Study Report - Exploring Graph processing with GraphX and Giraph

Sachin Badgujar

Department of Computer Science, University of North Carolina Charlotte

ABSTRACT

Processing extremely large graphs has been and remain a challenge, however, a forward move in Big Data technologies has made the task more practical. Social networks like Facebook and Twitter naturally follow the graph representation with their data. Application areas like network routing, website modeling or the study of disease spreading can be modeled as graph problems. In the E-commerce domain, recommendation systems and prediction systems for online shops also operate on high volume and high velocity [1]. To process the graph structured data, there exist numerous different frameworks. Here, I compare and explore two such platforms, Apache GraphX and Apache Giraph. Apache GraphX is a new component in spark for graphs and graph-parallel computation. At a high level, GraphX extends the Spark RDD by introducing a new graph abstraction. In addition, GraphX optimizes the representation of vertex and edge types when they are primitive data types as int and double, reducing the in-memory footprint by storing them in specialized arrays [2]. On the other hand, Apache Giraph is an iterative graph processing system built for high scalability. Apache Giraph is based on the Bulk Synchronous Parallel paradigm popularized by Google's Pregel project [3].

This report will analyze the GraphX and Giraph engines and will perform qualitative and quantitative comparison. This report is focused on measuring the relative performance and ability of the two systems to handle large graphs. In addition, the focus is on the usability of the system and how easy it is to program the algorithm on these two systems. To achieve this, I take simple graph algorithms with identical graph data in terms of nodes and vertices. I run the algorithms on both the systems to observe the performance, scalability, and usability.

1. INTRODUCTION

Analysing graph data is a critical capability for today's data scientist. Ability to compute graph queries such as reachability, shortest paths, and measuring connected components, as well as more sophisticated analytics for training machine learning models is required for analyzing and extracting knowledge from graphs. The graph algorithms are iterative in nature, iterating over the graph until reaching a threshold or fixed point [5]. Billion-node graphs that exceed the memory capacity of commodity machines are not well supported by popular Big Data. Tools like MapReduce, which are notoriously poor performing for iterative graph algorithms such as PageRank. In response, a new type of framework challenges one to "think like a vertex" (TLAV) and implements user defined programs from the perspective of a vertex rather than a graph. Such an approach improves locality, demonstrates linear scalability, and provides a natural way to express and compute many iterative graph algorithms. These frameworks are simple to program and widely applicable but, like an operating system, are composed of several intricate, interdependent components, of which a thorough understanding is necessary to elicit top performance at scale. Here, the two frameworks considered are Apache Giraph and Apache GraphX. Both are BSP-style graph processing system. Apache Giraph is

an open source implementation an open Pregel project. This is built on top of the Hadoop and inherits the scaling capabilities and its compatibility with the Hadoop-Eco system. Also, this enable the use of Hadoop input/output components like Flume, Kafka and hive.

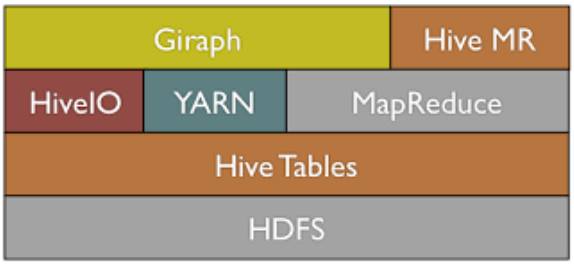

Figure 1: Giraph in Hadoop Eco-System [4]

GraphX is a distributed graph computation framework that unifies graph-parallel and data-parallel computation. Similarly, GraphX is built on top of Apache Spark which requires a cluster manager – native standalone spark cluster or Hadoop YARN, and requires a distributed file system like HDFS or Cassandra. I will be using both the framework with HDFS file system.

2. MODEL AND WORKING

Giraph implements BSP model on Hadoop and GraphX is developed on the small platform, which as you know, emphasizes on interactive in memory computations.

2.1 Giraph Model

In this model of distributed graph processing, data resides on edges and vertices, and each vertex repeatedly executes the same program. Thus, each vertex stores the result of its evaluation locally. A vertex program executes in three phases: i) receive messages from neighbour vertices, ii) compute and possibly produce new values for the vertex and outgoing edges, and iii) create messages for neighbour vertices. A Giraph program terminates when a user-specified termination condition occurs or there are no new messages to deliver. [5] A line from input file look like – A,10, [[B,2], source implementation of Google's [F,5]]. It is Vertex-centric BSP where each vertex has an id(A), a value (10), a list of its adjacent vertex ids (B and F) and the corresponding edge values (2 and 5). Each vertex is invoked in each superstep, can re-compute its value and send messages to other vertices, which are delivered over superstep barriers. User programs the master such that the workers are assigned a part of the input(parallel). Master instructs each worker to perform a superstep. In each superstep, the communication and computation cycles are repeated. After computation stops, master instructs worker to save its portion of the graph.

