CSP-595 Big Data Technologies

Project Final Paper

Spark-GraphX

-Sivasenthil Namachivayan

-A20391478

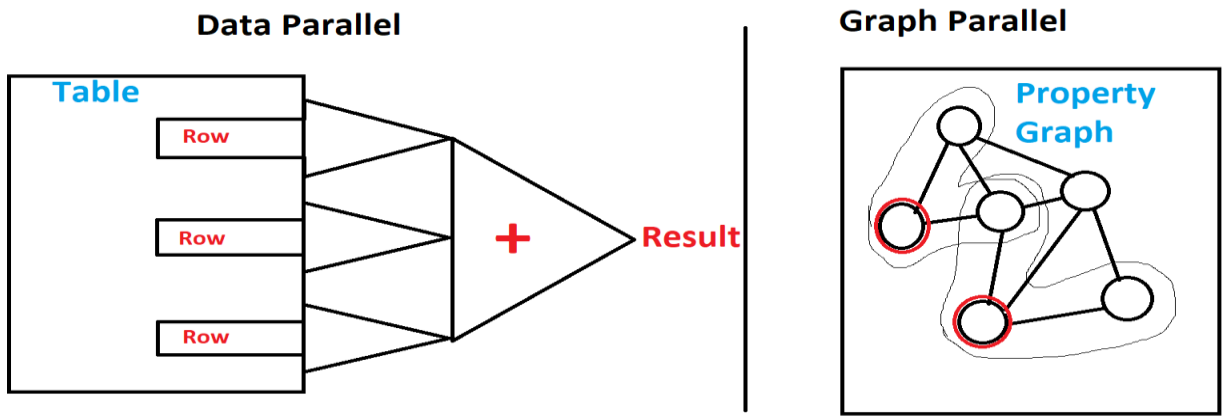
Spark GraphX

Big Data is changing the way the world sees the data in this present generation. With the various technologies which are in the boom to make use of the data which got accumulated over the years, this Apache Spark stands out with the way it analysis the Big Data to give different perception for the users in approaching the data. When ever we see the raw data, it is difficult to understand as well as scrutinize the needed information. Graphs make it easier for understanding and solve this issue. In big data, Spark bring optimizing algorithms for data mining, Machine Learning etc..,. Spark has sophisticated algorithms with tricky optimization and run code on distributed clusters to solve various problems like fault tolerance and parallel computations. As discussed earlier, the graphical representation of the data for social networks, advertising and so on are central to machine learning and data mining where directly applying existing data parallel tools to graph computation tasks are ineffective and burdening. To solve this problem and to have a scalable and intuitive tool for graphical computation, new graph parallel systems are designed to execute graph algorithms. This new system also couldn't help in removing the challenges in graph construction, transformations and fault tolerance. An innovative technology which is a hybrid of the above two methods named Spark- GraphX is introduced. This contains the advantages of data parallel and graph parallel systems to solve the existing issues and provide graph computation within the spark data parallel framework.

In this work, the effective method of distributing graphs as tabular data structures as innovative ideas in distributed graph representation. So, to over-come the disadvantages in the above-mentioned methods, we implement the PowerGraph and Pregel abstractions in minimal lines of code. Therefore, through Spark GraphX, the need for loading, transforming and computation of the big data is done using graphs.

Introduction

There are various distributed graph parallel frameworks which are introduced to cater the needs of graphical data representation for large sets of data. Few examples of the distributed graph parallel frameworks are Pregel, Giraph, GPS, GraphLab, PowerGraph etc.,. In each of these distributed graph parallel frameworks, a new programming abstraction is introduced which allows the users to describe the graph algorithms such as vertex Centrix model, Scatter- Gather, Gather-Sum-Apply-Scatter(GAS), Sub-Graph Centric, Filter- Process, PageRank, Belief Propagation. These frameworks also have their respective run time engine on which it executes these algorithms in a clustered and distributed environment. These frameworks give a sophisticated way of approaching the real-world problems in a pictorial representation to seek solutions and rectify the current shortcomings. These different methods have different view of graphical computation which is specific to certain domains or graph algorithms and applications. This difference is due to the diverse needs of the domain and various requirements that occurs at the time of runtime specific to the algorithms used.



