GRAPH VISUALIZATION USING THE NoSQL DATABASE

A Paper
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Kailash Raj Joshi

In Partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE

Major Department
Software Engineering

May 2013

Fargo, North Dakota

North Dakota State University
Graduate School

Title

GRAPH VISUALIZATION USING THE NoSQL DATABASE

By

KAILASH JOSHI

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

SUPERVISORY COMMITTEE:

Dr. Kendall Nygard
Chair

Dr. Kenneth Magel

Dr. James Cokendall

Approved:

05/21/2013	Dr. Brian M. Slator
Date	Department Chair

ABSTRACT

The relational database has been a dominant approach for organizing data into formally organized tables for years. Recently, with massive amounts of data being generated, a new type of database called NoSQL has emerged. NoSQL seeks to overcome the drawbacks of SQL, such as fixed schemas, JOIN operations and addresses the scalability problems. In this paper we have reviewed emerging technology called NoSQL and compared it with the traditional relational database. In the first part of the paper, we review the pros and cons of both the technologies and in the second, we tried to address issues involving data visualization. Characteristics such as flexibility, low latency, scalability, schema-less, fast query, and performance are some major advantages of a NoSQL database. To test the properties of NoSQL database, we have developed a graph-visualization application based on Neo4j, a graph database, along with accompanying technologies such as MapReduce and the REST web service.

ACKNOWLEDGEMENTS

My deepest appreciation and my sincere gratitude to my research supervisor, Professor Dr. Kendall Nygard, Computer Science Graduate Program Coordinator at North Dakota State University, for his enthusiasm on my chosen topic as well as for his patience and invaluable insights, encouragement, and valuable suggestions. Without his motivation, my research would not have been possible. My special thanks and sincere gratitude go to Dr. Mitch Keller, Project Technical Director of the Mathematics Genealogy Project, for his support and help in providing the dataset for my research project. I would also like to thank my research committee members, Kenneth Magel and Jim Coykendall, for supporting my research.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ACID	Atomicity, Consistency, Isolation, Durability
HDFS	Hadoop Distributed File System
HTTP	Hypertext Transfer Protocol
JSP	JavaServer Pages
MVC	Model View Controller
NoSQL	Not only SQL
RDMS	Relational Database Management Systems
REST	Representational State Transfer
SQL	Structured Query Language
URI	Uniform Resource Identifier

CHAPTER 1. INTRODUCTION

1.1. Background

The relational database has been part of our daily life since IBM has developed hierarchical database management system in 1969. It has helped companies solve the complex problems of storing data. The main idea of a relational database is to organize data into formally organized tables so that the database user can issue queries to obtain relevant information easily. As technologies improve, the amount of data generated by humans, the Internet, and machines have also increased enormously. To meet the demand of increasing data size, there was a need of highly scalable database. So a new type of database has emerged known as Not only SQL (NoSQL). Increase in volume of data has also increased volume of data to process, complexity of the data, and connectivity relationships among the data. Processing high volume of data is another aspect of data, which is beyond the scope of this paper. The term "NoSQL" should not be taken in a misleading way. Rather than replacing relational database, the NoSQL approach is intended to work in applications where there are scalability issues and where data need some structure. There are four emerging categories of NoSQL. The first one is the Key-Value stores, which is based on a research paper published by Amazon and called Dynamo [1]. Due to the high volume of data, Amazon built Dynamo, a Key-Value stores database, to solve the problem of scalability. The data model of the key-value stores is similar to hash table. The second category of NoSQL is the Column Family or the BigTable clones. This category is based on a Google paper titled "BigTable: A Distributed Storage System for Structured Data." [2]. The purpose for creating BigTable is similar to Amazon Dynamo, solving the problem of scalability. The main difference between Amazon Dynamo and BigTable is that every individual column in BigTable can have its own schema. Google's BigTable is a powerful tool to capture semi-structured data. Semi-structured data is the form of data where some of the fields are mandatory attributes but most of the fields are option attributes. The third category is documented databases. Again, the purpose of this category is the same as others, solving the problem of scalability. The most famous implementation of a Documented database is Couchdb, which was inspired by Lotus Notes, and the document is typically expressed in JavaScript Object Notation (JSON) or Extensible Markup Language (XML) format. The fourth category of NoSQL is graph database, the main focus of this paper. Graph theory inspires all graph databases. The data model of the database is similar to a Documented database. The Document database has JSON documents, and the documents are key-value pairs. These documents are called nodes in a graph database. Nodes can contain key-value pairs. The database helps where data are highly connected.

Another important portion of this paper is data visualization. We know that visualization is very important to humans, but to answer the importance, we turn to biology. The visual system of humans is extremely well developed. It is fundamental for human beings to analyze what and how they perceive things. There is a common belief that seeing or visualizing is one of the best ways of understanding and generating knowledge as well as learning things intuitively [4]. Pictorial representation of data is eye catching and can be more understandable than looking at numbers and text. Similar concepts have been used in today's computer and software world. The principles of human perception are used to visualize data effectively. Visualization techniques convert data to helpful information in a graphical form. It is influential in persuading people and presenting an argument, even after the data are seen in total. Visualization is useful for understanding patterns in data and for perceiving outliers that may be present. It can help to make decision and analyze the data in context. It can also help to answer questions in context and to support graphical calculations. Visualization is helpful in analyzing the data as a whole and can support reasoning based upon the information. Analysis can be about the process and calculation of data, reasoning about the data and their conclusions along with feedback and points of interactions. Visualization also supports the communication of information to others, sharing the conclusions and persuading the stakeholders. Finally, visualization helps to emphasize important aspects of the data [4].

Data visualization is important for gathering insights quickly and making decisions. It is important to transform those pieces into knowledge. We use the facts, such as data and reports, for decision making in the future. If we fail to represent the data properly, we could make the wrong decision or miss a great opportunity. Data visualizations help us understand complex problems in a familiar and easy way. One of the powerful means for discovering and understanding facts, followed by presenting them in an interesting and understandable manner, is data visualization. For the purpose of our graph-visualization project, we have used Neo4j as a graph database.

A graph database represents data as a structure providing a level of flexibility and resilience for the data. That characteristic of a graph database is a great match for today's fast-moving development methods. Data representation in graph databases shows data as nodes and relationships between nodes (things). These representation of data helps to closely understand the way we think about complex systems. Some of the best-known examples for using graph databases can be seen with well-known social networking sites such as Facebook and Twitter [5]. The flexibility and performance advantages of a graph database over a relational database for certain applications increase its popularity. The complexities and dynamics of today's fast-moving world are at the speed of the web, and call for new and easily understandable methods. The graph database has predominantly been used to accelerate development and massively speed up the performance of applications. It is particularly important to understand intricate and complex processes such as human behavior and interconnections in nature and on the web, which tend to not be static or predictable, and are ideal candidates for graph databases [6].

Many tools are available in the literature to organize unstructured or semi-structured data, such as relational databases or NoSQL. However, a standard database alone will not help to analyze data efficient way if the size of data is large. More efficient way of presenting data is graphical modeling of data, which helps researchers analyze data more efficiently. In our graph-visualization application, we have used Neo4j as a graph database. The application has a console where the result of query processed by the graph database can be visualized in a graphical model.

Rest of the paper is divided for two main objectives. The first objective is to review different scenario where NoSQL will be better option as compare to relational databases. The second objective is to show how visualization can help to better represent the query result of database in a graphical model. From the many graph-database available in the market, we choose Neo4j because it is an open-source database and is well documented. We will discuss, in more detail, methodology of our implemented using Neo4j in later chapters.

1.2. Motivation and Objective

With the increase in popularity of social networking and large volume of data associate with it, a need of new type of database was necessary, which can store and manipulate complex data. Our motivation of this project was to understand new type of database called NoSQL. There are two major objectives for this paper. The first objective is to review an emerging technology called NoSQL and implement a NoSQL-based application to compare the technology with an existing relational database. The second objective is to find an effective way of visualizing query result of graph database into a graphical model. The overall goal of the project is as follows:

a. Why it is appropriate to investigate an alternative type of database, such as HBase, Cassandra, Redis, MongoDB, Voldemort, CouchDB, Dynomite, Hypertable, Neo4j, and several others, as alternatives to a relational database

b. How a NoSQL database can help the scalability problem

c. Why visualization of data is an important issue

d. What can be done to improve visualization

Our methods and approaches of explaining above statements are explained in later chapters.

1.3. Roadmap

Chapter 2 gives a brief Literature Review about the pros and cons of SQL and NoSQL. In Chapter 3, we explain the methodology to implement our graph visualization project. The method includes the architecture of the software, design pattern used, class diagram, activity diagram, and a detailed discussion about how the dataset was received from the Math Genealogy Project and how it was used in the application developed for the project. In Chapter 4, we discuss various testing techniques utilized to test the application. It also consists of a sample case used to test the application. Finally, Chapter 5 discusses the future research work to improve the application's performance and the Conclusion of our project.

CHAPTER 2. LITERATURE REVIEW

NoSQL databases are relatively newer and evolving databases as compared to a relational database. Relational database are stable and fully tested database. In this section, we review several research papers, Internet articles, and expert opinions, which helped us, review the technologies more effectively.

2.1. Background of NoSQL

In 1998, Carlo Strozzi [7] was among the pioneers to have the “NoSQL” concept. The Strozzi model was created to distinguish his model from the Relational Database Management System (RDMS). RDMS uses Structured Query Language (SQL) to query the database. Strozzi used the term “NoSQL” because his model did not expose the SQL interface [8]. With the increase in data volume over time and to solve the problem of scalability, Google and Amazon came up with their own databases. Google came up with its database called Big Table, and Amazon called it Amazon Dynamo. One of the main aspects of Big Table is that it does not have features such as primary key, foreign key, JOINs, or relational calculus of any type. It is not a relational database but is a distributed, persistent, multidimensional-sorted map [2]. Mapping of data in BigTable is achieved by indexing row key, column key, and a timestamp. Each value in the map is an uninterrupted array of bytes. The main concept of a relational database is normalization of data, whereas BigTable de-normalizes the data. In BigTable, we can think of the table as a single-table database. All data are stored in a single table. Because all data are stored in one BigTable, the concept of JOIN does not apply to BigTable. The JOIN operation in a relational database is the process of combining records from two or more tables. BigTable database nature is distributed and sparse, so performing JOIN within the table can be difficult and inefficient. Another important property of BigTable is that it is a low-latency database because data are stored in the sorted order. Many NoSQL open-source projects have evolved in the market after Google published a paper about BigTable in 2006 [2].

2.2. Architecture of NoSQL

The key concept of NoSQL is to support high-performance, scalable data storage and to provide direct access to the programming language to manage the database from the application layer [8]. In NoSQL, the database user can manage data from both the application and database layers. Direct access of database from application layer provides more flexibility to maintain the application. But in relational database, database can only be managed database layers, not from application layers. This property of NoSQL provides more flexibility to maintain the application as well as the database. Relational database manages data by normalizing the data. Normalization is the process of managing fields and tables of database in such a way to avoid redundancy and dependency of data. By normalizing data into different tables, relational database management system (RDMS) can preserve key invariants, making it easy to maintain consistent data. Normalizing data of large dataset can have performance issues because quires to aggregate information can be complicated with lots of JOIN type query. But, NoSQL does not have ad-hoc JOIN type of query functionality as we can see in Figure 1. In NoSQL, user can also uses a programming model like MapReduce to process large volume of data parallel. NoSQL de-normalizes data; as a result, we can have significant improvement in the query time to traverse a database because less overhead while querying the database. NoSQL avoids the JOIN and aggregation operations, so the database users avoid O (n) aggregation operations. Due to data de-normalization, there is a chance of data inconsistency, such as duplication or redundant data. To avoid duplication, the application layer has to rely on data synchronization that avoids any inconsistency in the data copies. Another major difference in architecture of NoSQL and relational database is schema. With a relational database, you must define schema before adding any records.

Modifying schema or adding new schema to the database when the database is spread across the server is difficult [9].

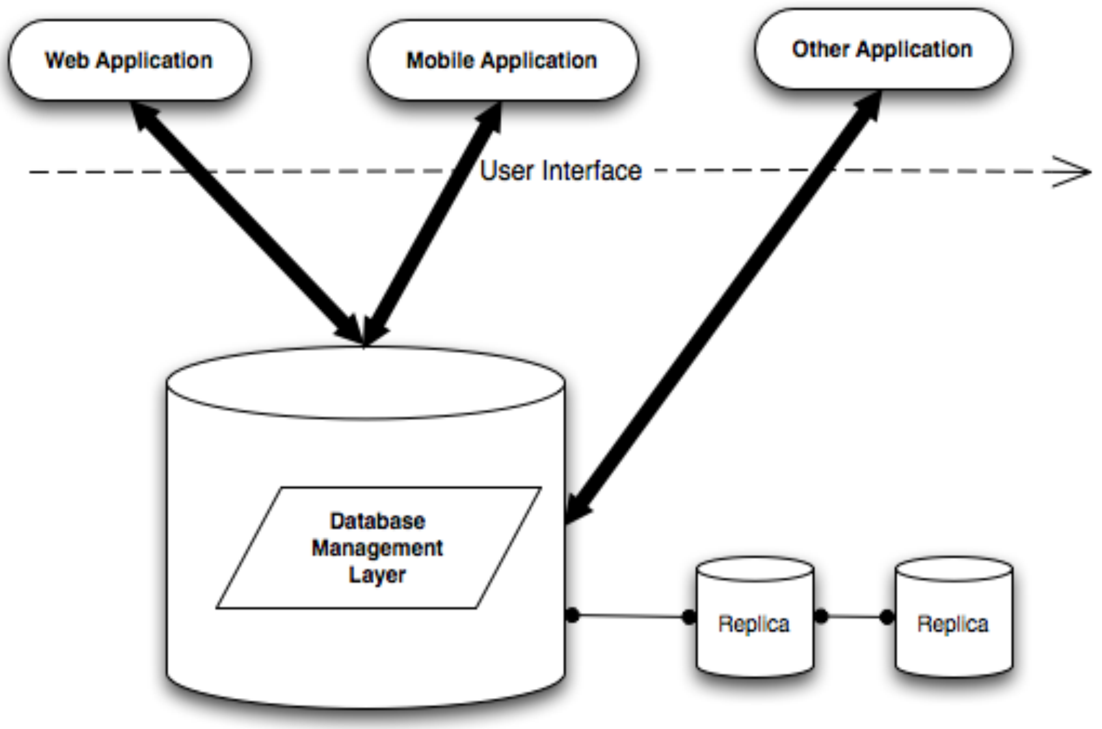


Figure 1. Typical setup of NoSQL in a distributed server

NoSQL database is a schema less database. The schema is added to database at the time when records are added to the database. This NoSQL property helps the database to scale out easily.

