2012 International Conference on Software and Computer Applications (ICSCA 2012)
IPCSIT vol. 41 (2012) © (2012) IACSIT Press, Singapore

Graph Database Model for Querying, Searching and Updating

Prashish Rajbhandari [1], Rabi Chandra Shah [2+] and Sonali Agarwal [3]

[1 2 3] Indian Institute of Information Technology, IIIT, Allahabad, India

Abstract. The field of database technologies has undergone drastic changes. The advent of complex connected data has given rise to database technologies that diverge from the traditional relational databases (RDBMS). Graph-based database, which abstracts data in the form of nodes, edges and properties, is one such innovative idea for highly associative data. In this paper, we have described a graph database model based on the graph theory and a prototype implementation of the same. The prototype implementation has a complete set of insertion, selection and deletion graph queries and a cache for fast data processing and retrieval.

Keywords: graph database; graph theory; graph-based database model

1. Introduction

Innovation sparks changes and in recent years the field of Information Technology has undergone drastic transformation. This change in technology and the form of data being used has led to massive changes in database technologies. Most applications today, represent data that are intensely associative i.e. structured graphs, texts, hypertexts, wikis, RDF and social networking. The traditional relational database model (RDBMS) has been proved to be ineffective while handling such associated data with problems regarding horizontal scalability and dynamic data handling. Graph database is one such database technology that is capable of handling such highly heterogeneous and dynamic data. It has gradually been gaining popularity with the giants of the Internet world Google, Twitter, and Facebook incorporating some form of graph database for computation, scaling and knowledge retrieval.

In this paper, we introduce our graph database model to manage and efficiently store data in a key-value form. The graph database is grounded on the concept of graph theory: abstracting data in the form of nodes, edges and properties. This model supports predefined as well as user-defined queries for various operations within the database. The basic predefined queries are inserting, updating and deleting nodes, edges and properties. And advanced queries comprises of finding all or specific property, find interconnected nodes/common nodes, Depth First Search (DFS), A*, Breadth First Search (BFS) etc. Since, a user can declare various user-defined functions within a graph database; various graph-operations can be efficiently executed for research in graph traversal [1]. A graph database should efficiently model the dynamic behavior of user-defined queries and advanced queries.

Scalability is a fundamental feature of the graph database. With the constant hassle of limitation of physical memory, any graph database that deals with associated and dynamic data should have an existing architecture to address scalability issues. The most prominent feature of our graph database model that deals with such scalability issues is the use of *Cache Layer* [Section 5.3]. The Cache Layer emphases on fast traversal and searching of graph data.

We focused on building architecture for graph database with our approach for storing and retrieving graph-data. We implemented a self-balancing tree as cache for fast retrieval of data and traversal.

[+] Rabi Chandra Shah. Tel.: + 91-9807128318;
E-mail address: iit2009026@iiita.ac.in

2. Outline of the paper

This paper describes the underlying architecture (model) [Section 4], the internal working [Section 6] and the work flow [Section 7] of the proposed graph database.

The paper is mainly divided into Sections [1 to 9]. Our proposed model is described in Section 3 with detailed explanation of each layer. Each layer described in Section 3 is then divided into sub-layers and each one is explained in Section 4. A prototype implementation along with storage system is explained in Section 5. In Section 6, complete workflow of our graph database is explained.

3. Related works

In recent time, there have been few works regarding graph database's architecture, storage, performance, scalability etc. Each paper presents their model and implementation [7] [8] [9] of their graph database. During our research, we found graph database model that were designed to perform specific works [4] [5] in specific fields like social network [5], video processing [4], bioinformatics [3] etc. We also found research papers that focused on querying languages [2] and traversal problems [1] in the current graph database models.

