CS425 MP4 Report

# Overall Design

#### **Algorithm**

We used Golang and introduced MP2 and MP3 as libraries to provide failure detector and distributed file system. Dataset input and final result are stored in SDFS of MP3. To build clean API, we used interface{} type in Golang.

Partitioning of graph is fully distributed and not dependent on master. For each vertex, we hash the vertexID and mod 10 to get the vmID it should be allocated. If the vm is not a worker, add 1 to vertexID and hash again.

1. Master
   1. Superstep synchronization: Each worker need to send ACK to master(and standby master) after it finishes its computation. Master sends command to start next superstep.
   2. Worker failure: When a worker fails, master sends out Start command to restart the job.
   3. Standby master: Standby master keeps track of ACK from workers and stepcount. When master fails, standby master continues listening on workers and sends out command.
2. Worker
   1. Communication with master: worker listens on a specific port for master. On Start command from master, worker runs the graph partitioning algorithm and stores the vertex-vm relationship locally, then run the first superstep. After finishing each superstep, worker sends ACK to master and waits for Run command for next superstep.
   2. Communication between workers: In each superstep, worker iterates all its active vertices and executes Compute() function. We use combiner to cache all outgoing messages to same vm and use mutex to provide mutual exclusion for combiner. After itearating all vertices, send out messages in combiner. We also use mutex to protect vertex status information and other critical section.
3. Client