2.3. Trade-off between NoSQL and SQL

In the above section, we have discussed various aspects of relational and NoSQL databases. In this section, we look into the trade-off between these technologies. We compare the trade-off between the two technologies from three perspectives: CAP theorem, relational data, and query language.

2.3.1. CAP Theorem

In 2000, Eric Brewer publishes a paper with the idea of fundamental trade-off between consistency, availability, and partition tolerance. Later the idea of trade-off widely knows as CAP Theorem that is widely been discussed to the date. Consistency means the data are same all across the cluster. User can get same data querying any node in the cluster. Consistency is a factor where two technologies are different. NoSQL is the less consistent database because of de-normalization of data. In order to increase consistency, the number of nodes needs should be increased. But increase in nodes also decreases latency of the database. According to the CAP Theorem [10], a system can only pick two of three factors, such as Consistency, Availability, and Partition tolerance. As shown in Figure 2, only two of the three aspects can be achieved at any given time. Relational databases are highly available and consistent database because data is stored on a single machine.

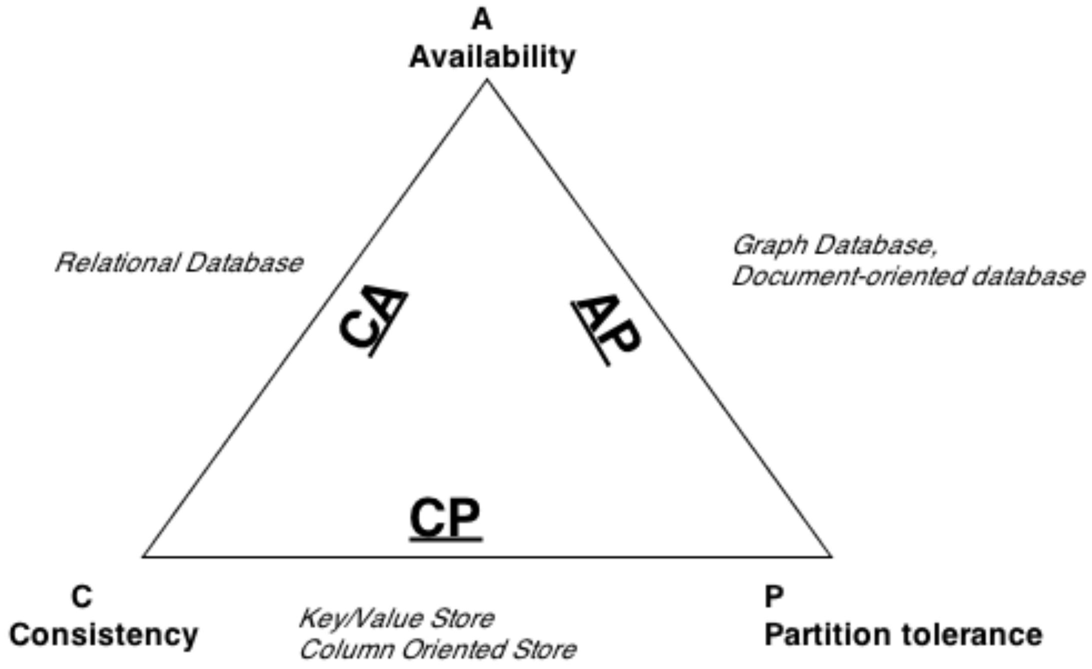


Figure 2. The CAP Theorem [10]

Most of the NoSQL databases tend to lose consistency to achieve better availability and partition tolerance to solve the scalability problem. Database users select the database according to the business-requirement needs. For example, Amazon Dynamo is based on high availability and partition tolerance because Amazon is a service-oriented company and providing continuous service to its customer is vital for the survival of a company. In high available key-value storage system of Amazon, the databases tend to lose consistency because of CAP theorem [10].

2.3.2. Relational Database

The relational database solves the problem of data redundancy and updated anomalies in relational schema by using normalization [11]. By applying normalization, the database ensures data consistency and avoids redundant data. When the data are distributed across the server, performance of the relational database is not effective. Often, the developer writes a complex query to obtain the desired outputs. In a distributed system, the most complex query operation is the JOIN operation. Performance of relational database is weak when mapping relational data with programming structures where programming model consists of complex data or hierarchical data such as Extensible Markup Language (XML). But NoSQL does not support normalization; hence no JOIN operation in NoSQL database. NoSQL database is suitable in an environment where the data are distributed across the servers and where consistency in not the main priority.

2.3.2. Query Language

Structured Query Language (SQL) was first developed by IBM to manipulate and retrieve data from relational database management system. The performances of SQL lag when data are of complex nature and distributed over the cluster. There are certain features in NoSQL database that cannot be expressed with the use of SQL. A new type of query language has to be written to query the NoSQL database. For example, in a relational database, a schema is written before records are added to the database, but in NoSQL, the database schema is written when records are added to the database. The limitation of writing schema before adding a record is that a new data type cannot flow into the database without major modification to database. To incorporate the new NoSQL features into query language, most of the database has its own query language, such as Cypher query used by Neo4j. The problem of introducing new language is that it is difficult to find an experienced developer for a company to maintain and use the query database efficiently.

2.4. Need for a Graph Database

All the implementation for this paper is based on the graph database therefore it is important for us to know the need of graph database. The graph database is helpful, especially when the volume of data is large and when data are highly connected. The graph database can be used not only in field like social networking, but it can also be used in area such as finance, banking, insurance, etc. With a graph database, the data are stored in a single-structured graph, not in tables [12]. Any typical graph contains a vertex and an edge:

$$G= (V, E),$$

where G is graph, V is vertex, and E is edge. Thus, the graph is a set of vertices and edges. Every vertex and edge in the graph database are linked to their adjacent vertices or edges. Due to this property of database, the complex JOIN operations are eliminated from the graph database. Every vertex and edge is connected to the adjacent vertex and edge, which may make it difficult to scale the graph database into distributed servers. This is one of the drawbacks of graph database. However, there is another main advantage, which will compensate disadvantage of graph database not being able to store over a distributed server. The main advantage of using graph database is all vertices and edges in database are connected with its adjacent vertex and edge. Hence cost of traversing the database is constant irrespective of database size. Graph databases are low-latency database because nodes are store in single-sorted structure and overhead cost such as sorting merging can be eliminated.

CHAPTER 3. METHODOLOGY

In the previous chapter, we discussed various advantages and disadvantages of NoSQL and relational databases. In this chapter, we discuss our approach for implementing NoSQL database. The chapter defines implementation details, such as programming language, programming environment, and application architecture. By implementing NoSQL, we are able to make appropriate comparisons between the two technologies. For the demonstration purposes, we choose to implement Academic Search. Academic Search, for the purpose of this project, shows various relationships between the adviser and advisee and how authors are connected with each other. With the graph visualization project, we are able to answer questions, such as shortest path between two persons, and the relationship of the node and its decedents. Not only storing of data but also presentation of any data is important because presentation determines the information flow. Therefore, we also show our approach for implementing visualization in a graph database.

3.1. Purpose

The main purpose for building this application is to illustrate both the advantages and disadvantages of using NoSQL over a traditional relational database. The purpose of the project is not to demonstrate which technology is better, but to demonstrate the circumstances in which the NoSQL database are suitable. We have also implemented graph visualization technique together with NoSQL database, which will help to analyze highly connected data easily.

3.2. Scope

The graph visualization project aims to build a database, which can take any type of data without having any fixed schema and scalable database where data are highly connected. As mentioned Chapter 2, the graph database stores semi-structured database where vertices and edges are connected with the adjacent vertex and edge. Semi-structure database model means database where some of the fields are mandatory and most of the fields are optional. The scope of this project is to highlight typical characteristic of graph database and compare graph database with relational database.

3.3. Overview

The graph visualization project is about Academic Search. Academic search engine helps to find relations between authors. We obtain the dataset for our project from the Math Genealogy project at North Dakota State University. The Math Genealogy project has been collecting information about mathematician from all over the world for the past one decade. The Math Genealogy project uses a relational database to store and query data but we have used a graph database to search and query data. We have used Neo4j, a graph database, in our project. For the purpose of reviewing the graph database, we have selected two different approaches. The first approach is using embedded Neo4j in our Java application. The second approach is using the Representational State Transfer (REST) web service to query the data running on standalone server. By implementing with two different approaches, we facilitate our technology review.

3.4. Use Case

The use-case diagram in Figure 3 shows the interaction between operations and actors in a graph database. Actors in the graph visualization project can be researchers, system administration, and general users. Actors of the graph database can easily create a new node in the database, delete unwanted nodes from the database, find the shortest path between two nodes, search the nodes based on their property, add a new property to an existing node, create a new relationship, and delete an existing relationship. We do not need to change any configuration in the database if we want to add a new type of data to the database. Cypher query of graph database allow users to interact with database and help to make ad-hoc query in the database.

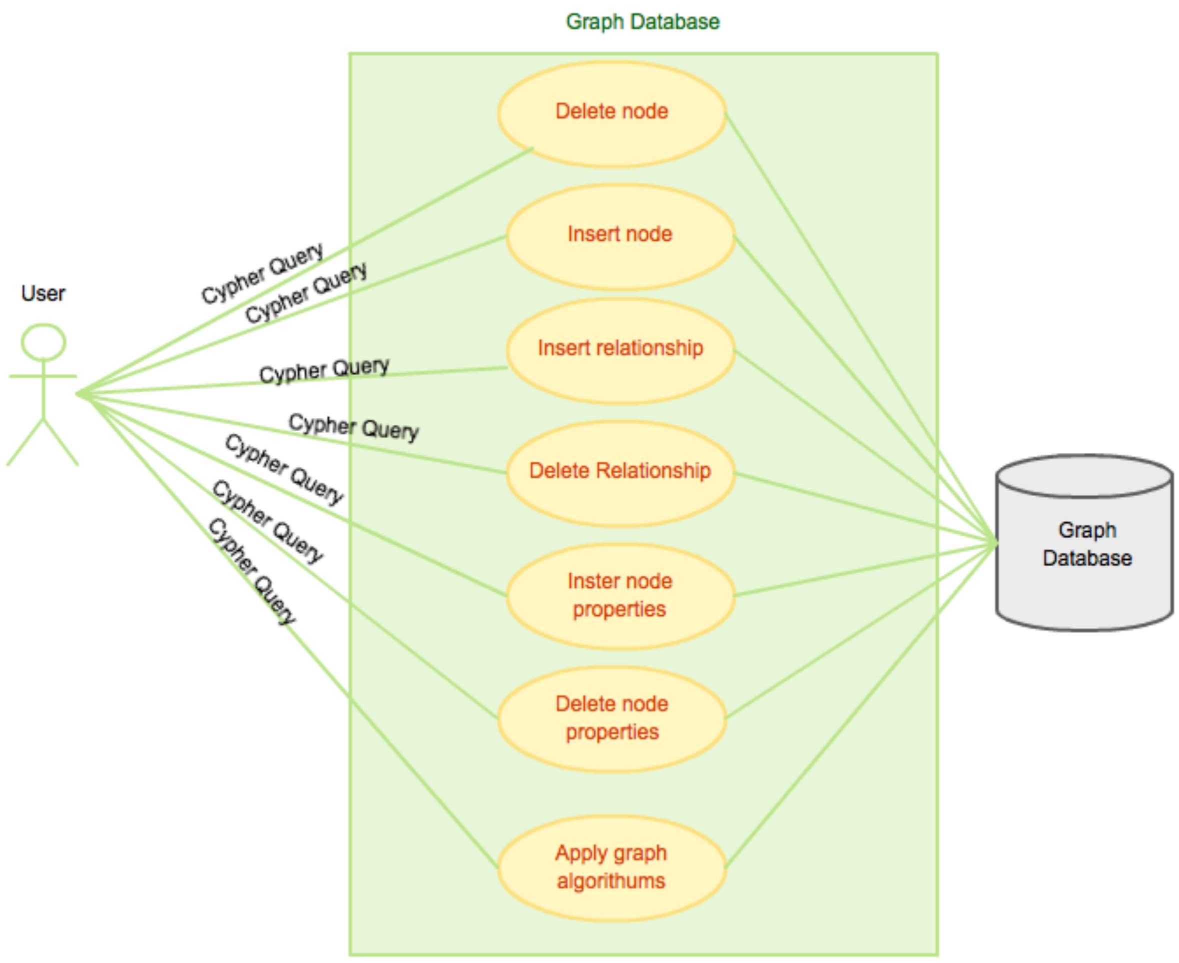


Figure 3. Use case of graph-visualization application

3.5. Software Design

This section contains the high-level design for the graph visualization project. We have used two different approaches: embedded Java application and web service. Embedded applications are useful for querying static queries written in Java application, and applications based on the REST API are to use to demonstrate use of web service to query database running in the server. In this section, we present the design for both approaches.

