A Study of Migrating Biological Data from Relational Databases to NoSQL Databases

by

Nawal N. Moatassem

Submitted in Partial Fulfillment of the Requirements for the Degree of

Master of Science

in the

Computer Information Systems

Program

YOUNGSTOWN STATE UNIVERSITY

August, 2015

A Study of Migrating Biological Data from Relational Databases to NoSQL Databases

Nawal N. Moatassem

I hereby release this thesis to the public. I understand that this thesis will be made available from the OhioLINK ETD Center and the Maag Library Circulation Desk for public access. I also authorize the University or other individuals to make copies of this thesis as needed for scholarly research.

Signature:

Nawal N. Moatassem, Student Date

Approvals:

Dr. Feng Yu, Thesis Advisor Date

Dr. John R. Sullins, Committee Member Date

Dr. Yong Zhang, Committee Member Date

Dr. Salvatore A. Sanders, Associate Dean of Graduate Studies Date

Acknowledgments

First off, I would like to thank my family and friends for supporting me through this graduate degree program. I would like to thank my fiance, Eric, who has always stood by my side and encouraged me when times got tough. Lastly, I would like to thank all of my professors who have taught me so much, especially my thesis advisor, Dr. Feng Yu who has believed in me and prepared me for my future success.

Abstract

The purpose of this research is to conduct a literature survey on various NoSQL Not-only-SQL) architectures. Included along with this literature survey is an experiment to do a comparison between a relational database management system (RDBMS) and a NoSQL DBMS. This work compares specifically MySQL and MongoDB, an RDBMS and NoSQL DBMS respectively, for the purposes of data migration. The migration is run on data sets for Youngstown State University's plantEST biological database. The idea is to demonstrate the need for shifting to NoSQL for management of large amounts of unstructured and semi-structured data, and to observe and record the insertion speeds of both databases using this custom plantEST schema.

Contents

Contents **v**

List of Figures **vii**

List of Tables **vii**

1 Introduction **1**

1.1 Database Management Systems 1

1.2 What is a Database? 3

1.2.1 ACID 4

2 Relational Databases **8**

2.1 What is Relational Database? 8

3 NoSQL **11**

3.1 What is NoSQL? 11

4 NoSQL Database Types **12**

4.1 Key-Value Store Databases 12

4.2 Document Databases 13

4.3 Column Store Databases 13

4.4 Graph Databases 14

5 Relational VS. NoSQL 15

5.1 Scaling 16

5.2 Data Structure 17

5.2.1 Structured Data 17

5.2.2 Semistructured Data 18

5.3 A Comparison Between Several NoSQL Databases with Comments and Notes 19

6 Benefits of NoSQL 21

7 Benefits of Relational databases 23

8 Big Data and NoSQL 25

8.0.1 Hadoop and NoSQL: Technologies and the Oracle Database 25

9 Experiment 27

9.1 MongoDB 27

9.2 MySQL 28

10 Experiment 29

10.1 Migration 29

10.2 Data-Retrieval 32

11 Conclusion and Future work **34**

References **35**

List of Figures

1 A Common NoSQL Task-Sharing Method Using Neo4j (courtesy of [3]) . 22

2 Migration speed of MongoDB compared to MySQL 31

3 Data retrieval speeds of MongoDB compared to MySQL 32

List of Tables

1 Relational Model 9

2 Structured Data 17

3 Adding a Table 17

4 `plantEST` Database 30

1 Introduction

In order to maintain a successful organization or business, a database is usually required. [17] states, "A database is a collection of data, typically describing the activities of one or more related organizations". This helps keep track of all transactions happening within a company or organization in order for it to run smoothly. There are several different types of databases which will be covered later on. Throughout this research we will be focusing on comparing and analyzing Relational Databases and NoSQL databases, which are currently among the most commonly used. Specifically this research aims at studying the differences between NoSQL databases and comparing one NoSQL architecture, namely MongoDB, against the performance of an RDBMS; in this case that RDBMS is MySQL. A difference in performance uses and architecture advantages and disadvantages will be discussed at length, as will the design purposes of various NoSQL implementations. The migration speeds and read speeds of the MongoDB and MySQL architectures will be discussed at length in the experimental section.

1.1 Database Management Systems

The definition of a database management system was best stated within [17], "A database management system, or DBMS, is software designed to assist in maintaining and utilizing large collections of data, and the need for such systems, as well as their use, is growing rapidly." Database management systems will continue to grow due to their being more and more data now accessible through computer networks. For example, one of the most common and simplest forms of databases are digital libraries. With database management systems, it is very easy for libraries to keep track of all items by using a database to show what items are checked out and how many are available.

Database management systems have been around since the early 1960s. Charles Bachman designed the very first general-purpose database management system at General Electric where is was originally known as the Integrated Data Store. [18] states, “In the 1980s, the relational model consolidated its position as the dominant DBMS paradigm, and database systems continued to gain widespread use. The SQL query language for relational databases, developed as part of IBM’s System R project, is now the standard query language.” In the late 1980s and the 1990s, advances have been made in numerous areas of database systems. Large vendors began extending their database systems at this time so they could store new data types into their systems such as written text and images, and also gain access to more complex queries.

Before database management systems, websites used their operating systems files in order to store their data which was being accessed through a web browser. Now that many DBMSs have been moved to the Internet, it is becoming widespread that websites use DBMSs for their storage. DBMSs play a very important role and continue to grow as they become more and more important in our every day lives.