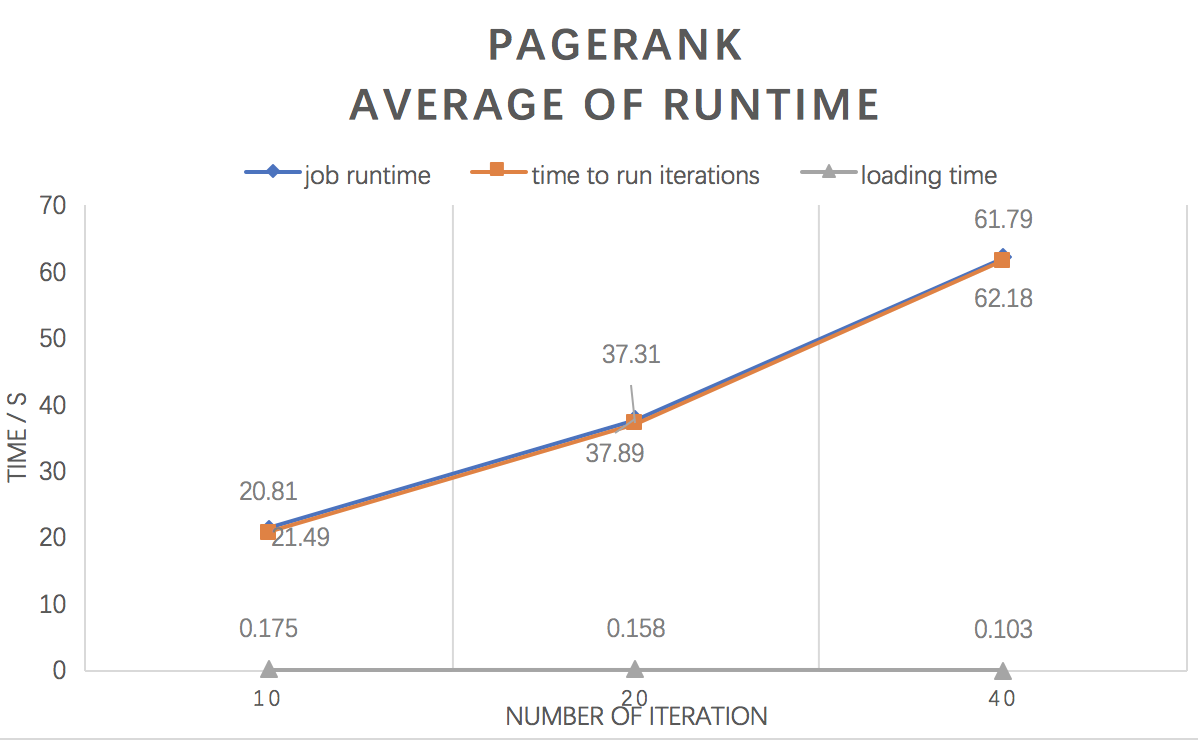
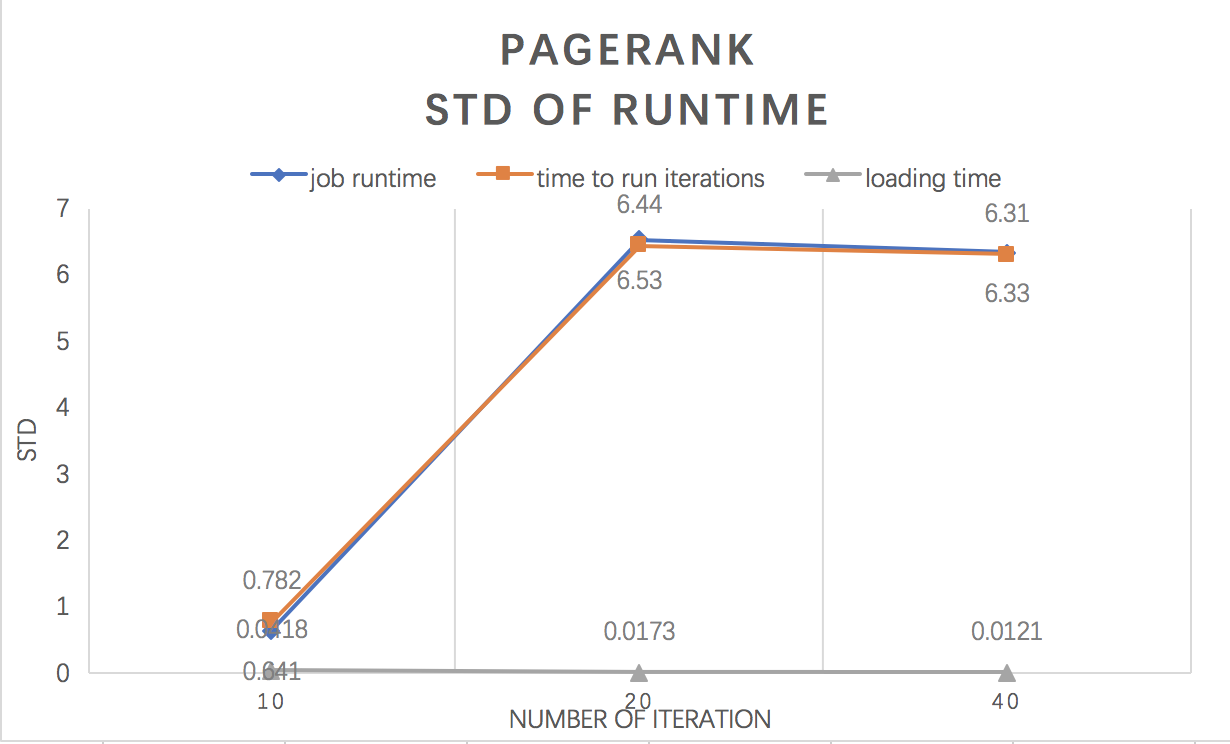
Client sends job name and dataset to server through protocol buffer, then waits for result.

