Managing and Mining Large Graphs: Systems and Implementations

Bin Shao
Microsoft Research Asia
Beijing, China
binshao@microsoft.com

Haixun Wang
Microsoft Research Asia
Beijing, China
haixunw@microsoft.com

Yanghua Xiao*
Fudan University
Shanghai, China
shawyh@fudan.edu.cn

ABSTRACT

We are facing challenges at all levels ranging from infrastructures to programming models for managing and mining large graphs. A lot of algorithms on graphs are ad-hoc in the sense that each of them assumes that the underlying graph data can be organized in a certain way that maximizes the performance of the algorithm. In other words, there is no standard graph systems based on which graph algorithms are developed and optimized. In response to this situation, a lot of graph systems have been proposed recently. In this tutorial, we discuss several representative systems. Still, we focus on providing perspectives from a variety of standpoints on the goals and the means for developing a general purpose graph system. We highlight the challenges posed by the graph data, the constraints of architectural design, the different types of application needs, and the power of different programming models that support such needs.

This tutorial is complementary to the related tutorial "Managing and Mining Large Graphs: Patterns and Algorithms".

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed Databases*

General Terms

Design, Performance

Keywords

Distributed Graph System, Memory Cloud, NoSQL, Graph Database

1. INTRODUCTION

Large graphs appear in a wide range of computational domains. For example, the World Wide Web contains over 50 billion web pages and one trillion unique URLs [5]. The friendship network of Facebook consists of 800 million nodes and more than 100 billion links [3]. LinkedData has 31 billion RDF triples and 504 million RDF links [4]. The de Brujin graph [19], wherein each node represents a DNA sequence of k base pairs, may contain up to 4^k nodes where k is at least 20.

We are facing challenges at all levels ranging from infrastructures to programming models for managing and mining large graphs. Recently, research on this topic has seen an explosive growth [6]. However, a lot of graph algorithms are ad-hoc in the sense that each of them assumes that the underlying graph data can be organized in a certain way that maximizes the performance of the algorithm. In other words, there is no standard or de facto standard graph systems based on which algorithms on graphs are developed and optimized. The situation is even more urgent for extremely large graphs with billions of nodes and edges: First, converting billion node graphs from one format to the other for different algorithms is extremely costly or totally infeasible; Second, many graph algorithms (e.g., subgraph matching algorithms or reachability queries that rely on super-linear graph indices) are not applicable to billion node graphs.

In response to this situation, a lot of graph systems have been proposed recently. In the tutorial, we discuss and compare several representative systems in detail. Still, the major focus of this tutorial is not about introducing a wide range of graph systems to our audience. Rather, we endeavor to offer perspectives from a variety of standpoints on the goals and the means for developing a general purpose graph system. We highlight specific challenges posed by the graph data, the hardware constraints for architectural design, the different types of application needs, and the power of different programming models that support such needs.

2. TUTORIAL OUTLINE

2.1 Data Space

Before we discuss the pros and cons of different designs of graph systems, it is important to understand the data for which the systems are designed. We describe data from three aspects: volume (how big is the data), variety (how complex is the data), and velocity (how fast is the data coming in). Here, we use Figure 1 to show the data space characterized by data volume and data variety (we will investigate all of the three aspects in the tutorial).

From Figure 1 (which is reproduced from [10]), it is clear that there is a huge problem space. RDBMSs, which have ma-

*The work was done at Microsoft Research Asia.

SIGMOD '12, May 20–24, 2012, Scottsdale, Arizona, USA.

