CS425 MP4 Report

Group 16: Jintao Jiang(jjiang43), Hao Yang (haoy4)

1. Design

- Vertex:

Think as a vertex! We first implemented an abstract vertex class supporting all related APIs and then expect user's application class to inherit it and provide a compute() function. Each vertex holds a list of its neighbors, its value and two message queues. If a vertex detects its neighbor as a vertex in other machine, it will send a message via TCP. Otherwise it will push message directly into message queue of its neighbor. At each iteration, a vertex only reads from the queue of last iteration and leaving a queue collecting message for this iteration.

- Worker:

Before doing the work, worker needs to parse the graph sent from the master. When working, each iteration will be divided into two stage: preparing stage and running stage. Preparing stage is for dealing with message queues and at running stage a worker will iterate through its vertices list and call compute() function. After each iteration it will sent a message to master indicating the finish of work and wait. At the end it writes logs to the local file system.

- Master:

Master needs to partition the graph and send the partition to each workers. The rule of partition is node_id % num_workers. Master also needs to synchronize workers. Master will only tell workers to start next iteration until all workers finish the current iteration.

- Fault Tolerant:

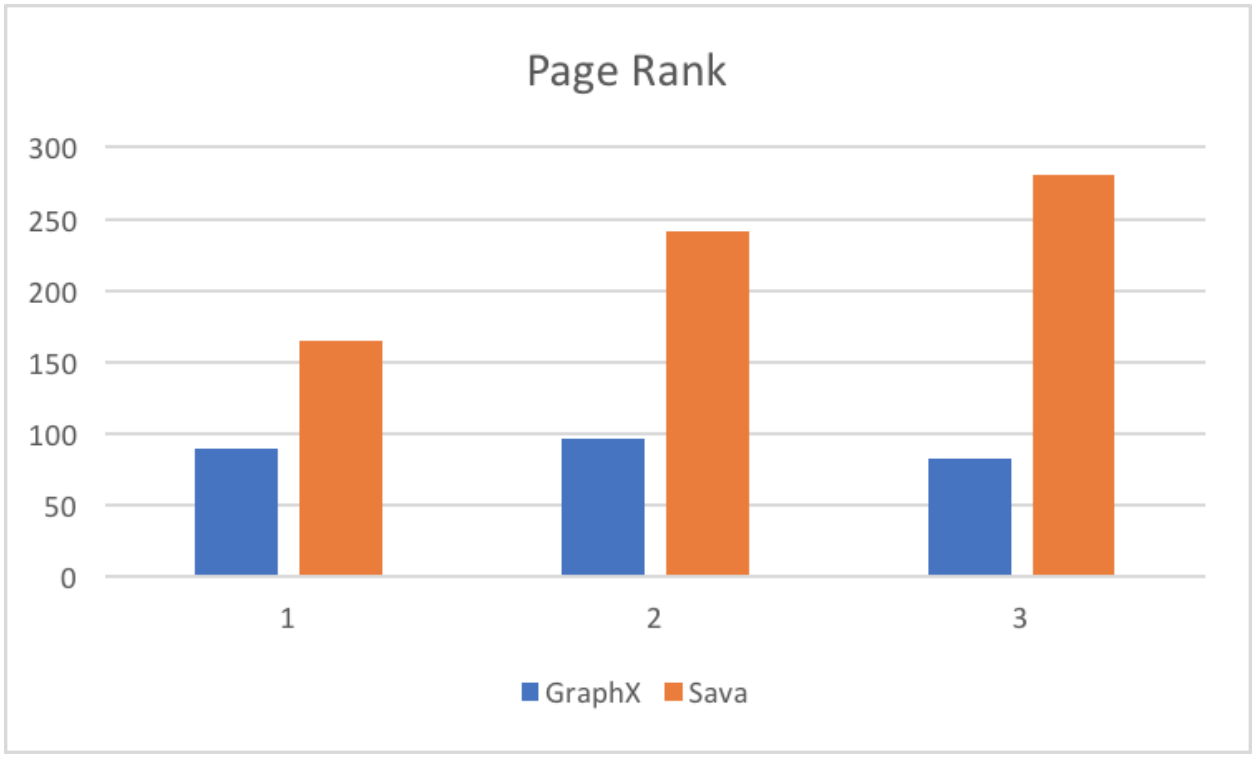
There are two cases. If master fails, then the standby master will take over the work. If workers fail, then master will repartition the graph and restart the task.

