NoSQL in TigreNoble

Thomas Ansill
Kemoy Campbell
Kabo Cheung

May 9, 2015

Chapter 1

Introduction

1.1 What is NoSQL?

NoSQL is a type of data storage mechanism. It is in contrast to Structured Query Language (SQL). The data in SQL is structured in tables with columns and indexes. NoSQL has many different kinds of implementations. Some use document-based structure to store their data, others use key-value based. There are several other different kinds of structure that NoSQL uses that is not previously mentioned. The advantages of using NoSQL over SQL is the flexibility of using data and speed of lookups and updates.

1.2 The Problem

My director for the TigreNoble project has recently been hearing rumors of NoSQL databases. He has heard many good things about them, but doesnt really know anything about them. He has asked me to convert our existing database to a MongoDB instance (which is a NoSQL database), but I have serious reservations (I dont like change, especially this late in the project). The goal of this paper is to outline the advantages and disadvantages of changing the project database from MySQL to MongoDB NoSQL database. Then the paper will explain if the switch will be beneficial to the project.

Chapter 2

MySQL vs MongoDB

2.1 Setup of the System

MySQL is fairly easy to set up. It is a popular choice for many WAMP, LAMP, MAMP, XAMPP stacks so the chances that you don't need to install MySQL yourself since it's preinstalled on the stack. If it's not installed yet, you can just install the MySQL package and follow instructions to set up your username and password, then you start the daemon and it's up and ready. Our project uses PHP scripting language to bring functionality to our website. The PHP already has built-in MySQL support using Mysqli or PDO commands. You don't have to do any further work with MySQL. It is easy to initialize our database with initial data with PHP by just uploading .sql file that we have already written into MySQL and the data will be ready for use.

MongoDB is fairly easy to set up too. Unlike MySQL, it's not preinstalled in most "AMP" stacks, but you can install the package yourself and follow instructions to set up your username and password and start the daemon. As of PHP 5.5, MongoDB is already supported by PHP. You don't have to install MongoDB driver. There's a minor gripe with using MongoDB is that we will need to convert our initial data from .sql file to MongoDB file. Once we have done that, the database initialization is easy with MongoDB using PHP. The real hassle of setting up MongoDB for this project is that we might have to rewrite most of our PHP scripts because we have already written most of our code to work with MySQL. MongoDB doesn't have stored procedures so that means we need to reimplement all stored procedures that we wrote for MySQL in PHP.

2.2 Tools

We control MySQL database engine in our project through command line interface or PHPMyAdmin. PHPMyAdmin is helpful to the developers to visually see their data in the table.

There is phpMoAdmin, the MongoDB version of PHPMyAdmin. We could use that to control MongoDB. The interface looks worse than PHP-MyAdmin. It is also more difficult to use. There is no other better web-based interface. There's software-based interface, but that is not what we need. Also, we can use command line interface, but some developers on the team don't like command line interface.

2.3 Libraries

We use PHP's PDO library for MySQL and we also have created our own library that uses PDO. PDO is an easy-to-use library that prepares the SQL query and cleans the users' input to prevent XSS injection.

There's a library for MongoDB installed in PHP 5.5. There's no input santization, so we may risk XSS injection unless we manually sanitize our input. There was already a PHP function that will sanitize the input. We just have to write it in for every MongoDB query. We will need to rewrite our library to work with MongoDB since we used PDO exclusively in our library.

2.4 Data Storage

Our data fits very well in MySQL's tabular relations. The data format is normalized to ensure that there is no unnecessary duplicate information within the database. Each SQL table is its own domain. No other data that doesn't belong can be found in any tables.

MongoDB's database engine benefits from data denormalization. This means we have to change our data structure for MongoDB to function better. Our new data structure will be using Students objects. Then there will be a book posting with book information together and that information will be a subset of a student object. The data structure is not optimal because there will be duplicate data of Book. There may be more than one book posting that sells the same book ISBN. The data of book description will appear in MongoDB more than once. Once our userbase and number of postings grow, our database space will get used up more quickly than our MySQL configuration. Also, we require that book posts to record the bids including the bidder. We will have to include the data of the name, address, email of that bidder under the post. This will duplicate the data of the Student. In our current MySQL implementation, our Ratings, Notification, Inbox data structure are related to Student which will result in more data duplication if we try to convert that structure to MongoDB.

The only way to prevent data duplication is to normalize the data. If we normalize the data using reference pointers and etc, the database's structure will eventually look like our MySQL configuration, then there's no real point of switching to MongoDB if the optimal project data structure will look like our MySQL configuration. MongoDB also have no concept of Join. Joins is useful on MySQL for denormalizing the book postings with book information upon retrieval. We also use Joins for bids and bidders' names and seller's name. It will be difficult to join data using MongoDB with normalized data.

2.5 Performance

If we continue to denormalize our data for MongoDB, data inserts will be faster on MongoDB compared to MySQL. For data updates, for the most time, it should be faster, but it depends on which data is being updated. If there are already a duplicated data sitting somewhere in the database like Student information and a duplication of Student information in Posts' bids, MongoDB has to update information in every duplicates in Bids too. It will perform worse than MySQL. If we decide not to update every duplicated data, the database will become corrupt. For retrieving the data, MongoDB will perform much faster than MySQL because the data is denormalized so there's no jumping back and forth via references. MongoDB will just go down the "trees" of data and find a node in the tree. MySQL will jump across tables through keys to find a specific data. Although, the performance in searching for the books in MongoDB might be worse because Book will be stored in individual Students's posts. MongoDB will need to search through all students then search students' posts for the book. MySQL can just find the proper book ISBN from the Book table without searching through Posts or Students.

If the data is normalized on MongoDB, performance differences between MongoDB and MySQL won't be significant because both will do exactly the same operations of inserting into groups of data. The data retrieval performance on MongoDB will be similar to MySQL since it will use references via keys so it have to jump between "trees" to find a data. Although, MySQL is more efficient in using key lookups. MySQL performs better for normalized data.

Chapter 3

Conclusion

3.1 MongoDB or MySQL?

MongoDB seemed like a great database for this project. It is quicker than MySQL in some cases. Unfortunately, we find that the data structure of the MongoDB is unsuitable for our project. Our project depends on multiple homogeneous relationships between objects. Using MongoDB will only result in unnecessarily duplicating the object while trying to retain the multiple relationships. This won't give us space and speed advantages from using MongoDB over MySQL. The space usage will be higher than MySQL while the speed is somewhat faster than MySQL. If we attempt to normalize the data in MySQL, the speed will be similar or worse than MySQL. MongoDB would work much better if many data is distinct but most of our data is not distinct like Books in each book postings might be the same.

In the addition, if we were to decide to switch to MongoDB, we will spend more of our time to reimplement our existing code to work with MongoDB and all we will get is insignificant performance gains with increased security risks since PHP's MongoDB library does not have automatic input santization.

In the conclusion, I do not think MongoDB is worth our time and sweat to reimplement our project so we can either suffer space consumption and benefit speed or have an insignificant performance and space difference to MySQL. For this project, we should continue using MySQL because it will serve us better in terms of space usage, convenience, redundancy, and consistent performance than MongoDB.