Examples for data parallel: Hadoop, Spark. Examples of Graph Parallel: Pregel, GraphLab

One of the main setbacks are that these frameworks fail to address the challenges of the data ETL preprocessing and construction, interpreting and applying results of computations. There are few frameworks which are built to support the interactive graph processing and computation which is done by the same frameworks instead of relying on any other plugins or software. The other major challenges for the distributed graph mining algorithms are computation parallelization, data partitioning and communication management. The graph applications are diverse and expose a variety of data access and communication patterns. Each abstraction is optimized for certain classed of graph applications as said earlier. For example, Popular vertex centric model is suitable for iterative value propagation algorithms. The neighborhood- centric model is suitable for support allocations on custom subgraphs, like ego networks. From this it is evident that, it is difficult to find a single model to support all the classes of graph applications. Also, there exists no qualitative comparison of the graph programming abstractions.

But, the concern of scalable data processing and graph construction(ETL) are addressed by the data parallel systems like MapReduce and Spark. Spark address the range of fault tolerant strategies and enable interactive data processing as well. The use of complex joins large data sets are becoming challenges, but this will not exploit the graph structures. The basic performance metric like measuring the speed, resource utilization and scalability to answer the question of which graph parallel processing framework is better suited for which application and datasets. The three widely used graph algorithms for benchmarking on the targeted computing systems are clustering coefficient, shortest path length and PageRank score. The run time increases with more computing nodes which is a degrading factor with respect to the scalability of the application. But the graph parallel computing framework like PowerGraph consistently exhibits better performance than the others mentioned above.

After careful evaluation of all these diverse developments and results, we address these challenges through GraphX, which is a graph computation system running on spark data parallel framework. GraphX extends Spark's Resilient Distributed Dataset (RDD) abstraction to introduce

the Resilient Distributed Graph (RDG). This RDG associates the records with the vertices and edges in a graph and provides a collection of expressive computational primitives. With these computational primitives we use PowerGraph (One of the consistently performing framework) and Pregel (one among the most used graph processing frameworks). Along with this, we also provide the new operations like view, filter and transform graphs which substantially simply the process of graph ETL and analysis. The GraphX RDG leverages advances in distributed graph representation and exploits the graph structure to minimize communication and storage overhead.

Resilient Distributed Datasets (RDD)

RDD is a fundamental data structure of Spark. It is an immutable distributed collection of objects. The dataset is divided into logical partitions which can be computed in different nodes of the distributed cluster in the RDD. The RDDs can contain any type of objects as in python, java or Scala or it can also contain the user defined classes. RDD can be created through deterministic operations on either data on stable storage or other RDDs. RDD is a read-only partitioned collection of records. The operations in RDD can be executed in parallel in the distributed systems.

There are two ways in creating the RDDs.

1. Parallelizing an existing collection in your driver program
2. Referencing a dataset in an external storage system (Shared file system, HBase, HDFS)

Spark uses this concept of RDD in-order to achieve better and optimal performances in MapReduce operations. Spark maintains the lineage of RDD and recovers the lost partitions by recomputing them from base data.

Data Sharing using Spark RDD

Data Sharing is slow in MapReduce due to replication, serialization and Disk IO. Most of the runtime is been wasted by the Hadoop applications on doing HDFS read write operations. This can be considered to more than 90% of their time. To resolve this problem, the idea of using the Resilient Distributed Datasets, was introduced which supports in-memory processing computation. The needed information is stored in the objects and these objects will be shared across the jobs. This type od data sharing within the memory is 10 to 100 times faster than the network and disk sharing. This also has the restriction, which depends upon the memory size.

Below is the comparison between the Iterative operation on Map reduce and on Spark RDD

