2012 SC Companion: High Performance Computing, Networking Storage and Analysis

Graph Database Design Challenges using HPC Platforms

Prajakta Kalmegh
College of Computing
Georgia Institute of Technology
Atlanta, USA
pkalmegh@gatech.edu

Shamkant B. Navathe
College of Computing
Georgia Institute of Technology
Atlanta, USA
sham@cc.gatech.edu

***Abstract*—Graph Database Management Systems, also called graph databases, have recently gained popularity in the database research community due to a need to effectively manage large scale data with inherent graph-like properties. Graph databases are representation, storage and querying systems for naturally occurring graph structures. Graph databases are finding increasing applications in social networks, computational geometry, bioinformatics, drug discovery, semantic web applications and so on.**

The intent of this paper is to introduce the role of graph database management systems in the context of high-performance computing platforms and present the options for designing high-performance graph databases. We also present a set of potential research directions and list the challenges in combining the research in the two fields of graph databases and high-performance computing.

***Keywords*-Database systems, high performance computing, graph theory, system-level design**

I. Introduction

A graph database models highly connected and associative data using nodes, edges and properties. The database architecture enables users to execute complex decision-based and knowledge management semantic queries like 'What is the most effective drug for a disease?', 'How many friends have I influenced positively?', 'What is the nearest train station with full day free car parking?' etc. A data model facilitates efficient storage, retrieval and mining of data from the underlying database.

A graph database provides flexibility by providing a schema-free data model. New information can be added to the database without changing the underlying schema. It is easier to model *One:Many* relationships without using primary key - foreign key indirections as in a relational database management system (RDBMS). A graph computation platform provides a combination of hardware and software to process large graphs. The software is implemented to exploit the underlying hardware architecture for efficient computation. The hardware architecture may provide a standalone, distributed, in-memory, multi-threaded or hybrid platform for processing of large graphs, often in parallel. This facilitates high-performance offline analytics for massive graph problems. It does not restrict the data model or representation for the graphs and gives no specification for the query language or interface. To provide an optimized utilization of the hardware and software resources is the responsibility of the software that sits on top of it.

The choice of a computation model (persistence of data, transactional guarantees and data distribution) plays an important role in the design of the graph database. A highly transactional all-in-memory data store can provide considerably high throughput and low latency in querying of graph data, whereas a scalable distributed disk-based store with an efficient communication subsystem and graph partitioning scheme can provide a robust large scale graph data analytics platform. Similarly, decisions about providing data integration capabilities, data security, availability and reliability can affect the performance of a graph database significantly.

In this paper, we aim at presenting the challenges in designing a graph database on high-computation platforms for large scale data analysis. The contributions of this paper can be summarized as below:

- to define a graph database management system (Section II)
- to present a set of characteristics that play an important role in influencing the design and implementation of a graph database (Section III)
- to present the characteristics that influence the design and implementation of a high-performance computing database and present future research directions in the advancements of a graph database (Section IV)

II. What is a Graph Database Management System?

A database system is a software to manage the entities in the database and to support queries and transactions over it. A database system typically gives no specification of the underlying hardware. A computation platform typically defines the system architecture and may also specify details on the software like the operating system, programming model and querying interfaces. Graph databases and graph computation platforms together provide a framework for storage and analysis of massive graphs. For the sake of simplicity, in the scope of this paper we refer to them together as **G**raph **D**ata**B**ase **M**anagement **S**ystems (GDBMS),

978-0-7695-4956-9 2013
U.S. Government Work Not Protected by U.S. Copyright
DOI 10.1109/SC.Companion.2012.160

or simply **Graph Databases**. We define GDBMS as the combination of the system architecture and software for management and analysis of massive graphs.

III. Graph Database Implementation Issues

A graph database manages the data entities like nodes, edges and properties of a graph using a graph data model. A graph database system should have several features to be able to manage, manipulate and query data. It should efficiently store and index data for faster access, apply concurrent updates for data accuracy, synchronize data between persistent and main memory, provide a query language to translate user defined queries into executable queries, and perform optimizations for efficient query processing. In this section, we present the issues in the design of a high-performance graph database. In effect, we discuss the features that constitute the building blocks and tradeoffs that govern the design of an efficient distributed graph database system.