1. API

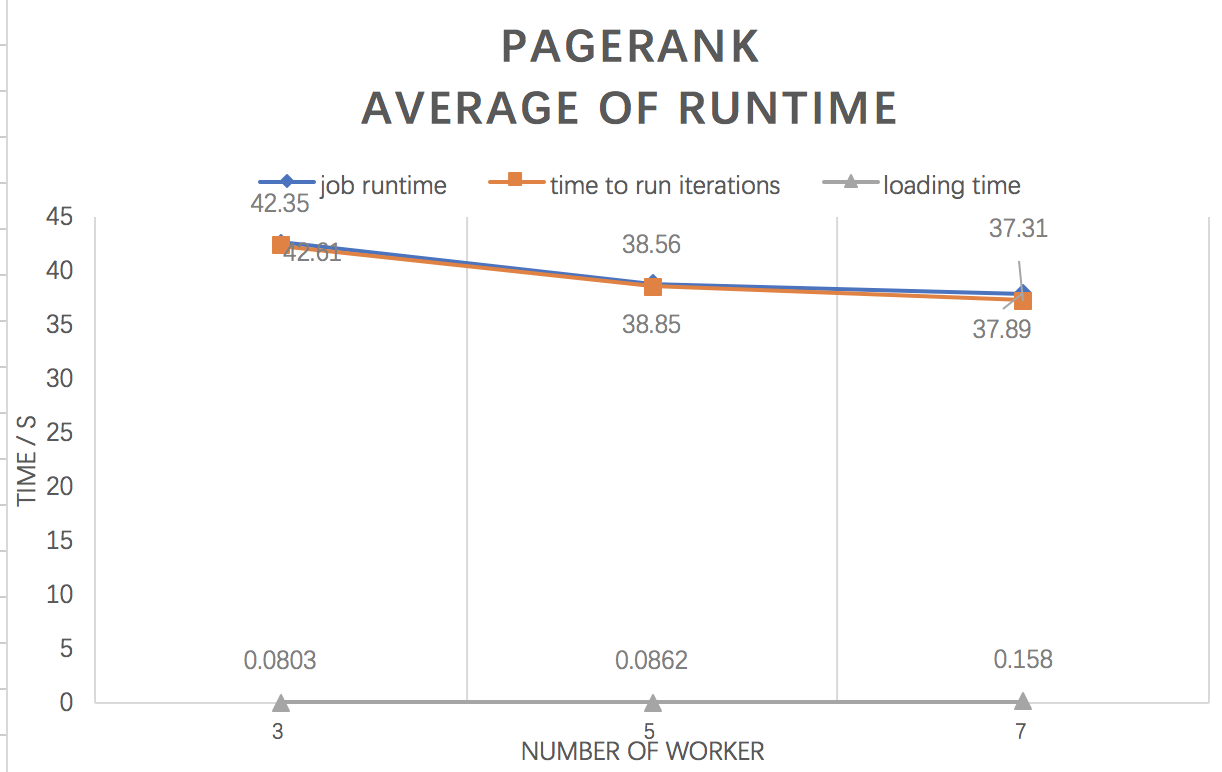
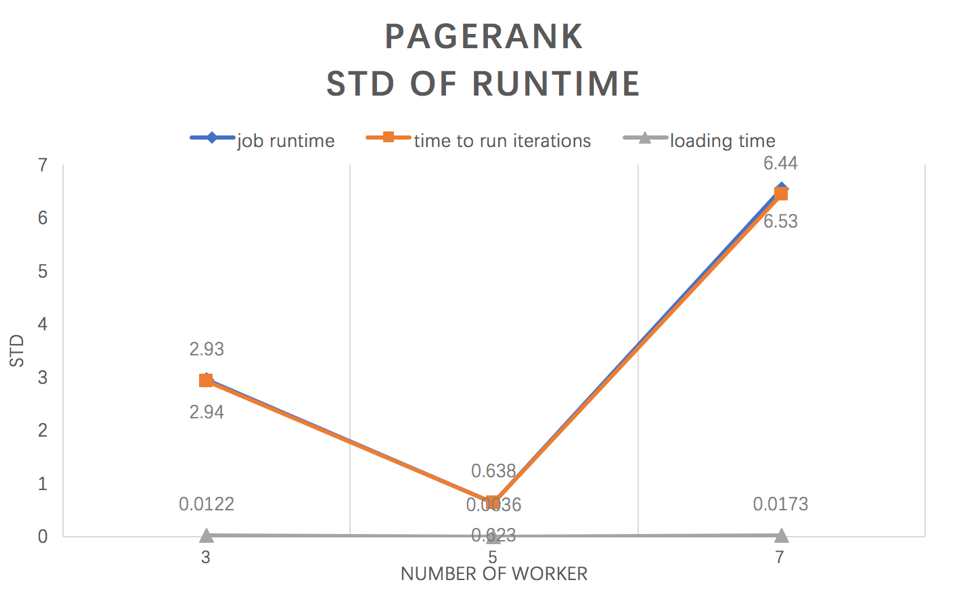
Define Vertex struct as an interface type in Golang to provide API. Each application simply need to define actual data types of variables and implement interface function in Vertex.