Figure 1: Iterative operation on Map Reduce

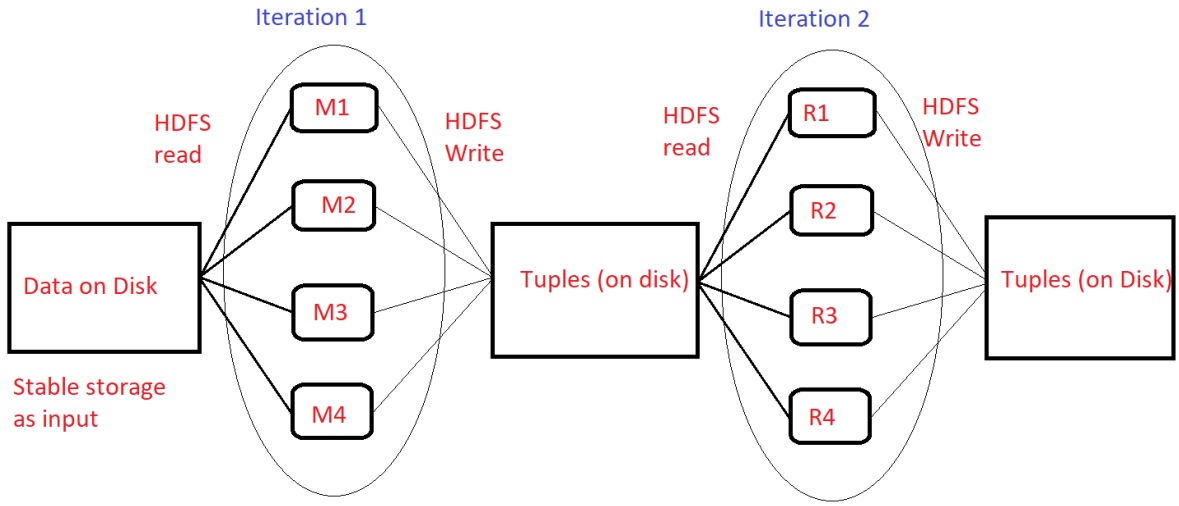


Figure 1 shows how the mapreduce framework works for the iterative operations. This will cause overheads due to data replication, disk I/O and serialization. This will affect the system performance and makes it slow.

Figure 2: Iterative operation on Spark RDD

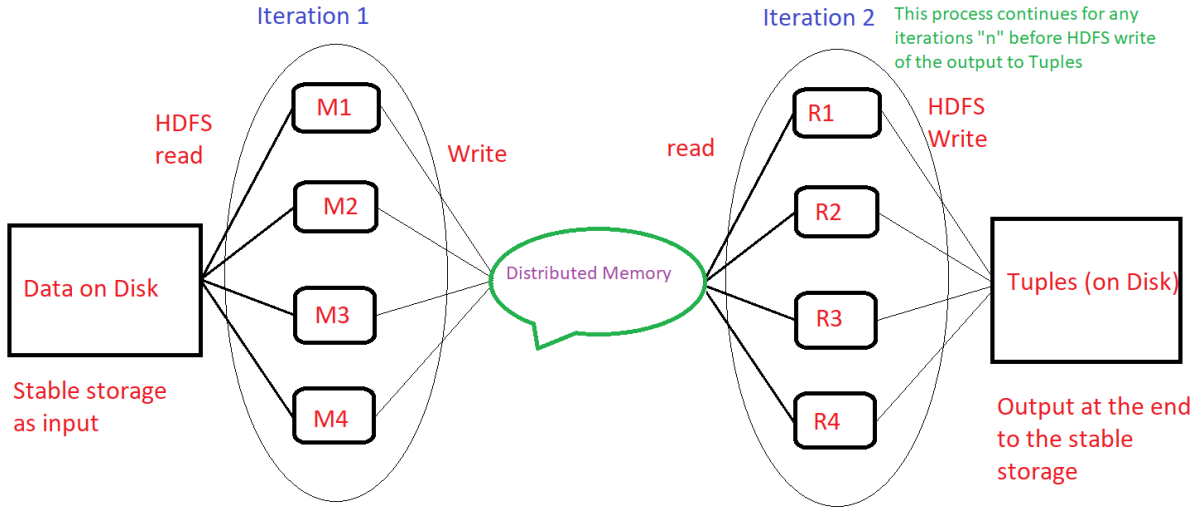


Figure 2 shows the iterative operation on Spark RDD where the intermediate results in a distributed memory instead of stable storage (Disk). This will improve the system in performance and execute the operations faster. As mentioned earlier, if the distribute memory is out of memory to store the intermediate results of the first iteration then it will be stored in the disk.

