Comparing Database Options for a Web Application

Anupriya Srivastava
Computer Science
University Of Hawaii
96813
anupriya@hawaii.edu

ABSTRACT

This project aims to find the most appropriate database for a web application. Database is very essential for quick storage and retrieval of information in any application. The paper concentrates on calculating the average time required to insert and fetch data by each database. The two databases used in the experiment are PostgreSQL relational database and MongoDB noSql database. The paper looks into the requirement of the application and thus decide the best database for it. Using two completely different databases adds to the clarity of the results. The paper also looks into indexing and implements Apache Lucene with each of the databases. The results were quiet predictable, MongoDB with Lucene faired the best.

1. INTRODUCTION

An affective and quality database design helps to reduce time and the overall cost of the system development process. Taking a correct approach to database design helps to understand the user requirements correctly and deliver a system the user desires. Therefore, to increase the overall performance of the application, I tested my application with two types of databases , Sql and Nosql. Further I tested each type with and without indexing. For indexing I used Apache Lucene as it is the most flexible and convenient open source search toolkit. Doug Cutting, who originated Lucene [11], describes his primary goal for Lucene as "simplicity without loss of power or performance," and this shines through clearly when using it.

The application is built on the MVC pattern architecture using the Play framework[4], with different backend databases. MVC stands for Model-View-Controller. Here the Model represents the application core (for instance a list of database records). The View displays the data (the database records). The Controller handles the business logic. Below is a brief description of how the MVC pattern works.

This paper aims to study the requirement of the Application (described further in section 2.1), and thus determine which is the most appropriate database for the Application. The databases tested were, PostgreSQL 9.3 for the Sql database type and MongoDb for the noSql type. Test cases were designed keeping in mind the advantages of both the databases,which are discussed further in the paper. The experiment is taken a liltle further by testing both types of databases with indexing using Apache Lucene. The results of the experiments was analyzed both quantitatively and qualitatively.

2. BACKGROUND

In this section, I describe the background: The application for which the databases were tested, about the databases and about the indexing java library Lucene.

2.1 The Web Apllication - Taggit!

Taggit attempts to help you unclutter your mind and your life by letting you organize and catalog your day to day experiences in a searchable, sortable and manageable online medium. Whether it is a website you like or thoughts rambling through your head, once you link them with Taggit, it will automatically parse, tag and contextualized that snippet of your life for future retrieval whenever you feel an itch in your brain to reminisce about a thought you can't fully recall.

While this seems like a very expansive undertaking, the idea is pretty simple. Whenever you come across moment in your digital or analog lives that you wish to hold onto just a little bit longer...just link it with Taggit, and carry on living the moment. Taggit will search for the relevant tags and save that snippet with the proper contextual information for future retrieval whenever you want to relive that moment.

Taggit is similar to several other apps out there that essentially serve as virtual diaries or scrap books. Apps like Evernote and pocket serve as repositories of your digital lives and allow both tagging and searching. However Taggit attempts to take this one step further through automatic contextualizing and keywords linking. Tagging and cataloguing are done without user input behind the scenes. It also aims to serve a different end purpose: more than just an online directory for content that the user himself organizes, it is meant to be a dumping ground for random moments from day to day life that are sorted and managed transparently.

2.1.1 Database Requirement

Information being stored for the application can come from a wide range of sources: website URLs, research papers, journal/ blog entries, social media comments, pictures

Entity	SQL Databases	NoSQL Databases
Structure	Table based databases	Document based, Key-value pairs
Schemas	Fixed	Dynamic
Scaling	Vertical	Horizontal
Consistency	Follows ACID	Follows BASE
Reliability	yes, eventually	yes
Support	not that good	yes, due to its legacy.
Complex query	not as good as SQL	yes

Table 1: Comparison of Sql and NoSql

and videos. The scope of the application includes text data, time data, image data and numerical data.

2.2 Databases Used

Relational database (RDBMS) like SQL has been the primary model for database management during the past few decades. But today, non-relational, NoSQL databases are gaining prominence as an alternative model for database management.

Introduced in 1970s, the relational model offers a very mathematically-adapt way of structuring, keeping, and using the data. This model organizes data into one or more tables (or ”relations”) of rows and columns, with a unique key for each row. Each database is a collection of tables, which are called relations, hence the name ”relational database”. Generally, each entity type described in a database has its own table, the rows representing instances of that type of entity and the columns representing values attributed to that instance. Because each row in a table has its own unique key, rows in a table can be linked to rows in other tables by storing the unique key of the row to which it should be linked (where such unique key is known as a ”foreign key”).

