RESEARCH REVIEW JOURNALS
Volume-03
Issue-01
January-2018

ISSN: 2455-3085 (Online)
RESEARCH REVIEW International Journal of Multidisciplinary
www.rrjournals.com [UGC Listed Journal]

An Analytics Research for Big Data Graph Analytics: Tools Techniques and Issues, Challenges

[1]Sai Prasad Padavala & [2]Dr. Suresh Chand Tyagi

[1]Research Scholar of Sri Satya Sai University
[2]Executive Director IDC Foundation

ARTICLE DETAILS

Article History

Published Online: 28 January 2018

Keywords
Big Data, Big Graph, Graph Processing, Graph Analytics, Graph Parallel Computing, Distributed Processing, Graph Algorithms

ABSTRACT

In the era of big data, interest in analysis and extraction of information from massive data graphs is increasing rapidly. This paper examines the field of graph analytics from a query processing point of view. Whether it be determination of shortest paths or finding patterns in a data graph matching a query graph, the issue is to find interesting characteristics or information content from graphs. Many of the associated problems can be abstracted to problems on paths or problems on patterns. Unfortunately, seemingly simple problems, such as finding patterns in a data graph matching a query graph are surprisingly difficult (e.g., dual simulation has cubic complexity and sub graph isomorphism is NP-hard). In addition, the iterative nature of algorithms in this field makes the simple MapReduce style of parallel and distributed processing less effective. Nowadays, graphs with billions of nodes and trillions of edges have become very common. In principle, graph analytics is an important big data discovery technique. Therefore, with the increasing abundance of large scale graphs, designing scalable systems for processing and analyzing large scale graphs has become one of the timeliest problems facing the big data research community. In general, distributed processing of big graphs is a challenging task due to their size and the inherent irregular structure of graph computations. In this paper, we present a comprehensive overview of the state-of-the-art to better understand the challenges of developing very high-scalable graph processing systems. In addition, we identify a set of the current open research challenges and discuss some promising directions for future research.

INTRODUCTION

The Big Data delineates huge data set to store, process and analyze. These data are generated by the daily aggrandizement of data from various sources, call for Big Data to knob these perplex data. Therefore, the conventional means of handling data are now obsolete, emerging Big Data with full-fledged, and qui vive for research in these data. Therefore, NoSQL is a prominent field in Big Data. The Big Graph is part of NoSQL which is gigantic in size, perplex to handle, and arduous to visualize.between objects are naturally modelled as graphs. Therefore, graphs have been used to represent data sets in a wide range of application domains, such as social science, astronomy, computational biology, telecommunications, semantic web, protein networks, and many more. In practice, graph analytics is an important and effective big data discovery tool. For example, it enables identifying influential persons in a social network, inspecting fraud operations in a complex interaction network and recognizing product affinities by analyzing community buying patterns. Nowadays, graphs with millions and billions of nodes and edges have become very common. For example, in 2012, Facebook has reported that its social network graph contains more than a billion users (nodes) and more than 140 billion friendship relationships (edges). The enormous growth in graph sizes requires huge amounts of computational power to analyze. In practice, distributed processing of large scale graphs is a challenging task due to their size in addition to their inherent irregular structure and the iterative nature of graph processing and computation algorithms. Graph algorithms are becoming increasingly important for analyzing large datasets in many fields. Real-world graph data follows a pattern of sparsity that is 1 not uniform, but highly skewed towards a few items. Implementing graph traversal, statistics and machine learning algorithms on such data in a scalable manner is quite challenging. As a result, several graph analytics frameworks such as Giraph, FlashGraph, GraphChi, X-Stream and many more, have been developed, each offering a solution with different programming models and targeted at different users.

