Graph Database Applications and Concepts with Neo4j

Justin J. Miller
Georgia Southern University
jm10197@georgiasouthern.edu

ABSTRACT

Graph databases (GDB) are now a viable alternative to Relational Database Systems (RDBMS). Chemistry, biology, semantic web, social networking and recommendation engines are all examples of applications that can be represented in a much more natural form. Comparisons will be drawn between relational database systems (Oracle, MySQL) and graph databases (Neo4J) focusing on aspects such as data structures, data model features and query facilities. Additionally, several of the inherent and contemporary limitations of current offerings comparing and contrasting graph vs. relational database implementations will be explored.

Keywords

Graph database, relational database, data model, property graph, vertex, edge, node, relation, traversal, attribute, ACID, locality, collaborative filtering, content based filtering

INTRODUCTION

The relational database model has been around since the late 1960s [4]. It has proven to consistently provide persistence, concurrency control, and integration mechanisms. Relational databases maintain tables which are defined by sets of rows and columns. A row can be perceived as an object while columns would be attributes/properties of that objects [15]. One of the weaknesses of the relational model is its limited ability to explicitly capture requirement semantics [14]. Big data problems involving complex interconnected information have become increasingly common in the sciences. Storing, retrieving, and manipulating such complex data becomes onerous when using traditional RDBMS approaches. Schema based data models by their very definition put in place limits on how information will be stored. There is an involved manual process to redesign the schema in order to adapt to new data. Where the RDBMS is optimized for aggregated data, graph databases such as Neo4j are optimized for highly connected data.

A graph is a data structure composed of edges and vertices [2]. Graph database technology is an effective tool for modeling data when a focus on the relationship between entities is a driving force in the design of a data model [3]. Modeling objects and the relationships between them means almost anything can be represented in a corresponding graph. A common graph type supported by most systems is the property graph. Property graphs are attributed, labeled, directed multi-graphs [2]. Figure 1 provides a visual example of a property graph which represents interactions between people and objects. A benefit to the multi graph is that it is the most complex implementation because every other type of graph consists of subsets of the property graph implementation. This means a property graph can effectively model all other graph types. The graph database is optimized for the efficient processing of dense, interrelated datasets [2]. This design allows the construction of predictive models, and detection of correlations and patterns [3]. This highly dynamic data model in which all nodes are connected by relations allows for fast traversals along the edges between vertices. A particular benefit is the fact that traversals are localized and do not have to take into account sets of unrelated data. A problem that is inherent in SQL [15].

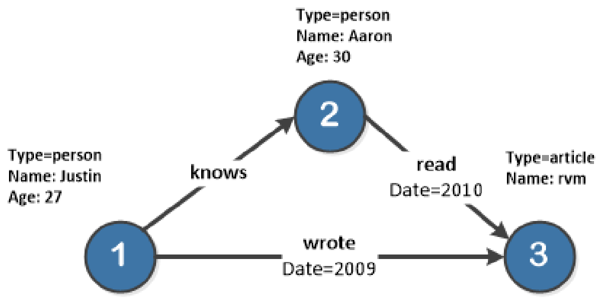

Figure 1 - The Property Graph Model

Despite the limitations involved in using a schema there are many benefits some which outweigh the restrictions implied by the RDBMS. These databases are well supported by their respective vendors, they are battletested and their strengths/limitations are well known, they have a common query language (SQL), there is a wide talent pool of trained professionals to pull from, and these platforms are extremely well documented. In contrast graph databases face a particular set of challenges inherent in their design.

CORE CONCEPTS

A graph is an object which contains nodes and relationships. Nodes have properties and are organized by relationships which also have properties. A traversal navigates a graph and identifies paths which order nodes. Figure 2 illustrates the individual components of a graph and gives a visual representation of how they relate to one another.

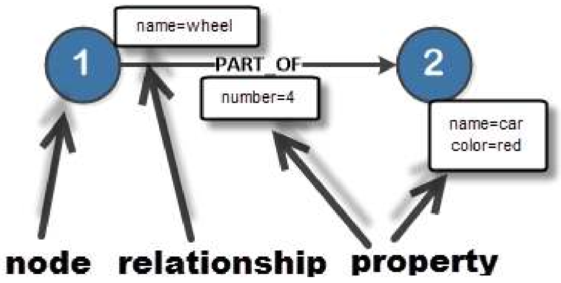

Figure 2 - Core Graph Concepts

Graph Processing vs. Graph Storage

Although they share a data model with the GDB, graph processing frameworks are designed to solve a different type of problem. This distinction is qualified in the comparison of online analytical processing (OLAP) and online transaction processing (OLTP). OLAP graph systems, such as Google's Pregel or Apache Hama, strive to swiftly answer multi-dimensional analytical queries focusing on high performance processing. On the other hand, the goal of OLTP systems, like Neo4j, is to facilitate and manage transaction oriented queries [6].