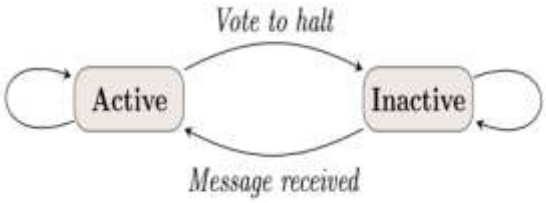

Figure 2: State machine of Vertex [4]

What are supersteps? It is Sequence of iterations and during each iteration the framework invokes a user defined function for each vertex call. This happens in parallel on the distributed system with the help of Master and worker. The function specifies behaviour at a single vertex and a single superstep. Algorithm terminates when no vertices change in a superstep.

2.1.1 Apart from Pregel

Additionally, Giraph adapts to the flexible inputs based on Vertex or Edge. Like in MapReduce I can write a custom input and output format. Giraph needs more resource sharing than MapReduce and to overcome this it parallelizes the computations using local multi-threading to take advantage of additional CPU. This reduces the TCP overhead significantly. To access the shared member variable a new concept of WorkerContext has been introduced. Memory optimization is done by serializing the edges of every vertex and messages

into byte array rather than native Java objects. The message combiner is optimized to reduce memory usage. The methods preSuperstep() and postSuperstep() are added to Giraph framework.

2.2 GraphX

GraphX extends the Spark RDD by introducing a new Graph abstraction: a directed multigraph with properties attached to each vertex and edge. GraphX API combines the advantages of data-parallel systems and graph-parallel systems to make powerful graph processing engine.

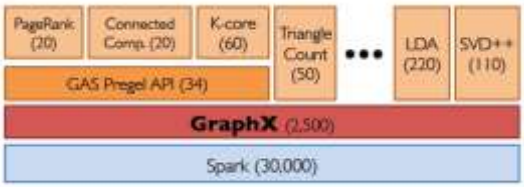

Figure 3: GraphX is a thin layer on top of the Spark general-purpose dataflow framework (lines of code) [8].

The GraphX system is motivated by the realization that (i) graphs can be encoded efficiently as tables of edges and vertices with some simple auxiliary indexing data structures, and (ii) graph computations can be cast as a sequence of relational operators including joins and aggregations on these tables. The basic representation of graph data and a core part of the GraphX data model is a combination of graph structure and properties form a property graph G(P) = (V; E; P). The GraphX data model consists of immutable collections and property graphs. The immutability constraint simplifies the abstraction and enables data reuse and fault tolerance [6].

In addition to the standard data-parallel operators (e.g., map, reduce, filter, join, etc.), GraphX introduces a small set of graph-parallel operators including subgraph and mrTriplets, which transform graphs through a highly parallel edge-centric API [6]. GraphX consist of data-parallel operators like filter, map, and reduceByKey. In addition, it consists of graph-parallel operators like Graph, vertices, edges, mapV, and many more. GraphX optimizes the representation of vertex and edge types when they are primitive data types (e.g., int, double, etc.) reducing the in-memory footprint by storing them in specialized arrays. From Figure 4, GraphX rather than splitting the graphs along edges, it partitions the graph along vertices which can reduce both the communication and storage overhead. Logically, this corresponds to assigning edges to machines and allowing vertices to span multiple machines [7].

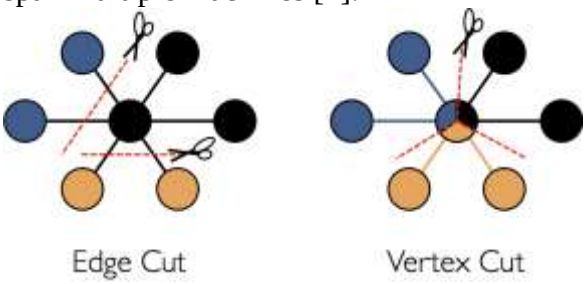

Figure 4: GraphX vertex based partition for optimization.

GraphX utilizes the power of RDDs, Computation DAGs, In-memory caching and Lineage based fault tolerance.

3. ENVIRONMENT

For this study report, I have both the systems installed on the Ubuntu 14.04 LTS operating system. The Hadoop with version 0.20.2 is deployed in Pseudo-distributed mode. For Giraph version 1.1.0 and Spark version 2.1.0 pre-build with user-provided Hadoop is used. The system configurations are – Intel core i5-5200U 2.20Ghz processor, 8GB RAM, NVIDIA GeForce 940M with 2GB of memory.

