ECE464 Databases – Problem Set 3: Appropriate DBMS for Potential Issues

Jongoh (Andy) Jeong

December 3, 2019

Abstract

Effective practices of appropriate database management systems (DBMS) against some potential issues that often arise in project development stages are crucial for performance efficiency in domains, such as maintenance and security. There are a number of problems that are commonly encountered upon designing a database system, and certain DBMS systems perform relatively better in handling such concerns. To determine which DBMS would be adequate for managing certain expected issues with regards to manipulating internet data, this paper surveys and analyzes some graph-like DBMS that would be good for certain situations in dealing with issues presented by Florescu et al. [6].

Contents

1 Introduction

A database is a collection of data, often describing the activities of related organizations. A database is structured by entity, such as students, faculty, and courses, and relationships between entities, such as course enrollments of students, courses taught by faculty, and times at which courses are run. In order to effectively make use of a database, a type of database management system (DBMS) is employed to assist in maintaining and utilizing large collections of data. By having a software for storing and retrieving users' data with appropriate security measures, one can achieve some advantages over simply storing data in an existent filesystem and writing application-specific code for management [1].

In the scope of our interest in working with the internet data (e.g. the World Wide Web, or WWW), we want a database to be capable of accommodating their characteristics. The domain of the WWW allows any type of information without much interference or coordination with other people, and the speed at which it grows has been increasing rapidly. Francis and Sato of NTT Software Lab suggest that a system using such global internet information should (1) follow the style of the Internet in a distributed, autonomous and scalable manner, and (2) be flexible enough for partial failures [7]. In addition, Florescu et al. notes that a large website should be viewed not just as a database, but an information system built around one or more databases with an accompanying complex navigation structure, implying a different data model than what we traditionally observed in relational models [6].

2 Growing Complexity in Web Data

As the fields of database applications are growing, the information increasingly relies more on the relations of data than on the entities themselves. In addition, the growing scale of data and the complexity of such graph-like data become challenging to manage as conventional data models may not always work efficiently [3].

Since typical web-based data are object-oriented due to the nature of the document object model (DOM) on which data resides, it is intuitive to treat data as graph-like structure. Otherwise, data can have either structured relational, semi-structured, or unstructured data models. Some relational databases, such as Oracle and PostgreSQL, support a mixture of structured relational data, XML- or JSON- format semi-structured data, and full-text indexing for unstructured data. In simpler cases, a key-value store can also work well, or if data is formatted as a document format, non-relational databases (NoSQL) such as MongoDB should be considered first because its purpose is to serve flexible data types, including structured ones. As an approach to dealing with web data which tends to be a bit more complex due to inter-site dependencies, one can start to look at graph database models.

Traditional relational databases faced with limitations in efficiency and covering requirements of current application domains led the development of NoSQL. NoSQL database can typically be described by four categories: Wide-column stores (following the BigTable model from Google), Document stores (oriented to store semi-structured data like MongoDB), Key-value stores (implementing a persistent map from key to value for data indexing and retrieval, such as BerkeleyDB), and Graph databases (oriented to store data in a graph-like structure).

In many cases, the internet data are often conglomerates of a website's database entries, HTML, XML pages and other structured files that have complex relationships among entities. The structure of such collected data is irregular and there is no fixed schema across multiple sources, and thus the

representation of some attributes may differ from source to source. This *semi-structured data* is often encountered in the scientific community, in which data (objects) are collected over heterogeneous sources, as Florescu et al. notes [6].

In the context of modelling web pages (either from multiple or single sites) and the links between them, Florescu et al. notes that a natural way of modeling in this case is with a *labeled graph data model*, in which nodes represent web pages, arcs represent links between pages, and the labels on the arcs would represent attribute names. An important feature of this graph data model they suggest is the support for regular path expression queries over the graph, enabling posing navigational queries over the structure [6].

A graph data model is often known to allow for a more natural, generalized modelling of graph-like data having multiple relationships. As it is intuitive from the term *graph*, it seems to make more sense to 'fit' such data into a graph-like generalized model, whose data structures for the schema and instances are directly modeled as graphs or their generalizations, and the graph-oriented operations along with type constructors manipulate such data efficiently in queries [3]. Considering the aforementioned nature of common web data types, it would be of our best interest to look more into representing such loosely defined semi-structured data with a graph data model. Florescu et al. presents a number of different ways of data representations in this context, and they are discussed in the subsequent sections [6].