Pregel is a distributed computer framework based on Valiant's Bulk Synchronous Parallel computing techniques for massive computations on graphs and matrices [12]. Pregel focuses entirely on the efficient processing of large graphs. It makes use of graph based algorithms to facilitate highly parallel solving of problems like page rank, shortest path, and bipartite matching [11]. In contrast, Neo4j strives to be the data backend for transaction based applications.

APPLICATIONS OF GRAPH DATABASES

Graph databases really shine when working in areas where information about data interconnectivity or topology is important. In such applications the relations between data and the data itself are usually at the same level [1]. Many companies have developed in-house implementations in order to cope with the need of graph database systems. Examples would be Facebook's Open Graph, Google's Knowledge Graph, Twitter's FlockDB, any many more. The following are a few examples of systems that would benefit greatly from graph database approach. RDBMS can be used for such systems but in a much more limiting and expensive way (expensive meaning processing power caused by recursive JOINs for friend of a friend type problems).

Social Graph

The social graph leverages information across a range of networks in order to quantify the relationships between individuals. Figure 4 represents a small social graph between friends and tracks the friendships, name, age, and favorite animal of each person. Figure 4 gives an inexpensive solution to finding the friends of Justin's friends. In traditional relational databases it is quite challenging to deal with recursive data structures where each traversal along an edge in the graph would be a join. The friend of a friend problem is an example of a problem that would require far less work for a graph database when compared to RDBMS. Joins are expensive on an RDBMS while a traversal across an edge costs very little.

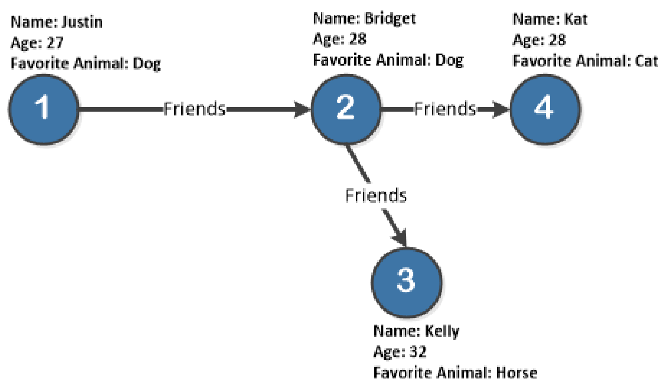

Figure 3 - Naive friend-of-friend graph

Recommender Systems

Recommender systems advise users on relevant products and information by predicting interest based on various types of information [10]. There are two correlation methods that naturally apply to the graph data model. Item-to-item correlation recommends products based on associations between new items and items the user previously expressed interest in. This is what is known as content based filtering [16]. In contrast, user-to-user (aka. Collaborative filtering) recommender systems give predictions based on correlations made by looking at how the user interacts with the system. A customer profile is built on past behavior and similarity functions are leveraged to find similar customers [10].

Combing both of these recommendation systems offers a very powerful way to wield large amounts of data effectively. Such systems allow Google to present only the most relevant ads their user is most likely to enjoy. An example of this system would be Facebook's friend suggestion functionality or Amazon's item suggestion engine.

Bioinformatics

Graph databases have seen extensive application in the field of bioinformatics. Bioinformatics uses graph databases to relate a complex web of information that includes genes, proteins and enzymes. A prime example is the Bio4j project. Bio4j is a framework powered by Neo4j that performs protein related querying and management. The project has aggregated most data available from other disparate systems (Uniprot KB, Gene Ontology, UniRef, RefSeq, ext.) into one golden source. The project is open source and the intention is for it to be easy to expand upon. This is made easier because the data is stored so that it semantically represents its own structure.

SOFTWARE DESIGN

It is common practice in software design to model the entities of a proposed project as a graph. Entities are proposed and assigned attributes. Those entities are then connected to other entities in various relations. Modeling in this way is a very natural process in contrast to the design considerations for the RDBMS.