3.5.1. Embedded Neo4j-Java Application Using Struts 2.0 Framework

In this approach we have embedded graph database in our Java based application. Cypher query can be passed to database using Java code. The embedded Neo4j-Java application is target to general user who does not have programming background. In this approach, users are not allowed to write their own query instead they can only query data that is already written in the embedded application. To implement the embedded Neo4j Java application, we chose the Struts framework. The Struts framework is an Apache open-source project for creating Java web applications. The framework is based on the Model View Controller (MVC) design pattern. The MVC design separates business logic from view layer of the application. The MVC design offers various benefits such as reuse of model class, less overheads in the application and efficient modularity. Only the application's business logic has direct access to the database.

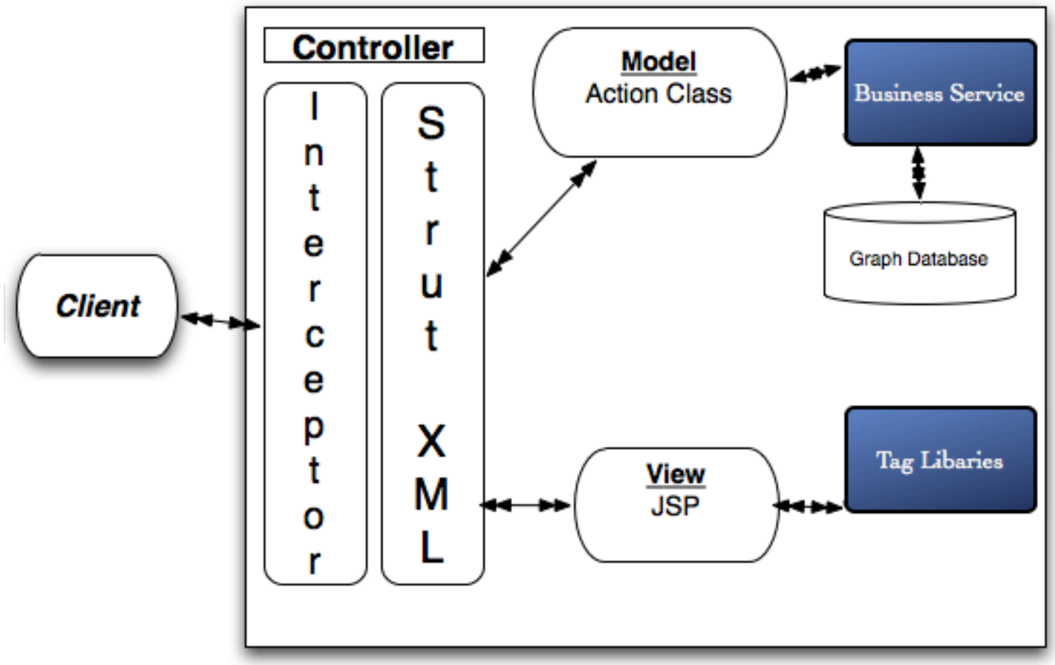


Figure 4. Architecture of graph-visualization application

In a typical servlet, it is difficult to decide how many servlets are needed for any given application. Generally, the number of servlets in an application is equal to the number of forms in the application. With the use of the Struts framework, controller accepts every request submitted by the user and forwards it to the appropriate action class. Here each action class server as a servlets. The framework is built with Java Platform, Enterprise Edition (J2EE) that uses technologies such as Java Filters, JavaBeans, Resource Bundles, Locales, and XML. Figure 4 shows the architecture of the framework. The main features of the Struts framework are as follows:

i. **Centralized Configuration**

Instead of writing complex code in the Java class, all complex code can be configured in an XML file, Struts properties file, or use of annotation. The complex code can be the initialization of an object or mapping of one object with another object. Therefore, code in the Java class is less and easy for debugging.

ii. **Form Bean**

The framework makes it easy to capture data from the forms. In typical servlets, request of user are executed using Hypertext Transfer Protocol (HTTP). Although struts framework uses HTTP to communicate but user does not need to write HTTP related code in their application. Struts framework handles communication between servlets and user internally.

iii. **Tag Library**

The Struts framework is shipped with its custom tag library. The tag library contains tags which allow user to access Java beans and their properties. The library allows user to read and write value of action class without writing Java code in the JavaServer Pages (JSP). There are three types of commonly used tag libraries: Bean tags, HyperText Markup Language (HTML) tags, and Logic tags. Bean tags are commonly used to access Java beans for the application. HTML tags are used to create HTML elements such as form, buttons, etc. Logic tags are used for iterating collections and adding conditional statement to outputs.

iv. **Field Validation**

Field validation is an important framework feature. The framework has a built-in capability to validate form value. If the form values do not correspond with the configured values, the framework displays a pre-configured error message and pauses further processing.

3.5.2. Request Processing Lifecycle

Web application written in Java should be run inside an application server such as TomCat, JBoss, etc. Application server is a program that manages all the web application running inside an application server. In our graph visualization project, we choose TomCat as the application server. Whenever a user makes a request to TomCat, the application server creates two objects, i.e., request object and response object. Request object contains information such as the name of the web application, user credentials, servlet name, action name, etc. The response object will be empty in the beginning because the request object has not been processed in the beginning. Inside the TomCat container, there may be many servlets or web applications running. Therefore, any user request should contain at least two things: first the name of the web application and second name of the servlets or actions. Once the user makes a request, the application server passes the information to the web application running inside TomCat. In our case, the web application is the graph-visualization application written in the Struts framework. Inside the web application, interceptors of the Struts framework map the request to the appropriate action class. Interceptors can also perform tasks such as logging, validating, file uploading, etc. After mapping a request with the action class, the action is executed, and the appropriate business logic is invoked. The application's business logic can directly communicate with the graph database and process the user-submitted query. The output of the action is rendered to the view component, which, in our case, is JSP. Finally, the application server sends the response object back to the user in an HTML format, and the result is displayed in the user web browser. Figure 5 shows the request flow made by a user in the graph-visualization application.

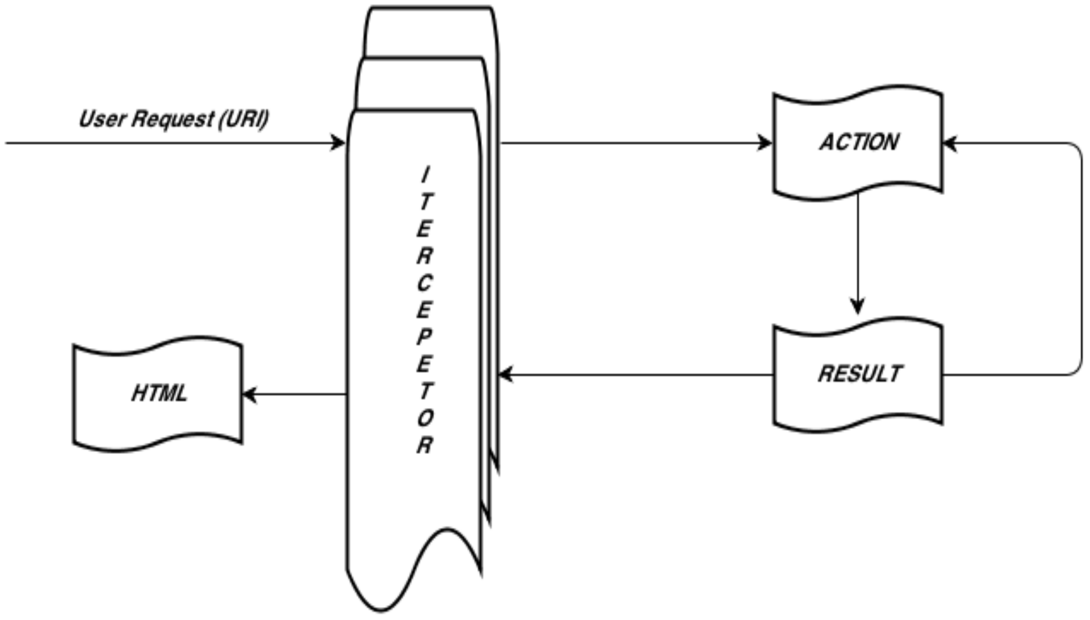


Figure 5. Diagram showing flow of user request in graph-visualization application

3.5.3. RESTFULL Web Service

REST stands for representational state transfer and it is an architectural style. An architectural style is an abstraction as opposed to a concrete thing. The architectural style defines characteristics and attributes. With REST web service, Universal Resource Identifier (URI) identifies resources, and manipulation of the resource can be achieved from representation. The REST architecture is designed based on the Hypertext Transfer Protocol (HTTP)[13]. The browser makes a request to the server, and the server gives a response to the browser. The response is a resource identified by a Uniform Resource Identifiers (URI). As shown in Figure 6, the REST service contains four different web verbs to communicate with services: GET, POST, PUT, and DELETE. GET is used to retrieve a resource; POST is used to append data into a resource; PUT is used to insert or update a resource; and DELETE is used to delete the resource. The request and response object can be in any form, such as JSON, XML, and HTML. The REST service is an independent platform, and it can be used with a variety of programming languages, such as JAVA, AJAX, or PERL. Figure 6 shows different languages supported by the REST web service.

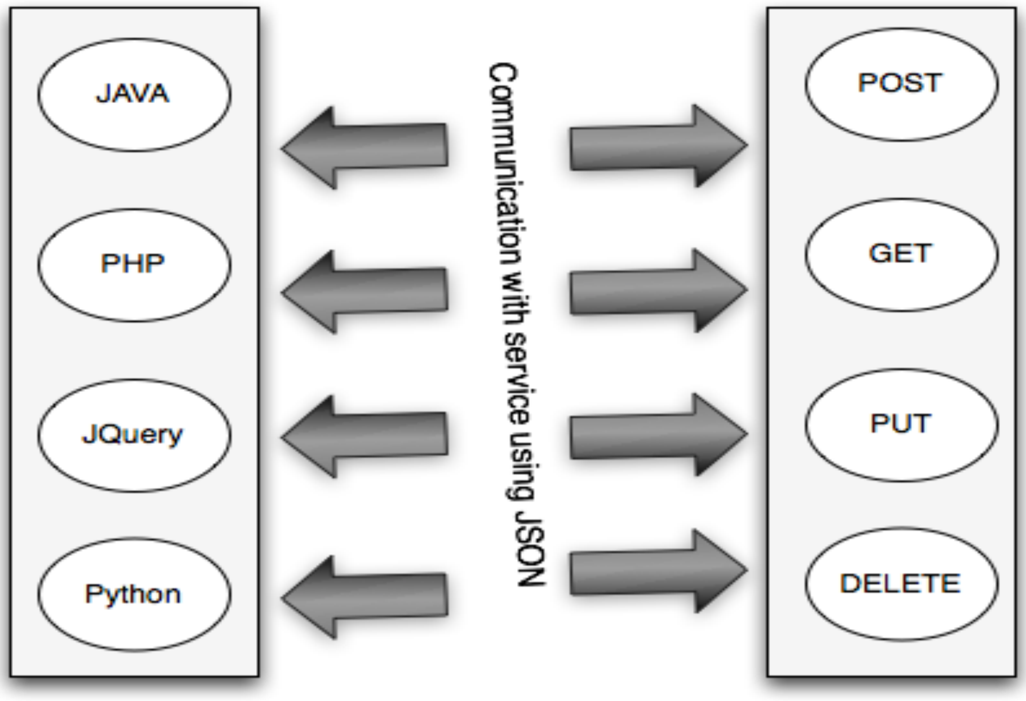


Figure 6. Communication to resources using various programming languages with REST web service

In our graph-visualization application, we have implemented REST service using AJAX as the programming language, which can query the graph database running on the standalone server. The application also converts query result of database into graphs and renders graph to user.

3.6. Graph Database

In a graph database, data are stored in nodes that can be inferred to be property of the node. A large number of properties can be added to a single node. This property of a graph database enables us to store any type of data in a database. Both relationship and nodes have key-value type properties. No fixed schema should be written to store data in a graph database. Schemas in the graph database are written when records are added to the node. The simplest form of any graph database should, at least, contain one node where data can be stored. To make the graph database organized and structured, each node in the database is organized with its relationship. Relationships in a graph database organize data into a predetermined, organized structure such as list, tree, map, or anything that has a richly interconnected structure. Figure 7 shows organization of node in a typical graph database. Graph database represent data as nodes. Nodes can be any objects such as a person, car, bank etc. and each nodes store information related to relationship with other nodes in the database. Relationships are directed but the speed of traversal is equal in both direction. Node properties are the behavior of nodes, such as name, age, sex, etc. A graph database also uses an index for an easy lookup of nodes. Each node in a graph database is indexed with a unique id number.