The structured approach of RDBMS database slows down performance as data volume or size gets bigger and it is also not scalable to meet the needs of Big Data. NoSQL is centered on the concept of distributed databases, where unstructured data may be stored across multiple processing nodes, and often across multiple servers. This distributed architecture allows NoSQL databases to be horizontally scalable; as data continues to explode, just add more hardware to keep up, with no slowdown in performance. This also means that unlike relational databases the data need not be normalized.

2.2.1 Sql Database - PostgreSQL

PostgreSQL is a powerful, open source object-relational database system. It has more than 15 years of active development and a proven architecture that has earned it a strong reputation for reliability, data integrity, and correctness. It runs on all major operating systems. It is fully ACID compliant, has full support for foreign keys, joins, views, triggers, and stored procedures (in multiple languages). As a database server, its primary function is to store data securely, supporting best practices, and to allow for retrieval at the request of other software applications.

Since the relational databases offer consistency and a higher learning curve as I am already familiar with it, I decided to test my application with PostgreSQL 9.3.

2.2.2 NoSql DAtabase - MongoDB

MongoDB is a distributed document database that provides high performance, high availability, and automatic scaling. It uses the concept of *collections* and *documents* to model data . A collection is a grouping of MongoDB documents which normally have similar schemas. A collection is analogous to a table in relational databases and a document is analogous to a table record. Documents are modeled as a data structure following the JSON format, which is composed of field and value pairs. Each document is uniquely identified by a *_id* field as the primary key. The values of fields may include embedded documents, arrays, and arrays of documents [7]. Relationships between documents can be modelled in two ways: references and embedded documents [6].

Since the amount of data in my application will become large overtime: linearly proportional to time and number of users, I decided to test it with MongoDB 3.0.3.

2.2.3 Lucene for Indexing

Apache Lucene indexes a document by breaking it down

Name	MongoDB	PostgreSQL
Description	One of the most popular document stores	Based on the object relational DBMS Postgres
Database model	Document Store	relational DBMS
License	Open Source	Open Source
Data scheme	Schema free	yes
Triggers	no	yes
Foreign keys	no	yes
Transaction	no	ACID
Concurrency	yes	yes
Durability	yes	yes

Table 2: Comparison of MongoDB and PostgreSQL

into a number of terms. It then stores the terms in an index file where each term is associated with the documents that contain it, kind of like a hashtable. Terms are generated using an analyzer which stems each word to its root form. When a query is issued it is processed through the same analyzer that was used to build the index and then used to look up the matching term(s) in the index. That provides a list of documents that match the query.

The version used in the experiments is Lucene 5.1.0

3. DATASETS

In this section we take a look at the dataset used for the experiment. The dataset was constructed my manual entry. The data on which I tested the databases consist of basically three data types: text, image, date and numerical data. The exact data model for each database is described in the subsections 4.1.1 and 4.2.1.

#**800** records were inserted in each of the databases.

#Following search queries were performed on the inserted data.

- Search for records with n keywords. (Since simple keyword index is required for optimum performance, PostgreSQL has an advantage on this)
- Find all keywords for a certain record. (Since complete entry retrieval is required, MongoDB has an advantage on this)
- Find records with any n common keywords (Added this for Lucene implementation as complete database retrieval and in memory processing is required)

4. EXPERIMENTS

In this section we first test the Application with both the databases, PostgreSQL and mongoDB. We then apply indexing using Lucene on each. We look into the data model for each case, the time it took to insert and retrieve the data and the ease of implementation.

We will also compare the execution time for each of these queries in MongoDB and PostgreSQL. We should note that each create query which is presented is executed 800 times and an average is taken to get an accurate execution time. And each read and search query presented is executed 100 and 500 times and respective execution time was recorded.

4.1 PostgreSQL

4.1.1 Data Design Modelling

In this section we design information architectures using entity-relationship models. For the POstgreSQL model, I decided on four entities, namely User Info, Entry, Keywords and Url Info. The User Info stores the name (text), email (text), password (text) and the keyword cloud associated with each user (image). The entry table stores each entry that the type of entry, text image or url, the timestamp of when the entry was saved and the email of the user. The email is the foreign key here. Since this application is currently implemented for saving websites only, we have a url info table which stores each url associated with the entry and the type of the url, image or text. The type of url has been added as the keyword extraction method for both image and text url are different. The fourth table is the keywords table. It stores the keywords associated with each entry and the relevance of each keyword.

The mapping between the tables is depicted through an ER Diagram in figure 1.

