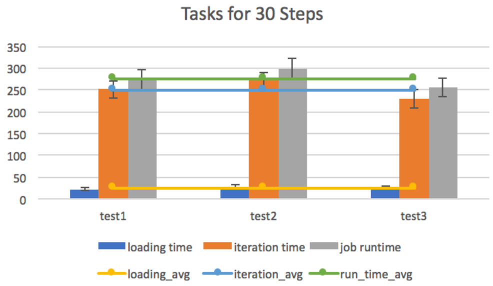
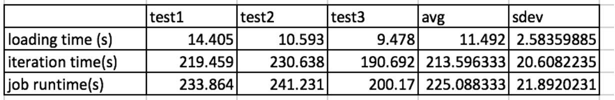


figure 3

Pagerank: 3 servers (20 steps)



	test1	test2	test3	avg	sdev
loading time (s)	14.405	10.593	9.478	11.492	2.58359885
iteration time(s)	219.459	230.638	190.692	213.596333	20.6082235
job runtime(s)	233.864	241.231	200.17	225.088333	21.8920231

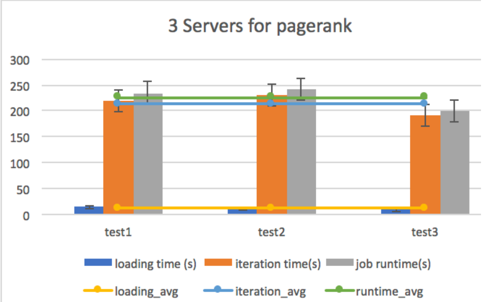
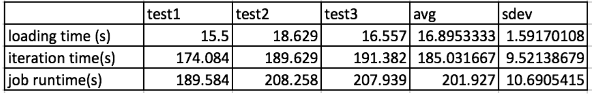


figure 4

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

Pagerank: 5 servers (20 steps)



	test1	test2	test3	avg	sdev
loading time (s)	15.5	18.629	16.557	16.8953333	1.59170108
iteration time(s)	174.084	189.629	191.382	185.031667	9.52138679
job runtime(s)	189.584	208.258	207.939	201.927	10.6905415

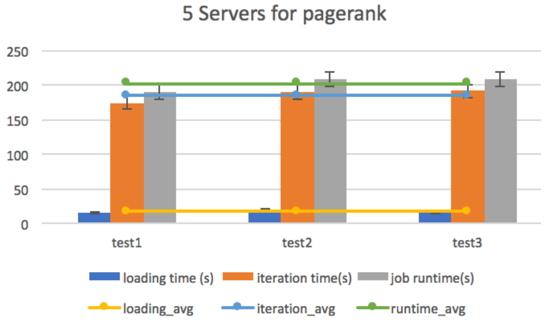
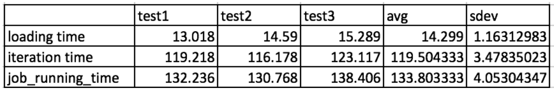


figure 5

SSSP 3 servers:



	test1	test2	test3	avg	sdev
loading time	13.018	14.59	15.289	14.299	1.16312983
iteration time	119.218	116.178	123.117	119.504333	3.47835023
job_running_time	132.236	130.768	138.406	133.803333	4.05304347

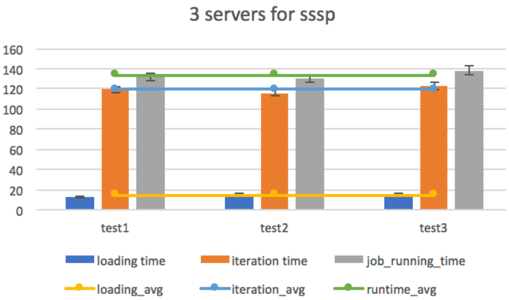
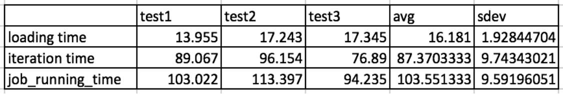


figure 6

SSSP 5 servers:



	test1	test2	test3	avg	sdev
loading time	13.955	17.243	17.345	16.181	1.92844704
iteration time	89.067	96.154	76.89	87.3703333	9.74343021
job_running_time	103.022	113.397	94.235	103.551333	9.59196051