4. EXECUTION

Both systems, Giraph and GraphX, undergo same algorithm with same input file.

4.1 INPUT

The input file consists of Facebook friends graph. The dataset is taken from ‘Stanford

Network Analysis Project (SNAP)' [9]. The sample data has 4039 Nodes and 88234 Edges. The data has 2 columns. The first column describes the node number and second column describes the friend node.

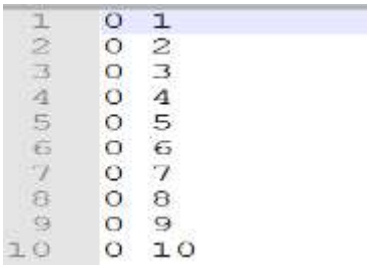

Figure 5: Data format for Facebook Data sample.

Here, node #0 is friend with node#1, node#2 and much more.

For Giraph, the input is reformatted as an adjacency list for every node. This input is consumed by the Giraph program as 'JsonLongDoubleFloatDoubleVertexInput Format'

4.2 ALGORITHM

The algorithm used for evaluation is Page Rank algorithm. PageRank is a way of measuring the importance of website pages [10]. The rank of a page is calculated by below formula-

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

where

1. PR(A) is the PageRank of page A,
2. PR(Ti) is the PageRank of pages Ti which link to page A,
3. d is a damping factor which can be set between 0 and 1.

For Giraph, we have taken a damping factor as 0.85. Whereas, for GraphX, we are using inbuilt PageRank algorithm by GraphX library.

4.3 RESULTS AND STATISTICS

The evaluation is done in a similar environment consist of Hadoop 0.20.2. The code for Giraph is as shown below.

Figure 6: Giraph code for page rank with max iteration as 30.

After running the 50 iterations (compute steps) following page rank has been produced.

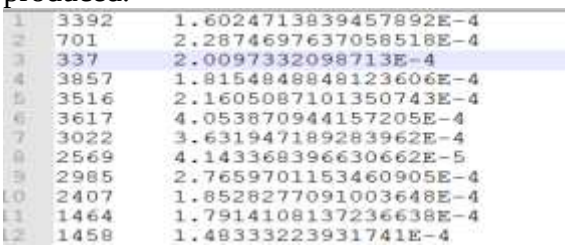

Figure 7: column 1 represents the node and column 2 represents the page rank of that node.

In GraphX, the input is given in a raw format as shown in figure 5. The program uses an inbuilt algorithm for page rank. The code snippet is shown below.

```
import org.apache.spark.graphx.GraphLoader
val fpath = "/face_in.txt"
val graph = GraphLoader.edgeListFile(sc, fpath)
val ranks = graph.pageRank(0.001).vertices
```

Figure 8: GraphX page rank code sinppet.

The algorithm iterates until the change in page rank become less than 0.001. The output given by the program is as below

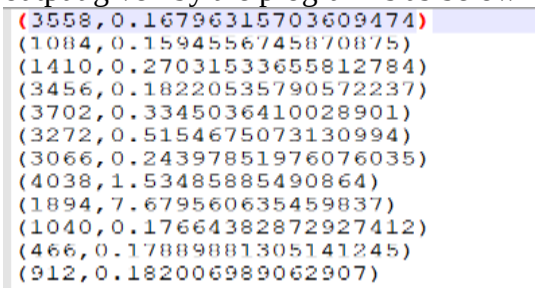

Figure 9: Page Rank produce by GraphX

5. DIFFERENCE

Although both the systems are made for processing large graphs, these systems vary in terms of models, optimization, ease of deployment, ease of programming, processing power and many other.

5.1 MODEL

The graph-parallel systems like Pregel, GraphLab, and Giraph can efficiently execute iterative graph algorithms orders of magnitude faster than the general data-parallel systems like MapReduce. However, the same restrictions that enable the performance gains also make it difficult to express many of the important stages in a typical graph analytics pipeline: constructing the graph, modifying its structure, or expressing computation that spans multiple graphs. Consequently, existing graph analytics pipelines compose graph-parallel and data-parallel systems using external storage systems, leading to extensive data movement and complicated programming model. [6] Basically, an operation which needs to move the data outside the graph topology or which require more global view are not feasible using graph-parallel systems. In contrast, data-parallel systems like MapReduce and Spark are well suited for these tasks as they place minimal constraints on data movement and operate at a more global view. By exploiting data parallelism, these systems are highly scalable. Furthermore, for Giraph, the input source is restricted to Hadoop input-output services as the system uses Hadoop platform. Whereas GraphX can be standalone, can use HDFS or non-Hadoop systems like Cassandra. The difference in partitioning strategies: GraphX uses vertex cuts while Giraph uses edge cuts. Vertex cuts split high-degree vertices across partitions, but incur some overhead due to the joins and aggregation needed to coordinate vertex properties across partitions containing adjacent edges.