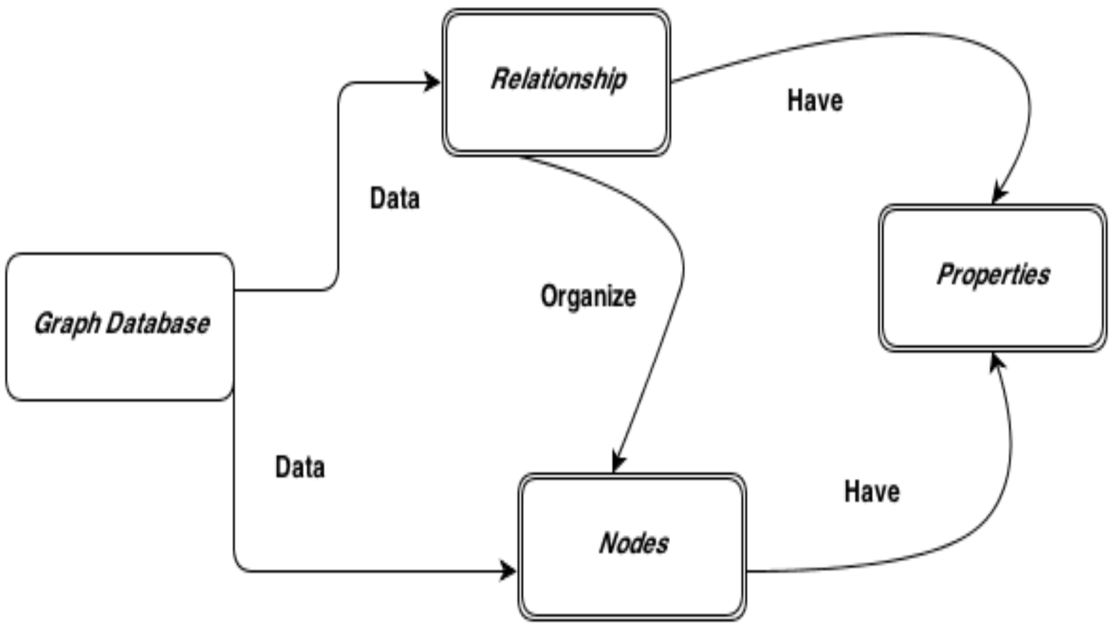


Figure 7. Organization of data store in graph database [14]

In graph database, data are stored in semi structured. Storing data in semi structure fashion means there are some mandatory fields in database but most of the other fields are optional.

3.7. Querying a Graph Database

Query languages are used to query any database. Both relational database and graph databases have their own query languages. Relational database uses Structured Query Language (SQL) to query data. Due to difference between two technologies i.e. NoSQL and SQL, new type of query language should be use to query NoSQL. Most of the NoSQL databases have their own query language such as Jaspersoft HBase Query Language is used to query HBase. HBase is a NoSQL database. Architecture of Hbase is similar to Google BigTable, a column oriented database. In graph database, we use a query language called cypher query. Queries in graph database are executed by traversing the nodes of database. Figure 8 shows different ways to query a graph database. In our graph-visualization application, we have used Neo4j as our graph database. Neo4j uses cypher query as a language to query graph database. The Cypher query language is a declarative graph query language to traverse a database without writing code for that action [14]. Cypher query is easy to read and learn, and it is suitable for both developers and researchers seeking to mine information from the database. The language is easy to learn because its syntax is similar to the SQL syntax.

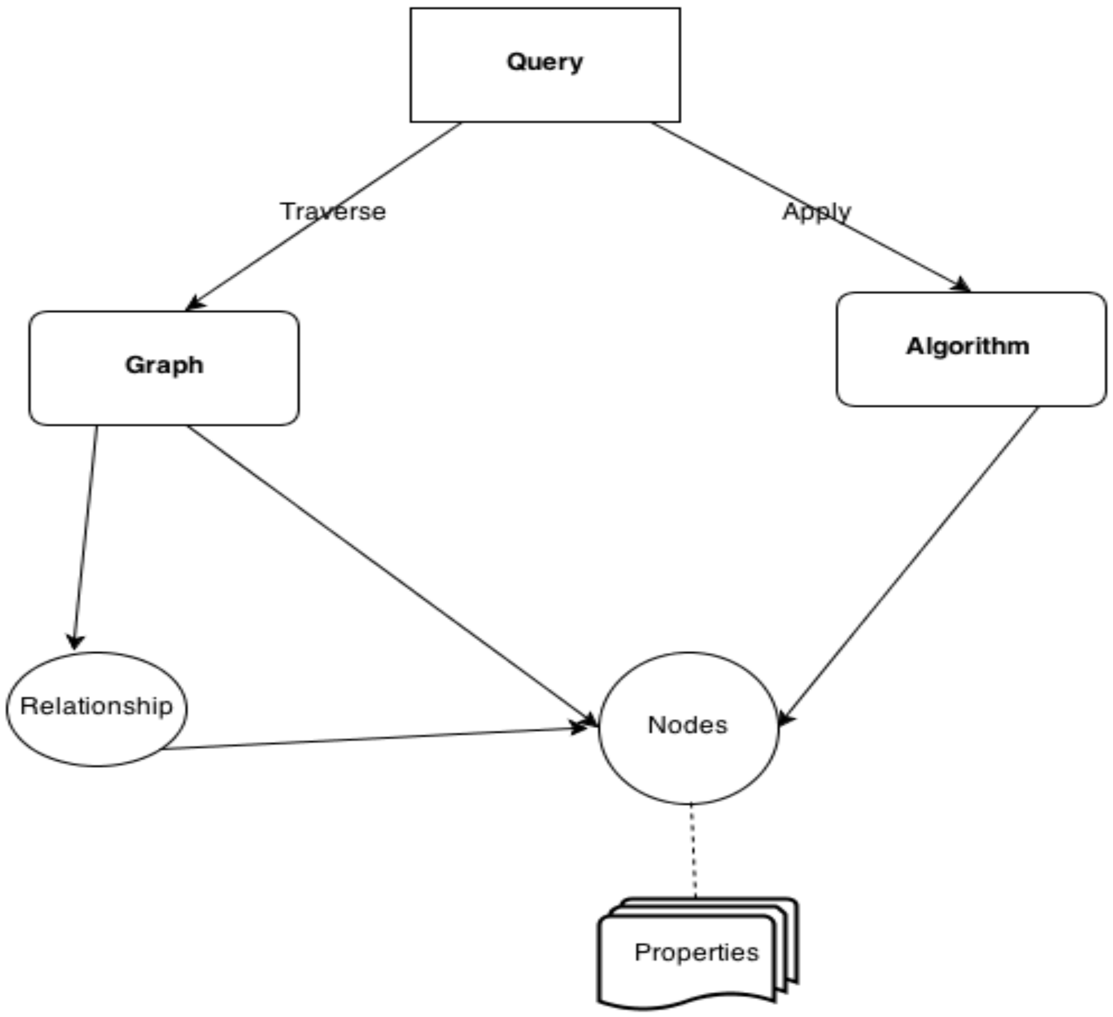


Figure 8. Querying graph database using algorithms or by traversing from node to vertices in a sequential manner [14]

Cypher query offer two different ways to query graph database. The first way is to query database by using built-in algorithm stored in a library of cypher query, such as the shortest path, all path, Dijkstra algorithm, etc. And second way is by implementing your own algorithm.

3.8. Implementation

In this section, we will describe the implementation of our graph-visualization project. As mentioned in above sections, we have implemented the project in two different ways: using an embedded graph database and using the REST web service. We will discuss each of them in details.

3.8.1. Graph-Visualization Application Using an Embedded Graph Database

The first application is built using a Java-embedded graph database and the Struts framework. In this approach the graph database is embedded inside a Java application. Combining the technology of Neo4j and the Struts framework makes our application secure and easy to debug. The primary purpose of building an application is to review the two different technologies, i.e., SQL and NoSQL. However, we have only added functionalities to the application that are useful for the purpose our review. We have not tested all the functionalities that are offered by graph database such as distributed graph database.

The GUI of an application takes two input parameters. Both parameters are the name of any two persons. The application takes each input parameter individually and searches for the property of each node in the database that matches the input parameter the user submitted. If the name property matches of a node matches with input parameter of the user, then the application gives a graphical representation of the shortest path, all path, and number of decedents at different depths between the two nodes (names). Figure 8 shows the application's result. When the user makes a request to the application, the application server receives the user's request. In our application, we have used TomCat as an application server.

TomCat receives the user request and checks for the graph-visualization application running inside the TomCat container. It is now the job of the interceptor of the Struts framework to map the user request to the appropriate action class. The action class in the Struts framework is similar to servlets. Servlets are Java class use to improve capabilities of server. In our graph-visualization project, the SearchAction class handles all user requests. User requests for the application are as follows:

```
search.action?firstNode="Kendall Nygard",secondName="Lester
Frair"
```

Once the action class receives data to process, it passes the request to the appropriate business logic. In our application, there are three different types of business-logic classes to process a user request. The three different units of business logic are ShortestPathFinder class, AllPathFinder class, and DepthFinder class. ShortestPathFinder locates the shortest path between two nodes. Our application is using the built-in algorithms of Neo4j to find the shortest path between two nodes. In our graph database, each node contains three distinct properties: the first name, middle name, and last name of the person. Each node also contains information about the relationship with other nodes in the graph database. The number of relationships that can exist in any graph database can range from 0 to infinity. Our application only has one type of relationship, i.e., the relationship between an adviser and advisee. As per situation, the graph database allow user to add any number of new relationship between the nodes. Application users can create or delete relationships directly from the application layer unlike traditional database.

In social networking data, nodes are highly connected with each other. Figure 9 shows highly connected data. In figure 9, node name Bill is connected with Jenny with multiple relationship such as both node share same location, Jenny and Bill have worked on same publication or share same research interest etc. Thus a single node can have multiple paths to traverse from one node to another. Out of several paths of the node in the graph database, the graph-visualization application should choose the shortest path. The pseudo code for finding shortest path between two nodes is as follow:

```
graphDb = new
GraphDatabaseFactory().newEmbeddedDatabase(DbPath);
indexService = graphDb.index().forNodes("nodes");
Node advisor = getOrCreateNode(firstName);
Node advisee = getOrCreateNode(secondName);
PathFinder<Path> finder = GraphAlgoFactory.shortestPath(
Traversal.expanderForTypes(ADVISOR, Direction.BOTH), 100);
Path foundPath = finder.findSinglePath(advisor, advisee);
sortestPath = Traversal.simplePathToString(foundPath,
NAME_KEY);
```

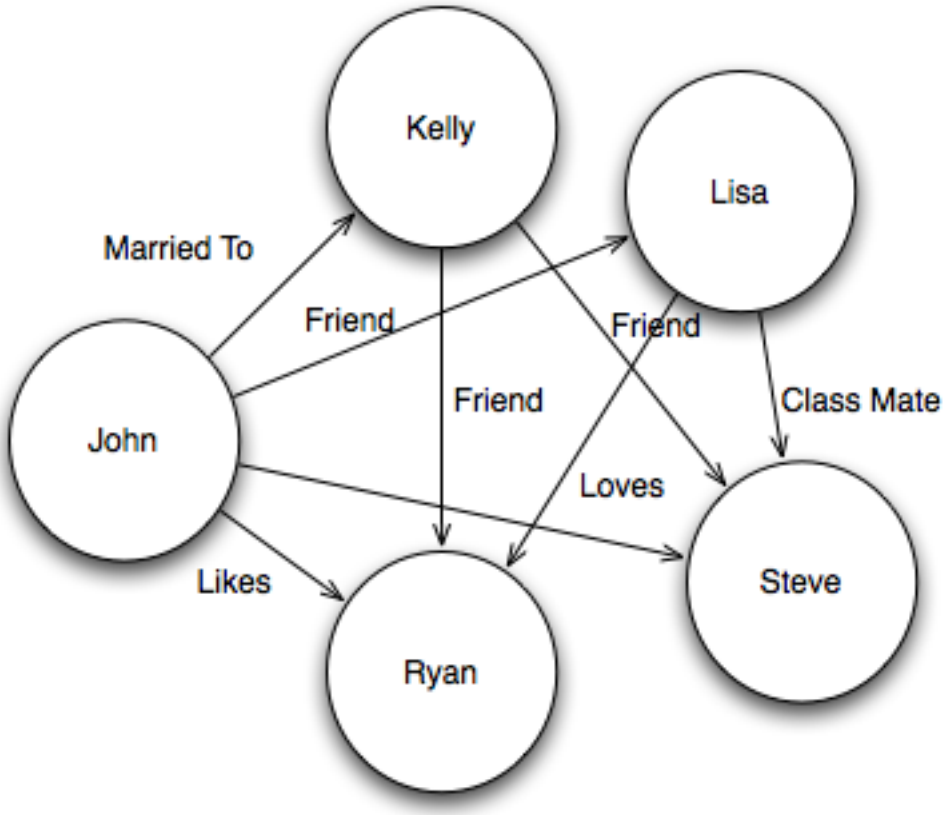


Figure 9. Highly connected data having more than one type of relationship

The graph algorithm factory of Neo4j has built-in algorithms, such as shortest path, all path, Dijkstra algorithm, etc. In order to use these algorithms, we first need to initialize the existing database with the graph database factory object. Once the instance of database is created, the database instance is mapped to GraphAlgoFactory, which allows the user to query any predefined algorithm from the graph algorithm factory. For the purpose of our project, we choose two algorithms: shortest path and all path. If the circumstances required, we can also write our own algorithm and pass it to the graph database using cypher query. Our purpose of writing the application is to review the technologies, so we did not write our own algorithm. The result of cypher query in the console of a graph database is as follows:

```
(Garrett Birkhoff)-->(Uta Caecilia Merzbach)<--(I. Bernard
Cohen)-->(Judith Victor Grabiner)
```

Our application processes the graph database console text result from the cypher query and converts the result to graphical form. Converting text result into graphical form is achieved by using the JavaScript InfoVis Toolkit framework (JIT). Figure 10 shows the snapshot of graph-visualization application showing shortest path between two nodes.