3 Graph Database

3.1 Overview

The data structures of most inter-related internet data may be generalized by graph structures consisting of entities or objects, specifically graphs, nodes and edges. A basic graph structure is described by a simple flat graph defined as a set of labeled/unlabeled nodes, or vertices, connected by directed/undirected and/or labeled/unlabeled edges, or binary relations over the set of nodes. To this simple graph, Angles, in his comparison of graph databases, claims that adding attributed nodes and edges can serve to improve the speed of retrieval for the data directly related to a give node. In addition, nested graphs – graphs whose nodes can form graphs themselves, called hypernodes – can model attributed graphs, whose nodes and edges can contain attributes that describe their properties, and hypergraphs, which allow an edge to relate an arbitrary set of nodes called hyperedge) [3].

	Schema			Instance					
Graph Database	Node types	Property types	Relation types	Object nodes	Value nodes	Complex nodes	Object relations	Simple relations	Complex relations
AllegroGraph					•			•	
DEX	•		•	•	•		•	•	
Filament					•			•	
G-Store					•			•	
HyperGraphDB	•		•		•			•	•
InfiniteGraph	•		•	•	•		•	•	
Neo4j				•	•		•	•	
Sones					•			•	•
vertexDB					•			•	

Figure 1: Schema- and Instance- Level Entities,
supported by example graph databases [3]

Figure 1 illustrates the supported entity definitions by several currently available graph databases. At instance-level, an object node represents an instance of a node type, a value node represents an entity identified by a primitive value like name, a complex node represent a more complex entity, such as a tuple and a set, an object relation is an instance of a relation type, a simple relation represents a node-edge-node instance, and a complex relation is a relation with special semantics, such as grouping, derivation and inheritance [3].

One can clearly observe that all examples in Fig 1 support value nodes and simple relations as the two are the most primitive representations of graph data. One possible inherent issue in graph databases that use objects is that it requires APIs at the high level to represent entities and relations. Also, using such objects means that entities/relations can be represented by specific object ID, but not the value-name of the entity/relation – in which case additional explicit property is necessary to specify the entity/relation name [3]. With regards to sharding, a graph database can give some difficulty in scaling issue due to its inherent structure of being highly mutable, leaving no predictable lookup. There are also competing requirements when sharding a graph across physical instances. For instance, one would not want to place too many connected nodes on the same database when co-locating related nodes for traversal performance, such that it doesn't become heavily loaded. The ideal approach for this issue would be to balance a graph DB instances with a minimal point cut for a graph, with nodes placed in a manner that only a few relationships span shards [11].

	RDBMS	Graph DB
Structure	Table	Graph
Entity	Row	Node
Description	Columns and Data	Properties and values
Association	Constraint	Relationship
Retrieval	Joins	Traversals

Table 1: Comparison between RDBMS and Graph Database

While graph databases allow efficient data retrieval by fetching by the object ID of entities or relations, some current databases also support reasoning and analysis. For instance, AllegroGraph supports reasoning through its own Prolog (a logic programming language for AI and computational linguistics) implementation, and typically analysis is done with special functions, such as shortest path, which queries graph properties. In terms of query languages used by current graph databases, there is no 'standard' language most use. For instance, AllegroGraph supports SPARQL (standard query language for RDF, or a type of graph database that stores data in a network form) based on graph pattern matching, G-store and Sones use SQL-based languages with some customized for querying graphs, and Neo4J uses Cyper, a query language for property graphs. Such diversity in query languages and lack of well-defined standards may cause difficulties adapting to one, and also inconsistencies when used for retrieval, reasoning and analysis of the data [4, 3].

In order to serve its purpose, a graph database should provide external interfaces (UI or API), database languages for data definition, manipulation and querying, query optimizer, database engine, storage engine, transaction engine, and management and operation features (tuning, backup, recovery). The aforementioned examples of graph database – AllegroGraph, Neo4J, Sones – satisfy these in general. Its inherent graphical nature has an advantage of being capable of scaling up, and it is typically taken that it can be robust in recovery upon failures [3]. We will now discuss several popular graph databases – **Neo4J and HyperGraphDB** – as well as two of DBMS with graph data models – **Lore** – as suggested by Florescu et al. [6].

