ABSTRACT

Migration of Legacy Web Application Using NoSQL Databases

by

Pouyan Ghasemi

April, 2013

Director of Thesis: Dr. Nasseh Tabrizi

Major Department: Computer Science

The Migration of the legacy web application to Content Management Systems (CMS) is inevitable because of the overload of managing the content. Traditionally, Content Management Systems are built with RDBMSes and the migration of the legacy web application is performed by transferring data from proprietary HTML pages to the corresponding database of the CMS. The newly introduced NoSQL databases beg the question as to whether SQL is the right choice for the migration. In this research, the performance of NoSQL was studied in a custom CMS and NoSQL was used to show how it can assist in the data migration process from legacy applications to the Content Management Systems. The performance and the storage requirements of NoSQL using Simple CMS were first tested in order to investigate the potential drawbacks of using NoSQL solutions. No negative effects on storage space and query performance were discovered. Along with these findings, the approach in this research incorporates document-oriented databases to enhance the process of data migration, and the result is that the dynamic schema of document-oriented databases makes the migration process more agile and accurate.

Migration of Legacy Web Application Using NoSQL Databases

A Thesis

Presented To the Faculty of the Department of Department of Computer Science

East Carolina University

In Partial Fulfillment of the Requirements for the Degree

Master of Science in Software Engineering

by

Pouyan Ghasemi

April, 2013

Migration of Legacy Web Application Using NoSQL Databases

by

Pouyan Ghasemi

APPROVED BY:

DIRECTOR OF DISSERTATION/THESIS:

M. H. Nassehzadeh Tabrizi, Phd

COMMITTEE MEMBER:

Junhua Ding, Phd

COMMITTEE MEMBER:

Sergiy Vilkomir, Phd

CHAIR OF THE DEPARTMENT OF COMPUTER SCIENCE:

Karl Abrahamson, Phd

DEAN OF THE GRADUATE SCHOOL:

Paul J. Gemperline, Phd

ACKNOWLEDGMENT

Special Thanks to

Dr. *Nasseh. Tabrizi*

Without whom this thesis would not have been possible

Table of Contents

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

In the early days of the Internet, it was a collection of static HTML pages linked together that created the Word Wide Web. First generation websites are those created by using static HTML. Organizations started taking advantage of HTML and the Internet to share their information assets by creating webpages and hosting them on webservers. Legacy web applications used their software of choice to apply practices such as WYSIWYG[1] and document editing that ensured ease of use for webmasters. By incorporating cascading style sheets they made web pages robust and visually compelling. Webservers hosted all of the files in the Server's File System and introduced routing mechanisms to map a particular URL to the path of the desired file on the server machine. Maintenance was the main disadvantage of this technology, as changes had to be applied one by one onto the web pages. Additionally, tracking the data and files on the server was cumbersome. This generation of web became known as the web 1.0 [1].

The above-mentioned paradigm has changed over the years. New tools and technologies have been introduced into the world of web development which revolutionized web applications. The main change has been the introduction of database layer within web applications. Databases offer a persistent layer for the web applications. Databases kept track of user activities and private data and separate the content and information from the logic behind the application. This new form of web application is referred to as Rich Internet Application (RIA) or the web 2.0 [1]. Web developers have embraced LAMP[2] [2] architecture as the dominant technology stack for creating web applications ever since and SQL[3] databases have been the primary choice of

[1] What you see is what you get
[2] Linux, Apache, MySQL, PHP/Perl/Python
[3] Structural Query Language

software developers. LAMP architecture has led to Content Management Systems. CMSes are efficiently designed to improve the process of sharing content and maintaining web applications [3].

The need for migration of the legacy web applications to an advanced system is inevitable. Enterprises are spending a substantial amount of time and assets towards migrating applications to their new platforms. In the case of static webpages, they are being migrated to CMSes. Currently, most of the Content Management Systems are built on top of RDBMSes such as MySQL or Oracle. Some examples of these are Joomla and Drupal. That is why SQL has been the number one and primary choice for software engineers with respect to migrating legacy web applications and websites to new Content Management Systems.