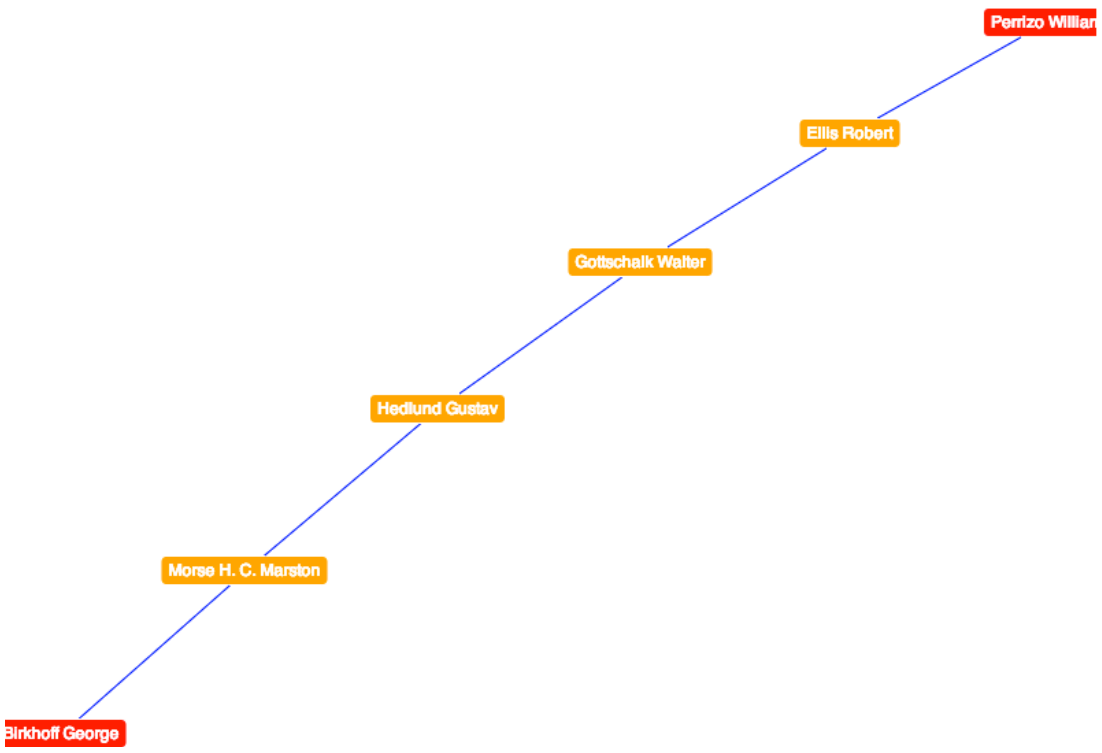


Figure 10. Snapshot showing the shortest path between two nodes

JIT is an open-source framework written in JQuery to support the visualization of various graph types. JIT supports graphs such as a bar chart, tree map, hyper tree, etc. For the purpose of our application, we have chosen the Force Directed Static Graph. Irrespective of the graph size, the force directed graph makes all graph nodes fit on a single page. The force directed graph of JIT framework also allow user to manually remove any unwanted nodes from the GUI of the application. The JIT framework takes JSON as its input parameter to convert it into the Forced Directed Static Graph. For converting the text result from the cypher query to JSON format, we wrote a parser that converts text result into JSON format. The pseudo code of parser is as follows:

```
Pattern p = Pattern.compile("-?\\-->|-?\\<--");
Matcher m = p.matcher(data);
While(true){
     temp = m.group(0);
     relation.add(temp);
}
for (int i = 0; i < nodes.size(); i++){
if (i < nodes.size() - 1) {
     if (relation.get(i).equals("-->")) {
               if(i==relation.size()-1){
                    json.append(nodes.get(i)).append(":{");
                    json.append(nodes.get(i + 1));
                    json.append(" :{'color':'blue'} }}");
                    }
               else{
               json.append(nodes.get(i)).append(":{");
               json.append(nodes.get(i + 1));
               json.append(" :{'color':'blue'} },");
                    }
          else {
               if(i==relation.size()-1){
               json.append(nodes.get(i+1)).append(":{");
               json.append(nodes.get(i));
               json.append(" :{'color':'blue'} }}");
else{
     json.append(nodes.get(i+1)).append(":{");
     json.append(nodes.get(i));
     json.append(" :{'color':'blue'} },");
}
```

The second unit of the business logic for the application is AllPathFinder class. AllPathFinder render graph to user similar to ShortestPathFiner class except it uses Google API to render graph. For the AllPathFinder class, we choose all path algorithms from the graph algorithm factory instead of the shortest-path algorithm. The algorithm returns a list of all paths and stores each path in an array. A parser converts each item of the array to JSON format and passes it to Google API. As a result Google API will render graph to the user. Figure 11 shows a snapshot of the output for the AllPath class.

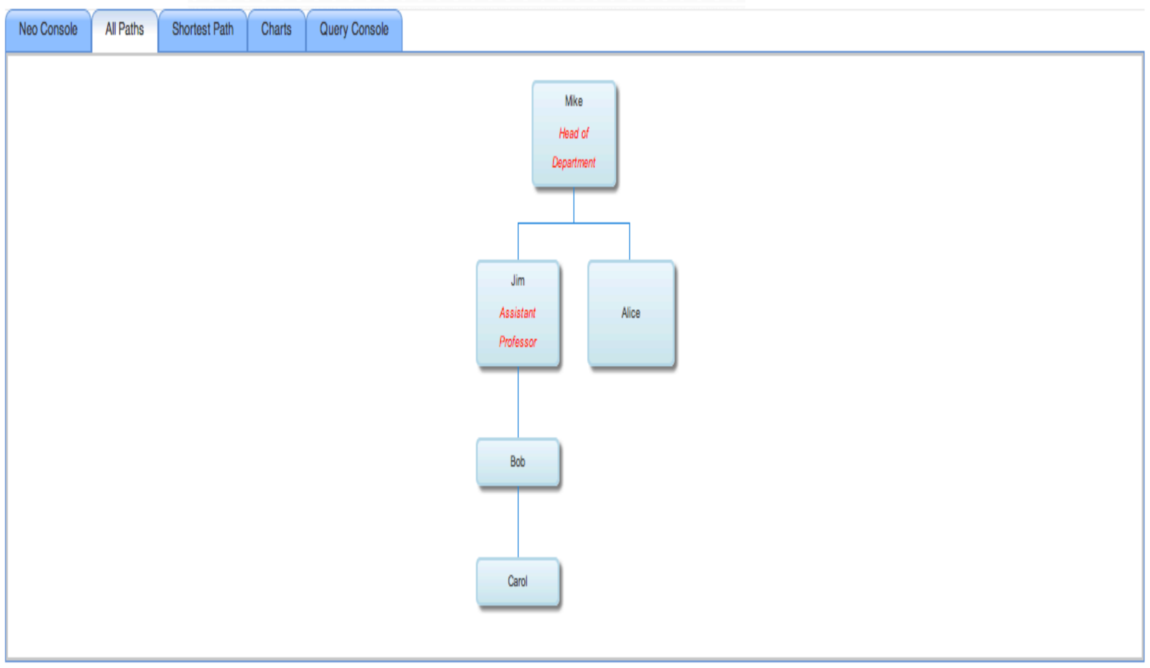


Figure 11. Snapshot showing all possible paths between two nodes

Some nodes in the database can have a large number of paths; for example, if we want to know all paths between two nodes, i.e., A and B, there might be multiple paths to reach from node A to B. For the purpose of simplicity, we have limited the number of paths to 20. If a node has more than 20 paths, then we randomly choose the first 20 paths and ignore rest of the paths. For example, when querying all paths between “Peter Werenfels” and “William Perrizo,” we found over 200 different paths. If we consider all 200 paths, then visualization will be complex and less effective. To reduce the complexity of graph visualization, we have limited the number of paths to 20. The Java code for finding all paths is as follows:

```
PathFinder<Path> finder = GraphAlgoFactory.allpath(
Traversal.expanderForTypes(KNOWS, Direction.BOTH), 20);
Path foundPath = finder.findSinglePath(neo, agentSmith);
String shortestPath =
Traversal.simplePathToString(foundPath,NAME_KEY);
```

The last unit of the business logic for our application is DepthFinder class. The purpose of DepthFinder is to show how any given node in the graph database is distributed at each depth. The DepthFinder will break length of any given node and show number of vertices at various depths and convert the result into graphical format. Figure 12 shows the snapshot of the application, showing descendants of two nodes at various depths. In a database, a length of a node can be more than 4, but for the purpose of simplicity, we have limited search of descendent at each depth to a maximum of length 4. The reason for limiting the number of depths to 4 is, again, for better and less complex visualization. The pseudo code of DepthFinder class is as follow:

```
GraphDatabaseFactory graphDb;
graphDb= GraphDatabaseFactory.newEmbeddedDatabase(DbPath);
indexService = graphDb.index().forNodes("nodes");
ExecutionEngine engine = new ExecutionEngine(graphDb);
ExecutionResult result = engine.execute("START n=node("+ id +
")MATCH (n)-[*.." + depth + "]-(x)RETURN count(x);")
for (Map<String, Object> row : result)
     {
       for (Entry<String, Object> column : row.entrySet())
       {
          data = column.getValue().toString();
       }
```

This method takes two parameters: the first parameter is the index number of the nodes for which user want to find decedents and the count number of descendants at each depth. And second parameter is maximum depth value the method will search for descendants. For the purpose of our application we have passed 4 as input parameter because we want to limit search to a length of 4. The result from above function is parsed and passed to Google API, which will render graph to a user. In Figure 12, the blue line shows the number of descendants of “George Birkoff” at various depths, and the red line shows the number of decedents of “William Perrizo” at various depths.

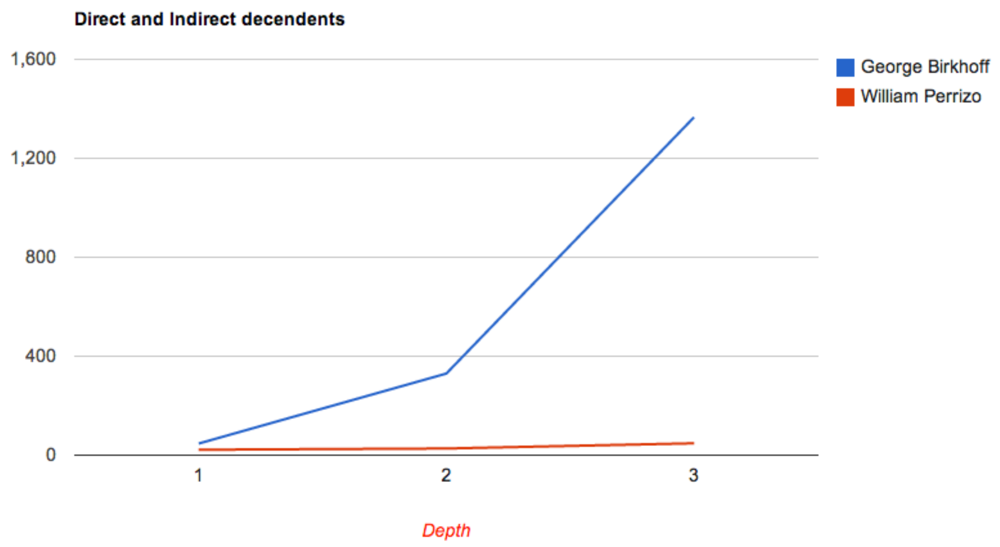


Figure 12. Snapshot showing number of descendants at various depths

To get the graph shown in Figure 12, first, the user makes a search request from a browser. The application looks for the requested action class. If it does not find the requested search-action class, it will redirect user to an error page. If the application finds the requested action class, it will invoke the appropriate business-logic class associated with relevant action class. Business logic is the only unit in the application that has direct access to the database. Second, the result received from the query is passed to the parser of business logic. The parser parse the data into JSON format and passed the parsed data to action class. As mentioned in the previous chapter, the Struts framework contains tag libraries that can access data from the action class directly to the JSP page without writing Java code on the JSP page. This property allow view object to access data of action class in the MVC architecture. Finally, parse data is passed by action class to viewer and viewer display graph to the user. Figure 13 shows the activity diagram of the Math Genealogy application using an embedded graph database.

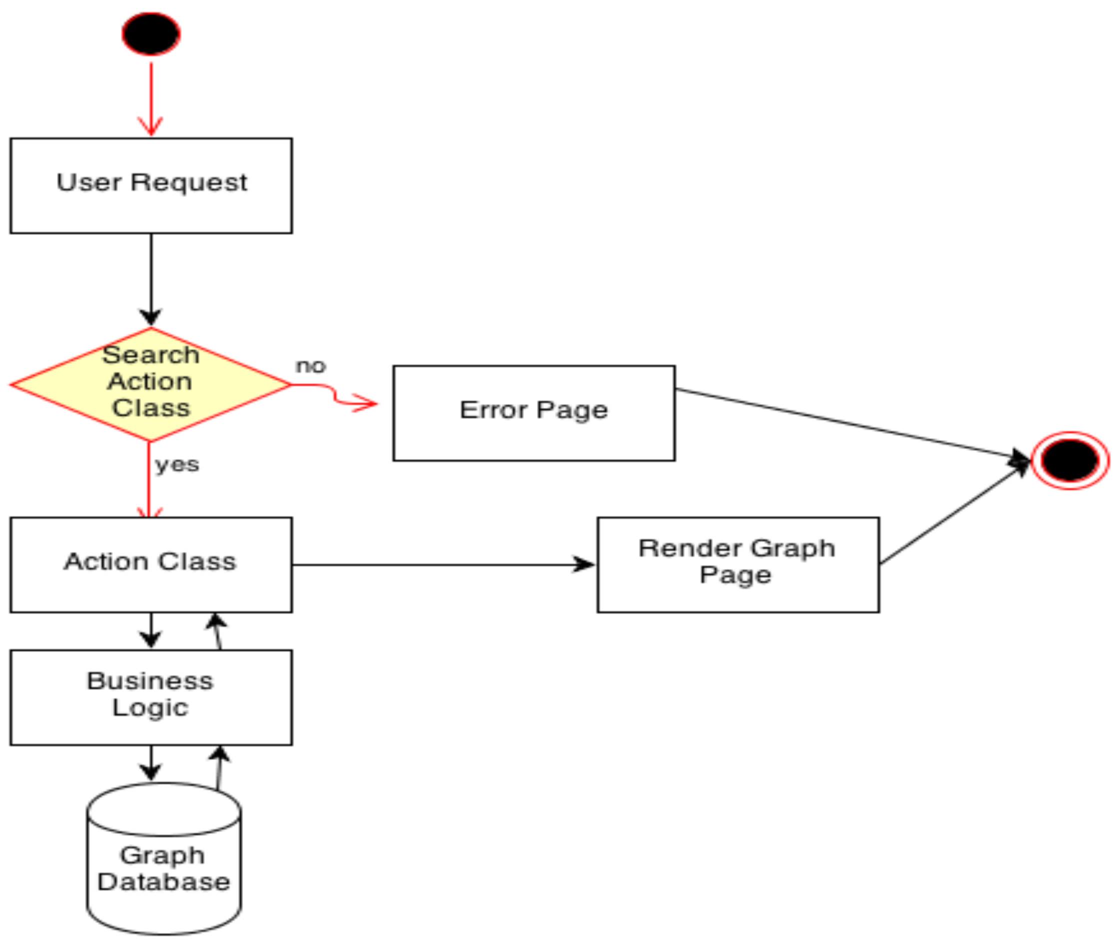


Figure 13. Activity diagram of graph-visualization application

3.8.2. Graph Visualization Using the REST Service

The purpose of using REST service is the web service open gateway to multiple programming languages such as Python, Java, and Perl to manipulate graph database. In this approach, the graph database will be running in a server. Users can manages the resource remotely using REST service. This feature allows programmer to work with database from application layer. For the case of the graph-visualization project, the resource is the Neo4j server running with the Math Genealogy project dataset. From the terminal, we first start the Neo4j server:

```
joshi-:neo4j kailashjoshi$ bin/neo4j start
WARNING: not changing user process [13823]... waiting for
server to be ready........... OK.Go to
http://localhost:7474/webadmin/ for administration interface.
```