3.2 Neo4j

Neo4j is currently leading the market in graph databases and gaining popularity for its advantages in its features. Its competency over other available graph databases comes from a number of features and support. First, its data model is pretty flexible and simple, yet still powerful in a sense that it can be configured to the application-specific purposes for the company. Neo4j follows a data model called native property graph model, where a graph contains nodes, or entities, that are connected by their relationships store as properties, or key-value pairs. This model is not constrained to a fixed schema, so one can easily add, remove and update properties and schema constraints. For this reason, it allows to represent connected and semi-structured data like data from the web. With a rich user interface called Neo4j browser that can be widely used for queries, visualization and data interaction, it also provides high availability for large enterprise with transactional guarantees, while also giving real-time data results (See example in Table 2). This real time retrieval is possible through easy representation of and traversal through connected data. In addition, similar to RDBMS, it supports full ACID compliance. Another merit comes from its scalability and reliability. One can increase the number of transactions without reduced processing speed or data integrity as it replicates instances for security and reliability purposes, which is always good from

a company's standpoint. In terms of the query language, it provides a declarative query language, called Cypher, to represent the graph visually using an ASCII-art syntax. This language is easy to learn and adapt, and can be used to create and retrieve relations without using complex queries like joins or indexing. Its wide compatibility with most programming platforms like Java, Nodejs, PHP, Python, .NET is also an enticing supporting feature. The integration support with Apache Spark, Elasticsearch, MongoDB, Cassandra, and Docker and open-source license leaves the company with some freedom in resources and money issues [5, 10].

Depth	RDBMS exec time(s)	Neo4j exec time(s)	Returned Records
2	0.016	0.01	2500
3	30.267	0.168	110,000
4	1543.505	1.359	600,000

Table 2: Execution Time Comparison between RDBMS and Neo4j Graph DB [13]

3.3 HyperGraphDB

HyperGraphDB is a general purpose, open-source data storage mechanism based on a powerful knowledge management concept called directed hypergraphs. It is, simply put, a generalized graph of entities, where links/edges (relationships) can point to an arbitrary number of elements rather than two in a regular graph, and links can be directed by other links as well. Hypergrahps are useful in domains where many-to-many relationships are common, which could be interconnected webpages. Figure 2 illustrates a hypergraph of Alice and Bob as owners of three vehicles. Using a hypergraph to model these relations would only require a single hyper-edge, whereas a typical property graph would require six.

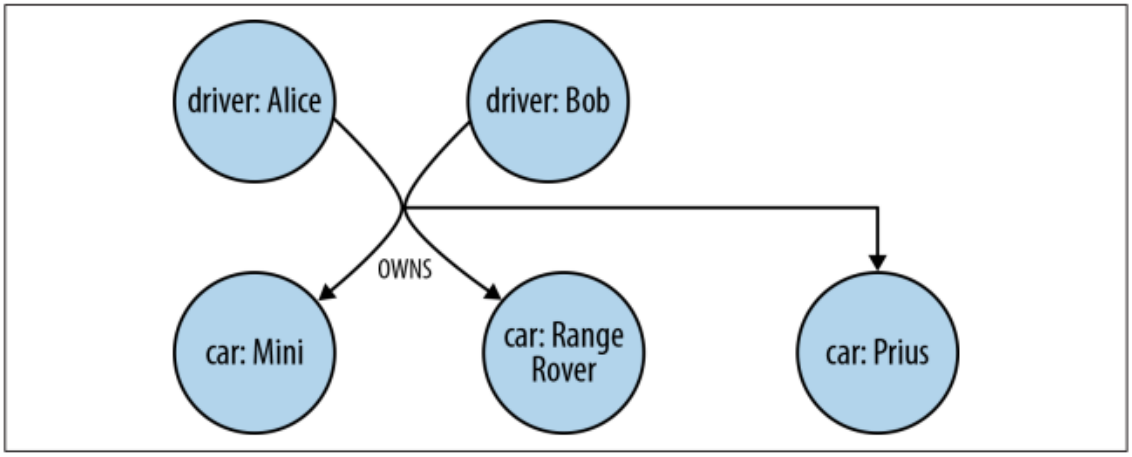

Figure A-7. A simple (directed) hypergraph

Figure 2: A simple directed hypergraph example [13]