When using an RDBMS, developers are forced to make software design considerations due to the inflexible nature of changing data model. Every new entity becomes a commitment to a table structure that limiting expansion of software functionality due to modeling limitations. For example, the Model View Controller (MVC) design pattern dictates that the presentation (view), logic (controller), and data (model) must all be completely separate from one another. The idea that the data model must be a separate entity tightly controlled with expected behavior is one inherent in the RDBMS approach. I believe this "separation of church and state" has occurred because of the rigidity and complexity involved in utilizing an RDBMS. Change must be tightly controlled and loop holes must be jumped through in the code to attempt to make the RDMBs match object and relation requirements required of it. This is the most blatant limitation of the RDBMS and has driven the creation of the graph model.

Frameworks have been developed to attempt to make RDBMS feel more natural. These frameworks are referred to as Object-Relational Mapping Systems (ORMs). An example of an ORM would be the Hibernate framework for Java. ORMs attempt to abstract the data model into objects and relations in order to make software development seem more natural. Preferably, the data model should not be a driving factor in software design. Instead software design requirements should be the driving force behind the design of the data model. The database should support flexibility and this is where the GDB excels.

Unchaining software developers from their RDBMS bonds could lead to a much more flexible and dynamic approach to software design. This flexibility lends itself well to the agile development methodology where requirements and solutions are supposed to evolve constantly. One powerful software design tool for Java and Neo4j is Spring Data. Spring data enables POJO based development for the Neo4j GDB. It maps annotated entity classes to the Neo4j Graph Database with advanced mapping functionality.

DATA QUERY AND MANIPULATION

A query language is a collection of operators or inference rules that can be applied to any valid instance of the data structures types of the model, with the objective of manipulating and querying data in those structures in any combinations desired [4]. RDBMS systems use a shared standard known as SQL (Structured Query Language) for data inserts, update and delete, query, and schema creation and modification. SQL is based upon relational algebra and tuple relational calculus. This general consistency in approach makes concepts learned from different RDBMS implementations extremely portable. There is a similar unified approach to working with graph databases.

Effectively retrieving information out of a graph requires what is known as a traversal. A graph traversal involves "walking" along the elements of a graph [2]. The traversal is a fundamental operation for data retrieval. One major difference in between a traversal and a SQL query is that traversals are localized operations. There is no global adjacency index instead each vertex and edge in the graph store a "mini-index" of the objects connected to it. This means that the size of the graph has no performance impact upon a traversal and the expensive grouping operations performed in SQL JOIN statements are unnecessary. It is important to note that global indexes do exist in Neo4J but they are only used when trying to find the starting point of a traversal. Indexes are required in order to quickly retrieve vertices based on their values. They provide a starting point at which to begin a traversal. Without indices determining if a particular element has a particular property would require a linear scan of all elements at a cost of O(n), n being the number of elements in the collection. Alternatively the cost of a lookup on an index is much lower at O(log2n) [2].

The world of graph databases has yet to standardize on a language for traversal and insertion. This lack of standardization has led to vastly different implementations and frameworks for data interaction. The implementations that are available range from the Neo4j's Java API, the REST interface, a DSL language named Gremlin, and another called Cypher. Gremlin implements traversals through a system called piping. Each piece of the query is a step to the next in a progressive fashion. However, Cypher attempts to use more of a keyword system trying to be more like SQL. This is weakness when compared to the more mature RDBMS SQL. There is a lack of consistency that requires one to learn all implementations before understanding what approach is most suitable to the problem.

Gremlin and Cypher are the two primary languages used to traverse Neo4J graphs. Gremlin is a domain-specific (DSL) programming language[1] focusing on graph traversal and manipulation. Gremlin is implemented on top of the Groovy programming language and has been adopted by most graph database vendors. Gremlin is backed by the Blueprints Java API and can work directly with Blueprints when required. Groovy operates on the JVM and is closely tied to Java. Gremlin seeks to be a standard language that can operate on any of the major graph database implementations.

Cypher is a declarative graph query language inspired by SQL that seeks to avoid the need to write traversals in code. Cypher is still under intense development with support recently being added for graph manipulation instead of just query ability.

Figure 5 provides a data model for the following example of traversing a graph model that stores information about users, their occupation, movies they have rated, and the genera's of the movies. The model will be a basis for example Gremlin queries to illustrate the power of traversals.