With they way technology is today, it is almost absolutely necessary that companies operate and maintain a database on a daily basis. Without a DBMS, a company would suffer and not be able to compete with other companies. [18] shows the importance of the

DBMS by saying “Today the field is being driven by exciting visions such as multimedia databases, interactive video, digital libraries, a host of scientific projects such as the human genome mapping effort and NASA’s Earth Observation System project, and the desire of companies to consolidate their decision-making processes and mine their data repositories for useful information about their businesses. Commercially, database management systems represent one of the largest and most vigorous market segments”. This kind of crucial need for database management gave rise to the importance of the company’s Database Administrator (DBA). The DBA helps to company to develop an appropriate schema to keep the RDBMS consistent and up-to-date. The DBA will manage the database and keep it secure against attacks, as well as fine-tune the database when needed.

1.2 What is a Database?

A database is a kind of system that is designed for storing data. Typically, this is a system made available on a network which handles many incoming requests, even overlapping requests. Most often, a database will be given a defined schema, which is a structure of the shape of the data. Usually the database will have mechanisms for database structure preserving. This way, the data cannot be inserted, modified, or removed in a way that will violate the schema.

Most databases handle requests through transactions. Transactions are an independent unit of work enacted upon the data. This occurs when clients are contacting the database and conducting transactions. Many clients can contact the database at the same time and create simultaneous transactions. The database makes sure those separate transactions do not interfere with each other. Ideally, transactions conform to whats prescribed by the acronym, ACID.

1.2.1 ACID

ACID stands for atomicity, consistency, isolation, and durability. The ACID formulation of transactions has been current for thirty years. [16] These four properties of transactions show why it is desirable to use databases.

Atomicity means for a transaction to be atomic. The entire transaction is performed all or nothing. Either all the modifications made in a transaction are committed to the database, otherwise all the modifications are rejected. If all of the modifications are rejected it will appear as though the transaction never happened. A transaction may be aborted due to several possibilities. In some cases another transaction could be interfering with transaction and one may have to be aborted. Other reasons could be power failure or some sort of database crash. In all of these circumstances, the important thing is that whatever work or modifications being made should be all or nothing. So either all of the modifications should be made, or none of them should be made and should be re-dropped.

"Consistency is best understood as a contract between the programmer writing individual transactions and the system that implements them" states [16]. Consistency in ACID means that when a transaction completes it should leave the database in a consistent state. This is a state consistent with all of the rules and constraints imposed in the schema. If the schema states that all the data should conform to such and such structure and the values should follow certain rules, then every transaction should leave the database conforming to those rules. “If the programmer ensures the consistency of every individual transaction, and also ensures that the initial state is consistent, then the system will ensure that consistency applies globally and forever, despite concurrency and failure” [16].

Isolation in ACID refers to the property that the transactions should be totally independent of each other. Overlapping transactions should not interfere with each other. If one transaction updates the value of one piece of data, you want that new value to then be subsequently read by any transactions that read that same piece of data. The issue with concurrency is when you have multiple actors trying to all act on the exact same data at the same time. Without proper isolation of transactions this could lead to issues where you have to overlapping transactions and where one transaction is updating two pieces of data, the other transaction is trying to read those two pieces of data but because of the timing, the transaction reading the data gets one of the values updated but not the other one. This could interfere with how the code is written so you assume when you update multiple values anyone reading the database will see the updates as a whole rather than getting the mix of new updated values and old out of date values. Databases typically offer different levels of isolation. There can be total isolation, or the rules can be relaxed to allow for more overlapping transactions to speed up the process and get on with their work.

Durability of a transaction simply refers to the property where by once a transaction completes, or commits, all of the changes it’s made to the database get preserved. The data should persist. The point of durability is that if a system were to lose power or shut down for any reason, it should be able to be restored to the last good state without losing any data or progress.

The relational database also has a need to maintain its ACID rules to prevent problems from arising within the database. One possibility associated with improper transaction controls is the potential for the DBMS to conduct a dirty read. A dirty read, [10], is when one transaction runs an update/write command without committing. Later when a second transaction reads the value of an updated/written record, but the original transaction is rolled back instead of committed, then the second transaction was working with dirty data. This type of operation can corrupt the consistency of the database.

Another problem is the lost update anomaly which can cause significant real life errors. In the lost update scenario, one update reads the value of a record it will use in a separate process, but before the finish can occur and a commit command can be sent to the DBMS, a second transaction reads the original value of the record for a separate process. A real-world example would be a banking transaction, where one account is updated twice without the first committing. If a client with an account of \$500 both transferred \$200 from a separate account, and then spent \$20 the result wouldn't be the correct \$680. In this case the first update would create a temporary value of \$700, but without committing first, the second transaction reads the original \$500 and applies its changes for a temporary value of \$480. The first commit is overwritten by the \$480 of the second transaction. The transfer of the \$200 might as well have not even happened. In this case the bank will have to refund that amount with this client, and if it happened in one case, there is a likely chance other clients have the same problem. There will be wasted time for the bank trying to get its account settled and correct. This is a good example of why, for many businesses and institutions, that ACID rules are so crucial.

2 Relational Databases

E.F.Codd, an employee at IBM, invented the term relational database in 1970 and he first used the term in a paper he wrote called “A Relational Model of Data for Large Shared Data Banks”. In the late 60s and early 70s, Universities in Michigan and Massachusetts were some of the first to have relatively faithful implementations of the relational model. “In 1974, IBM began developing System R, a research project to develop a prototype RDBMS. Its first commercial product was SQL/DS, released in 1981. However, the first commercially available RDBMS was Oracle, released in 1979 by Relational Software, now Oracle Corporation.[6] Other examples of an RDBMS include DB2, SAP Sybase ASE, and Informix.” [7] Since the 1980s, Relational database management systems have been a popular choice for storing information in databases used for personnel data, financial records and other applications.