# Sava Performance

#### **App 1: PageRank : Graph with 334563 vertices and 925872 edges**

1. Different number of tasks(iterations)

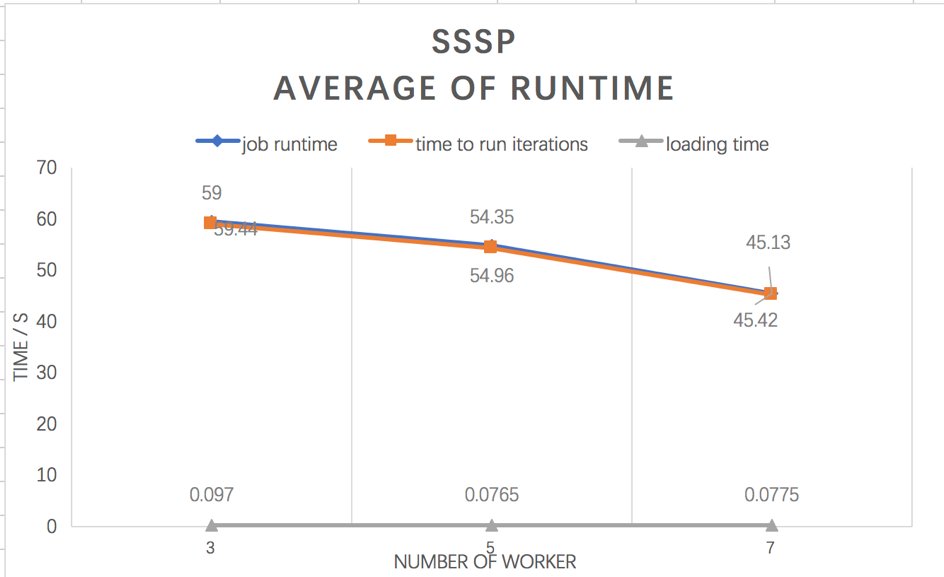
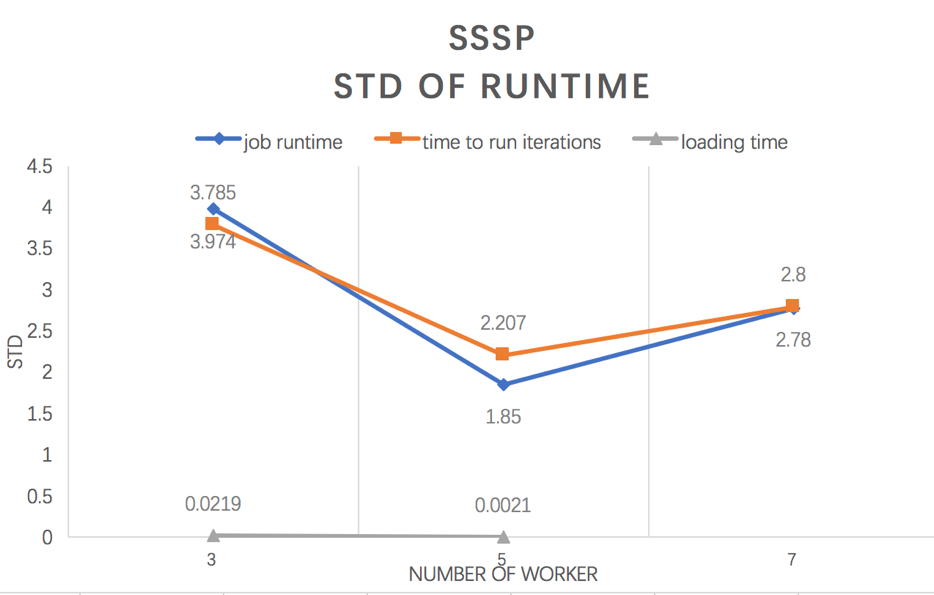
* Analysis

1. The job runtime and time to run iterations are almost the same. This means the loading time and others are quiet small.
2. With increased number of iterations, the job runtime and time to run iterations increased. Loading time remains nearly unchanged, because the input size is the same.
3. As for Std, the std of loading time remains nearly unchanged. Std of job runtime and time to run iterations may be quiet different, this is because the network congestion is unpredictable.
4. Different number of workers

* Analysis

1. From the average figure, we can find out that with increased number of worker, the job runtime and time to run iteration both slightly decrease. This is the amount of calculation work will be evenly distributed to workers. So each worker has smaller amount of work.
2. The std of runtime is quiet different and seems to have no relationship with number of workers We think it is more related to network situation.