HyperGraphDB itself is an embedded database that relies on a key-value store underneath. It

it currently also full-fledged object-oriented database based natively in Java, and its storage layout, indexing and caching are support graph traversals and pattern matching. It is a very versatile data model that is good at dealing with graph-oriented problems with large datasets, while also supporting object-oriented database properties (having well-developed type systems). Some other notable, unique features include: higher-order, n-ary relations and open-architecture in which the storage layout is open and documented with customizable indexing, type handling back-end storage and distribution algorithms [12]. Figure 3 shows some differences that set HyperGraphDB from Neo4j.

	HyperGraphDB	Neo4j
Data model	Graph-oriented	Graph-oriented, schema-less, NoSQL
Data storage	volatile memory	data on disk, volatile memory, memory-mapped file
Query Language	HGQuery-API	API calls, REST, Cypher
Data Types	Java Object	Java Primitive Array, Strings
Integrity	MVCC	ACID, Log Replication
Atomicity	Yes	Yes
Consistency	N/A	Yes
Isolation	N/A	Yes
Durability	N/A	Yes
Horizontal Scalability	N/A	No
Replication	N/A	Master-slave
Sharding	No	No
Cloud support	N/A	Yes
Community Support	N/A	Yes

Table 3: HyperGraphDB vs. Neo4j Comparison [8]

3.4 Lore

While there are options to use a graph database directly in dealing with web-queried data, another approach is enforcing an efficient query language on top of a conventional object-oriented DBMS, yielding a more flexible system suitable for managing both structured and semi-structured data. As summarized by Florescu et al., Lorel is a user-friendly, effective query language in the Object Query Language (OQL) style based on a labeled graph data model. It is distinguished from other query languages in that it extensively relieves the user from the strict typing of OQL, which is often inappropriate for semi-structured data like the web data. Also, it supports powerful path expressions, allowing a flexible form of declarative navigational access and making it suitable to use when the details of the structure are not known to the user. This specific declarative query language is originally developed for XML type data on Lore DBMS, which uses specialized indexing and a cost-based query optimizer to process each query as efficiently as possible. Lore DBMS also provides a new keyword-based search technique that exploits XML's data structure format [2, 9].

Currently, Lore DBMS exclusively supports Lorel language because it was initially developed for the purpose of being used for Lore. Lore supports multi-user, crash recovery, materialize views, bulk loading of sets of related XML documents, and a declarative update language. It also has external data manager allowing XML from external sources to be fetched in a dynamical manner and combine with local data during query processing. Using this XML-specific model, one can generally view any

set of XML documents as a directed, ordered graph of inter-related data elements and attributes because XML offers special and IDREF attribute types to allow any element to reference another [2, 9].

In the context of semi-structured data that we typically see in the internet, Lore leverages SQL's familiar syntax like SELECT to augment the language with powerful path expressions for specifying traversals through the XML graph, using Lorel. This approach allows to take advantage of how search engines (whose data are more rigidly structured than XML) efficiently use indexing on many documents for keyword-based searches and also advantage of the rich structure and data relationships realized by XML. Another useful feature of Lorel is its feature to help users write queries when they are not sure of the database's structure because Lore DBMS does not assume any particular structure or patterns exist within the XML data. In addition, it is useful to have Lorel handle cases when data is incomplete by ignoring the path expression and continues to process the rest of the query, though Lore does still have some warning in the system for system diagnosis [2, 9].

The main points of Lore DBMS are matching path expressions using a top-down execution strategy – where the system from the root exhaustively checks all possible paths to find the elements that satisffy the requested query – and support in different types of indexing to speed up specific types of database accesses. In general, while indexing helps improve the speed of query performance drastically, it can be costly for database updates since the index must also be updated. Lore's value index enables fast location of values that satisfy the conditions, edge index quickly locates elements or attributes with a specific name, and path index enables fast identification of all elements reached by a specific path. By exploiting these various indexing techniques, one can design around the database structure to maximize its performance [2, 9].