2.1 What is Relational Database?

“A relational database management system (RDBMS) is a database management system (DBMS) that is based on the relational model as invented by E. F. Codd, of IBM’s San Jose Research Laboratory” states [7]. This type of database “Unlike the other data stores, relational DBMSs have a complete pre-defined schema, a SQL interface, and ACID transactions” [9].

Relational databases are the most dominant. This means that most of database management systems used today are based on the relational data model. The relational model in terms of describing the data is referred to as a schema. In the relational model, the schema for a relation specifies its name, the name of each field, which is the attribute or column, and the type of each field. [17]

For example, a book within a library database would be stored as the following:

Books(BookNum: integer, Author: string, Title: string, Publisher: string)

This schema indicates that there are four fields that include the names and types. This schema would appear in Table 1.

Table 1: Relational Model

BookNum	**Author**	**Title**	**Publisher**
5023	E.L. James	Fifty Shades of Grey	Vintage
4018	Sheri Anderson	Secret in Salem	Days of our Lives Publications
789	Edgar Allan Poe	The Works of Edgar Allan Poe	Flying Fish

The term RDBMS refers to the software itself. The reason it is called a management system and not just a database system is because you can have multiple separate databases all under control of one database system.

The standard language used by most databases is known as SQL (pronounced "sequel" by many) which stands for, Structured Query Language which clients use when they talk to the server. The term query refers to a request for data, like when a client sends a request to retrieve data from the database. SQL is not restricted to just these queries. A client may also make requests to insert or modify data in the database. Not all databases use the same exact SQL language, they all have their own variations, which could make it difficult to transfer data from one type of database to another.

There are dozens of different relational database management systems. Oracle DB [4], Microsoft SQL Server [1], MySQL [2], PostgreSQL [5], and SQLite [6] are among the most popular out there. In this experiment we will be focusing mainly on MySQL, which will be discussed more later on.

3 NoSQL

There are different kinds of database management systems. Here we will mainly be comparing and contrasting NoSQL databases, along with RDBMS databases. Carlo Strozzi used the term NoSQL in 1998 to name his lightweight, open-source relational database that did not expose the standard SQL interface. Strozzi suggests that, as the current NoSQL movement "departs from the relational model altogether; it should therefore have been called more appropriately 'NoREL', referring to 'No Relational'." (wiki) Later on in early 2009, Johan Oskarsson organized an event to discuss open-source distributed databases where the term NoSQL was also reintroduced by a man named Eric Evans.

3.1 What is NoSQL?

NoSQL is a database management system and is also defined as 'Not Only SQL'. The basic motivation of NoSQL was to make it easy to build and deploy applications. It makes it easy to scale and operate these systems by having a distributed design. It can be more cost effective for some institutions not requiring strict ACID rules in place for transactional controls. This type of database handles unpredictable and unstructured data and is a more cloud-friendly approach, because a traditional RDBMS model usually requires strict normalization. In the mid-1980s started the rise of the relational database which brought many benefits, and focused on upholding ACID rules.

4 NoSQL Database Types

All these different types of NoSQL databases serve different purposes. We have the ability to store the data in a combination of these NoSQL databases as well as relational databases. Indeed, these different NoSQL databases can be used in combination in certain companies to offer versatile functionality and greater parallelism for real-time applications with less required overhead.

4.1 Key-Value Store Databases

One type of NoSQL database is the key-value store databases. “Key-value stores are the simplest NoSQL data stores to use from an API perspective. The client can either get the value for the key, put a value for a key, or delete a key from the data store. The value is a blob that the data store just stores, without caring or knowing what’s inside; it’s the responsibility of the application to understand what was stored. Since key-value stores always use primary-key access, they generally have great performance and can be easily scale”. These are similar to document oriented databases but do not allow you to do a query inside the document without actually having the key first. For example, the user cannot have a query to find a certain object such as all birthdays in July without having the Key ID. This type of database is useful because it has very fast access to data. These are used for storing customer browser history and serve a better user experience. The most common is known as Redis and is used for popular sites such as twitter to quickly present the information on a user’s twitter stream.

4.2 Document Databases

Another type of NoSQL database is a document databases, which is what we will be focusing on mostly in the experiment section, is the most common. With a document oriented database it enables the user to make a query within the document to find certain data. The language used is a lot more object oriented which is very familiar to programmers. For example, a query statement in SQL would look something similar to:

```
SELECT* FROM Users WHERE FavColor = ‘Green’;
```

However, when using a document oriented database such as MongoDB, that same statement would look more like:

```
db.users.find(FavColor:  "Green")
```

This type of database is great for analysis and creates a user friendly language that is familiar for programmers.

4.3 Column Store Databases

Another type of NoSQL database are column store databases. The most commonly used column store database is Cassandra which is an open source distributed database management system designed to handle large amounts of data across many commodity servers, providing high availability with no single point of failure. [12] This type of database enables the user to effectively add additional information for each record due to them having rows across the top instead of having fields. The field names are down the side of the column database. Although it allows you to add more information, the downside is that there is a limit to how big each record can get.