	# of Nodes	# of Edges	Size of Graph Topology	# of Machines		
				Online/Offline	Offline	Sampling
US Road Graph	23.9×10^6	58×10^6	808 MB	1	1	1
Facebook Social Graph	800×10^6	104×10^9	787 GB	25	3	1-25
Web Graph	20.1×10^9	160.3×10^9	1494 GB	47	15	10-47

Table 1: Graph sizes and number of machines needed to deploy on Microsoft's Trinity (each machine has 32G RAM)

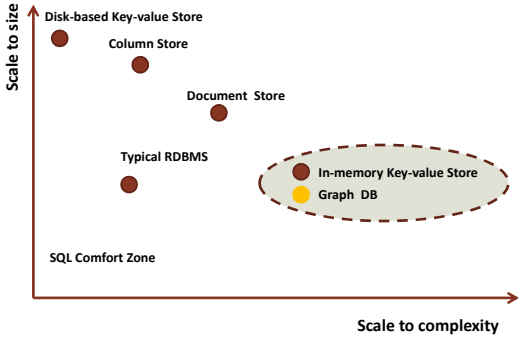

Figure 1: Data Space

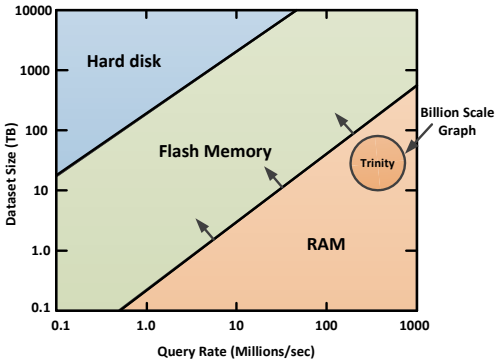

Figure 2: The territory of memory-based systems

tured after more than 40 years of continuous improvements, only handle data of relatively small size and low complexity. It is unlikely that a single system can cover the entire data space. For example, for data of extremely large volume but very low complexity, Map/Reduce systems and key/value stores may be appropriate solutions. In contrast, graph systems handle data of much smaller size but much higher complexity.

The high complexity of the data (i.e., many relationships exist among the data) means applications require flexible data accesses. One distinguishing characteristics of graphs is that graph accesses have no locality: As we explore a graph, we invoke random, instead of sequential data accesses, no matter how the graph is stored. In other words, any non-trivial graph query will have poor performance if the locality issue is not properly addressed. For example, although relational models or key/value stores can be used to manage graph data, they do not provide efficient query support, because random accesses are achieved through join operations.

2.2 Architecture Design

To a large extent, the size and the complexity of the graph data determines the architecture of a graph system. Figure 2 (which is reproduced from [16]) shows under what conditions the total cost of ownership will be the lowest. As we mentioned, graph exploration requires random accesses. This is equivalent to having a high rate of queries in different parts of the data. It will be desirable if the graph is in memory, which provides fast random access support.

As Table 1 indicates, it is entirely feasible to host web-scale graphs in the memory of a few dozens of machines, because even for billion-node graphs, the size of the graph topology is still much less than 10 TB (the topology of the current Facebook friendship graph takes about 1.5 TB memory space). Furthermore, Figure 2 shows that the dividing line between RAM and Flash Memory is moving up as RAM technology advances, which means memory-based systems are a suitable choice for large graphs.

Still, it is unlikely that we can store the topology of a web-scale, billion-node graph in the memory of a single commodity off-the-shelf machine. Thus, a lot of design issues will arise in building in-memory graph systems on a COTS cluster. The tutorial will discuss these topics in detail.

2.3 Applications Needs

Generally speaking, in graph processing, application needs can be classified into two large categories: online query processing, which requires low latency computing, and offline graph analytics, which requires high throughput computing. For example, deciding instantly whether there is a path between two given persons in a social network belongs to the first category while computing pagerank for the WWW belongs to the second.

As we will analyze in the tutorial, these two kinds of tasks have different data access patterns, which lead to challenges and opportunities for optimization. Still, many sophisticated applications have both needs. For example, although community analysis on social networks or link analysis on click graphs are mainly analytical tasks, it is important that the system also supports interactive user activities such as graph browsing and querying, as these activities may provide actionable insights for tuning and testing large analytical jobs. On the other hand, online query processing, for example, approximate shortest distances, etc., often relies on indices or sketches derived from the data, and building such indices or data sketches is analytical jobs.