2. Experiments

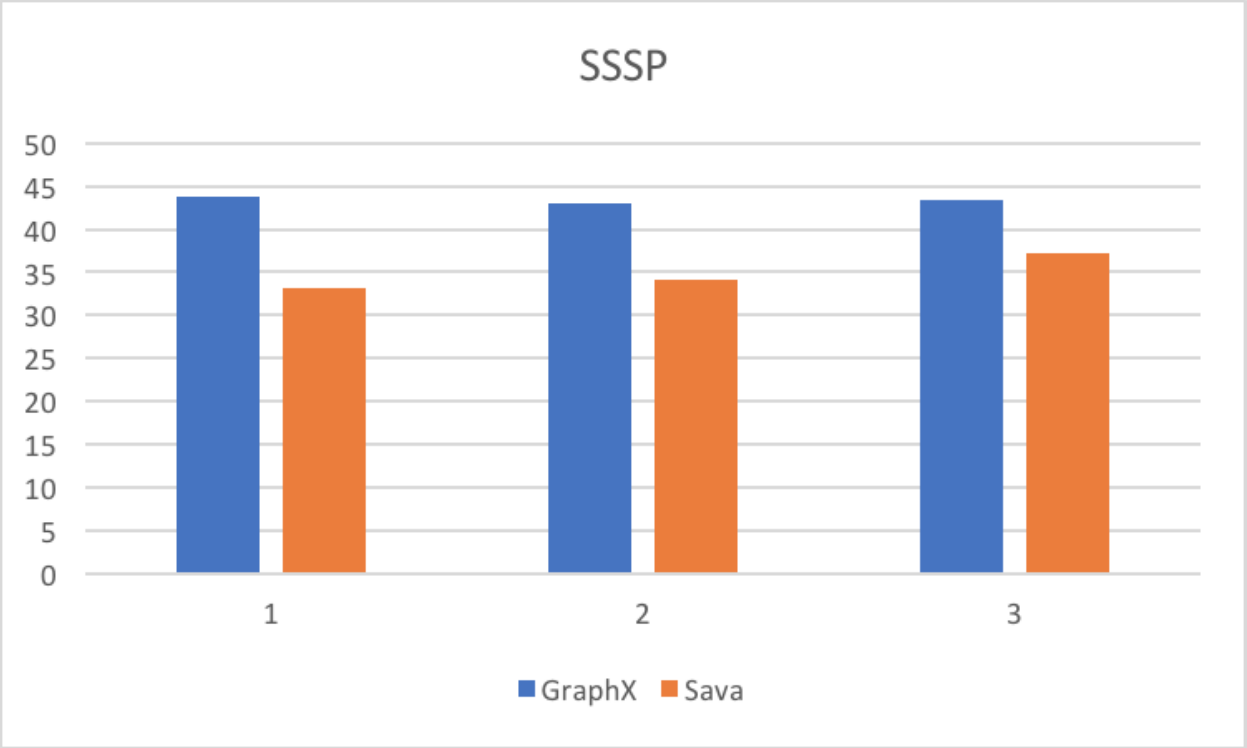
- Results (in seconds):

GraphX	PageRank	num of worker	7	7	7	5	5	5	3	3	3
		Reading	18	25	16	18	24	18	16	15	19
		Running	66	93	51	73	78	65	59	51	79
		Sum	84	118	67	91	102	73	78	69	101
		Avg			89.6666667			96.5			82.6666667
Sava	PageRank	num of worker	7	7	7	5	5	5	3	3	3
		Reading	2.4	2.5	2.4	2.6	2.5	2.4	2.4	2.5	2.4
		Running	160	149	172	232	240	238	288	267	278
		Sum	162.4	151.5	181.4	234.6	242.5	245.4	290.4	269.5	283.4
					165.1			240.833333			281.1
GraphX	SSSP	num of worker	7	7	7	5	5	5	3	3	3
		Reading	16	14	18	15	16	18	16	15	19
		Running	25	30	28	29	26	28	31	26	23
		Sum	41	44	46	44	42	46	47	41	42
		Avg			43.6666667			43			43.3333333
Sava	SSSP	num of worker	7	7	7	5	5	5	3	3	3
		Reading	2.8	2.3	2.6	2.4	2.7	2.4	2.3	2.2	2.1
		Running	31	29	32	32	33	30	35	34	36
		Sum	33.8	31.3	34.6	34.4	35.7	32.4	37.3	36.2	38.1
					33.2333333			34.1666667			37.2

- Comparison of PageRank:



- Comparison of SSSP:



3. Discussion

Comparing to Sava, GraphX has a more consistent performance in general. In Page Rank application, with the decreasing of workers, the total running time is almost the same, which is about 90 seconds. At the same time, with the decreasing of workers, Sava's performance is decreasing, too. The graph suggests that the tendency is linear.

In the experiment of SSSP, Sava and GraphX both have a consistent performance. We believe that was caused by the less number of calculating. In the Page Rank application, the number of communication between nodes during each iteration is greater than SSSP application. Because in SSSP we start from a source, and only update the neighbors with sended messages.

Although Sava is winning in the SSSP application a little bit, we believe it was because the size of Spark jar files are huge. And to run GraphX, the big size of jar files which is about 300mb need to be send to all the workers from the master. And that is the reason why GraphX need about 20 seconds for reading.