4.4 Graph Databases

The last type of database is graph databases. The biggest and most commonly used type of Graph databases is known as Neo4J. This type is commonly used in social media. For example, when a site such as Facebook is recommending a user to be friends with another user because they have a number of friends in common or recommending them to ‘like’ a page because their current friends like the same page is due to the graph database linking these connections together to make suggestions.

5 Relational VS. NoSQL

Albeit the relational model and the NoSQL model are different in their data-structure and how they scale, the purposes of these databases fit different business needs. In [14] the authors argue that if a business needs strong transactional controls that guarantee data is up-to-date then it is a good argument for relational databases that maintain strict ACID rules. If the business requires large-amounts of data that don't necessarily need to be up-to-date, and not all nodes in the distributed system need to always maintain ACID rules than NoSQL is the better choice. If a business wants the best of both worlds, with ACID maintained where needed, and more efficiency in areas where this isn't required and an eventually-consistent data-set is acceptable, then both can be used.

Achieving 'polygot persistence', where applications are designed with many languages and databases in mind, can be a worthwhile task. The example used by the authors is an E-Commerce site, that uses and RDBMS such as Oracle to maintain the sites inventory and pricing, while NoSQL architectures are used for a variety of other tasks: key-value stores for shopping-cart and session data, document-stores for completed purchases, and graph-stores to hold such things as customer reviews. One major corporation taking the polygot approach is Amazon with their Dynamo NoSQL architecture. Various functions of their website are controlled by different applications within Dynamo that allow for the most efficiency and the least overhead for various operations. Spreading out this workload through many different types of databases helps control the very-real chance that a single architecture would be bogged-down by requests or cause inconsistencies that might cost money and ruin customer experiences.

5.1 Scaling

Relational databases are designed to scale up while NoSQL databases are designed to scale out. This means that the scaling ability is limited for relational databases to 10 terabytes or 100 terabytes. NoSQL databases can reach up to 1000 terabytes by just adding more boxes parallel to each other.

The vertical scaling of relational databases involves adding more hardware power. This often means buying expensive, high-powered, proprietary servers. These servers can handle a lot of traffic and a heavy workload but begin to get bogged down when dealing with potentially millions of simultaneous users, such as with big web names like Facebook or Amazon. To spread out this workload, would require a large number of servers that can work concurrently while maintaining consistent data storage, which isn't possible on a massive scale. NoSQL makes use of horizontal scaling, where workloads and data storage can be maintained upon a large number of standard machines or even virtual machines. This horizontal scaling allows the workload to be spread over a potentially large number of nodes, making web services like Facebook capable of handling potentially a hundred-million or more concurrent users.

5.2 Data Structure

Another key difference is the data structure. In relational databases, the type of data that is stored in the database is predefined when designing a program. The nature of the data must be understood.

5.2.1 Structured Data

Relational databases deal with structured data. For example, a basic user table may contain a User ID, a Username, User age and favorite color. This table will appear similar to the following:

Table 2: Structured Data

ID	Name	Age	FavColor
3	Jake	23	Purple

Now let's say that the same user had a second favorite color, and both needed to be stored. Originally, the database was only designed to cope with one favorite color to be entered into the system. In order to fix this problem, a new table would need to be created for the favorite colors separately.

Table 3: Adding a Table

ID	FavColor
3	Purple
3	Orange

By having two different tables, it physically requires moving the data to another place on the disk and having a pointer that points from the user table to the new table created. This issue slows down the server runtime and is expensive. Due to the data being transferred, the program would also have to change to accommodate and may require the data system to go offline while the transfer takes place.

5.2.2 Semistructured Data

In a NoSQL database this implementation differs a lot from relational databases and contains semistructured data. By using a NoSQL database, this same data can be stored in a more simpler way. For example, the following shows the same data stored in a document type database, such as MongoDB.

```
USERS ID:3, Name; ‘Jake’, Age:  ‘23’, FavColor:  ‘Purple’
```

Now, to make the same change to this database as we did the table, it is very easy. All that needs to be done is to add the extra data into the same document by just adding square brackets along with the additional information, as shown below. This makes it fairly simple because no additional document or table needs to be created.

```
USERS ID:3, Name; ‘Jake’, Age:  ‘23’, FavColor:  [‘Purple’, ‘Orange’]
```

This database type allows you to add information or make changes to the data for a single record without affecting the structure of any other records within the database. There is no structure in MongoDB. The structure is defined per record which allows you to add objects within objects without limitation.

5.3 A Comparison Between Several NoSQL Databases with Comments and Notes

According to [20] there are over 120 implementations of NoSQL solutions in existence, and focusing on open-source products. Typically these implementations do not need to follow any set schema and therefore often attempt to avoid join queries. While many of these NoSQL platforms seek to depart from the traditional RDBMS system, many others such as Google's HBase and BigTable seek to keep more traditional ACID qualities. Other vendors have taken to mimicking certain desirable NoSQL features such as Microsoft offering snapshot isolation methods and Oracle 11g coming with single instances of NoSQL. In some cases these open-source implementations come with sophisticated means to detect failures such as Apache Hadoop's library software that allows for computing queries across multiple clusters.

When tested against the open-source RDBMS for MySQL, there were significant findings. MySQL was compared against two NoSQL implementations specifically, namely HBase and Cassandra. In dealing with throughput, MySQL suffered when reading/writing over 7000 operations per second while the NoSQL implementations showed little change; HBase being better at writes, while Cassandra showed to be more read-capable. Despite this MySQL and similar implementations retain strong read-oriented strength, but this is only true for smaller data sets. When the size of the MySQL database grows too large there can be performance penalties that will make the NoSQL implementations more favorable. Cassandra and HBase retain better write-optimizations, even when compared at a smaller dataset size.