There are many publicly released graph databases like InfiniteGraph, InfoGrid, DEX, AllegroGraph but the most famous one is Neo4j Graph Database. They have features like scalability, high-performance [8], RDF graphs, garbage collection etc. that makes them different from one another. These databases are usually general-purpose databases that store graph-data for applications like social networks, route planning system, collaborative filtering, P2P networks etc.

We studied other graph database models [10] and we say that our model varies in terms of approach, architecture and data storage system.

4. Proposed model

In this section, the architecture along with its working is explained. Our model (Figure 1) consists of three basic layers: Presentation Layer, Application Layer and Storage Layer.

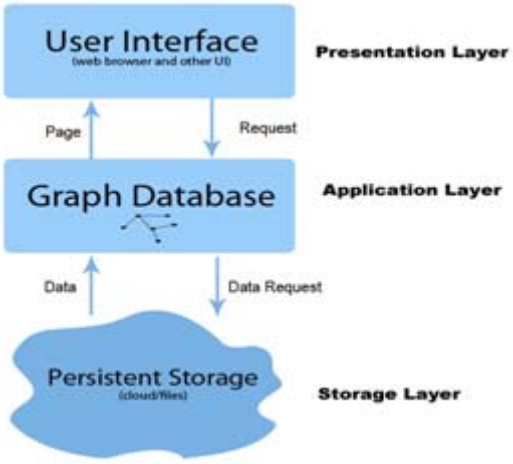

Fig. 1: Level 1 Layer Diagram of the proposed model

4.1. Presentation Layer

It is the user-interface (application, browser) that takes in user input and request information from the Application Layer.

The information or page is then retrieved back to this layer after processing. The layer then presents the output in a user understandable form. Presentation Layer is the only layer that the use can interact with.

4.2. Application Layer

Requests are sent to the Application Layer for retrieval or processing of graph data. First, Graph Database Engine translates request or data into a form understandable by the Graph Database. The translated

form is then used for various operations *(View Engine, Store Engine)* within the Application Layer. The Application Layer directly interacts with the Storage Layer for storing manipulated data inside file.

The primary tasks of the Application Layer are:

- Translate raw data.
- Process the data for operation.
- Send data to Storage Layer for storing in files.
- Implementation of Graph Operations.
- Implementation of Algorithms.
- Generate data and return to the Presentation Layer.

4.3. Storage Layer

Storage layer is responsible for managing the storage of graph data. The request or data from the Application Layer are handed to the Storage Layer with some prior storage instructions. This layer then interprets the instructions and accordingly, creates, appends or deletes raw data files.

The primary tasks of the storage layer are:

- Create, delete and edit data and files.
- Manage data.
- Encryption of data.
- Maintain the configuration file.

5. Complete Proposed model

In this section, the layers described in Section 4 are divided into sub-layers and each sub-layer is explained.

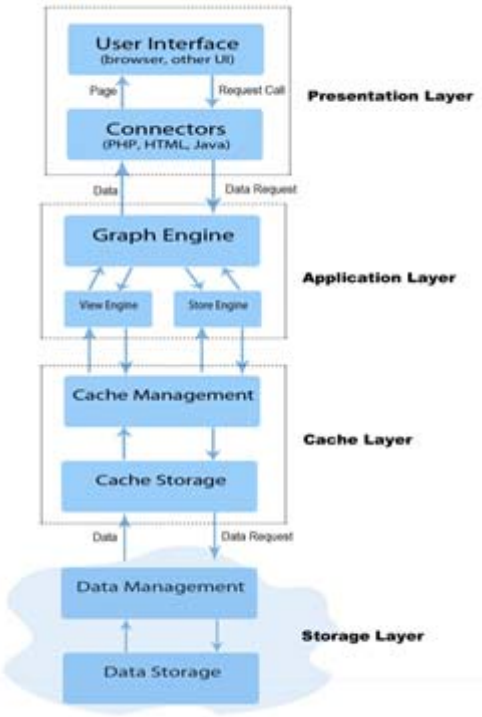

Fig. 2: Level 2 Layer Diagram of the proposed model

5.1. Presentation layer

5.1.1. User Interface

The User Interface is any interface that directly interacts with the user. It takes in raw input from the user, provides control of operation and displays output to the user.