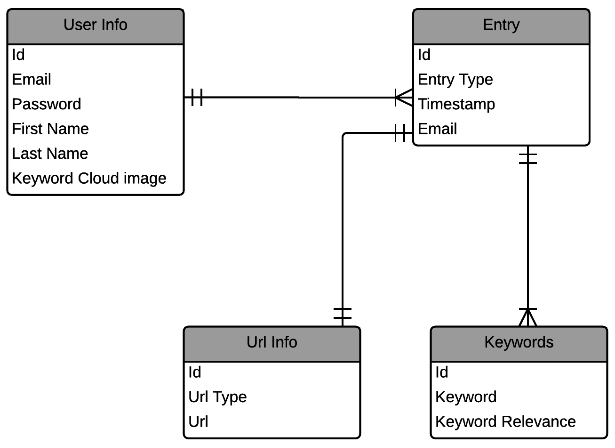

Figure 1: PostgreSQL ER Diagram

4.1.2 Data Design Implementation

In this section we implement persistent models using the PostgreSQL relational database management system. Since the application is built in Play framework, I made use of the Ebean objectrelational mapping and JPA annotations. Ebean uses the same mapping as JPA with its @Entity, @Table, @OneToMany etc. annotation. The mapping of Entity beans should be compatible between Ebean and JPA. The querying is doe using the find() method of the Ebean ORM.

4.1.3 Queries

In this section, we will discuss various queries which were written in the development of this application with PostgreSQL. The queries can be mainly categorized as CRUD (Create, Read, Update and Delete) queries.

- **Data Insertion** Since I used Ebean ORM, the data was inserted through JAVA. The entity setter methods add the various objects to the entity and then by calling the entity.save() method, the data is saved to the database. The entity.update() method is used to update the keyword cloud each time the user adds a new entry.

  The code snippets for the same are attached in the appendix section.

- **Data Retrieval** The data retrieval was done using the entity.find() method of the Ebean ORM in Java. The retrieval is done once the user searches for the website entries using the keywords from the keyword cloud. The results are compared in table 3.

  As described in section 3 , three types of queries were used. The code snippet is attached in appendix.

4.2 MongoDB

4.2.1 Data Design Modelling

Data in MongoDB has a flexible schema. Unlike SQL databases, where you must determine and declare a tableâĂŹs schema before inserting data, MongoDBâĂŹs collections do not enforce document structure. The key decision in designing data models for MongoDB applications revolves around the structure of documents and how the application represents relationships between data [6]: through **references** or **embedded documents**.

References store the relationships between data by including links or references from one document to another. Embedded documents capture relationships between data by storing related data in a single document structure. MongoDB documents make it possible to embed document structures in a field or array within a document. These denormalized data models allow applications to retrieve and manipulate related data in a single database operation [6].

For my application, the Entry document *refers* User document and *embeds* Keyword and Url document. The mapping between the tables is depicted through an ER Diagram in figure 2.

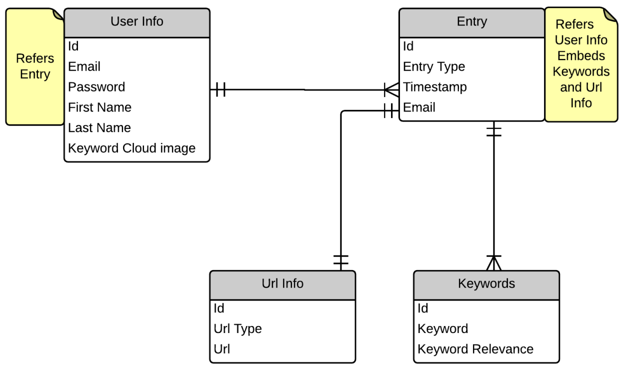

Figure 2: MongoDB ER Diagram

4.2.2 Data Design Implementation

In this section we implement persistent models using the MongoDB noSQL database. As discussed earlier, since the application is built in Play framework, I made use of the Ebean objectrelational mapping and JPA annotations. Ebean uses the same mapping as JPA with its @Entity, @Table, @OneToMany etc. annotation. The mapping of Entity beans should be compatible between Ebean and JPA. The querying is doe using the find() method of the Ebean ORM.

Each document was an Entity, so basically we have two entities, *User Info* and *Entry*. The Entry entity embed the keyword and url docements, therefore in java they are added as arraylists.

4.2.3 Queries

In this section, we will discuss various queries which were written in the development of this application with MongoDb noSQL database. The queries can be mainly categorized as CRUD (Create, Read, Update and Delete) queries.

- **Data Insertion** Since I used Ebean ORM, the data was inserted through JAVA. Each document is saved

Name	PostgreSQL	PostgreSQL with Lucene	MongoDb	MongoDB with Lucene
Insertion of 300 records	0.63	4.08	0.48	0.93
Insertion of 500 records	1.30	2.23	0.96	1.23
Retrieval of records with 20 keywords	1.23	0.96	1.36	1.28
Retrieval of all keywords for 1 record	1.32	0.87	0.93	0.83
Retrieval of records with 2 common keywords	1.56	0.79	1.93	0.68
Image update	1.35	1.381	2.492	2.52

Table 3: Experiment Result. Note: All results are in seconds.

manually using entity.save() method of Ebean. So in all 800 such entries were saved manually.