[11] gives an explanation of Eric Brewer's principle that web services, like DBMSs, attempt but cannot achieve consistency, availability, and partition-tolerance simultaneously. An RDBM, though always offering consistency and availability, rarely has extensive abilities in partition-tolerance. The idea of vertical-scaling, with stacking more hardware onto a central, often proprietary server, helps to respond to large numbers of users while keeping the data consistent and usually in a single up-to-date format. Some partition-tolerance is available but usually only through increased complexity. Most NoSQL databases offer availability and partition-tolerance, but lack certain abilities to remain constantly consistent. Environments that perform database sharding, where scalability is horizontal and therefore information kept at a local level, can guarantee easy expansion of the database network and easy availability, but cannot always guarantee a truly up-to-date result for each query. The only way to make all three a certainty might be a mixed approach, where some functionality is provided by an RDBMS while NoSQL provides other functionality to an application, but it cannot guarantee all three for each subset of functionality.

6 Benefits of NoSQL

Unlike relational databases, NoSQL contains the lack of explicit data scheme. In a relational database the data is structured whereas NoSQL databases could have structured data or unstructured data. The major problem was that after relational databases were created they ran into the problem where they could not handle big data. NoSQL was the solution to this problem. NoSQL is focused to provide scalability, performance, and high availability. With scalability NoSQL can handle large amounts of data. Although there will be less functionality compared to Relational database, there will be better performance. Flexibility is a major benefit which relational databases do not have. Relational databases require the user to go back and design the database in order to cope with changes that could easily be made in NoSQL.

One major advantage of NoSQL architectures is that there are a large number of open-source or easily available platforms. One of the most famous of these, which was put into use for Facebook, is called Casandra. It is a column-store NoSQL database, which means that it is particularly well suited for read-optimization. This doesn't mean that it needs to sacrifice write-speeds, however, as [8] showed that its write-speeds exceeded even MongoDB's write-speeds; additionally their read speeds were very similar and efficient. Having been used for Facebook's database needs, Casandra shows a high aptitude for being able to handle millions of concurrent users. There is no single point of failure for Casandra and reads/writes can be done on any node; a popular feature of several NoSQL architectures. Like other NoSQL services, Casandra's nodes automatically make use of their resources and each node contains its own data set that the node is responsible for and in charge of. This load balancing can be seen in more detail in Figure 1, showing an illustration of Neo4j's similar technique to spread workloads across multiple nodes. These features that make Cassandra a powerful database architecture can be seen commonly in different but similar forms in other NoSQL architectures, making Cassandra an excellent and famous example of a non-relational database.

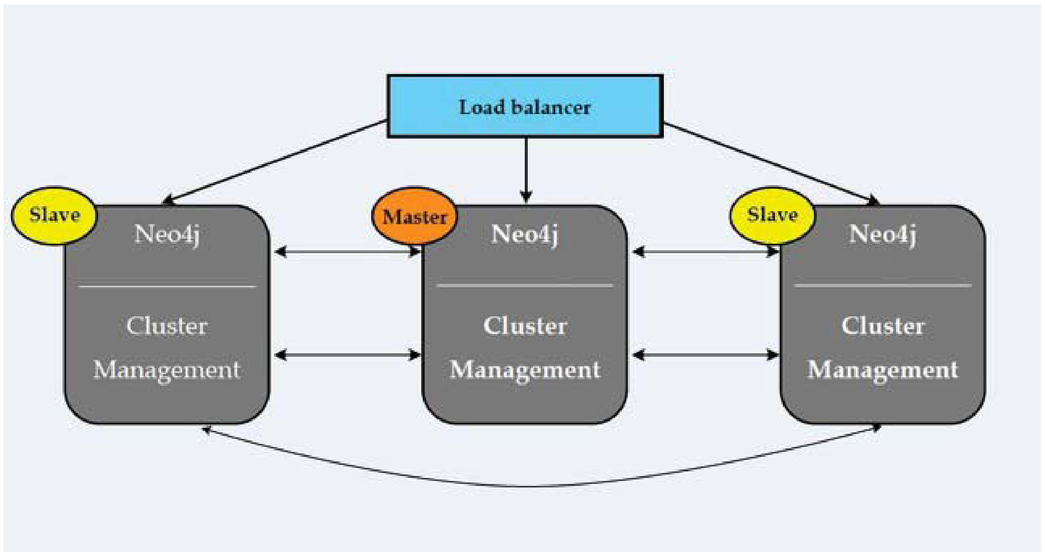

Figure 1: A Common NoSQL Task-Sharing Method Using Neo4j (courtesy of [3])

7 Benefits of Relational databases

One of the main benefits of an RDBMS is the consistency of the data and the guarantee that data is up-to-date at all times. While NoSQL applications offer an eventually-consistent model for its data, relational models guarantee through strict adherence of ACID principles that data viewed is always the correct version. One of the main advantages that users cherish in the relational database models are the security principles. NoSQL databases, such as MongoDB and Casandra, as displayed in [15], have certain cracks in their security that make their use risky by companies and institutions with sensitive or private data. Information often isn't encrypted in NoSQL architectures; information is even sometimes sent over the network as plain text. Any malicious user can make use of packet sniffers to find out crucial information, such as account passwords or social security numbers that are transferred between nodes in this way. They are also often acceptable to SQL injection and denial of service attacks that might well be caught in an RDBMS. Therefore, many institutions trying to protect crucial data may opt for an RDBMS over a NoSQL implementation.

