DS 256 - Scalable systems in Data Science
Assignment - 1

Sheshadri K. R.
Dream Lab (CDS)
06-02-02-10-12-18-1-16336

March 15, 2019

1 Experimental Setup

System Environment of Turing Node:

- **Java Version :** 1.8.0_112 (Oracle Corporation)
- **Spark Version :** 2.1.1.2.6.1.0-129
- **Scala Version :** 2.11.8
- **Giraph Jar:** 1.3.0-SNAPSHOT
- **CPU:** Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40GHz
- **OS architecture :** amd64
- **Number of Processing Cores :** 12
- **Total installed RAM :** 47GB

HDFS Environment:

- **Block size :** 128MB
- **Replication Factor :** 2

Software Setup

- **Java version**: 1.8.0.144
- **JSON library org.json** , version 20180813

Data description

- **soc-LiveJournal1.txt**
- **Total Vertices:** 4847571
- **Total Edges:** 68993773
- **Type:** Directed

- **cit-Patents.txt**
- **Total Vertices:** 3774768
- **Total Edges:** 16518948
- **Type:** Directed

- **orkut.txt**
- **Total Vertices:** 3072441
- **Total Edges:** 117185083

- **Type:** Undirected

2 Algorithm 1 : PageRank.java

The following formula has been used to compute PageRank:

$$PR(A) = (1 - d) + d(PR(T1)/C(T1) + ... + PR(Tn)/C(Tn)) \quad (1)$$

In the spark implementation, the pagerank of a vertex is propagated to the neighbors, by creating an adjacency list. The pagerank computed is passed to each vertex in the adjacency list, if there is any change in the computation of the page rank.

In order to converge, the pagerank computed between 2 iterations shouldn't exceed a epsilon value. To see that the code doesn't get into too many iterations, a check of iterations is also placed. The code terminates either if the number of iterations expire or there is a convergence of page rank by all vertices, whichever happens first is considered.

In Giraph implementation, the input is converted into the vertex input json format, this is a pre-processing step which is done before the execution of the actual algorithm.

The dampening factor for the algorithm is set to 0.85.

The pagerank code was ran for **cit-Patents** ,**soc-Livj**.

The X axis indicates the number of iterations and the y-axis indicates the time taken in minutes.

For the orkut graph, Spark took **13.5 mins** and **20.5 mins** for 5 iterations and 10 iterations respectively for execution of PageRank.

The spark job was submitted with the following configuration:

- Number of executors = 4 and 6
- Number of cores = 2
- Memory = 8GB

The giraph job was submitted with the following configuration:

- Number of executors = 4 and 6
- Number of cores = 4
- Memory = 8GB

In Fig 1-2 , the number of workers are kept to 4, and 6 from Fig 3-4, with 8GB of memory per each worker. We can see that Giraph performs better than Spark in overall time of execution of pagerank for citpatents and soclivj.

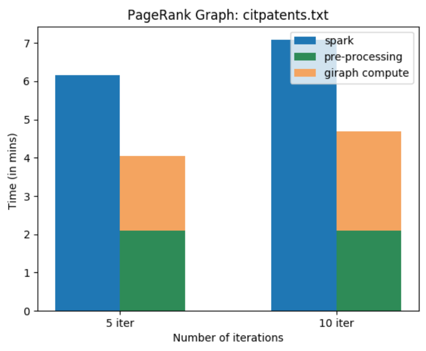

Figure 1: Cit-patents, 4 Workers , 8GB per Workers, Spark vs Giraph

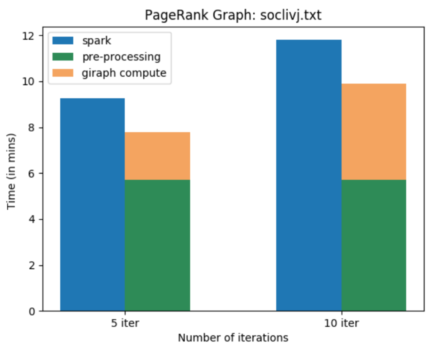

Figure 2: soc-livj, 4 Workers, 8GB per Workers, Spark vs Giraph

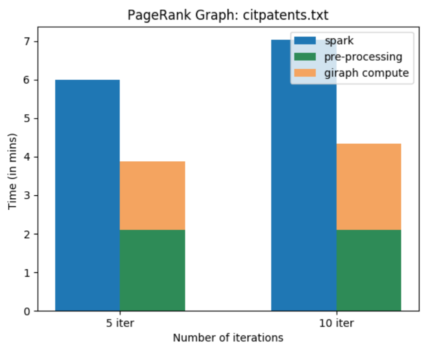

Figure 3: Cit-patents, 6 Workers , 8GB per worker, Spark vs Giraph

2.1 Strong Scaling experiments for Pagerank

A strong scaling experiment was performed for both the graphs, by increasing the workers from 4 to 6. Keeping the total memory per worker to be 8 GB. Fig (5,6) show PageRank execution for Spark on citpatents and soclivj. Fig (7,8) show PageRank execution for Giraph on citpatents and soclivj

The x-axis indicates the number of iterations and the y-axis indicates the time taken in minutes.

Expectation: Spark and Giraph was not expected to strongly scale.