A. Graph Data Storage and Representation

A graph database should provide organization of graph data using low-level data structures that define a data representation and manipulation interface. This process involves definition of basic entities in the graph representation, identification of a suitable data structure for physical organization of data on disk, and providing basic data manipulation operations to query and update the data.

There are many ways to store the graph itself. A matrix representation provides fast access to data at the cost of huge memory requirements, whereas lists require less memory but slower access to data. Due to this reasoning, lists are preferred for sparse data and matrices are preferred for dense data representation. Adjacency list, incidence list, adjacency matrix, incidence matrix, and laplacian matrix representations allow storing directed graphs. For distance matrix and degree matrix, this information needs to be stored in an additional data structure. Additionally, data can be stored in row-based or column-based stores in relational database systems. Data is also stored as tables of triples *(subject,predicate,object)* in RDF databases.

B. Data Management

Once the data representation and storage mechanisms are in place, it becomes important to define techniques to efficiently manage and query the data. This section describes graph data management using database principles.

1) Persistency and Transactional Models: Any database system can be divided in two broad categories based on the mode of data storage. The data can be stored on disk (persistent) and loaded in main memory at the time of querying; or can be stored in main memory itself (in-memory).

A database that supports transaction processing needs to meet the ACID properties (**A**tomicity, **C**onsistency, **I**solation, **D**urability) [1]. A graph database can be fully transactional(guarantees all ACID properties), partial transactional (supports only some of ACID properties with at least a minimal support for atomicity and consistency), non-transactional (no guarantees for data reliability and integrity) and distributed transactional (supports ACID in compliance with the CAP theorm -**C**onsistency, **A**vailability, **P**artitioning [2]).

2) Distributed Data Management: A distributed graph database serves as a parallel computation platform for processing queries in parallel on massive scale graphs. The need for parallel computation arises from the need to support a huge dataset for offline analytics or for providing online ad-hoc query execution environment [3] for a large number of users. The use of distributed relational databases for managing massive graphs is inefficient [4] due to the overhead in processing graph structure, search and mining queries.

Data distribution using the partitioning principles of the relational model results in excessive overhead in communication across the servers where the queries require data from more than one server. This communication can be minimized by using efficient graph partitioning approaches [5], [6] to distribute data across servers for parallel computation. An efficient graph data distribution and querying engine also needs to preserve locality in graph data access across servers and hence needs to define data caching mechanisms.

C. Query Processing and Optimization

Query processing techniques for a graph database are significantly different than those for a relational database. To retrieve the underlying data we need efficient access methods for the graph data structures. The graph queries are mostly of the reachability or connectedness types and hence require an algebra different from traditional relational databases. We need to have parallel query execution mechanisms in place to process queries over distributed graph databases with high performance and availability of data.

Query processing and optimization techniques for a distributed graph database aim to minimize the total operating cost of executing a query, which is the sum of processing costs on individual nodes and the total communication cost across the system. The query optimization [7] schemes choose a query plan that minimizes the costs discussed above. Data shipping (information propagation) and query shipping (query propagation) [3] are popular techniques for query placement across the partitions. Query optimization can be achieved with efficient indexing schemes as well. In the design of graph databases, defining indexing techniques is of utmost importance due to the complex structures and massive amount of data to be analyzed. A graph database should implement an indexing strategy to support indexing of both nodes and edges.

Research in the design of parallel graph processing techniques has been done to demonstrate performance gains in typical graph operations such as breadth-first search (BFS) [8], finding spanning trees [9], shortest path algorithm [10], graph coloring schemes [11], partitioning schemes [12], [6], etc. But these techniques work on graph structures that do not use the transactional power of a database system and do not define any data management or querying models.

It is a challenge to support parallel execution of these queries for massive graphs in graph databases. Some of these query types like reachability, keyword search, and pattern matching are online queries and ad-hoc queries that result in random disk accesses; while others like graph clustering, classification, topological, geometric and metric operations are examples of offline analytical query processing on graph databases. Most of the graph databases like Neo4j [13], InfiniteGraph [14] have a navigational query processing API and lack a declarative query model. Graph computation platforms like Pregel [4], etc. execute queries in batch and do not provide any online query processing interface.