Figure 3: Interactive operation on MapReduce

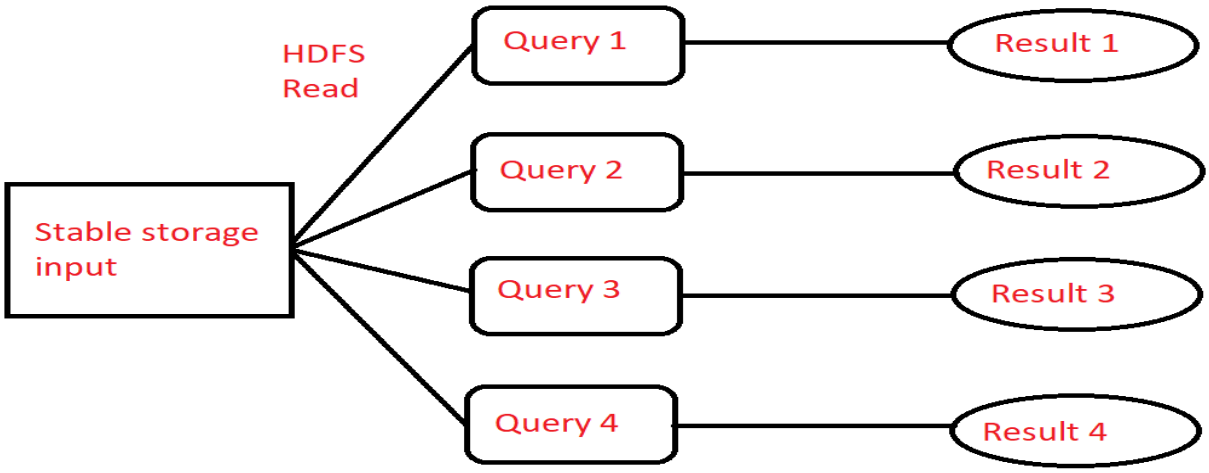


Figure 3 shows the interactive operations on MapReduce where the user executes the ad-hoc queries on the same subset of data. In this type of execution, the queires will do the disk I/O in the stable storage which causes the application to consume more execution time. This is a huge setback for the performance.

Figure 4: Interactive operation on Spark RDD

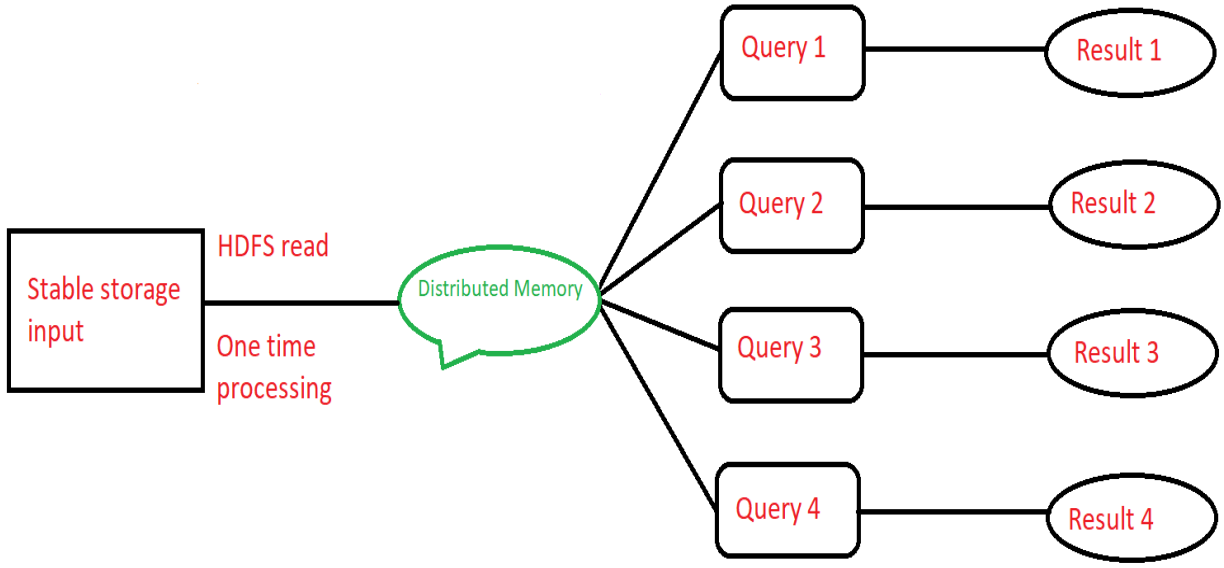


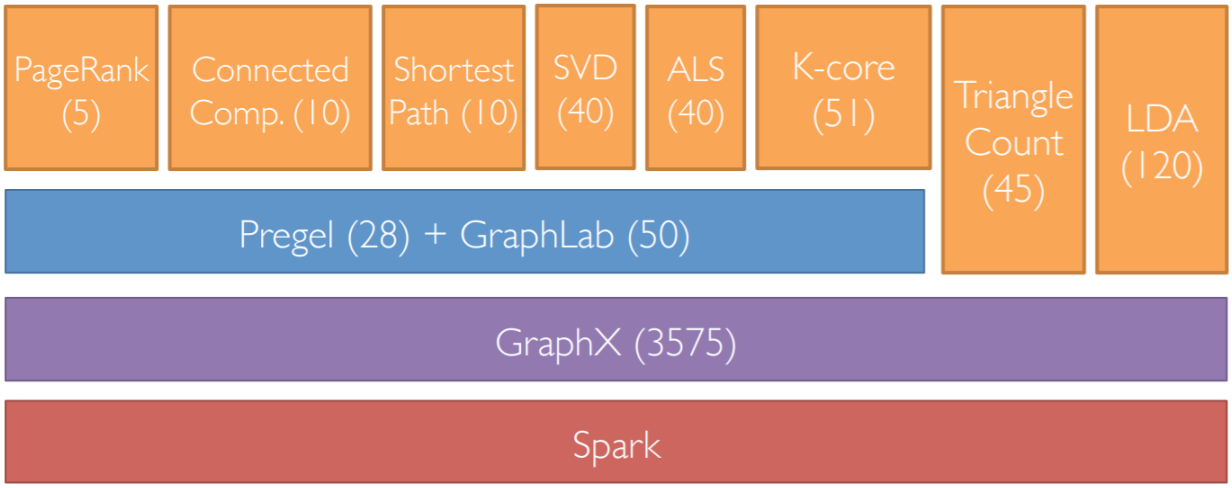
Figure 4 shows the interactive operation on Spark RDD which depicts that if different queries are run on the same set of data set for repeated times, then it will be kept in the memory for better execution time which we saw as the set back in the figure 3. Each transformed RDD may be