GRAPH ANALYTICS

When relationships between data items take center stage (e.g., social networks), big data analytics often takes the form of graph analytics, in which the data items are represented as labeled vertices, and the relationships as labeled edges. Many problems in graph analytics may be formulated in terms of labeled multi-digraphs. A labeled multi-digraph allows multiple directed edges

between any two vertices, so long as they are differentiable labeled.More formally, a labeled multi-digraph may be defined as a 4-tuple $G(V,E,L,l_v)$ where

$$V = set\ of\ vertices$$
$$E \subseteq V \times L \times V \quad (set\ of\ labeled\ edges)$$
$$L = Set\ of\ labels$$
$$l_v: V \to L \quad (vertex\ labeling\ function)$$

The corresponding class definition in ScalaTion stores the graph as an array of edges, where each edge is a triple Tuple3 [Int, TLabel, Int]. Vertex labels are stored as an array of labels. The vertices are implicitly 0, … ,$n - 1$.

```
class    TDigraph    (edge:  Array  [Triple],
                      label:  Array  [TLabel],
                      name:   String)
```

In ScalaTion, a simpler alternative representation allows edge labels to be vector valued. The alternative definition, then becomes a 5-tuple $G(V,E,L,l_v,l_e)$ where

$$V = set\ of\ vertices$$
$$E \subseteq V \times V \quad (set\ of\ edges)$$
$$L = Set\ of\ labels$$
$$l_v: V \to L \quad (vertex\ labeling\ function)$$
$$l_e: E \to L^{k_e} \quad (edge\ labeling\ function)$$

The connections between vertices are characterized by a set of edges. For a multi-digraph, $uv \in E$ means that there is a directed edge from vertex *u* to *v*. If the triple representation is used, a labeled edge is denoted as $u\lambda v \in E$. When the edge labels are scalar or non-existent, the multi-digraph becomes a digraph.

The corresponding class definition in ScalaTion stores the graph using an adjacency set/list representation. Again, the vertices are implicitly 0, … ,$n - 1$.

```
class    MDigraph    (ch:        Array  [SET  [Int]],
                      label:     Array  [TLabel],
                      valelabel: Map    [Pair,  TLabel],
                      inverse:Boolean,
                      name:      String)
```

A simple way to characterize the connectivity is in terms of children and parents, as defined by the following two setvalued functions.

$$ch(u) = child(u) = \{v: uv \in G.E\}$$
$$pa(u) = parent(u) = \{w: wu \in G.E\}$$

Many of the problems in graph analytics involve finding paths, patterns or partitions in very large data graphs (e.g., graphs with a billion edges). We measure the size of a graph in terms of the number of vertices and the number of edges.

$$\eta = |G.V| = the\ number\ of\ vertices$$
$$\epsilon = |G.E| = the\ number\ of\ edges$$

The path, pattern or partition problems are strongly interrelated. A path may be viewed as a simple linear pattern and partitioning is needed for both path and patternproblems, when graphs become too large to store or process on a single machine or single thread.

BIG GRAPH

We can simply define Big Graph as,

"Big Data + Structure = Big Graph"

Big graphs are ubiquitous, ranging from social networks and mobile call networks to biological networks and the World Wide Web. The sources of real-world large-scale graphs include:

- Social graphs (Facebook, Twitter, Google+, LinkedIn, etc.)
- Endorsement graphs (web link graph, paper citation graph, etc.)
- Location graphs (map, power grid, telephone network, etc.)

The size of large scale graphs used in the recent literature is given in table 1.

Table 1: Large Scale Graphs in current literature

Name	Nodes	Edges
Web Graph	More than 20 billion nodes (pages)	More than 160 billion edges (hyperlinks)
Facebook	More than a billion nodes (users)	More than 140 billion edges (friendship relationships)
LinkedIn	Almost 8 million nodes	Almost 60 million edges
SemanticWeb	3.7 million nodes (objects)	400 million edges (facts)

BIG GRAPH PROCESSING

The growth of graph-structured data in modern applications such as social networks and knowledge bases creates a crucial need for scalable platforms and parallel architectures that can process it in bulk.