Once the server has been started, users are now ready to query database remotely. The REST application contains three main JQuery files: Graph.js, Model.js, and RestAPI.js. Graph.js script convert result of cypher query into forced directed graph. Model.js helps to parse the result to appropriate JSON format. It also help user to edit graph from GUI. RestAPI.js is responsible to pass cypher query of user to the database server running remotely. The script is also responsible to provide result of query to parser of Model.js. These files are responsible for manipulating the graph database. The user submits the cypher query through the browser. The request is sent as follows:

```
GET http://localhost:7474/db/data/ Accept: application/json
```

The Neo-server processes the cypher query and sends a response to the web browser in JSON format. The example of a response object by the Neo-server is as follows [14]:

```
200: OK
Content-Type: application/json
{
"extensions" : {
  "node" : "http://localhost:7474/db/data/node",
  "reference_node":"http://localhost:7474/db/data/node/6260",
  "node_index" : http://localhost:7474/db/data/index/node
}
```

The response object is sent to a parser of the visualization application where parser of graph-application converts the response object into an input parameter of JIT. The JIT task is to display a graph to the user in a visualized manner. We have chosen a force-directed graph for the purpose of our application. Figure 14 shows the flow for the REST service.

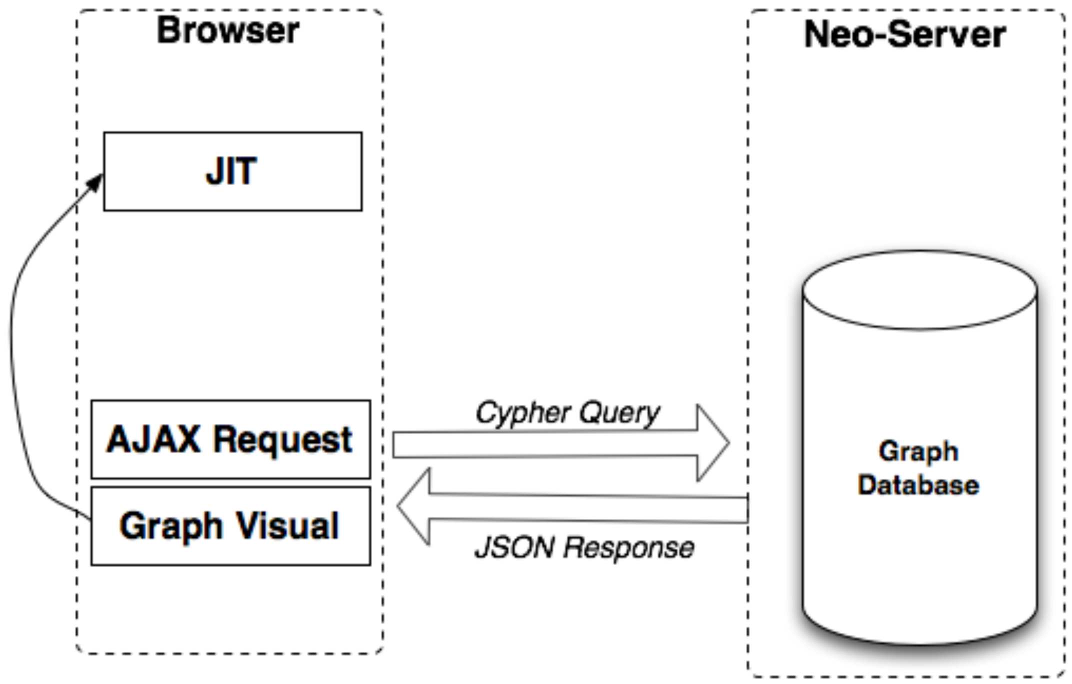
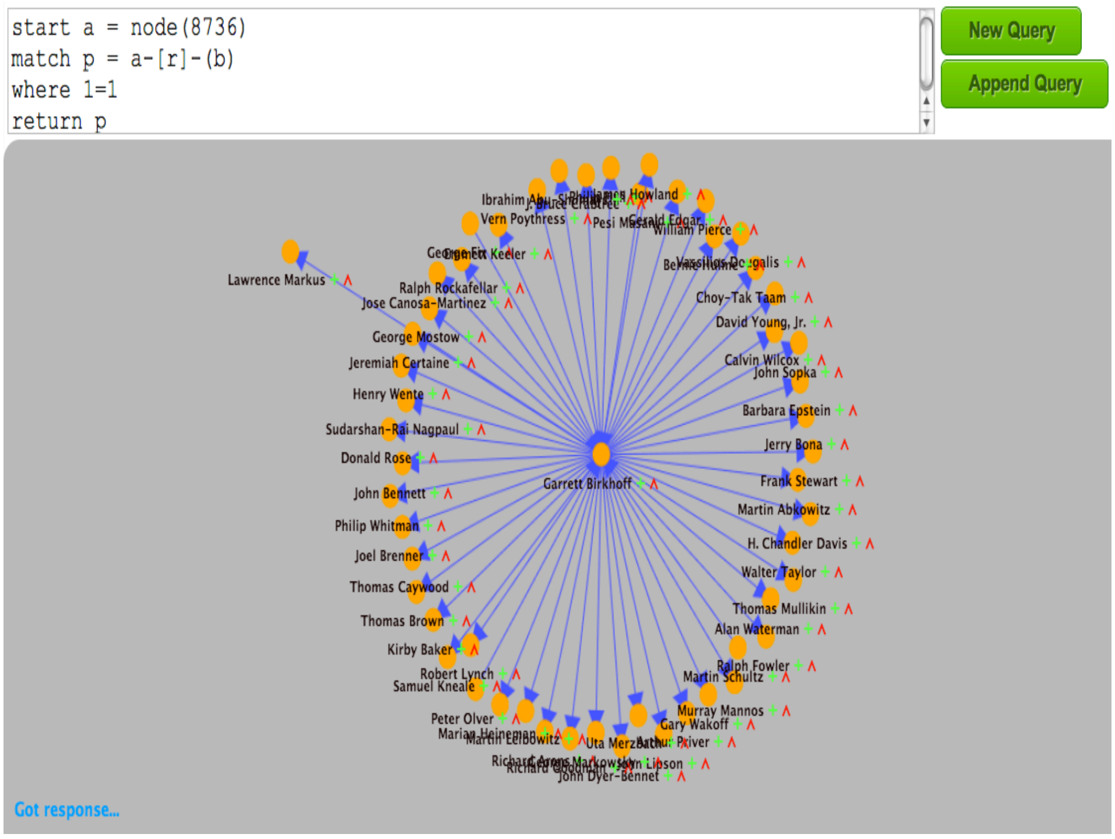


Figure 14. Interaction of graph-visualization application with REST service

The GUI of the REST service-based application contains buttons, a textbox, and a graph-visualization console board. Figure 15 shows the GUI of the graph-visualization application. The New Query button clears the graph display console board before adding a new graph to the console. Append Query button is responsible to appends the new graph to an existing console board as shown in Figure 15. User of the application can manually delete unwanted node from the console board. User can also move entire graph anywhere inside the console. All these properties of GUI of the application make user to analyze data effectively from the graph database.



```
start a = node(8736)
match p = a-[r]-(b)
where 1=1
return p
```

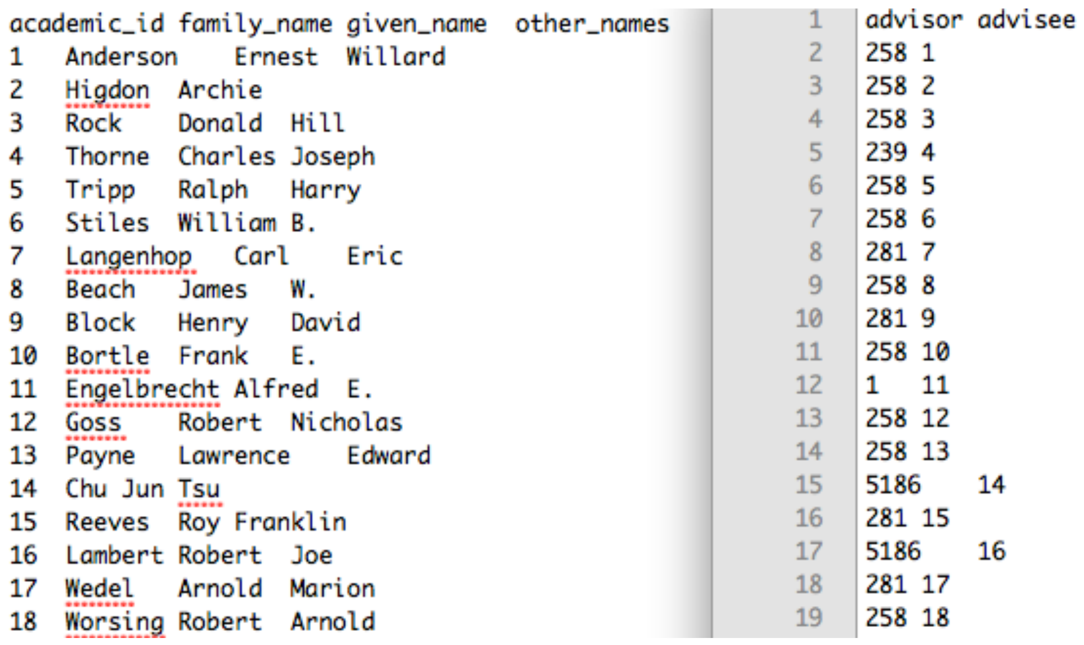
Figure 15. Snapshot showing graphical console of graph database

3.9. Dataset

The Math Genealogy Project at NDSU approved us to use its dataset for testing our application. The Math Genealogy Project has been collecting data about all mathematicians around the world for almost a decade. The dataset contains the relationship between the adviser and advisee as well as other information related to them. The dataset has information about 165,124 different advisers and advisees, and 171,324 different relationships between the adviser and advisee.

3.9.1. MapReduce Job and Lookup Table

The dataset from the Math Genealogy Project contained two sets of tables. The first table contained detail information about advisers and advisees indexed by the unique ID number. The second table contained the adviser and advisee relationships in terms of ID number. Figure 16 shows table structure of Math Genealogy Project. Our goal is to process data from two tables to upload the date to graph database.