5.1.2. Connectors

The **Connectors** are languages that the applications (that uses Graph Database as back-end) are developed in. The Connectors can be applications in any language i.e. Java, PHP etc. Later, the Graph Engine translates data from the Connectors to a form understandable to the graph database.

In our implementation, we have developed two applications in Java (later described in Section 6).

5.2. Application layer

5.2.1. Graph Engine

Data from the Connectors arrive to the Graph Engine for translation into a form understandable by the Graph Database. When the processing inside the Application Layer is complete, the Graph Engine again translates back the data into language understandable to the Connectors. This layer acts as a language translator.

In our implementation, we developed the Graph Engine in Java. It means that the data from Connectors is translated to form understandable by Java for manipulation within the database.

5.2.2. View Engine

The View Engine manages graph data retrieval from the database files. It takes care of all graph queries that are related to *reading* into the database files. This layer carries out queries such as selecting nodes/edges, traversing graphs, finding nodes/edges.

5.2.3. Store Engine

The Store Engine manages the storing of information into the database. It takes care of all graph queries that are related to *writing* into the database files. This layer carries out queries such as inserting, appending, deleting nodes/edges.

5.3. Cache layer

Cache layer is an integral part of our Graph Database Model. The data requested by basic queries or any advanced queries are received from this layer. This layer also maintains all the modification, insertion or deletion of data to the Storage layer.

When the database server starts, a cache session is instantiated. The session loads all data with same node or edge properties into the memory for fast data retrieval. When a query is fired, the Application Layer processes the data in the Cache layer, extracts the required information and finally presents it to the user.

Also, any modification such as insertion, deletion or updating is carried out in the cache session itself i.e. in the memory and when session is terminated, the cache informs the storage layer to carry out all the changes to in the physical memory. Multiple Caches can be instantiated to speed up the query processing. One can also instantiate Cache prior to any user's queries.

Cache Layer should be fast enough to respond to user queries and to load the graph data into the memory. Hence, it should incorporate an efficient data structure to model massive data with minimum insertion, updating and deletion time. In our implementation, we have made use of a *self-balancing tree*, AVL Tree. However, any other data structure can be used for the Cache layer.

5.4. Storage layer

5.4.1. Data Management

The Data Management Layer takes in data from the Cache Layer and assigns unique IDs to the graph data. It also returns the raw data to the Cache Layer when requested by the user.

Its primary tasks are:

- Assign ID to newly created nodes, edges and properties.
- Place data to its respective location.
- Encryption of data.

5.4.2. Data Storage

The Data Storage stores data into the different files. The actual writing of raw data into the hard disk is done in this layer.

6. Implementation

We explain the implementation of our proposed Graph Database Model.

First of all, we assign a unique identification number (Node ID) to a node when it is created. A *main index file* maintains the Node ID and all its properties in a CSV file in (ID; <list of properties separated by ';'>) format. Another *table file* maintains the mapping of a property with its *property file* in (ID; filename) format. After mapping the *property file*, we can retrieve the property value of the node using the Node ID assigned to the node. The *property file* is a CSV file in (ID; value) format that stores all the property values.

When the Application Layer requires these values of a particular node, the Storage Layer makes use of this identification number to distinguish the values of the required node from the values of other node. It then forwards these values to the Application Layer via Cache Layer.

A brief illustration of single node storage is shown in the figure below. The node has three properties: *Name, Hobby* and *Salary* with values Gopal Prasad, Football and 10000 respectively. A file *maintablehash.gdb* stores the ID of the node along with all its properties. Similarly, a file *tablehash.gdb* maps the property and the property filename. Hence, we have three files for each property. *10001.prt* file contains values of *Name* property. *10002.prt* file contains the values *Hobby* property and *1004.prt* contains all the values of *Salary* property. Property of a specific node is matched by Node ID in the Property file. In our case, *ID 1025* is matched with the IDs in the Property file *(10001.prt, 10002.prt and 10004.prt)* to get its property values.