#### **App 2: Single-Source Shortest Path: Graph with 334563 vertices and 925872 edges**

1. Different number of workers

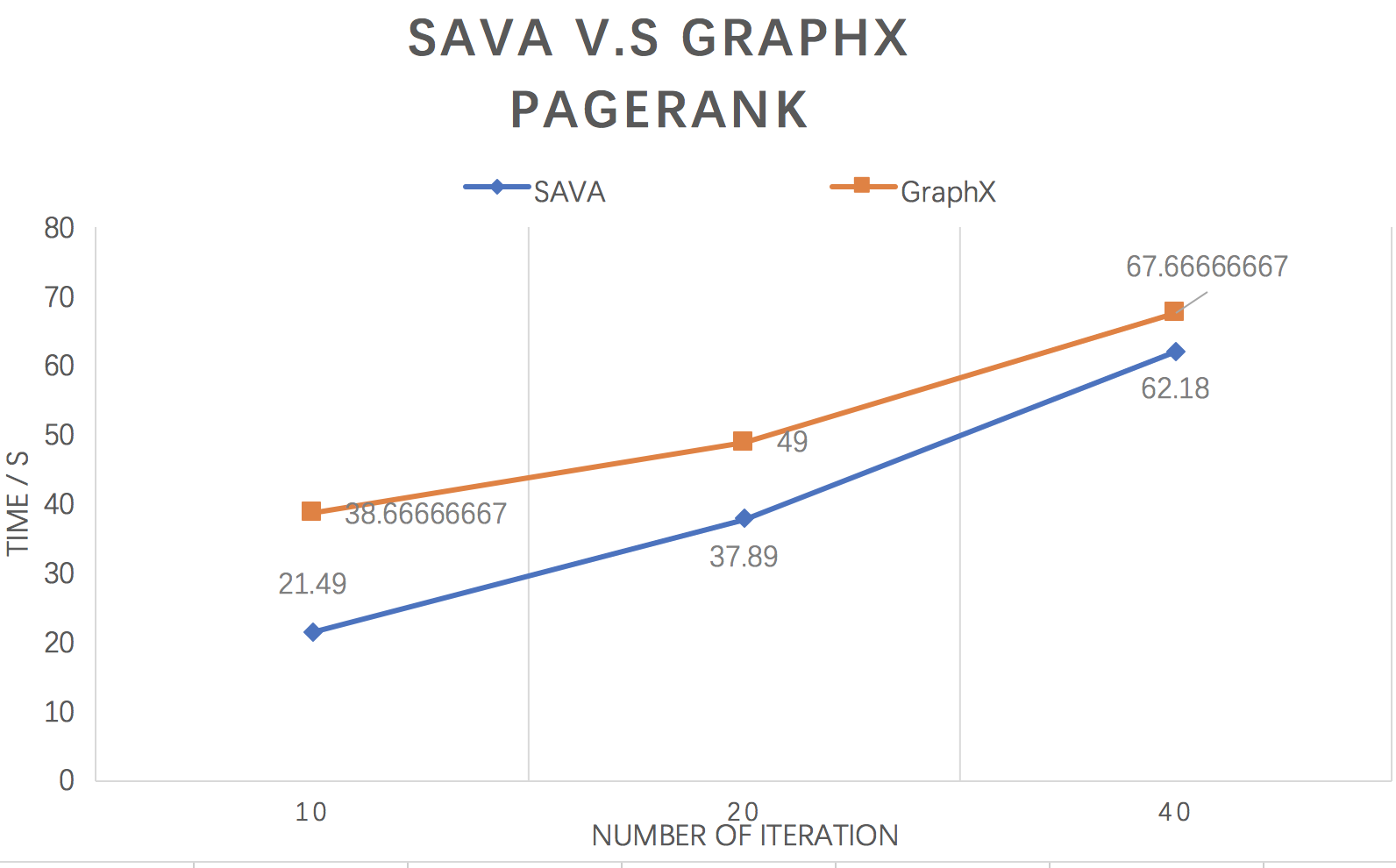
* Analysis

1. From the average figure, we can find out that the job runtime and time to run iterations still decrease with increased number of worker. And the loading time still very small and nearly unchanged with same size of input
2. We think the std of data is related to network.