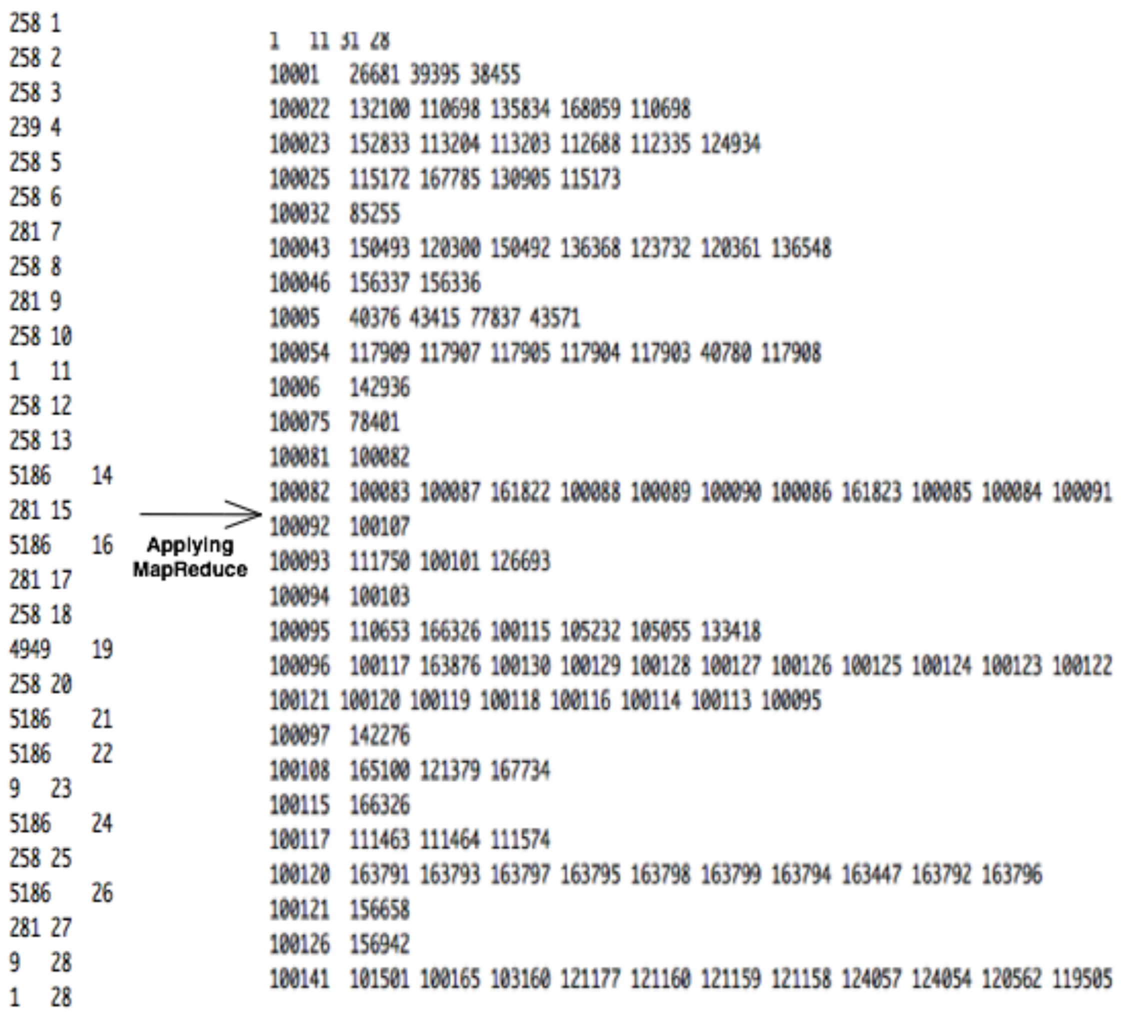


```
academic_id family_name given_name  other_names
1    Anderson    Ernest  Willard
2    Higdon  Archie
3    Rock    Donald  Hill
4    Thorne  Charles Joseph
5    Tripp   Ralph   Harry
6    Stiles  William B.
7    Langenhop   Carl    Eric
8    Beach   James   W.
9    Block   Henry   David
10   Bortle  Frank   E.
11   Engelbrecht Alfred  E.
12   Goss    Robert  Nicholas
13   Payne   Lawrence    Edward
14   Chu Jun Tsu
15   Reeves  Roy Franklin
16   Lambert Robert  Joe
17   Wedel   Arnold  Marion
18   Worsing Robert  Arnold
```

```
advisor advisee
258 1
258 2
258 3
239 4
258 5
258 6
281 7
258 8
281 9
258 10
1   11
258 12
258 13
5186    14
281 15
5186    16
281 17
258 18
```

Figure 16. Schema of Math Genealogy Project using relational database

To convert relational database dataset into graph database format dataset, first we need to process the format of data, de-normalize the dataset, and uploaded into our graph database. The reason for processing dataset before upload it to database is because received data is distributed into two different tables. If we upload the dataset without processing them it will take long time to upload 200 thousand nodes into graph database because for finding properties of any given node, we have to iterate entire property file. Now, the challenge was to reduce such iteration while uploading data into graph database. To reduce the number of iteration, we used two different techniques, MapReduce and Lookup Table. Using MapReduce, we were able to shrink the size of dataset. By running the MapReduce job, we were able to reduce 171,324 iterations of relationships into 38,937 iterations. Figure 17 shows the output of our MapReduce model.



```
258 1
258 2
258 3
239 4
258 5
258 6
281 7
258 8
281 9
258 10
1   11
258 12
258 13
5186    14
281 15
5186    16
281 17
258 18
4949    19
258 20
5186    21
5186    22
9   23
5186    24
258 25
5186    26
281 27
9   28
1   28
```

Applying MapReduce

```
1   11 51 28
10001   26681 39395 38455
100022  132100 110698 135834 168059 110698
100023  152833 113204 113203 112688 112335 124934
100025  115172 167785 130905 115173
100032  85255
100043  150493 120300 150492 136368 123732 120361 136548
100046  156337 156336
10005   40376 43415 77837 43571
100054  117909 117907 117905 117904 117903 40780 117908
10006   142936
100075  78401
100081  100082
100082  100083 100087 161822 100088 100089 100090 100086 161823 100085 100084 100091
100092  100107
100093  111750 100101 126693
100094  100103
100095  110653 166326 100115 105232 105055 133418
100096  100117 163876 100130 100129 100128 100127 100126 100125 100124 100123 100122
100121 100120 100119 100118 100116 100114 100113 100095
100097  142276
100108  165100 121379 167734
100115  166326
100117  111463 111464 111574
100120  163791 163793 163797 163795 163798 163799 163794 163447 163792 163796
100121  156658
100126  156942
100141  101501 100165 103160 121177 121160 121159 121158 124057 124054 120562 119505
```

Figure 17. Dataset after applying MapReduce programming paradigm

With MapReduce model, our new dataset consists of node and its relationship with other node in an array form. Our next task was to map each index number of a node to relevant properties of the node so that our database will have complete dataset. This way, we were able to upload a entire data to graph database in a single iteration. In order to reduce the iteration of findings, the value of the index, we created a lookup table.

3.9.2. Lookup Procedure

The lookup table was also necessary to avoid a memory issue while uploading data to graph database. Our dataset size was 6.3 MB, so we divided 6.3 MB of data into 51 individual files. This way we were able to reduce property lookup time for any give ID as well as memory related issues. The result received from MapReduce job gives array of relationship among node in a compact form. To create a lookup table, we have divided the property table received from Math Genealogyproject into 51 small files. We label each files as index-Chunk-idnumber. Each Index-Chunk file was labeled as IC-1, IC-2, etc. In lookup table we have mapped each range of node index number with one indecx-Chunk file. For example, for a range of index value of 1 to 3,300, IC-1 file is mapped in a lookup table so that search iteration using lookup reduce each iteration from 165,000 to 3,300.

After creating lookup table, we now need to upload data into graph database. First we start a database. To start a database, first we should create an instance of the GraphDatabaseFactory class, which will give an option whether we want to update an existing database or create a new database. In our case we choose to create new database. The graph database applies a lock system, which means multiple instance of database cannot be created at a time. This feature helps to guarantee that the user will always have an updated database.

```
GraphDatbaseService graphDb = new
GraphDatabaseFactory().newEmbeddedDatabase( DB_PATH );
```

As mentioned above, this database uses a lock system when updating the database. In some instances, if the previously opened session was not closed properly, the database will not allow us to update the database or create a new database. To avoid this dread lock, we add a shutdown hook before starting the database. The shutdown hook ensures that database is properly closed before starting the database. The pseudo code of the shutdown hook is as follow:

```
    Runtime.getRuntime().addShutdownHook( new Thread(){
        @Override
        public void run()
        {
            graphDb.shutdown();
        }
        });
```

3.9.3. Why Ne04j?

There are various types of graph databases available on the market. We look for an open source as well as a database that fits our purpose. Neo4j is ACID (Atomicity, Consistency, Isolation, Durability) database and consistency is the factor we are looking for our application therefore we choose Neo4j, a graph database, for the purpose of our project. Neo4j is a conventional, distributed database because it can be embedded into various programming languages, such as Java, Python, Ruby, etc. In terms of performance, Ne04j running on a commercial machine allows us to do 1.2 million traversals per second. Here, a traversal means moving from a node to its edge. This property will allow us to explore the depth of the database in one instance and in a very short time period. Another reason for picking Neo4j is because Neo4j is a well-documented database. The Neo4j cluster is very similar to MySQL. This cluster is fault tolerance [15] because it periodically checks for the presence of any corrupt file and replaces the corrupt file, if present, from its backup file. Figure 18 shows typical Neo4j cluster. It consists of several Neo4j instances that are either embedded or running in server mode. A configuration file is created in the cluster so that nodes can communicate with each other over the network. One disadvantage of using Neo4j is that, in a network, it can only read the database; it cannot write over the distributed server. Although the Neo4j team is working to make it writable, this problem is yet to be resolved. On the other hand, manipulation of graph database is done from an application layer. The Neo4j model is created more to read than to compute, so the amount of reading is more than the amount of writing for a graph database.

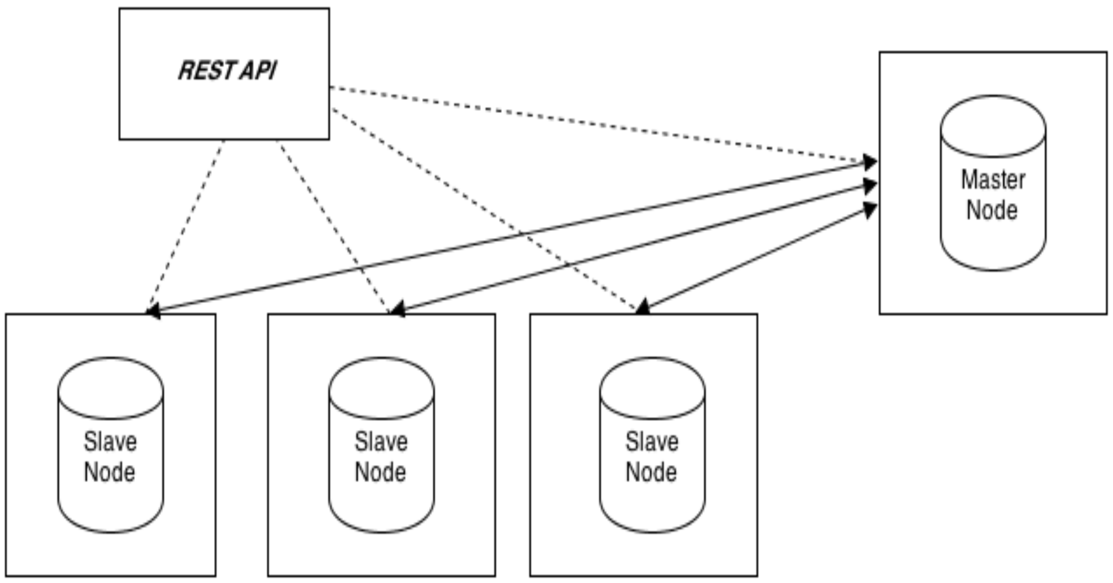


Figure 18. Typical setup of Neo4j database running in a distributed server

The graph database model is gaining popularity because of its flexibility and rapid development time. It is easier to quickly add any new functionality without affecting previous deployments, which helps to design new features. Most of the NoSQL databases in the market are scalable database. Although graph database is NoSQL database but graph database are not consider highly scalable database because database are stored on a single machine. This is the major disadvantage of using graph database.

CHAPTER 4. ANALYSIS AND EVALUATION

In this chapter, we show test the performance of relational database and NoSQL database. We will also evaluate our application in terms of efficiency and time complexity.

4.1. Analysis of the Graph Database

The graph database is used for highly connected data efficiently. To demonstrate the point, let us take an example. Let us assume that a set of data contains profile of 1,000 different people from a social networking site and each person has an average of 40 friends. We note the query time of both relational database and graph database to find a person name and list of friends of a person. We have repeated our test twenty times to avoid any biasness in the query time. Our test result shows that the relational database took an average of 367.385 milliseconds (ms) to complete the task while the graph database took an average of 1.45 milliseconds (ms) to complete the task. Figure 19 shows the bar chart of relational database and graph database in terms of difference in query time. As we can see in the figure that query time for graph database is the rage of 1 ms to 2 ms where as query time for relational database is in the range of 350 ms to 525 ms. If the dataset is smaller in size, the query time for both relational database and graph database may not be significantly different. It will only be comparable if the dataset is large. Data in a graph database are stored in a structured and sorted manner; therefore, traversal time for the database will be constant. The query time in relational database depends upon number of friends that a person have. If the person has large number of friends, then the query time will increase. But in graph database, irrespective of number of friends person has, the query will be constant because all information related to friends are stored within the node of a given person.