recomputed each time when we run on it. This operation also supports persisting RDDs on disk or replicated across multiple nodes in the cluster of nodes.

GraphX

The growing scale and importance of graph data has driven the development of numerous specialized graph processing systems including Pregel, PowerGraph and many others. By exposing specialized abstractions backed by graph-specific optimizations these systems can be naturally expresses and efficiently executed using iterative graph algorithms like PageRank and community detection on graphs with billions of vertices and edges. As a sequence, graph processing systems typically outperform general-purpose distributed data flow frameworks like Hadoop MapReduce by orders of magnitude.

GraphX systems consists of immutable graph which consists of both directed adjacency structure as well as user defined attributes associated with each vertex and edge. Programs in the GraphX system describe transformations from one graph to the next either through operators which transform vertices, edges or both in the context of their neighborhoods (adjacent vertices and edges).

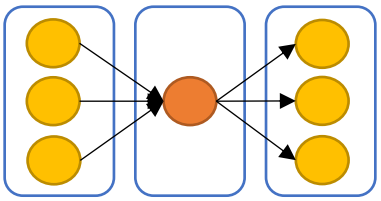


The above image is the pictorial representation of GraphX Stack. The lines of code involved in it. This image was referred from Stanford presentation given by Reza Zadeh

Partitioning

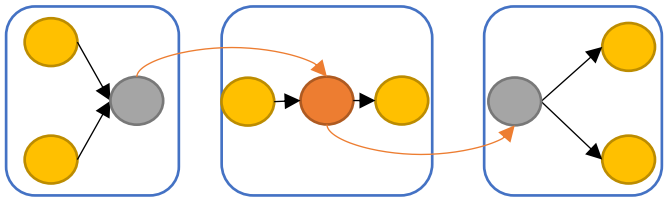
Graph Parallel computation requires each vertex or edge to be processed in the context of its neighborhood. Each transformation depends on the results of distributed joins between vertices and edges. Whenever there is some segmentation or distribution of data, indexing serves as an important part in the implementation. The proper allocation of layout for the data and its location is always prominent in achieving the effective performance. The graph structure describes the data movement, distributed graph computation systems rely on graph partitioning

and efficient graph storage to minimize communication and storage overhead to ensure balanced computations.



Edge-cut (Giraph model)

An edge cut splits the graph along edges while a vertex-cut.



Vertex-cut (GAS model)

A Vertex-cut splits the graph along vertices.

In these 2 types we partition the graph across three machines

Property Graph

Tables and Graphs are composable views of the same physical data. Each view has its own operators that exploit the semantics of the view to achieve efficient execution.

Table View <------ GraphXUnifoed Representation -------> Graph View

The property graph is represented using the two spark RDD

Edge Collection: VertexRDD

Vertex Collection: EdgeRDD

Class Graph[VD,ED] {

Val vertices: VertexRDD[VD]

Val edges: EdgeRDD[ED] }

Where VD is the type of the vertex attribute and ED is the type of the edge attribute

The primitive data types involved in this are

Vertex Collection:

class VertexRDD[VD] extends RDD[(VertexId, VD)]

Edge Collection:

class EdgeRDD[ED] extends RDD[Edge, [ED]]

class class Edge[ED](srcId: VertexId=0, dstId: VertexId=0, attr:ED= null.asInstanceOf[ED])