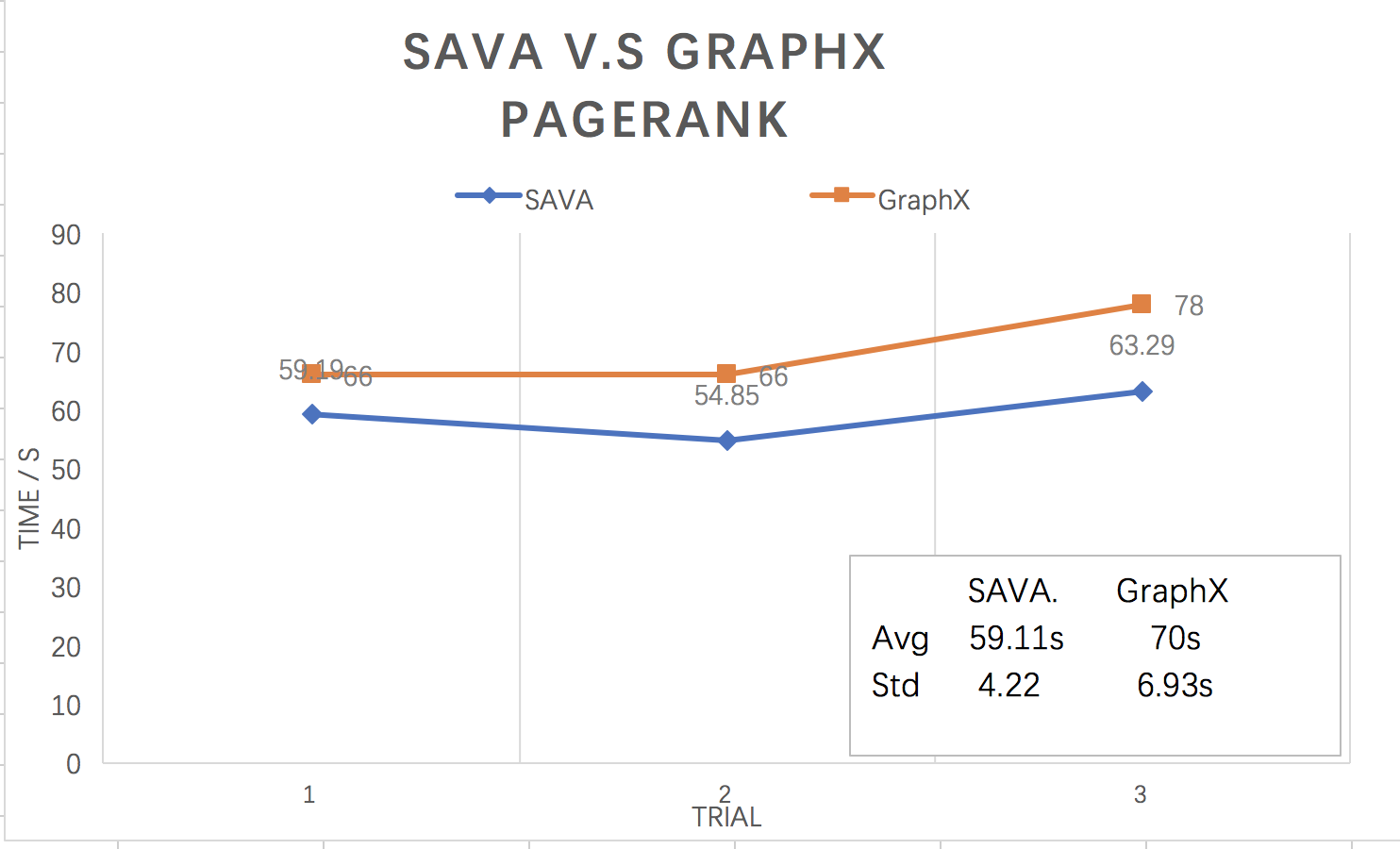
# Sava performance V.S. GraphX

#### **PageRank**

#### We run PageRank using graph with 334563 vertices and 925872 edges for 10 iterations, 20 iterations and 40 iterations. We run at least three times to get the data point. We only compare job runtime for both platform because we can get the loading time and computation time for GraphX.



#### **Single-Source Shortest Path**



1. Analysis

Our platform SAVA can definitely beat GraphX, no matter what the number of iteration is.

* 1. As we can see in both platform running PageRank, the computation time is increased with increasing number of iteration. This is obvious because each iteration takes similar amount of time.
  2. We assume that GraphX is slower based on three following reasons.
* GraphX provides a more general API and can process more general graph and problem. We guess it uses a more complex protocol so that it takes more time to transmit messages with bigger overhead.
* It may use a more complex superstep synchronization algorithm, because it need to scale to more machines.
* It uses Java, we use Golang. Golang is usually considered to be faster than Java. However, Golang is not suitable to implement API compared to Java.