Engineers rely on two methods when it comes to migrating legacy web applications:

- Ad-hoc migration: data will be migrated to the new system as a bulk migration. Ad-hoc migration is planned in advance and requires a shorter amount of time. This kind of migration needs direct access to all of the data from the server.
- Lazy migration: data will be migrated when user requests the data. The migration process may take some time and occurs as a dynamic process. The advantage of lazy migration is that only the most recently accessed data is migrated [4].

Using lazy migration for migrating data from a legacy website that contains valuable information is not ideal. Lazy migration is usually reserved for migrating data between two modern databases as occurs between MySQL and Oracle. Conversely, harvesting the servers to extract webpages is typically a time consuming activity and it may be quite difficult to figure out the puzzle pieces and maintain a code that is capable of handling exceptional cases. In case of the some legacy web applications changes are applied to static web pages. Over the years they have experienced many iterations of adding features to a number of the pages. Those functionalities are usually introduced in terms of web scripts mostly on the server side (PHP, ASP). Overall, software architects tend to rely on the data that is accessible to the client since it may be a great reference for what data should be migrated versus what data should remain on the old server.

In both cases of migration, flat HTML pages are being read by the migration software and then after parsing the data, the proper information is written into the database. When using SQL base data structures, a software architect needs to know the data model and its structure in advance in order to define the proper tables for the RDBM of his/her choice. The process of finding and categorizing different HTML pages and creating efficient tables within the database for those data forms requires additional programming and design practice. If the migration process confronts an unknown HTML form, the software is unable to determine where to import the data considering the pool of available tables. This limit is being imposed by the fact that SQL databases restrict data entry based upon their initial definition.

During the past couple of years there has been a shift from SQL database to a new generation of databases called NoSQL. NoSQL stands for either "Not Only SQL" or "No SQL" [5]. NoSQL questions the limits of the SQL databases and attempts to find answers to the challenges and difficulties introduced as a result of using SQL family databases in the process of software product development and maintenance of the applications.

NoSQL refers to a vast category of new databases that do not follow the relational database paradigm. Even though relational databases or RDBMSes have been the number one choice for secure banking and financial applications, they impose their limits to the application developers in terms of defining the structure of the data in advance. With the rise of social media and cloud computing, SQL databases have imposed their shortcomings of lack- of scalability and poor performance when dealing with large tables of data. The NoSQL family of databases is typically categorized by the following categories [6], which will be described briefly in chapter 2:

- Key-value store
- BigTable
- Document-oriented Databases
- Graph Databases

In this thesis the focus is on migration of legacy web applications to a tailored CMS using document-oriented database. One of the main advantages of document-oriented databases is that they do not impose a restrict schema on developers. MongoDB [7], a document-oriented database developed by 10gen [8] has been selected as the database here. MongoDB is a one of the most promising databases of the future offering such built-in features such as auto-sharding, horizontal scalability, full index-support, and map/reduce. This thesis is organized as follows:

Chapter 1 will provide an introduction to the thesis. It will contain information about legacy web applications, the migration of legacy web applications, and a brief introduction to NoSQL databases.

Chapter 2 will provide the related work to the research introduced in Chapter 1 and will deliver some background about NoSQL databases. It will also contain an introduction to MongoDB, the document-oriented database chosen for this research. The new aspects and functionalities of MongoDB will be studied; primarily, the features in comparison to their counter parts in MySQL.

Chapter 3 will include a survey on different CMSs and their features. In this chapter, Simple CMS, a light-weight CMS developed for this thesis, will be introduced, as it is specifically tailored for benchmarking MySQL and MongoDB. This section has been included in order to ensure the system’s performance was not compromised.

Chapter 4 will focus on benchmarking the databases using Apache JMeter. JMeter has been used to run performance RESTfull queries against both databases in their CMS incarnation. This chapter will also include information about the software used to run the benchmarks.

Chapter 5 will introduce a new migration practice in using MongoDB. MongoDB features will be incorporated into a middle tier platform to enhance the current data migration practices. Steps of migrating data between MySQL and MongoDB will be provided.

Chapter 6 is a conclusion and will summarize the research included in this thesis.