1

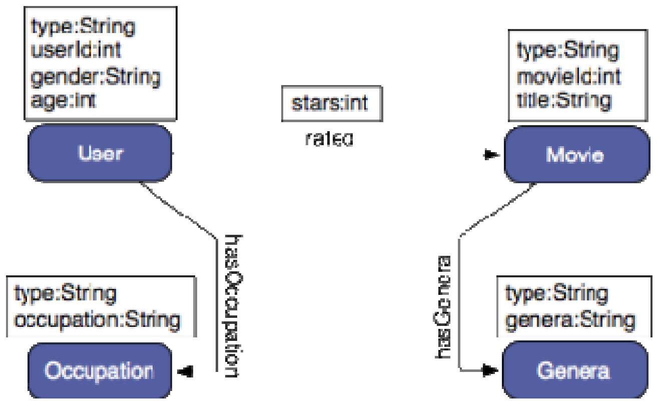

Figure 4 - MovieDB Data Model

[7]

Which users gave Men in Black 5 stars and what other movies did they give 5 stars to? (Only return 5 results)

Query:

```
g.V.filter{it.title=="MeninBlack(1997)"}.inE('rated').filter{it.getProperty('stars')==5}.outV.outE('rated').filter{it.getProperty('stars')==5}.inV.title[0..4]
```

Returns:

```
==> Saving Private Ryan (1998)
==> Die Hard (1988)
==> Braveheart (1995)
==> Enemy of the State (1998)
==> Jurassic Park (1993)
```

Walking the Graph Step-by-Step:

1. Start from Men In Black - g.V.filter{it.title == " Men in Black (1997)"}.
2. Get the incoming rated edges - inE('rated')
3. Filter out those edges whose star property is not 5 — filter{it.getProperty('stars') ==5}
4. Get the tail user vertices of the remaining edges – outV
5. Get the rating edges of those user vertices- out(‘rated’)
6. Filter out the edges without a rating of 5 – filter{it.getProperty(‘stars’) == 5}
7. Get the head movie vertices of the remaining edges – inV
8. Get the string title property of those movie vertices – title
9. Emit only the first 5 titles – [0..4]

This example illustrates one of the strengths graph databases have over the RDBMS, the power of recommendations formed from association. The above is an example of what is called collaborative filtering. Collaborative filtering is a technique of automating predictions of user interests by collecting information from a large sample set. This is different from content based filtering which focuses more on analyzing item characteristics, finding similar items, and recommending those items. An example of this would be seeing someone likes a movie and using that information to find movies in the same genre and recommending those movies to the current user. Combining both of these techniques can yield powerful results.

RELIABILITY

In database storage, **ACID** (**atomicity, consistency, isolation, durability**) is a set of properties that guarantee that transactions are processed reliably. Proper ACID behavior is one of the primary requirements of data reliability. Much like Oracle Neo4j is fully ACID compliant [6]. Lack of ACID compliance has prevented many systems from being considered production ready for the enterprise.

- Atomicity: If any part of a transaction fails, the database state is left unchanged.
- Consistency: Any transaction will leave the database in a consistent state.
- Isolation: During a transaction, modified data cannot be accessed by other operations.
- Durability: The DBMS can always recover the results of a committed transaction.

Transactions

There are some important details about transaction management in Neo4j.

- All changes to Neo4j data are wrapped in transactions.
- Queries are not protected from modification by other transactions.
- Only write locks are acquired and held until the end of a transaction.
- Locks are acquired at the node and relationship level.
- Deadlock detection is built into the core transaction management system.

High Availability

The Neo4j HA solution follows a master-slave design for coordination and replication. The caveat to this setup is that a write to one slave is not immediately synchronized with all other slaves. This means there is a danger of losing consistency for a short time. Because of this limitation HA is promoted mainly as a solution for increasing the capacity for reads instead of writes. Neo4j depends on Apache Zookeeper to coordinate nodes. Zookeeper controls coordination of all interaction between nodes in the cluster. Figure 6 illustrates the master, slave, and router relationship used by Neo4J.

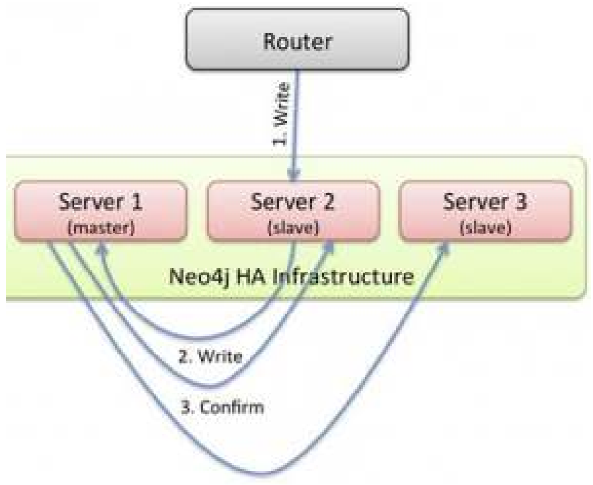

Figure 5 - Neo4j HA Implementation

Backups

Backups are a required part of any enterprise data storage solution and Neo4J Enterprise Edition supports both full and incremental backups. Using the command line interface you can specify the type of backup (full or incremental) and the cluster domain names. Backups are only supported in the enterprise version of Neo4J.