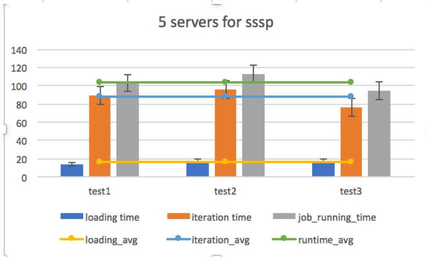
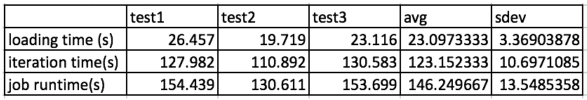


figure 7

SSSP 7 servers:



	test1	test2	test3	avg	sdev
loading time (s)	26.457	19.719	23.116	23.0973333	3.36903878
iteration time(s)	127.982	110.892	130.583	123.152333	10.6971085
job runtime(s)	154.439	130.611	153.699	146.249667	13.5485358

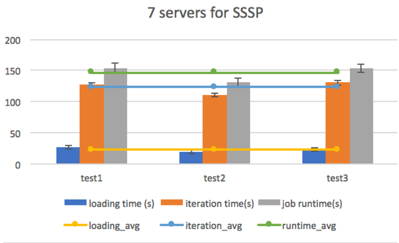


figure 8

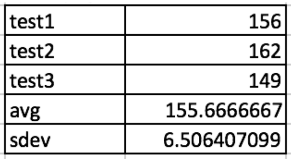
From the plot above, when servers number is given, the execution time will increase with step number. The reason is that our system is not a converge system. In other words, when the superstep is achieved the setting number, the system will force to stop and all workers will return the results to the 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:

see figure 2

Spark pagerank performance:



test1	156
test2	162
test3	149
avg	155.6666667
sdev	6.506407099

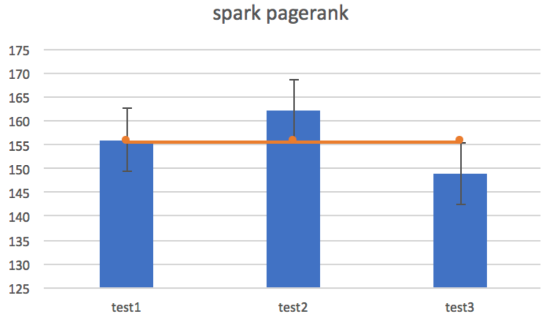
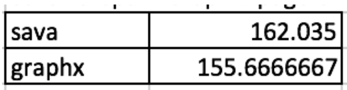


figure 9

sava vs graphx: pagerank



sava	162.035
graphx	155.6666667

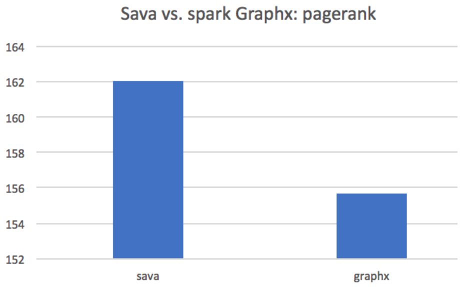
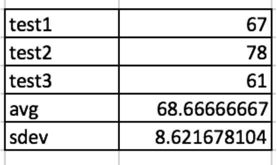


figure 11

Sava SSSP performance:

see figure 8

Spark SSSP performance:



test1	67
test2	78
test3	61
avg	68.66666667
sdev	8.621678104

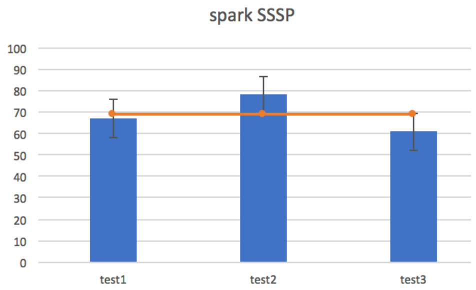
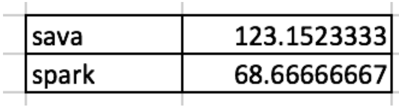


figure 10

sava vs. graphx sssp



sava	123.1523333
spark	68.66666667

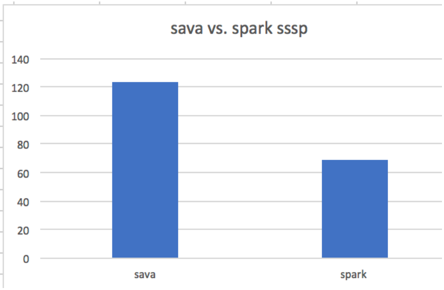


figure 12

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.