1. Pregel

Pregel[3] is a scalable, general-purpose system for implementing graph algorithms in a distributed environment. It is known as the first Bulk Synchronous Parallel (BSP), an implementations that provides a native API specifically for graph algorithms using a "think like a vertex" computing paradigm. The basic computation model of Pregel is shown in figure 1.

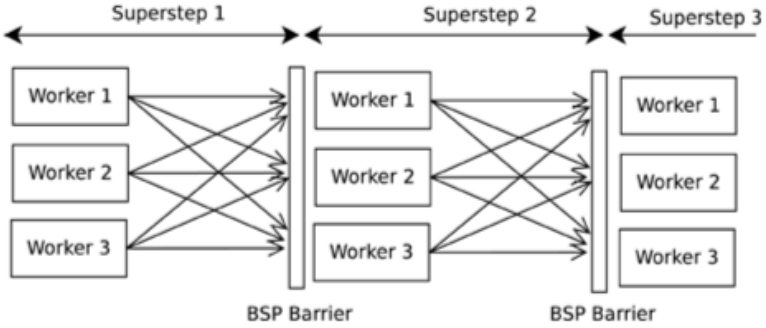

Figure 1.Pregel Computation Model with three supersteps and three workers

In Pregel, programs are expressed as a sequence of iterations (supersteps), in each of which a vertex can receive messages sent in the previous iteration, send messages to other vertices, and modify its own state and that of its outgoing edges. The input graph is loaded once at the start of a program and all computations are executed in-memory.

2. Giraph

Apache Giraph[1] is an iterative graph processing system built for high scalability. It runs workers as map-only jobs on Hadoop and uses HDFS for data input and output. Giraph adds several features including master computation, sharded aggregators, edge-oriented input, out-of-core computation, and more. It also uses Apache ZooKeeper for coordination, checkpointing, and failure recovery schemes. With a steady development cycle and a growing community of users worldwide, Giraph is a natural choice for unleashing the potential of structured datasets at a massive scale. For example, it is currently used at Facebook to analyze the social graph formed by users and their connections.

3. Mizan

Mizan[8] is a scalable framework for supporting graph mining algorithms in large parallel computing infrastructures. It is a layer between the users' code and a variety of computing infrastructures, such as Linux clusters, cloud environments and supercomputers. Mizan examines the graph structure and decides the placement of the data and the low level message passing mechanism transparently to the users' code. It uses message passing and implements an instance of the Bulk Synchronous Parallel model.

4. GPS

GPS[9] is an open-source system for scalable, fault-tolerant, and easy-to-program execution of algorithms on extremely large graphs. It is a distributed system, designed to run on a cluster of machines, such as Amazon's EC2. GPS offers Large Adjacency List Partitioning (LALP), an optional performance optimization of algorithms that send to all of its neighbors the same message.

GPS also features an optional dynamic migration scheme. Dynamic migration repartitions the graph during the computation by migrating vertices between workers, to improve workload balance and network usage.

5. GraphLab

GraphLab[11] is a framework for asynchronous parallel graph computations in machine learning. It differs from Pregel in that it does not work in bulk synchronous steps, but rather allows the vertices to be processed asynchronously based on a scheduler. The vertex functions can run at any time as long as specified consistency rules are obeyed. It is therefore well-suited for the machine learning types of applications for which it is defined, where each vertex accumulates information from its neighbors' states and updates its state, possibly asynchronously. It extends the shared memory GraphLab abstraction to the distributed setting by refining the execution model, relaxing the scheduling requirements, and introducing a new distributed data-graph, execution engines, and fault-tolerant systems.

6. Graph Sample and Hold

Graph Sample and Hold (gSH)[7] is a stream sampling framework that supports analytics over big graphs. The nice property of gSH consists in building theoretically sound unbiased estimators derived from the sample graph, but still robust for the estimation of different properties of the target (big) graph, yet conveying in high accuracy and tolerable approximation errors.