4 Conclusion

While we reviewed some graph databases that use distinct query languages and exploit certain database structure, graph databases in general may not be as useful for operational use cases as they are often inefficient at processing high volumes of transactions and they are not good at handling queries that span the entire database. Because they are not optimized to store and retrieve business entities such as customers or suppliers, it would be a general approach to combine a graph database with a relational or NoSQL database for effective data management – preferably MongoDB to exploit its automatic replicated servers, community support and flexible data storage. To manipulate the internet data that consists of various types, a more flexible DBMS would be recommended, and it would largely depend on, for example, how much support, security and integrity the system is required to have. If trying to improve the scale up the system without degrading the processing speed or data integrity using an easy-to-learn query language (Cypher), then Neo4j (which has very useful user interface and support) would be ideal to start with. As it is noticeable in Table 3, HyperGraphDB lacks some functionalities compared to Neo4j in complying to ACID properties and replication/sharding, and thus if one seeks to make use of a graph database, Neo4j would be a good initial approach. However, if the data of interest is specifically structured in XML format, it would be ideal to use a DBMS that is designed for it – Lore – which exploits path expression effectively [14].

References

[1] Guru99 2019. *What is DBMS? Application,Types, Example,Advantages, Disadvantages.* https://www.guru99.com/what-is-dbms.html. (Visited on 11/24/2019).

[2] S. Abiteboul et al. "The Lorel Query Language for Semistructured Data". In: *Journal on Digital Libraries* 1.1 (1996). URL: http://ilpubs.stanford.edu:8090/162/.

[3] Renzo Angles. "A Comparison of Current Graph Database Models". In: *International Conference on Data Engineering Workshops* (2012). DOI: 10.1109/ICDEW.2012.31.

[4] Pablo Barcelo. "Querying graph databases". In: 32nd symposium on Principles of database systems (PODS). ACM, 2013. DOI: 10.1145/2463664.2465216.

[5] Manoj Bisht. *A QUICK LOOK INTO THE POPULAR GRAPH DATABASES.* https://www.3pillarglobal.com/insights/a-quick-look-into-the-popular-graph-databases. (Visited on 11/26/2019).

[6] Daniela Florescu, Inria Roquencourt, and Alberto Mendelzon. "Database Techniques for the World-Wide Web: A Survey". In: *ACM SIGMOD Record* 27 (June 1999). DOI: 10.1145/290593.290605.

[7] Paul Francis and Shin ya Sato. "Design of a Database and Cache Management Strategy for a Global Information Infrastructure". In: International Symposium on Autonomous Decentralized Systems. IEEE, 1997. DOI: 10.1109/isads.1997.590632.

[8] Roy Goldman, Jason McHugh, and Jennifer Widom. *HyperGraphDB vs. Neo4J Enterprise.* http://vschart.com/compare/hypergraphdb/vs/neo4j. 2018. (Visited on 11/26/2019).

[9] Roy Goldman, Jason McHugh, and Jennifer Widom. *Lore: A Database Management System for XML.* https://www.drdobbs.com/web-development/lore-a-database-management-system-for-xm/184404068. 2000. (Visited on 11/26/2019).

[10] *Neo4j - Overview.* https://www.tutorialspoint.com/neo4j/neo4j_overview.htm. (Visited on 11/26/2019).

[11] *On Sharding Graph Databases.* https://jimwebber.org/archive/2011/02/on-sharding-graph-databases/. 2011. (Visited on 11/26/2019).

[12] Alex Popescu and Ana maria Bacalu. *What is HyperGraphDB?* https://nosql.mypopescu.com/post/942594519/what-is-hypergraphdb. 2010. (Visited on 11/26/2019).

[13] Ian Robinson, Jim Webber, and Emil Eifrem. 2nd ed. O'Reilly, 2015.

[14] *The Good, The Bad, and the Hype about Graph Databases for MDM.* https://tdwi.org/articles/2017/03/14/good-bad-and-hype-about-graph-databases-for-mdm.aspx. 2017. (Visited on 11/26/2019).