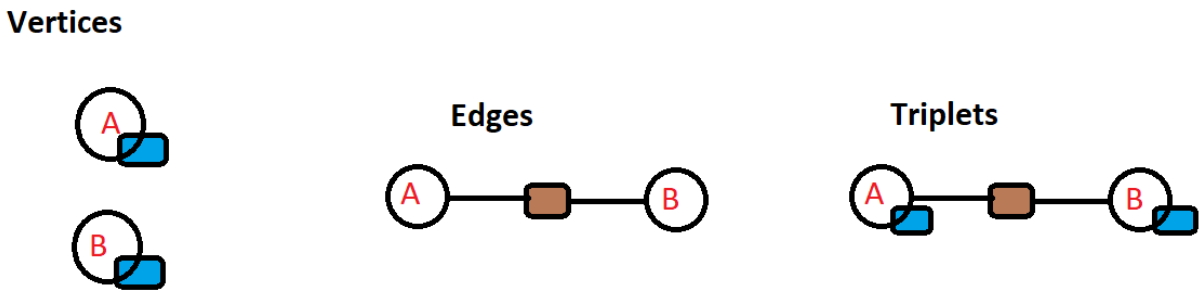
Edge Triple:

This represents an edge along with the vertex attributes of its neighboring vertices

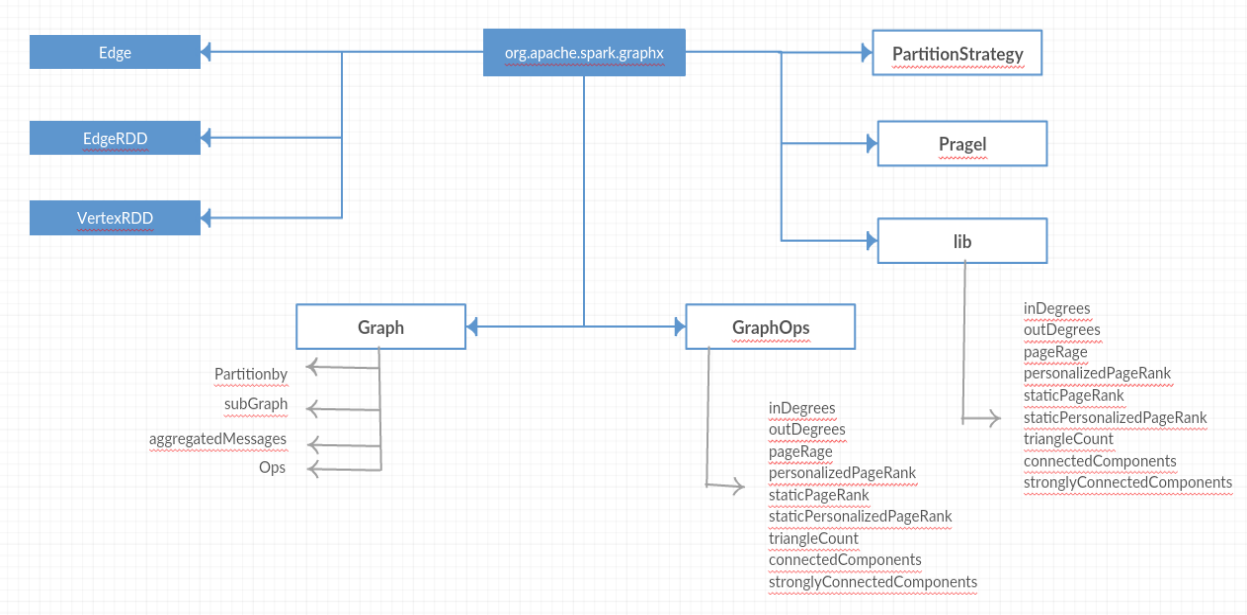
class EdgeTriplet[VD,ED] extends Edge[ED]

Unlike a relational table, graph processing is contextual with respect to a neighborhood. It maintains locality, equal size partitioning. The communication and storage overhead of an edge cut is directly proportional to the number of edges that are cut where as in vertex cut it is directly proportional to the sum of the number of machines spanned by each vertex. Vertex cut strategy by default is with the minimum replication of edges. The AlphaGO Tweets Analytics pipeline works in the following way. Collect data-- > Store -→ Transform and extract features→ GraphxModel→ GraphX Algorithms.

The representation of graphs using two RDDs namely edge-collection and vertex collection are called distributed graph representation. Vertex cut partitioning is happening as in PowerGraph. Each vertex partition contains a bitmask and routing table. A routing table is a logical map from a vertex id to set of the edge partitions that contains adjacent edges. Bitmask enables the set of intersection and filtering. Vertices bitmasks are updated after each operation. To consider this we can take the example of mapreduce triplets. Vertices which are hidden by the bitmask do not participate in the graph operations. One of the major similarity between the powergraph and GraphX is both are vertex partitioning but PowerGraph operates on GAS programming model and GraphX is unifying data-parallel and graph-parallel analytics.