The code snippets for the same are attached in the appendix section.

- **Data Retrieval** Here again, the data retrieval was done using the entity.find() method of the Ebean ORM in Java. The retrieval is done once the user searches for the website entries using the keywords from the keyword cloud. The results are compared in table 3.

  The code snippet for the three types of queries discussed in section 3 is attached in appendix.

4.3 Indexing Databases with Lucene

In both of these cases, I used a Lucene index on my data. I stored the data as described in the above sections, but then build a corpus of Lucene documents with fields corresponding to the data I needed to find frequently the keywords. Anything that is put in a field is indexed and can be queried in ad-hoc ways. Then, I queried my Lucene index.

I did not face the biggest challenge of deciding how to represent data in Lucene as I had already modeled my data for mongoDB. So renormalizing my tables into a flat Lucene. Document was not a big hurdle.

The code snippet for Lucene indexing is attached in the appendix and the results are compared in table 3.

5. ANALYSIS OF EXPERIMENT RESULT

Table 3 presents the results for various reads and writes on both the databases with and without indexing. Since the dataset was not massively large, the results do not vary too much.

Insertion is faster in mongoDB as it stores the whole record as a document. Lucene does not fare well when inserting data. This is because inserting nodes in Lucene is costly.

Looking at various retrieval queries, we see that Postgresql performs better than MongoDb when a fixed number of keywords are given. This is because PostgreSQL uses foreign key to find and retrieve the records.

MongoDB fairs better when all keywords for a particular record are to be fetched. This is because mongoDB embeds the Keywords in the Entry entity.

We see that Lucene does well in all kinds of retrieval but fairs particularly well when records with common keywords are to be fetched. This is because we added indexing on keywords and Lucene creates an inverted index which makes retrieving data quite fast.

As we can see that updating a row in PostgreSQL is faster than updating a row in MongoDB database. Since PostgreSQL update query uses primary key to find the row to be updated, it is faster than MongoDB update query.

Other factors that I think should be taken into account are:

- Ease of Implementation: Both PostgreSQL and MongoDB have amazing documentation. Since PostgreSQL has been around for a longer time, the support is quite good. MongoDB's documentation is quite elaborative but implementing it with Lucene was a bit difficult as not has been written about it.

- Cost: Both the databases are open source.

6. FUTURE WORK

Due to time constraint I did not look into Graph database (e.g. Neo4J), so this can be done. For a better evaluation the database can be indexed and then compared with Lucene indexing. I decided to omit this as all the literature suggests Lucene is the best.

7. CONCLUSION

After comparing all the databases, I concluded that MongoDB with Lucene is the best option for my Web Application. MongoDb provides ease of insertion and Lucene makes up for its delay in keyword based search. Since the data in my application will grow linearly with time and increase of users, and there is no template for standard data entry, i.e. schemaless, noSql database is best suited for it.

8. REFERENCES

[1] K. Chodorow and M. Dirolf. *MongoDB: The Definitive Guide Powerful and Scalable Data Storage.* O'Reilly Media, September 2010.

[2] P. M. David Hows, Eelco Plugge and T. H. (Author). *The Definitive Guide to MongoDB: A complete guide to dealing with Big Data using MongoDB.*

[3] T. A. S. Foundation. Apache lucene - index file formats, 2013.

[4] P. Framework. The main concepts.

[5] M. Heydt. *Instant Lucene.NET.* Packt, July 2013.

[6] M. Inc. Data modeling introduction., 2015.

[7] M. Inc. Introduction to mongodb., 2015.

[8] M. Inc. Mongodb, 2015.

[9] M. Inc. Mongodb glossary, 2015.

[10] RobertMuir. How to make indexing faster.

[11] Wikipedia. Doug cutting, 2015.

[12] Wikipedia. Relational database, 2015.

APPENDIX

A. CODE SNIPPETS

The Code snippets for various implementations are attached below.

A.1 Insertion

- Postgresql
  ```
  entry.save(); urlInfo.setUrlEntryId(entry.getEntryId());
  urlInfo.save();
  ```
- MongoDB
  ```
  repository.save(new Entry(“url“, “anupriaa@gmail.com“,“20150505_180801“,
  “http://www.asecolab.org/“,“text“));
  ```

A.2 Retrieval

- Postgresql
  ```
  List<UrlInfo> urlList = UrlInfo.find().select(“url“).where().in(“urlEntryId“,
  finalIdList).findList();
  ```
- MongoDB
  ```
  for (Entry entry : repository.findAll())
  System.out.println(entry);
  ```