MP4 REPORT

DESIGN

Architecture

- **File system**:
  - The SAVA system is built on top of the SDFS system implemented in MP3.
- **Membership**:
  - There is only one active master during execution. However, there is another inactive master as the standby master. The standby master takes charge when it detects the failure of the active master.
  - All the other members in this cluster are workers.
- **Components**:
  - **Client**: User can use our programming framework(API) provided below to write their own vertex-centric application. It should be implemented in a single C++ file, and client upload it to master. After compiling this cpp file with other necessary components, master generates a single binary executor "runner". Executor will then be distributed to each server.
  - **Executor**: In SAVA system, the executor will handle the main execution of each vertex in each iterations. The worker use TCP connection to communicate with each executor that resides on the same server. When a new SAVA task comes, the executor will be called by worker and initialized with the splitted dataset from master.
  - **Worker**: The worker is the bridge between master and its executor. The worker uses TCP connection to communicate with master to sync the superstep, and communicate with the executor to keep track of the execution progress in each iteration.
  - **Master**: The master is the center of this task. It will first partition the graph dataset using self-defined strategy, and distributes these partitions to each worker. It will also sync all the workers so they can follow the superstep strategy in Pregel model. It listens to the user command and report the progress back to the user.
- **Graph partition**:
  - In our experiment, we found out that using random partition for our SAVA will be sufficient to beat the GraphX in runtime.

Programming Framework

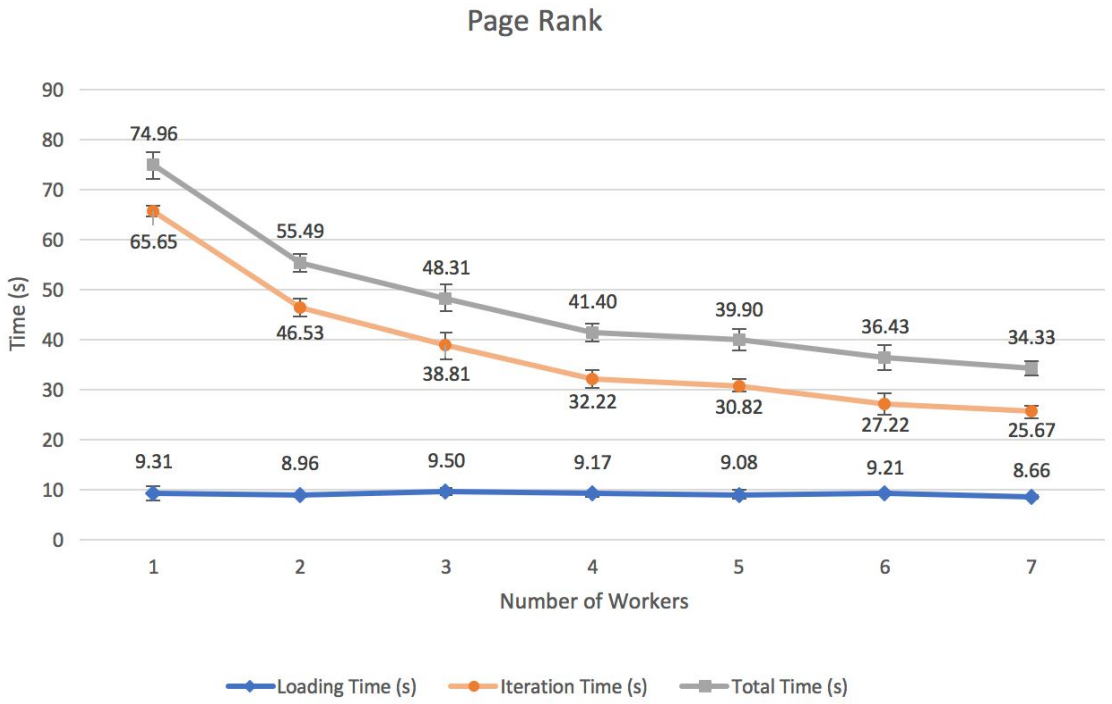
- User will only need to implement the Compute() function for each vertex in the graph dataset.
- Predefined Vertex APIs:
  a. Value(): return value of this vertex / VertexID(): return id of this vertex
  b. SendMessageTo(int id, double val): send message with value val to vertex id
  c. GetMessages(): return all the input messages related to this vertex
  d. GetOutEdges(): return all the out-edges of this vertex
- Client API:
  a. Format: python mp4/sava.py app-name path-to-app-code remote-graph-filename result-set(TOP-K: max k vals, TOP+K: min k vals) combinator(SUM/MIN/MAX)
  b. "python mp4/sava.py PageRank mp4/apps/PageRank.cpp rinput TOP-25 SUM"

MEASUREMENTS (Based on 3 trials each & Using SNAP dataset - 1.08G)