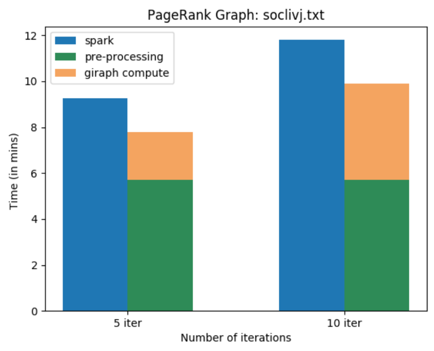

Figure 4: soc-livj, 6 Workers , 8GB per worker, Spark vs Giraph

Observation: When the number of workers were increased, the spark didn't scale, this is due to shuffling activity between more worker nodes. Spark showed an increased execution time with increase in the number of executors (workers). Even Giraph didn't strongly scale with the increase in worker nodes.

The conclusion is that just by increasing more machines, we cannot speed up the execution of the algorithm.

3 Algorithm 2 : Weakly Connected Components

In the spark implementation of the Weakly Connected components, an adjacency list was constructed and a maximum id in the adjacency list is passed as a message to all the neighbors of the considered vertex.

All the vertices which are present in the same connected component will eventually have the same maximum vertexId. In such a condition, where all the vertices in the graph has the maximum vertex the algorithm would converge and terminate.

The spark job for Weakly Connected Components, on the cit-Patents graph took 1hrs, 12mins. (not shown in the plot)

The spark job was submitted with the following configuration:

- Number of executors = 4
- Number of cores = 2
- Memory = 8GB

The giraph job was submitted with the following configuration:

- Number of executors = 4
- Number of cores = 4
- Memory = 8GB

The Giraph Job for Weakly Connected Component was ran for citpatents and soclivj.

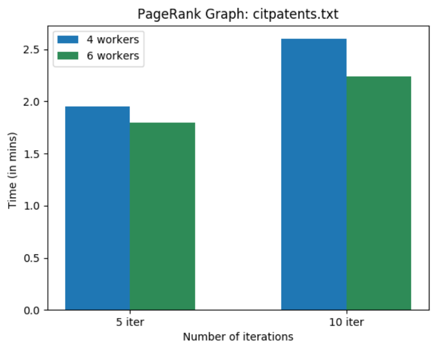

Figure 5: Cit-patents, Spark PageRank, 4 vs 6 workers

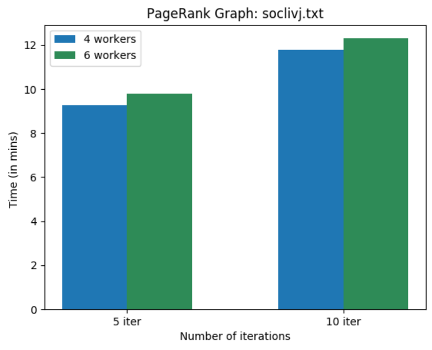

Figure 6: soc-livj, Spark PageRank, 4 vs 6 workers

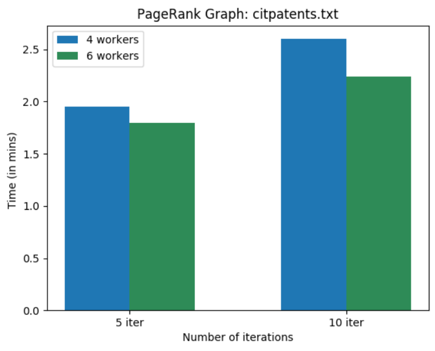

Figure 7: Cit-patents, Giraph PageRank, 4 vs 6 workers

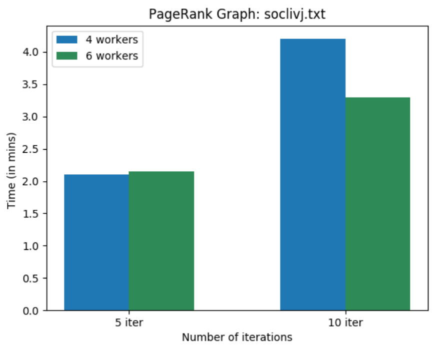

Figure 8: soc-livj, Giraph PageRank, 4 vs 6 workers

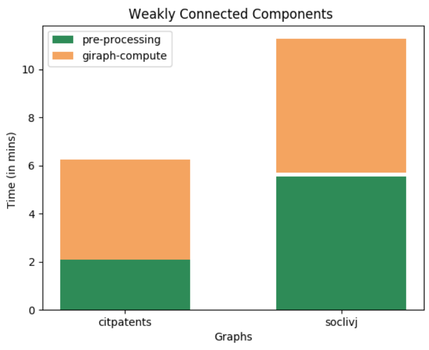

Figure 9: Giraph Weakly Connected , citpatents, soclivj

Fig 9 shows a plot of execution time of weakly connected component in giraph for citpatents and soclivj, X-axis indicates the graph, Y-axis indicates the time taken to execute Weakly connected component code on it.

A strongly scaling experiment was performed on giraph for weakly connected components for the soc-livj.txt graph. In Figure 10, the X-axis represents the number of workers, Y-axis represents time, when the number of workers were increased from 4 to 6. The giraph job completed faster by 2mins. Only in this case was the memory alloted was 12GB.

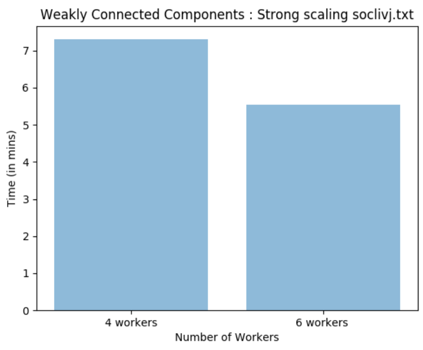

Figure 10: soc-livj, 4 vs 6 Workers , 12GB per worker, Giraph

4 Acknowledgements

I would like to thank Swapnil for quick responses and the giraph primer to help kick-start the assignment.