5.2 INTERFACE

GraphX raised a few interesting points about the programming interface and ease of development. In this respect, GraphX inherits several features from the Spark ecosystem that make it easy to develop graph applications. GraphX can be integrated with Hive to use SQL-like query. This makes it easy for column transformation and preprocessing of the graph. However, Giraph does not support such transformations. It expects input to be in specific format and representations like GiraphTextInputFormat, JasonLongDoubl-eFloatDoubleVertexInputFormat, TextEdgeInputFormat, etc. The user needs to take extra programming efforts to convert input format. Interactive environment of sparks facilitates checking of output while developing and testing an algorithm.

5.3 PERFORMANCE

This section elaborates the difference in the performance of the two systems for a given environment and input dataset. The results can vary with respect to change in configuration and algorithms. It is majorly affected by the type of data consumed by the programs.

For Giraph, loading a job is slow process as compared to GraphX. Moreover, because of high computations initialization and first superstep takes approximately the highest time.

Giraph Stats	Aggregate bytes loaded from local disks (out-of-core)	0	0	0
	Sent message bytes	0	0	0
	Aggregate bytes stored to local disks (out-of-core)	0	0	0
	Current workers	1	0	1
	Last checkpointed superstep	0	0	0
	Aggregate sent messages	4,411,700	0	4,411,700
	Aggregate finished vertices	4,039	0	4,039
	Aggregate vertices	4,039	0	4,039
	Aggregate edges	176,468	0	176,468
	Superstep	26	0	26
	Aggregate sent message bytes	70,590,950	0	70,590,950
	Current master task partition	0	0	0
	Sent messages	0	0	0
	Lowest percentage of graph in memory so far (out-of-core)	100	0	100

Figure 10: Giraph program statistics for PageRank Algorithm

The total time taken by the program is 11330ms.
For GraphX, the total time taken to compute the page rank was 5900ms

Figure 11: GraphX job details snapshot.

The algorithm converges when the delta in page rank gets less than 0.001.

5.4 DEPLOYMENT

The GraphX clearly wins in the deployment convenience. GraphX need not to reside on Hadoop and can run as a standalone application. The input files to the program can reside in the local file, HDFS or at a remote location.
The Giraph has many constraints for the deployment. The major one is a Hadoop version. Giraph needs a Hadoop with version 0.20.X and memory configurations if you are running it on single node cluster. Also, it needs configuration for the worker nodes and IPs of the worker to communicate. The Cloudera manager alleviates the installation process to some extent.

6. CONCLUSION

The conclusion highly depends on the input, environment, and algorithm used. This experiment clearly shows that GraphX is quick in producing the page ranks for the given input. However, when data is non-numeric and has more columns, GraphX performance gets unstable. For this experiment, Giraph deployment and installation was a timely process. Giraph is not convenient when it comes to the input formats. The setup is similar map reduce jobs where the user needs to provide the custom class for the unusual input format. But, Giraph has common input format classes already defined in the framework. The advantage is the programming language used. Giraph programs are written in Java in contrast to the Scala programs in GraphX.
Hence, if users have medium scale, computationally heavy problem GraphX is a suitable system. If users have a very huge graph and already has a compatible Hadoop installation, then Giraph is the way to go.

7. REFERENCES

[1] *"Evaluating the Scaling of Graph–Algorithms for Big Data using GraphX"* 2016 2nd International Conference on Open and Big Data
[2] GraphX Programming Guide
[3] Giraph.apache.org
[4] Avery Ching, Sergey Edunov, Maja Kabiljo, "One Trillion Edges: Graph Processing at Facebook-Scale"
[5] Walaa Eldin Moustafa, Ken Yocum, "Datalography: Scaling datalog graph analytics on graph processing systems"
[6] Reynold S. Xin Daniel Crankshaw Ankur Dave, "GraphX: Unifying Data-Parallel and Graph-Parallel Analytics"
[7] Start with GraphX - http://spark.apache.org/docs/latest/graphx-programming-guide.html
[8] Joseph E. Gonzalez*, Reynold S. Xin GraphX: Graph Processing in a Distributed Dataflow Framework
[9] SNAP- https://snap.stanford.edu/data/egonets-Facebook.html
[10] Page Lawrence and Brin Sergey and Motwani,Terry (1999) '*The PageRank Citation Ranking: Bringing Order to the Web.'*