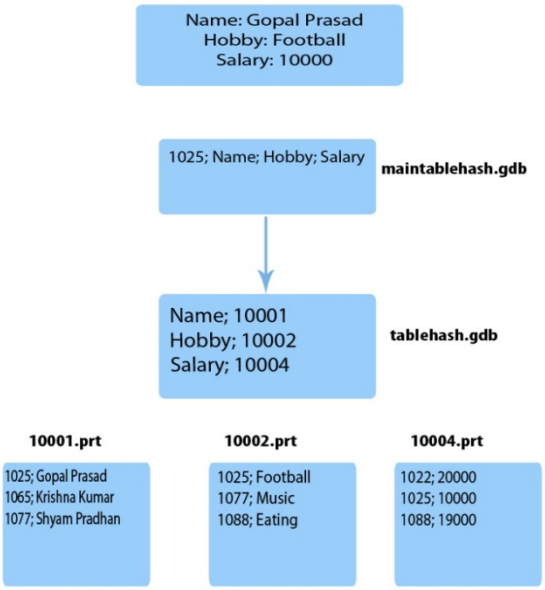

Fig. 3: Figure representing how nodes are stored in separate files and interlinked by the main file.

In the same manner, edges are stored in our Graph Database.

7. Acknowledgement

We would like to thank our honorable Director (IIITA), Dr. M.D. Tiwari, for providing us with a wonderful opportunity and infrastructure to work under this topic.

8. References

[1] MANNINO and M.V. SHAPIRO, "L.D.: Extensions to Query Languages for Graph Traversal Problems," *IEEE TKDE 2,* pp. 353–363, 1990.

[2] AMANN B., SCHOLL and M. GRAM, "A Graph Data Model and Query Language. In: European Conference on Hypertext Technology," *ACM Press,* pp. 201-211, 1992.

[3] J. Huan, D. Bandyopadhyay, W. Wang, and J. Snoeyink,"Comparing graph representation of protein structure for mining family-specific residue-based packing motifs," *Journal of Computational Biology (JCB)*, 12(6): pp. 657–671, 2005.

[4] J Lee, J. Oh and S. Hwang, "Strg-index: Spatio-temporal region graph indexing for large video databases," *SIGMOD*, pp. 718–729, 2005.

[5] L. Kirchhoff, K. Stanoevska-Slabeva, T. Nicolai and M. Fleck, "Using social network analysis to enhance information retrieval systems," *ASNA'08, Zurich*, 2008.

[6] Gyssens, M., Paredaens, J., Bussche, J. and Gucht, D, "A Graph-Oriented Object Database Model" *IEEE TKDE 6 (1994)*, pp. 572–586, 1994.

[7] Poulovassilis, A. Hild and S.G., "Hyperlog: a graph-based system for database browsing, querying, and update," *Knowledge and Data Engineering, IEEE Transactions*, pp. 316 – 333, Mar/Apr 2001.

[8] Martinez-Bazan, N., Gomez-Villamor, S., and Escale-Claveras, F., "DEX: A high-performance graph database management system," *Data Engineering Workshops (ICDEW), 2011 IEEE 27th International Conference on April 2011,* pp. 124 – 127, 2011.

[9] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge and Jamie Taylor, "Freebase: a collaboratively created graph database for structuring human knowledge," *SIGMOD '08 Proceedings of the 2008 ACM SIGMOD international conference on Management of data,* 2008.

[10] Renzo Angles and Claudio Gutierrez, "Survey of graph database models," *ACM Computing Surveys (CSUR) Surveys Homepage archive Volume 40 Issue 1 February*, 2008.