Additional benefits to an RDBMS platform is its use of SQL, which isn't the main language used in many NoSQL applications. This gives organizations that use an RDBMS access to many professionals that have worked with SQL; in some cases for more than a decade. Another benefit, as displayed in [13], comes with the development of 'NewSQL,' which attempts to mimic many of the best features of a NoSQL implementation. The idea here is to maintain the valued consistency of an RDBMS, while offering the users more partition-tolerance through mimicking NoSQL. Some of the features copied through

NoSQL and other implementations of RDBMS include: column data storage, in-memory processing, symmetric multiprocessing, and massively parallel processing.

8 Big Data and NoSQL

Hadoop enables users to process large amount of data and break it down to send it across the network to multiple computers. This consists of a process known as MapReduce. The map process maps the workload from the data stores to the computers and then the Reduce process reduces the workload from the computers to a final result. Hadoop breaks up the data and divides the data among many computers so the result can be searched more quickly.

8.0.1 Hadoop and NoSQL: Technologies and the Oracle Database

In [19], it focuses on Hadoop with MapReduce as a viable option for big-data workloads. This worked focused a lot of effort on explaining the differences in scaling between a NoSQL platform and a traditional RDBMS. In a traditional setting the focus was on vertical-scaling. In this case the idea was to add the most powerful hardware possible at a server level to deal with query processing. It could be very expensive, however, as costs on more efficient and powerful hardware were often steep, limiting the capabilities of most institutions to what they could afford. At times it could even cause bottlenecking at the server with network traffic on a distributed DBMS as merging at server. This can make an RDBMS less viable for institutions lacking in funding, especially those needing to run big-data queries efficiently.

When it comes to NoSQL, the approach is to use horizontal scaling. In the case of horizontal scaling, the objective isn't to add more powerful hardware, but to spread out a workload using clusters of smaller machines. This is more possible due to the NoSQL capability of not needing a schema, allowing for database sharding which can see a single table spread around the hard drives of the various machines inside the clusters. One way Hadoop is made viable is by making use of MapReduce to keep track of which data is on which machine and to spread the workload accordingly. If a table has a million rows that are spread out across ten machines, MapReduce keeps track of which data is one which machine. When a query is being processed, the optimizer sends tasks to the various machines holding the necessary data. In this way, Hadoop is able to use MapReduce to aid in horizontal scaling, which in its nature is often more cost efficient for institutions lacking the funding necessary to attempt vertical scaling associated with most traditional RDBMS models. Some of the drawbacks to this method, however, lies in the data storage model associated with horizontal scaling. Given that data is kept locally on many machines, and that many NoSQL platforms lack schemas, it can be a heavy burden on system resources to perform join queries.

9 Experiment

Our experiment was to help benefit the Biology Department at Youngstown State University by making their database run more efficiently. The issue was that the department is currently using the relational database, MySQL, which will become very slow once the database reaches 2 Gigabytes as they continue to add data into their database. Our solution to this problem was to migrate all of their data from MySQL to a NoSQL database, and in this case we chose to use MongoDB. After all the tables and the data within the tables were migrated into MongoDB we ran tests comparing both input and data retrieval for both MySQL and MongoDB.

9.1 MongoDB

MongoDB is an open source NoSQL implementation and a document database. It is a schema-free database that keeps a JSON-like format called BSON, a binary format. Like Cassandra, MongoDB is highly scalable and has no singular point of failure. It is comprised of an arbiter, master node, and many slave nodes. MongoDB has many powerful features that makes it quite attractive, [8]. It can perform automatic sharding, a DBMS driven partitioning across various servers. Many shards hold data replicas, allowing access to data even if the main node queried for the information should fail. The ease of the MongoDB commands has also allowed it to gain popularity and become the most popular NoSQL implementation since 2011, [13]. There exists in MongoDB a simple syntax such that many queries written in SQL can be easily translated. This easy to learn syntax has aided both the popularity of MongoDB and its ease of use for professionals.

9.2 MySQL

The MySQL database got it's name because the original programmer named it after his daughter, 'My'. The programmer and his partner founded the company and released MySQL under an open source license. MySQL is still the most popular open source database today. Written in C/C++, MySQL was developed by Oracle developers and is one of the most well-known and widely used SQL products. With a large host of implementations and various graphic interfaces, it is available in many forms to meet a host of RDBMS needs.

10 Experiment

Here we seek to do an experiment that will compare performance differences between MongoDB and MySQL. The categories for comparison will be both migration speeds for the `plantEST` schema between the two architectures as well as general read-speeds. Firstly with the migration, the speeds are measured as the schema's data is passed through CSV files from one architecture to the other and then back. Then an experiment will be conducted to see how fast each DBMS can read data in the schema and return it to the user. It is expected that these experiments will show the strengths of each architecture and how MongoDB can lead to an improvement in performance over the current MySQL implementation for Youngstown State University's biological data. While implementations of MongoDB often have multiple nodes or clusters, this experiment was run using a single MongoDB node to compare direct performance to MySQL.

10.1 Migration

The migration portion of our experiment consisted of migrating a total of 13 tables from a database called `plantEST` from MySQL to MongoDB. Table 4 shows a list of tables included in the migration process and the size of each table.