D. Performance

Various metrics and benchmarks have been defined and used in evaluating the performance of relational database systems. These measurements quantify each of the key characteristics to collectively identify the performance capabilities of a database system. We need to use similar measurements for benchmarking and evaluating the performance of a graph database. Data analysis and online querying for many application domains like social networks, information networks, biological and chemical networks, technological networks, etc. should be done on state-of-the-art graph database technologies for high performance and high throughput for the applications.

1) Data Size: In relational databases, the data size of a database instance is governed by the schema and table sizes. In graph databases, with no notion of a structured schema and storage not confined to table records, the data size depends on the graph entities - nodes, relations, properties and types.

2) Scalability: Scalability of a system is determined by its capacity to grow with growing data sizes. In database systems, the scalability is a measure of its speedup and scaleup capabilities [15]. In the design of distributed graph database systems, it is important to meet system scaleup and speedup requirements to handle an increasing size of graph structures (as in the case of social and information networks). For most graph databases, high scalability comes from defining clustered deployment with high throughput across the system.

3) Reliability and High-Availability: High availability for a system can be achieved by providing modularity, redundancy and fault-tolerance [16]. Traditionally, this has been achieved by means of various replication and clustering strategies [17] and a communication and synchronization system.

IV. Graph Databases for High performance Computing

The design of graph databases gained a lot of momentum first in the industry and is an evolving topic in the field of research. BigData [18] gives a distributed high-performance architecture for computing petabyte scale massive data analysis on commodity hardware. DEX [19] demonstrates high-performance results on the R-MAT benchmark [20]. FlockDB [21] is a relatively new database and provides a scalable and distributed environment for online, low-latency and high-throughput applications. Since it is built on top of MySQL, it provides a high-performance computation platform with all of MySQL capabilities. Data Analytics Supercomputer [22] is a highly scalable and high-performance distributed computing platform for data-intensive applications. STINGER [23] is a high-performance graph analysis platform on massively multi-threaded architecture.

The high-performance computing community has made significant adavancements in introducing new architectures for petabyte scale massive graph analysis. However, the graph databases have not been able to exploit the underlying architectures till date. There is much scope in combining the research in these fields in the areas of data modeling, data distribution schemes, parallel query processing and optimization techniques, and definition of benchmarks.

We need to define data models that can provide flexibility to define higher order graphs, power law graphs, hypergraphs. etc. and also manage the metadata for these graphs. Researchers could consider incorporating concepts from data models in object-oriented databases and XML databases to make graphs richer in information content. In the implementation of distributed graph databases, the foremost challenging task is to define efficient data partitioning schemes for parallel query execution.

The choice of an appropriate transactional model is made difficult by the need to distribute and query highly associative data in the graphs. In the context of real-time applications like traffic control, the need to query or frequently update graph entities residing on multiple nodes in the cluster needs solutions that combine ideas from database transaction processing, concurrency control, as well as communication systems research.

The research in this field needs to address scalability issues not only due to large data size, but also due to overly complex heterogeneous data sources [3]. The research in defining a *true database* benchmark for graph database systems is still open. We strongly believe there is a need to define benchmarks for graph databases which can be widely accepted as industry standard since graph database systems will soon become an important platform for performing data analysis and posing ad-hoc queries in a variety of applications.

References

[1] T. Härder and A. Reuter, "Principles of transaction-oriented database recovery," *ACM Comput. Surv.*, vol. 15, no. 4, pp. 287–317, 1983.

[2] S. Gilbert and N. Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services," *SIGACT News*, vol. 33, pp. 51–59, jun 2002. [Online]. Available: http://doi.acm.org/10.1145/564585.564601

[3] INRIA(Sophia-Antipolis-Mediterranee) and UPC. (2012, October) Scaling up graph databases for network-centric applications (picasso and color projects). [Online]. Available: http://www.dama.upc.edu/research-2/color

[4] 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 ACM SIGMOD International Conference on Management of data*, ser. SIGMOD. New York, USA: ACM, 2010, pp. 135–146. [Online]. Available: http://doi.acm.org/10.1145/1807167.1807184