CHAPTER 2: RELATED WORK AND BACKGROUND

On the first part of this chapter we provide information about the related work to the research. The second part is dedicated to the NoSQL databases and their use cases. On the third section we introduce MongoDB which is the document-oriented databases being used for the legacy web application migration in this research and finally we provide comparison between MongoDB and MySQL.

2.1 Related Work

Because of the newness of NoSQL solutions there has been little effort in explaining and exploring their potential. Ironically, the migration of web applications needs to be performed with the latest technology though SQL is still being utilized to perform the migration activities. Seth [3], in his master thesis explored the idea of migrating a legacy web application to Joomla CMS. The effort would involve justifying the migration of the legacy applications, along with proposing a tool to handle the migration process. MySQL has been used for RDBMS databases. The author is restricted to design Joomla's schema within the initial phase. It may be a daunting challenge when the data within the legacy web application is not consistent or has no particular structure. More research in this field is mentioned in Doug's [9] work in reimagining library web guides in Content Management Systems. The result of his work is satisfying but then again, in that research, MySQL is used to perform the migration process. The work starts with the assumption that every aspect of the legacy data is already known; however, the result of their effort is rather complicated, as they create several tables and relationships between those tables. This potentially may have a direct negative impact on the migration process.

Yunhua's article published in 2011 [10] looks at the problem statement from the perspective of a web crawler. In this work, MongoDB features were incorporated into the design of a web crawling application. The application enabled them to store web crawled information from web pages from the Internet and stored them in a NoSQL solution (*MongoDB*). No visible effort of data migration has been seen from a legacy web application to a NoSQL solution. This thesis research studies the migration process from legacy application in detail.

Along with the use cases studies of NoSQL solutions, there have been great strides in studying the performance of NoSQL and in comparing it with SQL [11]. These results are especially satisfying especially in cases where the data sets are large and horizontal scalability is required or relations between data sets are beyond SQL's ability [10]. In Chapter 4 of this thesis, examples of benchmarking CMSes with NoSQL databases can be found; therefore, questions about the performance of NoSQL solutions in the form of a CMS to ensure that there would be no performance drawbacks for the migration.

2.2 Background

In this section it is assumed that the reader has basic knowledge of SQL databases which is why only a brief introduction to the concept is provided along with more details about NoSQL solutions, further cementing the reason why NoSQL solutions were studied to determine that it is a better fit for the migration process.

2.2.1 Databases

As Chamberlin [12] mentions in his article "Early History of SQL," the main challenge during the early 70s was to find a system and language that could handle data persistency. Relational Database Management Systems and Structural Query Language are the result of that effort. In general, RDBMSes store the data in two dimensional arrays called tables. Databases usually contain several tables and developers design relations between tables to achieve the required functionality. Most RDBMSes ensure the ACID[4] transaction which is a great fit in use cases such as in financial and enterprise applications; yet, as is mentioned in Chapter 1, RDBMSes no longer fit all of the requirements of the new generation of applications. With the rise of cloud computing and social networks, application developers and innovators are searching for new databases that fit their specific use cases. NoSQL database designers are attempting to fill this gap by introducing databases that are more customized for the contemporary use cases.

SQL databases such as MySQL and Oracle convey two types of information: the data itself that is stored in tables, and the relationship among these tables. Tables may be seen as flat files, while rows in the flat files represent a collection of data values. The entire row is referred to as tuples. Rows are capable of storing data of any type such as String, Integer. Schema in a SQL database is the table with its attributes [6]. ACID transaction is the principle that RDBMSes are based upon; ACID transaction ensures the following [13] [14]:

- Atomicity: refers to the atomic transaction which means that either all of the changes or none will be applied.
- Consistency: in a consistent transaction where each transaction ensures that the data is in a consistent state after and before the transaction.
- Isolation: concurrent transactions appear to be serialized and two running transactions will not interfere with one another.
- Durability: the effect of a transaction will be persistent and it will not be lost even in case of the system failure.