7. Pregelix

Pregelix[6] is a dataflow-based Pregel-like system built on top of the Hyracks parallel dataflow engine. It combines the Pregel API from the systems world with data-parallel query evaluationtechniques from the database world in support of large scale graph analytics. This combination leads to effective and transparent out-of-core support, scalability, and throughput, as well as increased software simplicity and physical flexibility. To the best of our knowledge, Pregelix is the only open source Pregel-like system that scales to out-of-core workloads efficiently, can sustain multi-user workloads, and allows runtime flexibility.

PARTITION PROBLEMS

Besides direct problems calling for partitioning a graph into subgraphs, such as community detection in social networks, partitioning is often used as the first step in placement of graph data on the nodes of parallel/distributed graph computation clusters such as Pregel (Malewicz, et al., 2010), GPS (Salihoglu&Widom, 2013) and Giraph (Giraph website, n.d.)

The partitioning problem can be formally stated as follows. Given a graph G, $G, partition(G)$) is

$[\, G_1, \dots, G_k \,]$ $s.t.$ $\cup_{i=1}^{k} G_i.V = G.V.$

These edges are usually referred to as inter-partition edges. The simplest partitioning strategy, which is often the default in many distributed graph computation clusters, is the random partitioning. As the name suggests the assignment of nodes to partitioned subgraphs is random. Random partitioning has the advantage of being simple and fast. It also is known to yield good load balancing when used for data placement in distributed graph computation clusters. However, it usually results in large numbers of interpartition edges (also referred to as “cut edges”). Having too many cut edges is generally not preferred because it can result in very high communication overheads in distributed graph computation clusters. Min-cut partitioning, on the other hand, aims to minimize the number of cut edges. METIS is a popular mincut partitioning algorithm (Karypis& Kumar, 1995). Pure min cut algorithms often result in poor load balancing in distributed graph computations clusters (Abdolrashidi, Ramaswamy, &Narron, 2014). Thus variants of min-cut algorithms have been developed to balance the number of vertices in each partition while maintaining the min-cut property (Karypis, 2003). ScalaTion currently supports three efficient algorithms for partitioning very large graphs: Random Partitioning, Ordered Partitioning and Label Propagation Partitioning. Random partitioning has the advantage of being fast and balanced, but is not concerned about whether edges are cut and tends to perform poorly in this regard. Ordered partitioning performs similarly to random partitioning unless the graph is built in a way that vertex ids of neighboring vertices tend to be clustered, in which case it can have far fewer cuts than random partitioning. Label propagation partitioning tends to perform better than the other two in terms of reducing the number of edge cuts, but balance may be reduced. Again, when graphs are built in an ordered way, ordered partitioning will do better than label propagation partitioning. For label propagation partitioning (Wang, Xiao, Shao, & Wang, 2014), each vertex is initiallygiven a unique integer label ‘ilabel’. On each iteration, each vertex will have its ‘ilabel’ reassigned to the most popular/frequent ‘ilabel’ in its neighborhood (which includes its children, parents and itself). There are other algorithms that perform better at reducing edge cuts than these. Unfortunately, many graph partitioning problems and consequently, partitioning very large graphs is quite challenging. For example, bisecting the vertex set into two subsets of equal (or off by 1) size that minimizes the number of edge cuts is (Garey, Johnson, &Stockmeyer, 1976). Also, METIS (Karypis& Kumar, 1995) is one of the better packages for graph partitioning, but currently will not work on graphs with several tens of millions of vertices (Wang, Xiao, Shao, & Wang, 2014). “METIS works in three steps: (1) coarsening the graph; (2) partitioning the coarsened graph; (3) uncoarsening.” (Wang, Xiao, Shao, & Wang, 2014).

graph partitioning. However, several problems remain open and additional research is needed to address them. While it is known that graph partitioning and placement have significant impact on graph computations, the impact of graph partitioning on individual types of queries is, to our best knowledge, not comprehensively studied. Graph queries vary widely with respect to the manner in which the computation flows across the different topological regions of the graph as well as the communication pattern among the vertices. For example, in the vertex-centric PageRank algorithm, all the vertices are active in the very first superstep, and many of them remain active for a large fraction of the supersteps. In contrast, in the vertex-centric single source shortest path algorithm, the computation starts from a single vertex (i.e., a single vertex is active in the first superstep), and the computation migrates to other vertices in later supersteps. Clearly, these computations need different types of partitioning strategies. Characterizing the computation and communication behaviors of various types of queries, studying their performance with different partitioning strategies and developing computation-specific graph partitioning strategies are important research problems.