The objective of this migration was to move Youngstown State University's `plantEST` biological database from MySQL to a NoSQL, MongoDB implementation. As stated above, NoSQL implementations are better equipped to handle large amounts of

Table 4: `plantEST` Database

Tables	Size in MB
EST_FASTA	4307.18
SignalP	3247.09
Orfout	1276.93
Phobius	973.00
Plant_Info	950.57
TMHMM	787.80
Rpsout	711.92
TargetP	584.75
Prosite	393.59
Annotator	278.39
GPI	24.27
Predict_Secreted_Old	20.07
Predict_Secreted	9.28

unstructured and semi-structured data, and `plantEST` had grown to over 13GB, with certain tables being over a full gigabyte. Since MySQL isn't made to handle such large amounts of data, it is assumed that that this non-proprietary, open-source RDBMS will begin to grow too costly to be able to manage `plantEST` without significant delays to response-time.

The migration process for using Java began by opening connection to the MongoDB and MySQL databases. Then one table is read at a time and each row read from the MySQL tables are converted and inserted into MongoDB's collections. This method worked, but the coding was very intricate and could take a long time to execute.

The preferred method involved the use of CSV files, where each new line represented a new row in the table. A program could read a row from the CSV files that contained the table rows and information and easily input into MongoDB. Separate commands are issued to export the data from MySQL into the CSV file and then import them into MongoDB.

An example of the command to import into MongoDB from a CSV file is:

```
mongoimport -d plantEST -c EST_FASTA -type csv -headerline -file
/tmp/EST_FASTA.csv
```

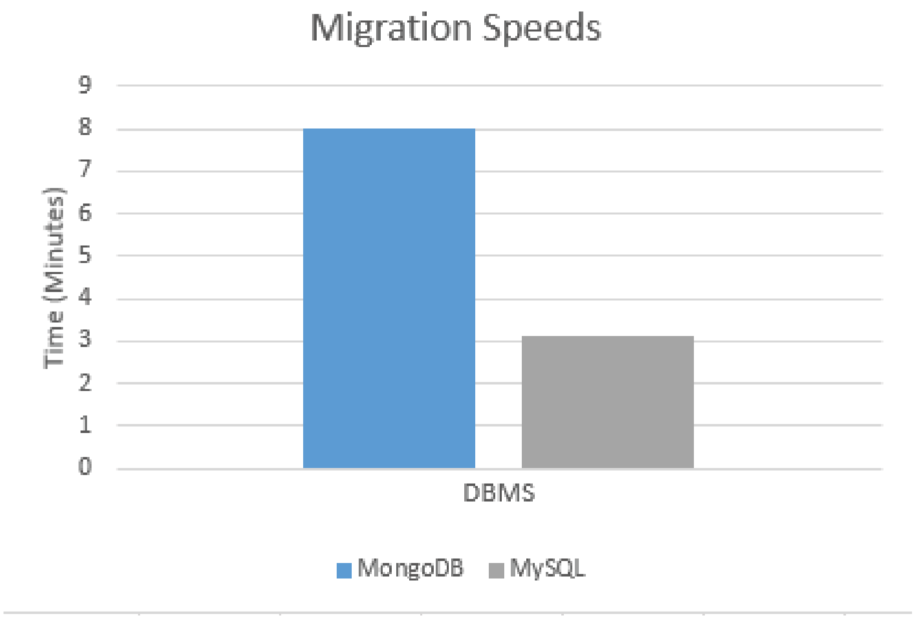

Figure 2: Migration speed of MongoDB compared to MySQL

The importation of data was successful and the speed of insertion was tracked throughout this experiment and replicated by reading from the files into MySQL to compare which DBMS has the faster rate of insertion. The results of this can be seen in Figure 2. It was discovered that the entire process took MongoDB a flat 8 minutes to insert all of the data provided from the CSV files. Interestingly enough it took MySQL only 3.12 minutes to insert the same amount of data. While research shows MongoDB is capable of working more efficiently querying larger amounts of unstructured and semi-structured data than MySQL, it appears the rate of insertion is actually in favor of a standard RDBMS. Part of the reason may be that this experiment yielded slower results was because of using only one node for working the inserts in MongoDB. MySQL is able to use multi-threaded methods to run some queries simultaneously while writes for MongoDB must sometimes be placed in a queue to await processing