Unfortunately, existing graph systems, such as Neo4j [1] and Pregel [14], are designed for one type of the needs. Then, a natural question is whether it is possible and what it takes to build a system that can support both needs. In this tutorial, we will investigate this problem by looking into several

	Native graphs	Online query processing	Memory based exploration	Distributed parallel processing
Neo4j [1]	Yes	Yes	No	No
HyperGraphDB [11]	No	Yes	No	No
FlockDB [2]	No	Yes	No	Yes
MapReduce [9]	No	No	No	Yes
PEGASUS [12]	No	No	No	Yes
Google's Pregel [14]	No	No	No	Yes
Microsoft's Trinity [15]	Yes	Yes	Yes	Yes

Table 2: Some representative graph systems

application areas, including basic graph operations such as subgraph matching [17, 7], knowledgebases [18], social networks, etc.

2.4 Computation Model

Recently, a variety of computing models have been suggested for graph analytics. For instance, a few systems represent graphs by their adjacency matrices, and model graph operations as matrix operations [12]. Another common approach is to support graph analytics in the Map/Reduce framework. Pregel [14] supports offline vertex-based computation under the BSP model. More recently, Trinity [15] extends the vertex-based computation model into two more finery model by i) restricting message passing among neighboring nodes, and ii) restricting message passing among nodes on a single machine.

In the tutorial, we will introduce and compare these computation models by focusing on their expressive power, and ease of use. We will also discuss their generality – that is, whether the computation model is friendly to all kinds of application needs. For example, it is difficult to provide online query processing in the Map/Reduce framework.

2.5 Cost of Ownership

There is one more important factor about graph systems, which is the cost of ownership. Table 1 shows three representative real life graphs. Clearly, the real size is not big at all – the topology of the Facebook friendship graph and even the entire topology of the WWW can fit in a single hard drive of modern standard. However, MapReduce and Pregel require hundreds or thousands of machines in order to process such graphs. This means not many organizations (including universities, companies, and government agencies) can afford to deploy such systems.

Thus, the cost of ownership becomes an important factor in evaluating a graph system. Currently, Trinity can perform efficient graph analytics on web-scale, billion-node graphs using as few as 20-30 commodity machines. Furthermore, a large variety of computations, such as density estimation, connected components discovery, etc., can be performed locally even on a single machine. This is shown in Table 1, where the right three columns show the number of commodity machines needed to process the graphs in online, offline, and approximate computation mode.

2.6 Representative Systems

In the tutorial, we try to provide perspectives on the goals and the means of developing a graph system by analyzing the existing representative systems. Table 2 lists several important systems although the tutorial also covers many systems we mention in Section 3. Currently, there are two representative graph systems for the two types of application needs we described in Section 1.3. Neo4j [1] focuses on supporting online transaction processing (OLTP) on graph data. Neo4j is like a regular database system, only with a more expressive and powerful data model. However, Neo4j is not distributed: It does not handle graphs that are partitioned over multiple machines. This limits the size of the graphs Neo4j can handle *efficiently*. This is so because data access on graphs has no locality [13], in other words, exploration on graphs incurs mostly random data access. For large graphs that cannot be stored in memory, disk random access becomes the performance bottleneck. Furthermore, a single machine also does not have enough computation power compared with a distributed, parallel system. Thus, it is difficult for systems such as Neo4j to handle web-scale graphs.

On the other end of the spectrum is MapReduce [9] and Pregel [14]. Both are high latency, high throughput platforms. MapReduce is for simple data, while Pregel is for graphs. Unlike Neo4j, MapReduce and Pregel do not support online query processing, instead, they are optimized for analytics on large data partitioned over hundreds of machines. MapReduce computations on graphs depend heavily on interprocessor bandwidth, as graph structures are sent over the network iteration after iteration. Pregel mitigates this problem by passing computation results instead of graph structures between processors. In Pregel, analytics on the graphs are expressed using a vertex based computation mechanism under the Bulk Synchronous Processing (BSP) model. Although some well known graph algorithms, including Pagerank, shortest path discovery, etc, can be implemented through vertex based computing with ease, there are many sophisticated graph computations, including for example, multi-level graph partitioning, cannot be expressed in a succinct and elegant way.