Security

Graph databases have brought about a new way of modeling and traversing interconnected data that is unparalleled in information storage. With the advent of production grade systems such as Neo4j using GDB problems can be addressed without resorting to a limiting implementation on a RDBMS. The graph database has a natural application for biological, semantic, network, and recommender systems that require the type of data model only they can offer.

Is this graph database ready to completely replace the RDBMS? That this is the wrong question to ask. RDBMS are suitable for the majority of data storage needs. They are well documented, supported, and proven stable. The decision to use a graph database should not be made based on a desire not to use RDBMS. The requirements of the system should be taken into consideration if there is a need for a dynamic data model that represents highly connected data then a graph database is the best solution. For the majority of common problems the RDMBS is a suitable solution which many more architectural options than the GDB realm can currently offer.

Conclusion

Graph databases have brought about a new way of modeling and traversing interconnected data that is unparalleled in information storage. With the advent of production grade systems such as Neo4j using GDB problems can be addressed without resorting to a limiting implementation on a RDBMS. The graph database has a natural application for biological, semantic, network, and recommender systems that require the type of data model only they can offer.

Is this graph database ready to completely replace the RDBMS? That this is the wrong question to ask. RDBMS are suitable for the majority of data storage needs. They are well documented, supported, and proven stable. The decision to use a graph database should not be made based on a desire not to use RDBMS. The requirements of the system should be taken into consideration if there is a need for a dynamic data model that represents highly connected data then a graph database is the best solution. For the majority of common problems the RDMBS is a suitable solution which many more architectural options than the GDB realm can currently offer.

REFERENCES

1 C. Berge, "The theory of graphs and their applications", Wiley (1962)

2 Rodriguez, M.A., Neubauer, P., "Constructions from Dots and Lines" Bulletin of the American Society for Information Science and Technology, American Society for Information Science and Technology, volume 36, number 6, pages 35-41, doi:10.1002/bult.2010.1720360610, ISSN:1550-8366, August 2010.

3 Silvescu, Adrian and Caragea, Doina and Altramentov, Anna *Graph Databases.* Iowa State University.

4 Edgar F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377-387.

5 B. Amann and M. Scholl. Gram: a graph data model and query languages. In *ACM Hypertext*, 1992.

6 Webber, Jim. Graph Processing versus Graph Databases, World Wide Webber. Retrieved August 24, 2011.http://jim.webber.name/2011/08/graph-processing-versus-graph-databases/

7 Codd E.F., Codd S.B., and Salley C.T. (1993). "Providing OLAP (On-line Analytical Processing) to User-Analysts: An IT Mandate". Codd & Date, Inc. Retrieved 2008-03-05..

8 R. Angles and C. Gutierrez, "Survey of graph database models," in ACM Computing Surveys Vol. 40 Issue 1, February 2008.

9 M. Newman. Why social networks are different from other types of networks. Physical Review E, 2003

10 HUANG, Z., CHUNG, W., ONG, T.-H., AND CHEN, H. 2002. A graph-based recommender system for digital library. In *Proceedings of the 2nd ACM/IEEE-CS Joint Conference on Digital Libraries* (Portland, Ore.). ACM, New York, 65–73

11 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 PODC, 2009

12 Leslie G. Valiant, A Bridging Model for Parallel Computation, Comm. ACM 33(8), 1990, 103-111

13 Vicknair, C. ,A Comparison of a Graph Database and a Relational Database. ACM SE 10 Proceedings of the 48th Annual Southeast Regional Conference Article No. 42

14 Hull, R. AND King, R. 1987. Semantic database modeling: Survey, applications, and research issues. *ACM Comput. Surv. 19,* 3, 201-260.

15 Rodriguez, M.A., Neubauer, P., "*The Graph Traversal Pattern*," Graph Data Management: Techniques and Applications, eds. S. Sakr, E. Pardede, IGI Global, ISBN:9781613500538, August 2011.

16 B. Sarwar, G. Karypis, J. Konstan and J. Riedl. "Itembased Collaborative Filtering Recommendation Algorithms." *Proc. 10th International World Wide Web Conference, 2001*