Structural Queries – Indegrees, vertices Attribute Transformers- mapVertices Structural Transformers, Join- Reverse, subgraph Connecting Mining- ConnectedComponents, triangles Algorithms- Aggregate messages, PageRank. There are four ways to build a graph. GraphLoader.edgeListFile(…), from RDDs, fromEdgeTuples <- id tuples and fromEdges<- edgeList



In the above diagram GraphOps is a separate algorithm from the Graph implementation and library is used to implement the analytical function

Graph Parallel Algorithms

Collaborative Filtering

- Alternating Least Squares
- Stochastic Gradient Descent
- Tensor Factorization

Community Detection

- Triangle-Counting
- K-core Decomposition
- K-Truss

Structured Prediction

- Loopy Belief Propagation
- Max- Product Linear Programs
- Gibbs Sampling

Graph Analytics

- PageRank
- Personalized PageRank
- Shortest Path
- Graph Coloring

Semi-supervised ML

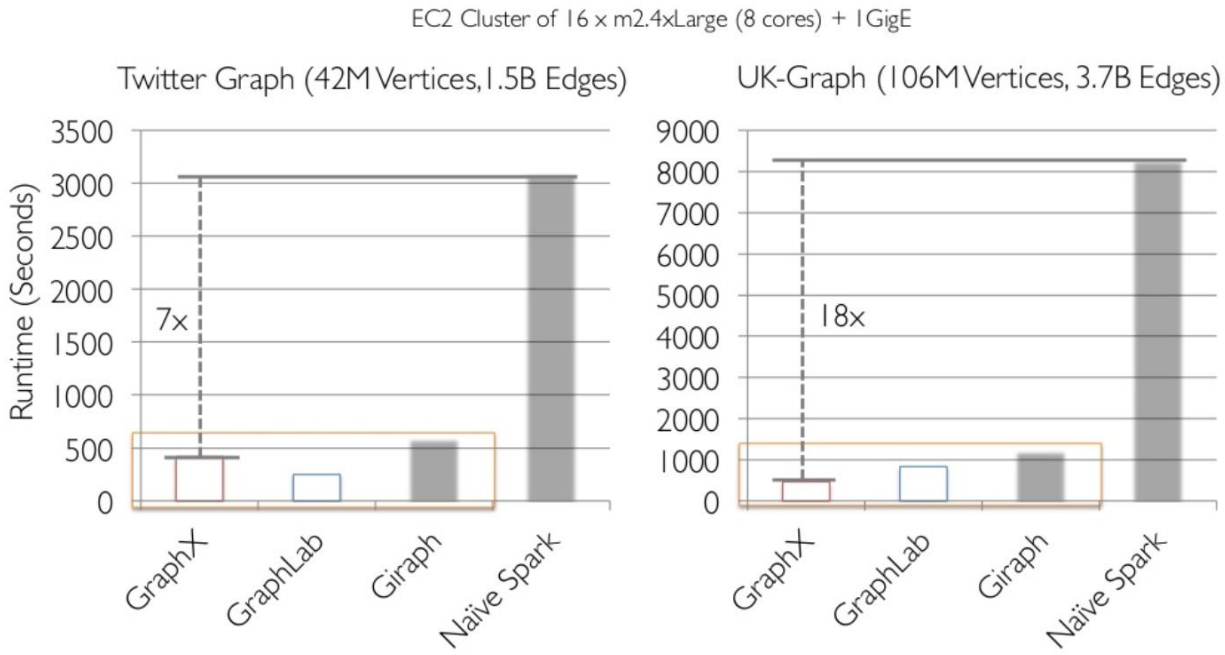
- Graph SSL
- CoEM

Classification

- Neural Networks

PageRank Benchmark

GraphX performs comparably to the state of the art graph processing systems a per the details mentioned in the below analytics



PageRank and Pregel

PageRank is one of the prominent reason on why Google Search works so fast and precise now a day. It is the considered to be our black box. Essentially, PageRank is all about the importance of a node in a Graph which is used for Link Analysis. The main bottom line to extract from this is, In-Links are votes and In-Links from important node are more important which is recursion.