Besides Neo4j and Pregel, dozens of graph systems have been proposed in the last few years. Table 2 compares a few of them. The tutorial will discuss these systems in more detail. We focus on 4 important questions. First, does the graph exist in its native form, or does it follow other models, including RDBMS or key/value stores? When a graph is in its native form, graph algorithms can be expressed in standard, natural ways [8]. If not, we need a complete rethinking of the problem in order to develop analogous implementations in the new model, e.g., MapReduce. Second, does the system support in memory graph exploration? Because data

access on graphs has no locality, and random access on disks leads to performance bottlenecks, keeping graphs memory resident is important for efficient query processing. Third, does the system support low latency query processing on graphs? Although several existing systems in Table 2 are capable of providing OLTP support, few of them can handle billion-node graphs. One of the reason is that indices for graph processing usually have super-linear complexity, and building such indices for billion-node graphs is infeasible. Finally, does the system support high throughput offline analytics? Systems such as MapReduce and Pregel are capable of supporting computations such as Pagerank on extremely large graphs. However, not all graph computations can be implemented in MapReduce or in Pregel's vertex-based computation model in succinct and elegant ways.

3. REFERENCES

[1] `http://neo4j.org/`.

[2] `https://github.com/twitter/flockdb`.

[3] `http://www.facebook.com/press/info.php?statistics`.

[4] `http://www.w3.org/`.

[5] `http://www.worldwidewebsize.com/`.

[6] C. C. Aggarwal and H. Wang, editors. *Managing and Mining Graph Data*, volume 40 of *Advances in Database Systems*. Springer, 2010.

[7] J. Cheng, J. X. Yu, B. Ding, P. S. Yu, and H. Wang. Fast graph pattern matching. In *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, ICDE '08, pages 913–922, Washington, DC, USA, 2008. IEEE Computer Society.

[8] J. Cohen. Graph twiddling in a mapreduce world. *Computing in Science & Engineering*, pages 29–41, 2009.

[9] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, Jan. 2008.

[10] E. Eifrem. A nosql overview and the benefits of graph databases. Nosql East 2009.

[11] B. Iordanov. Hypergraphdb: a generalized graph database. In *Proceedings of the 2010 international conference on Web-age information management*, WAIM '10, pages 25–36, Berlin, Heidelberg, 2010. Springer-Verlag.

[12] U. Kang, C. E. Tsourakakis, and C. Faloutsos. Pegasus: A peta-scale graph mining system implementation and observations. In *Proceedings of the 2009 Ninth IEEE International Conference on Data Mining*, ICDM '09, pages 229–238, Washington, DC, USA, 2009. IEEE Computer Society.

[13] A. Lumsdaine, D. Gregor, B. Hendrickson, and J. W. Berry. Challenges in parallel graph processing. *Parallel Processing Letters*, 17(1):5–20, 2007.

[14] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 international conference on Management of data*, SIGMOD '10, 2010.

[15] B. Shao, H. Wang, and Y. Li. The Trinity graph engine. Technical Report 161291, Microsoft Research, 2012.

[16] J. Ousterhout, P. Agrawal, D. Erickson, C. Kozyrakis, J. Leverich, D. Mazières, S. Mitra, A. Narayanan, G. Parulkar, M. Rosenblum, S. M. Rumble, E. Stratmann, and R. Stutsman. The case for ramclouds: scalable high-performance storage entirely in dram. *SIGOPS Oper. Syst. Rev.*, 43:92–105, January 2010.

[17] Z. Sun, H. Wang, B. Shao, H. Wang, and J. Li. Efficient subgraph matching on billion node graphs. In *PVLDB*, 2012.

[18] W. Wu, H. Li, H. Wang, and K. Zhu. Probase: A probabilistic taxonomy for text understanding. In *Proceedings of the 2012 international conference on Management of data*, SIGMOD '12, 2012.

[19] D. R. Zerbino and E. Birney. Velvet: algorithms for de novo short read assembly using de bruijn graphs. *Genome Research*, 18(5):821–9, 2008.