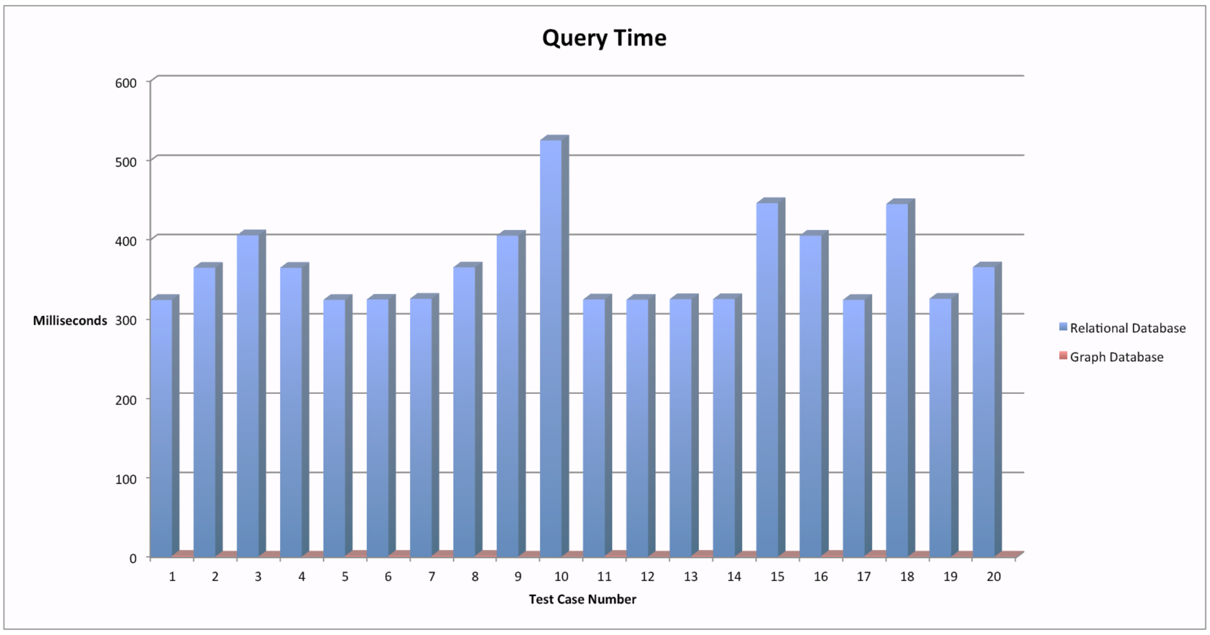


Figure 19. Bar chart showing difference in query time

4.2. Evaluation of the Project

Let us take Math Genealogy dataset to evaluate the time complexity between SQL and NoSQL. The Math Genealogy Project at NDSU uses a relational database to process the data. For simplicity, let us assume that the database only contains two tables. The first table stores information about the adviser and advisee, such as first name, middle name, and last name. Each row of this table is indexed with an ID number. The second table contains the relationship between the adviser and advisee. Table 2 shows table structure of the Math Genealogy Project using a relational database. We have calculated and compare complexity of query process using relational data and graph database. Our query for both relational database and graph database let is to find all students name for an advisor, Kendall Nygard, from the database. First we will calculate and evaluate complexity using relational database.

Table 1. Schema of Math Genealogy project using relational database

Properties			
ID	familyName	givenName	middleName
680	…..	……	….
681	….	…..	….
690	Kendall	Nygard	

Relationship	
Advisor	Advisee
690	87
690	88
690	600

4.2.1. Relation Database

In relational database we store data into tables and create relationship among table. If we want to find all students for an advisor, Kendall Nygard, from the table we have to follow following steps

1. In the beginning, query familyName field of property table to find the name “Kendall Nygard.” The time complexity for this process is O (Logn).
2. Once the name familyName is located, find the index associated with the name. The time complexity of this step is O (1).
3. In the relationship table, find all relationships associated with the ID number found in step 2. Let us suppose that the total number of rows in a table is n, so the time complexity of the process will be O(logx):x<<n. The size of x should always be less than n.
4. Form the list found in step 3, get the ID number for each advisee in the relationship table. The complexity for this process is O(X).
5. Go to the Properties table and locate the ID number in the table of all IDs for the list collected in step 3. The time complexity for this step is O(Xlogn).
6. Find the familyName of each index from the step 5. This operation yields all names of direct descendent for Dr. Nygard. The time complexity for this process is O(x).

In a relational database, although data are indexed and organized properly, the graphs are not a relational structured but, rather, is constructed using indexed intensifying operations. In the above example, while only a subset of data is required, the entire table needs to be traversed. The reading time of the relationship in our example is O(Logn), which is fast as long as data set is not large. Users with a small dataset might not notice the performance difference, but the performance can be observed with larger datasets.

4.2.2. Graph Database

Graph databases have three different main elements: nodes, relationship, and properties. Each node in a graph is managed with indexes similar to a relational database. Now, let us try to solve the previous problem of finding all direct descents of Dr. Nygard from the graph database. The steps and time complexity are as follows:

1. Find the index number of a node, which has a property name, familyName, equal to "Nygard." The time complexity of this step is O(Logn).
2. Let us say the vertex retrieved from the first step gives x number of edges. The time complexity to access each edge is O(x).
3. For the list received from the second step, get k number of properties from each edge received in first step. The time complexity for this step is O(kx).

The above operation is efficient because, in a graph database, there is no JOIN operation. And data are stored in semi structure form. The vertices are directly connected with their adjacent node, so it is quicker to access the edges. Figure 19 shows the structure and organization of graph database.

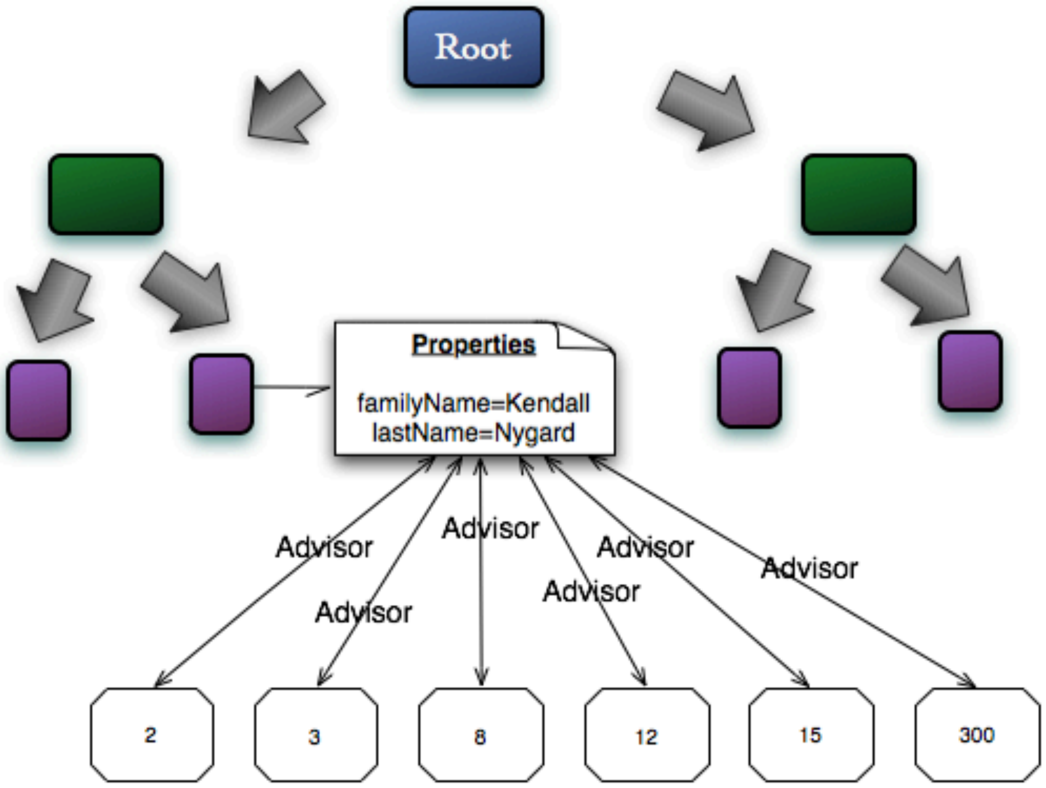


Figure 20. Structure of graph database using Neo4j

In a graph database, traversing from one vertex to another vertex has a constant time, so the total traversal time for a graph database is the total nodes traversed by a query multiplied by the time to travel from one vertex to another [12].

CHAPTER 5. CONCLUSION AND FUTURE WORK

5.1. Conclusion

To the date the popularity of relational database is wider as compare to NoSQL database because relational database is a mature and stable database. On the other hand, NoSQL database are evolving hence not stable database. But the popularity of NoSQL is growing rapidly. Evolution of NoSQL database is the result of increase in data size to process regularly. For example, Facebook process more than 500TB of data every day. Facebook is a social networking site so the nature of Facebook data is highly connected. As we know that graph database is designed for highly connected data therefore company like Facebook is regular user of graph database. NoSQL also seeks to overcome the drawbacks of SQL, such fixed schemas and JOIN operations, and addresses the scalability problems. There are four emerging categories of NoSQL i.e. Key-Value stores, ColumnFamily, documented databases and graph database. All of the four categories are governed by CAP theorem and different category is suitable for different situations. For example, if the nature of data is highly connected, we choose graph database. Graph database are suitable for highly connected data. Most of the highly connected data are social networking data. Graph database is efficient as compare to relational database because in graph database data are stored in a semi structure form hence query time is constant irrespective of size of the database. Through extensive literature review we have tried to determine whether NoSQL can be better in a situation such as Academic Search where the relationship among nodes is complex or relational database.

To test two different databases, we have created a sample data set of 1000 profiles. Each profile in our sample has an average of 40 friends. We then load the sample data into relational database as well as graph database and run the query with each database to find all friends of any given profile. Our test result shows that average query time of relational database to find all friends of any given profile is 367.385 ms while the average query time to perform same task in graph database is 1.45 ms. Our test results give us an indication that if the dataset are of the nature of social media and highly connected, graph database could be more appropriate. From our test result, we have also conclude that when the size of dataset is not large and there is a fixed type of data flowing into database, then relational database can be more efficient because of the concept called normalization. Use of normalization in relational database reduce redundancy of data hence performance improvement in query time of the relational database. User will not find significant difference in the performance of graph database if the dataset is small. But if the dataset is sufficiently large, the user of the graph database will see the performance of database in action because data in graph database are stored in semi structure form. Hence the query time of graph database is constant irrespective of the size of the dataset.

To review relational database and NoSQL database, we have also built an application using graph database. Math department of North Dakota State University provides dataset for our project. In choosing the type of database for our project, first we looked the nature of data of we received from the math department. The nature of data received from the math department is social networking data and is highly connected data. As we know from our literature review that graph database are suitable for highly connected and social media data. Therefore we choose graph database for our project. Our first task was to convert relational database tables into key-value format. Graph database stores its properties using key-value. To convert relational database table into key-value we have used various technique like MapReduce and Lookup table. Our next task was to build a dashboard where a result of the graph database could be represented into different type graphical format to provide maximum information to the user using our application. Our dashboard shows user shortest path between any two nodes in the graph database. The dashboard of our application also breaks length of any given node in graph database and shows number of vertices at different depths in graphical form. The dashboard will help user to understand how nodes are distributed in the graph database at various depths and find the shortest path between any two nodes. We have also built a convenient console using graph database where any query passed by user will return result in a graphical format. The main logic for building the console is to provide various options to users using graph database. Typically, results of graph database queries are returned to users in a textual format but we gave an option to users to view result in a graphical format. We have also given user the flexibility to manipulate graph from the console. Any user using our console can add, delete or update graph node from the console of our application.

5.2. Future Work

The dataset for the Math Genealogy Project is outdated and incomplete. A manual procedure was used to record the data into a database. In order to make it complete, we have to find alternative ways of getting data. The procedure for getting data should be more automatic than manual. One solution to make the database complete is to mine the researcher's profile from web and digital libraries such as Association of computing machinery (ACM) and Institute of Electrical and Electronics Engineer (IEEE) [14]. Data collected from a web and digital library have complex relationships with many different properties. In this type of situation, our application model best fit. The database needs to store a large collection of data that lack fixed schema. Our application uses NoSQL, meaning that schemas need not be fixed. Second, if we get a complete database for an Academic Search, then the size of the database can be large. Microsoft Academic Search has over a million nodes. As the database size grows, hosting a complete database on a single server might have performance issues. This arise the problem of scalability. To solve the scalability issue we can use High Availability of Neo4j. Neo4j High Availability is fault tolerance database architecture. Another important issue that can be addressed in future is security. In our graph visualization application, we have ignored security. Any user of our application can add, delete, or modify the database. To make an application effective we can create different user group such as faculty, researchers, students, system administrators, and general users, each group have different read and write permissions.

REFERENCES

1. DeCandia, Giuseppe, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, and Peter Vosshall. "Dynamo: amazon's highly available key-value store." *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles* 41 (2007): 205-220.
2. Chang, Fay, Jeffrey Dean, Sanjay Ghemawat, Wilson Hsieh, Deborah Wallach, Mike Burrows, Tushar Chandra and Fikes Andrews "Bigtable: A Distributed Storage System for Structured Data." *Google Inc*. (2006).
3. Card, Stuart, Jock Mackinlay and Ben Shneiderman. "*Data Visualization The Value of Visualization.*" Web. 26 Feb. 2013.
4. Few, Stephen. "Data Visualization for Human Perception. "*The Encyclopedia of Human-Computer Interaction*. Web. 26 Feb. 2013.
5. Kandel, Sean, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. "Enterprise Data Analysis and Visualization: An Interview Study." *Visualization and Computer Graphics, IEEE* 18.12 (2012): 2917-962. 08 Oct. 2012. Web. 28 Feb. 2013.
6. Eifrem, Emil. "Graph Databases: The New Way to Access Super Fast Social Data." *Mashable*. Web. 26 Feb. 2013.
7. Carlo, Strozzi. "NoSQL A Relational Database Management System." Weblog post. *NoSQL: A Non-SQL RDBMS*. Web. 26 Feb. 2013.
8. Zyp, Kris. "NoSQL Architecture." *sitepen*. Web. 26 Feb. 2013.
9. " A Database for the Web" *Apache CouchDb*. Web. 28 Feb. 2013.
10. Gilbert, Seth, and Nancy Lynch. "Perspectives on the CAP Theorem." Web. 26 Feb. 2013.

11. Hussain, T., Shamail, S., Awais, M.M., "Eliminating process of normalization in relational database design," *Multi Topic Conference, 2003. INMIC 2003. 7th International* (2003): 408-13.
12. Marko A. Rodriguez and Peter Neubauer. "The Graph Traversal Pattern." *Cornell University Library* (Apr. 2010).
13. Fielding, Roy T. Architectural Styles and the Design of Network-based Software Architectures. Diss. University Of California, 2000.
14. "The Neo4j Manual v1.9-SNAPSHOT." *Neo4j the graph database.* Web. 26 Feb. 2013.
15. Ivan Herman, Guy MelancËon, and M. Scott Marshall. "Graph Visualization and Navigation in Information Visualization: A Survey." *Visualization and Computer Graphics, IEEE* 6 (2000): 24-43.