The important of a node is the probability that a random walker fall on a given node. So it depends on the following

- The probability that he lands into one of its neighbor
- The probability that he crosses a link from the neighbor to it
- An arbitrary probability of teleportation

The solution for this will be

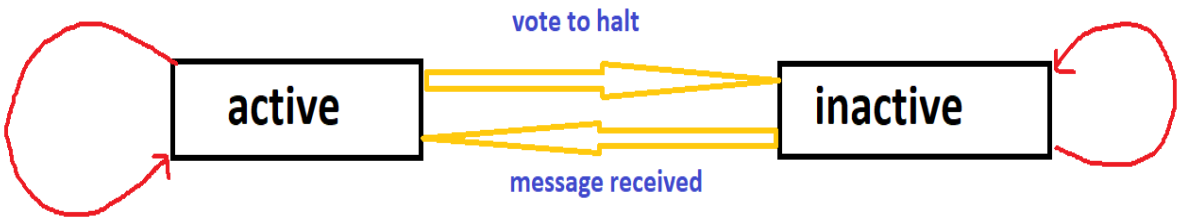
Power Method / Iteration (recursive)

r_new= A x r_old

Here it take a lot of effort to find out the required result. Matix algebra is always a pain in the distributed environment. So here the process is rather graph oriented.

Pregel- which is also been implemented in google. This is based on the Bulk Sync Parallel process(BSP). BSP works like message passing style. During the superstep I, a vertex can

- Use messages received from Superstep i-1
- Execute a function
- Send messages
- Vote to halt



PageRank with Pregel

```
//Define the three functions needed to implement PageRank in the GraphX

//Version of Pregel

def vertexProgram(id: VertexId, attr: (Double, Double), msgSum: Double): (Double, Double) = {

        val (oldRP, lastDelta) = attr

        val newPR = oldPR + (1.0 -resetProb) * msgSum

        (newPR, newPR – oldPR)

}

def sendMessage(edge: EdgeTriplet[(Double, Double), Double]) =  {

        if (edge.srcAttr._2 > tol) {

                Iterator{(edge.dstId, edge.srcAttr._2 * edge.attr)}

        } else {

                Iterator.empty

        }

}
```

Here we can execute a dynamic version of pregel.

```
Pregel(pagerankGraph, initialMessage, activeDirection = EdgeDirection.out) ( vertexProgram, sendMessage, messageCombiner).mapVertices((vid, attr -> attr._1)
```

```
//Load into a graph

val graph = Graph.fromEdgeTuples(edges, 0)

//calculate page rank and put in cache

val pageRank = PageRank.run(graph, 5). Vertices.map(_._2)

val minRank = pageRank.min

val maxRank = pageRank.max

println(minRank + " " + maxRank)
```

Summary

Spark is faster than Hadoop's MapReduce where in various cases Spark outperformed by 100 times better than MapReduce. Spark is easily integrated and flexible in compatibility with efficient memory management. Another major uptake from this technology of Big Data is that, Spark's community is one of the most active community building and debugging various Spark releases. Spark has big libraries of Spark SQL, GraphX which is major use of data in pictorial form. Using the GraphFrames, it is possible to turn data tables into graphs with just few lines of code. The full power of the Pregel API implemented in GraphX is available in combination with spark SQL. As a result, raw data stored in Hadoop in assorted flavors can easily be combined into huge multi-layer graphs with graph analysis all done via spark.

Future of GraphX

This GraphX is a combination of advantages from both graph parallel systems and data-parallel systems. GraphX uses the vertex cut partitioning which enables minimized movement of data during graph construction. GraphX provides RDG, based on which we implement Pregel and PowerGraph. There are potential improvements with the GraphX modelling. The language support for the JavaAPI and PythonAPI are taking lime light. We can try to implement GraphX on top of the distributed relation databases. Along with the current algorithms LDA, Correlation clustering algorithms are also worked on. Even though these are the soon to be achieved, the real test comes in the implementations streaming and time varying graphs and development of graph database like queries.

References

The below links are used for the basic understanding of ***GraphX API and its operators***

http://ampcamp.berkeley.edu/big-data-mini-course/graph-analytics-with-graphx.html

https://spark.apache.org/graphx/

The below link deals with the topic of ***setup and implementation of graph***

https://docs.databricks.com/spark/latest/graph-analysis/graph-analysis-graphx-tutorial.html

The below link is referred for the ***RDD and Spark Concepts***

https://jaceklaskowski.gitbooks.io/mastering-apache-spark/content/spark-rdd.html

https://spark.apache.org/docs/2.2.0/rdd-programming-guide.html

The below link is referred for the ***PowerGraph Concepts***

https://www.usenix.org/node/170825

https://people.eecs.berkeley.edu/~jegonzal/assets/slides/powergraph_osdi12.pptx

The below links are studied for the ***Pregel Concepts***

https://endymecy.gitbooks.io/spark-graphx-source-analysis/content/docs/pregel-a_system_for_large-scale_graph_processing.pdf

https://stanford.edu/~rezab/classes/cme323/S16/notes/Lecture16/Pregel_GraphX.pdf