1. Sava Performance detail

Page Rank

Num of Workers	1	2	3	4	5	6	7
Loading Time (s)	9.31	8.96	9.50	9.17	9.08	9.21	8.66
Stdev	1.59	0.43	0.72	0.40	0.96	0.36	0.36
Iteration Time (s)	65.65	46.53	38.81	32.22	30.82	27.22	25.67
Stdev	1.16	1.74	2.57	1.68	1.29	2.19	1.22
Total Time (s)	74.96	55.49	48.31	41.40	39.90	36.43	34.33
Stdev	2.69	1.84	2.62	1.84	2.19	2.40	1.39

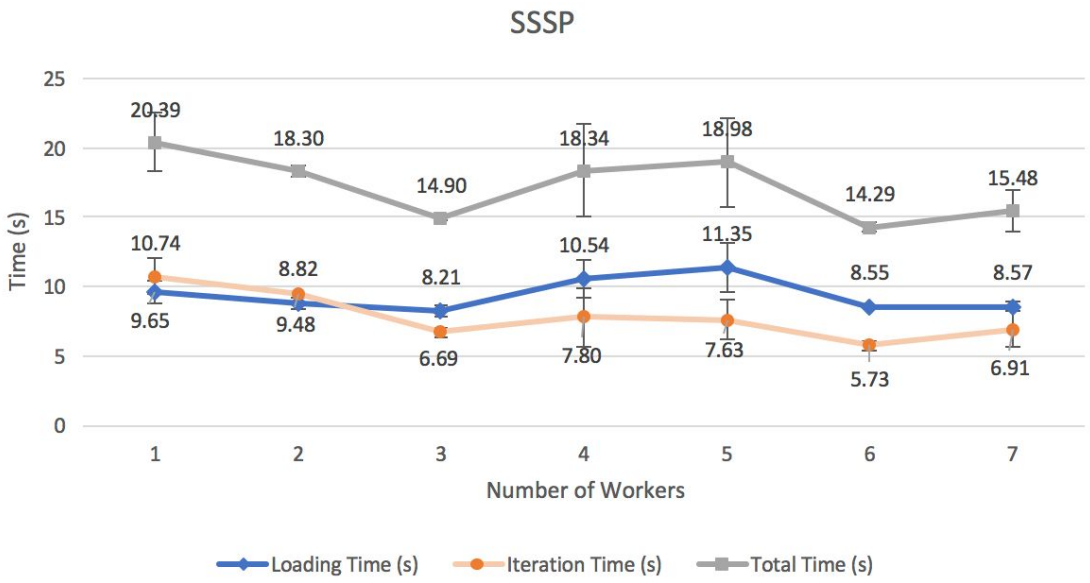


As we can see from the plot, the Loading Time of Pagerank barely varies with the change of number of workers, while the Iteration Time reduces when number of workers increases. The former is because the main time consuming part of load time is read the dataset from SDFS and partition it to several sub-graphs, which is only related to the master. However, the more workers SAVA has, the less workload each worker will have and thus it decreases the runtime of each iteration. The decrease in time is less obvious in SSSP application, because the computation of SSSP is relatively low, and the active vertices in each iteration is much less than PageRank.

SSSP

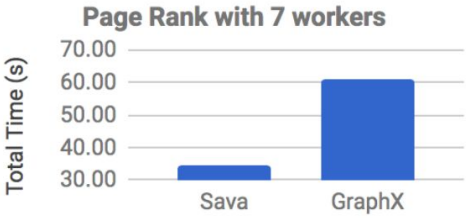
Num of Workers	1	2	3	4	5	6	7
Loading Time (s)	9.65	8.82	8.21	10.54	11.35	8.55	8.57
Stdev	0.81	0.38	0.39	1.32	1.74	0.06	0.37
Iteration Time (s)	10.74	9.48	6.69	7.80	7.63	5.73	6.91
Stdev	1.33	0.05	0.31	2.08	1.47	0.31	1.29

Total Time (s)	20.39	18.30	14.90	18.34	18.98	14.29	15.48
Stdev	2.10	0.37	0.19	3.33	3.20	0.30	1.46



2. Comparison

Page Rank	Sava	GraphX
Total Time	34.33	61.08
Stdev	1.39	2.33



We can see the sava is about 170% faster than GraphX.

SSSP	Sava	GraphX
Total Time	15.48	62.36
Stdev	1.46	3.39

Sava is about 400% faster than GraphX.

The reason why our Sava is faster than GraphX may be: (1) Sava is implemented in C++, while GraphX is in Scala. C++ is faster than scala in general(if algorithms are the same). (2) Sava is designed to address specific problems, while GraphX is designed to handle various and complex situations, not limited to Pregel computation model. (3) Because our dataset and is relatively small (compared to industrial ones), so the starting time is a big part of the total time in GraphX. If using a bigger dataset, our data structures may need to be redesigned to support larger dataset, and in that case GraphX's performance would be better.