CONCLUSIONS

With the increasing importance and growing size of graph stores and databases, recent research activity in graph analytics has increased substantially. Progress has also been substantial, but many challenges remain for future research. Both graph pattern matching and graph partitioning will require continued progress in research in order to scale to the billion vertex threshold.The usage of large scale graph processing platforms is rapidly expanding in both academia and industry. In principle, large scale graph processing platforms are increasingly important as more and more problems require dealing with graphs. To this end, we presented a thorough survey of the state-of-the-art of the emerging platforms in this domain. In addition, we have provided an overview of the recent studies for benchmarking and evaluating some of the existing platforms. Finally, we identified and presented a set of the current open research challenges and also presented some of the promising directions for future research. In general, we believe that there are still many opportunities for new innovations and optimizations in the domain of large scale graph processing. Hence, we consider this article as an important step in helping researchers to understand the domain and guiding them towards the right direction to improve the state-of-theart.

REFERENCES

1. Abdolrashidi, A., Ramaswamy, L., &Narron, D. S. (2014).Performance modeling of computation and communication tradeoffs in vertex-centric graph processing clusters. Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2014 International Conference on (pp. 55--63). IEEE.
2. Angles, R., & Gutierrez, C. (2005).Querying RDF data from a graph database perspective. In The Semantic Web: Research and Applications (pp. 346--360).
3. Springer. Barceló Baeza, P. (2013). Querying graph databases.Symposium on Principles of database systems (pp. 175- 188).ACM.
4. Bialecki, A., Cafarella, M., Cutting, D., & O'Malley, O. (2005).Hadoop: A framework for running applications on large clusters built of commodity hardware. Retrieved from Wiki at http://lucene.apache. org/hadoop
5. Conte, D., Foggia, P., Sansone, C., & Vento, M. (2004). Thirty years of graph matching in pattern recognition. International journal of pattern recognition and artificial intelligence, 265--298.
6. Corby, O., &Faron-Zucker, C. (2007). Implementation of SPARQL query language based on graph homomorphism. Springer.
7. Dean, J., &Ghemawat, S. (2014). MapReduce: Simplified Data Processing on Large Clusters. In OSDI.ACM.
8. Deville, Y., Dooms, G., &Zampelli, S. (2008). Combining two structured domains for modeling various graph matching problems.In Recent Advances in Constraints (pp. 76--90).
9. Fan, W., Li, J., Ma, S., Tang, N., Wu, Y., & Wu, Y. (2010). Graph pattern matching: from intractable to polynomial time. Proc. VLDB Endow. (pp. 264--275).VLDB Endowment.
10. Fan, W., Li, J., Ma, S., Wang, H., & Wu, Y. (2010). Graph homomorphism revisited for graph matching. Proceedings of the VLDB Endowment, 1161--1172.
11. Fan, W., Wang, X., & Wu, Y. (2014). Answering graph pattern queries using views. ICDE, (pp. 184--195).
12. Fard, A., Abdolrashidi, A., Ramaswamy, L., & Miller, J. A. (2012).Towards efficient query processing on massive time-evolving graphs. 8th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), (pp. 567 -574).
13. Fard, A., Manda, S., Ramaswamy, L., & Miller, J. A. (2014).Effective caching techniques for accelerating pattern matching queries. Bid Data Conference (pp. 491--499). IEEE.

14. Fard, A., Nisar, M. U., Miller, J. A., &Ramaswamy, L. (2014). Distributed and Scalable Graph Pattern Matching: Models and Algorithms. International Journal of Big Data (IJBD), 1-14.