[5] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM J. Sci. Comput.*, vol. 20, pp. 359–392, December 1998. [Online]. Available: http://dx.doi.org/10.1137/S1064827595287997

[6] K. Schloegel, G. Karypis, and V. Kumar, "Sourcebook of parallel computing," J. Dongarra, I. Foster, G. Fox, W. Gropp, K. Kennedy, L. Torczon, and A. White, Eds. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003, ch. Graph partitioning for high-performance scientific simulations, pp. 491–541. [Online]. Available: http://dl.acm.org/citation.cfm?id=941480.941499

[7] N. Kabra and D. J. DeWitt, "Efficient mid-query re-optimization of sub-optimal query execution plans," in *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, ser. SIGMOD. New York, USA: ACM, 1998, pp. 106–117. [Online]. Available: http://doi.acm.org/10.1145/276304.276315

[8] A. Yoo, E. Chow, K. Henderson, W. McLendon, B. Hendrickson, and U. Catalyurek, "A scalable distributed parallel breadth-first search algorithm on bluegene/l," in *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, ser. SC '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 25–. [Online]. Available: http://dx.doi.org/10.1109/SC.2005.4

[9] D. A. Bader and G. Cong, "A fast, parallel spanning tree algorithm for symmetric multiprocessors (smps)," *J. Parallel Distrib. Comput.*, vol. 65, pp. 994–1006, September 2005. [Online]. Available: http://dx.doi.org/10.1016/j.jpdc.2005.03.011

[10] J. Crobak, J. Berry, K. Madduri, and D. Bader, "Advanced shortest paths algorithms on a massively-multithreaded architecture," in *Parallel and Distributed Processing Symposium, 2007. IPDPS. IEEE International*, March 2007, pp. 1 –8.

[11] D. Bozdağ, U. Catalyurek, A. H. Gebremedhin, F. Manne, E. G. Boman, and F. Özgüner, "A parallel distance-2 graph coloring algorithm for distributed memory computers," in *Proceedings of the First international conference on High Performance Computing and Communications*, ser. HPCC. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 796–806. [Online]. Available: http://dx.doi.org/10.1007/11557654_90

[12] K. D. Devine, E. G. Boman, R. T. Heaphy, R. H. Bisseling, and U. V. Catalyurek, "Parallel hypergraph partitioning for scientific computing," in *Proceedings of the 20th international conference on Parallel and distributed processing*, ser. IPDPS'06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 124–124. [Online]. Available: http://dl.acm.org/citation.cfm?id=1898953.1899056

[13] Neo4j. (2012, October). [Online]. Available: http://neo4j.org/

[14] InfiniteGraph(Objectivity). (2012, October). [Online]. Available: http://www.infinitegraph.com/

[15] D. DeWitt and J. Gray, "Parallel database systems: the future of high performance database systems," *Commun. ACM*, vol. 35, pp. 85–98, June 1992. [Online]. Available: http://doi.acm.org/10.1145/129888.129894

[16] J. Gray and D. P. Siewiorek, "High-availability computer systems," *Computer*, vol. 24, no. 9, pp. 39–48, sep 1991. [Online]. Available: http://dx.doi.org/10.1109/2.84898

[17] M. T. Özsu and P. Valduriez, *Principles of Distributed Database Systems 3rd Edition*. Reading, Mass.: Springer, 2011.

[18] BigData(SYSTAP). (2012, October). [Online]. Available: http://www.systap.com/bigdata.htm

[19] DEX. (2012, October) (sparsity technologies). [Online]. Available: http://www.sparsity-technologies.com/dex

[20] D. Bader, J. Gilbert, J. Kepner, and K. Madduri, "Hpc scalable graph analysis," October 2012. [Online]. Available: http://www.graphanalysis.org/index.html

[21] FlockDB(Twitter). (2012, October). [Online]. Available: https://github.com/twitter/flockdb

[22] LexisNexis. (2012, October) Data analytics supercomputer (das). [Online]. Available: http://www.lexisnexis.com

[23] D. Ediger, K. Jiang, J. Riedy, and D. Bader, "Massive streaming data analytics: A case study with clustering coefficients," in *Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, april 2010, pp. 1 –8.