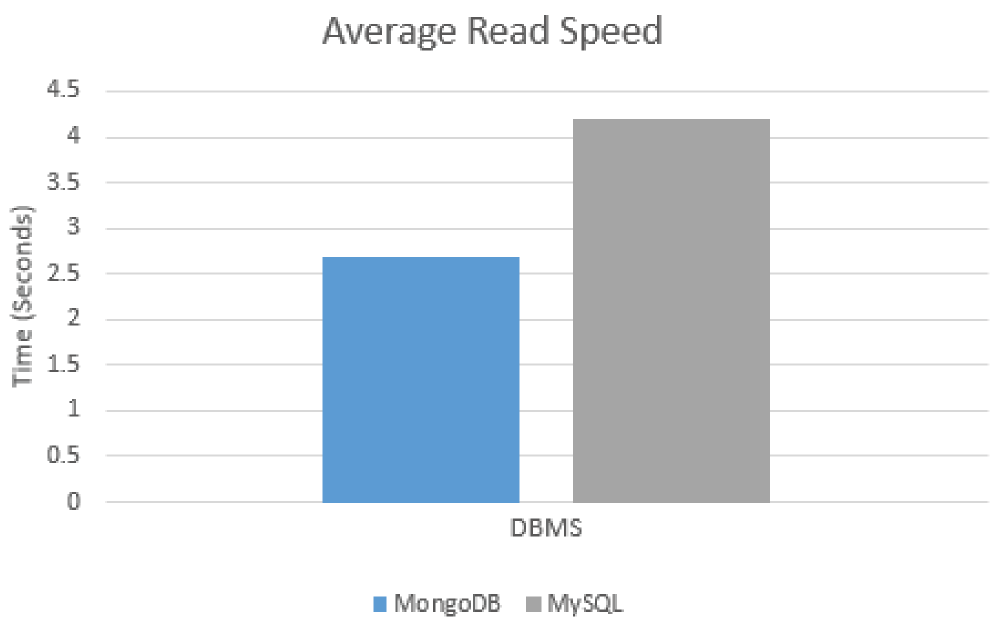

Figure 3: Data retrieval speeds of MongoDB compared to MySQL

10.2 Data-Retrieval

In order to test the effectiveness of MongoDB against MySQL for standard DBMS workloads we decided to test the read-speeds of each architecture. Five aggregation queries to pull the number of rows from the `Phobius` table. The average of these five queries was taken for each implementation. The query in question is an aggregate query that counts the number of rows within the table. MongoDB proved the faster DBMS with a time of 2.68 seconds as compared to 4.186 seconds for MySQL, meaning MongoDB took 64% of the time MySQL needed to complete on average. These results can be viewed in Figure 3. The reason for the speed increase for MongoDB is that MongoDB is optimized to handle simple queries very quickly and therefore has a boost in performance over MySQL. The reading through JSON-like format and the ability to swiftly read unstructured data gave MongoDB a significant advantage over MySQL in this case.

11 Conclusion and Future work

In conclusion, we have researched both relational and NoSQL databases and compared their advantages and disadvantages. We have worked along side a fellow Youngstown State graduate student with migrating a total of 13 tables of the biological database, `plantEST` from MySQL to MongoDB. The migration for this experiment was a success. We then ran importing and exporting data tests to compare the speed between a relational database and a NoSQL database.

We found that NoSQL is the better option, especially when working with large, unstructured data. When importing the data, MySQL seemed to have a faster speed than MongoDB but when retrieving data using queries MongoDB was more successful and showed better speed times. For future work, we will continue to research NoSQL databases further in depth to retrieve even better results when compared to MySQL. We plan on using Cassandra, a column-store database as my next experiment in hopes of getting much better results.

References

[1] Microsoft SQL Server. http://www.microsoft.com/en-us/server-cloud/products/sql-server/. [accessed on July 2015].

[2] MySQL. https://www.mysql.com/. [accessed on July 2015].

[3] Neo4j. http://neo4j.com/neo4j-scales-web-enterprise/. [accessed on July 2015].

[4] Oracle Database. http://www.oracle.com/index.html. [accessed on July 2015].

[5] PostgreSQL. http://www.postgresql.org/. [accessed on July 2015].

[6] SQLite. https://www.sqlite.org/. [accessed on July 2015].

[7] Relational Database Management System. https://en.wikipedia.org/wiki/Relational_database_management_system/, 2000. [accessed on June 15, 2015].

[8] C. Bazar and C. S. Iosif. The Transition from RDBMS to NoSQL, A Comparative Analysis of Three Popular Non-Relational Solutions: Cassandra, MongoDB and Couchbase. *Database Systems Journal*, 5(2):49–59, 2014.

[9] R. Cattell. Scalable SQL and NoSQL Data Stores. *SIGMOD Record*, pages 1–27, Dec 2010.

[10] L. Frank. Databases and Applications with Relaxed ACID Properties. *Doctoral Dissertation, Copenhagen Business School*, pages 10–15, 2008.

[11] S. Gilbert and N. Lynch. Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services. *Massachusetts Institute of Technology*, pages 1–12, 2002.

[12] A. Lakshman. Apache Cassandra. http://en.wikipedia.org/wiki/Apache_Cassandra, 2015. [accessed July 5, 2015].

[13] A. B. M. Moniruzzaman and S. A. Hossain. NoSQL Database: New Era of Databases for Big data Analytics Classification, Characteristics and Comparison. *International Journal of Database Theory and Application*, 6(4):1–13, 2013.

[14] C. Nacne, T. Losser, R. Iype, and G. Harmon. NoSQL vs RDBMS - Why There is Room for Both. *SAID 2013*, pages 111–116, 2013.

[15] L. Okman, N. Gal-Oz, Y. Gonen, E. Gudes, and J. Abramov. Security Issues in NoSQL Databases. *International Joint Conference of IEEE TrustCom-11*, pages 541–547, 2011.

[16] P. K. Pandya and J. Radhakrishnan. An Equational Theory for Transactions. *FSTTCS 2003:Foundations of Software Technology and Theoretical Computer Science*, pages 38–49, 2003.

[17] R. Ramakrishnan. *Database Management Systems*. Tom Casson, December 1998.

[18] R. Ramakrishnan and J. Gehrke. *Database Management Systems*. McGraw Hill, 2000.

[19] G. Smith. Hadoop and NoSQL: Technologies and the Oracle Database. *Oracle White Paper*, pages 1–11, Feb 2011.

[20] B. G. Tudorica and C. Bucur. A Comparison Between Several NoSQL Databases with Comments and Notes. In *Roedunet International Conference (RoEduNet), 2011 10th*, pages 1–5, 2011.