[4] ACID (Atomicity, Consistency, Isolation, Durability)

Consistency of the SQL is guaranteed by supporting ACID transactions. In his article, Leavitt [15] formulates relational database limitations by pointing to the fact that SQL is not scalable; they can be quite complex when the data has no structure. In such a database, handling large feature sets is a problematic task for developers.

2.2.2 NoSQL Databases

Some believe that NoSQL databases started with Berkeley DB which was developed during late eighties and early nineties. Berkeley DB has now been acquired by Oracle but the real movement of NoSQL databases started after 2008, and since then, it has been rigorously advancing. There is not a single definition to cover all of the aspects of the NoSQL databases, as they cover variety of features. However, one thing that they have in common is that they do not follow the mainstream SQL convention [16]. Currently, there are a number of applications written in databases other than SQL and that trend appears to be growing [5]. The main categories of NoSQL databases are to follow.

2.2.3 Key-value Store

Amazon's Dynamo and Berkeley DB fall into this category. Key-value store databases act as dictionary**,** a familiar data structure for programmers. Data is retrieved using the key associated with the value. As a result of that key-value store databases are highly scalable but they lack the consistency enjoyed by other databases [17]. The database API is easy to learn and intuitive in key-value concept since it provides a set of simple operations [15]. Key-value store is ideal for holding both structured and unstructured data [15]. The following shows a simple set of structures performed by Dynamo DB [17]:

- get(key): returns a list of objects.

- put(key, context, object): for adding the content

This simple set of operations is similar to Map API in Java and other programming languages. Redis is the most famous open source key-value store database sponsored by VMware [6]. Databases such as Redis are typically used to make databases faster by storing the data in memory[5].

2.2.4 BigTable

BigTable databases are sometimes referred to as "Column Family Store" [6] or as just CF. Facebook's Cassandra [18] and Google's BigTable are great examples of these types of databases. These databases tend to be scalable and highly available, and they store the data in a column rather than a row. Also, they store structured data in a way that is linearly scalable. Google's BigTable does not support tables association but different flavor of these databases have different features. Concurrency appears to be the winning factor for these databases. Each column thread is being processed by one procedure that results in concurrent I/O [19]. The Cassandra sample API set is displayed below [17]:

- "get(table, key, columnName)
- insert(table, key, rowMutation)
- delete(table, key, columnName)"

One thing that may be seen as a trend in NoSQL databases is that they have a tendency to keep the database API as simple and as close to the mainstream programming languages such as Java or JavaScript as possible.

[5] Memcached

2.2.5 Document-oriented Databases

Document-oriented databases are the main focus of this research. These databases store data as a collection of documents. High performance read and write is not on top of their priorities [19]. The fact that they follow a no-schema design with no restrictions makes them a great fit for the development of web-applications [15]. They allow more complex data structures than both Key-value store and BigTables. Apache's CouchDB and 10gen's MongoDB are two well-known databases in this category. These two databases have the following aspects in common:

- CouchDB uses JavaScript Object Notation (JSON) [17] as a format for transferring data and MongoDB uses BSON which is a Binary JSON. BSON and JSON both support embedded documents and arrays [20].
- Both databases support and provide REST[6] API for accessing data.
- CouchDB and MongoDB are both schema-free and do not constrain developers to design relations before using the database.

Because of the way CouchDB communicates with the application and file system, ACID transactions can be applied to it. Even though MongoDB does not provide an ACID transaction it provides some features like Atomic update that ensures some basic transaction capabilities [7]. More details on MongoDB and its features will follow in this chapter.

2.2.6 Graph Databases

In some cases researchers do not consider graph databases to be a separate category of NoSQL databases [15] [17]. In other cases graph databases are considered to be a separate

[6] Representational State Transfer

category since they offer a rich set of features to manage and to store specific sets of data. As may be gleaned from their names, graph databases are used to store and retrieve data that can be represented as a graph. Social networks and the connections between people are a great example of how they can be used. Researchers have used SQL databases to represent graph data in the past, and that has had some negative effects on the performance of their algorithms, and it has made the problem even more complicated.

In graph databases data is represented as a mathematical graph with the following representation $G = (\mathrm{V}, \mathrm{E})$, in which V is a set of nodes and E is a set of relations between nodes of the graph [6] . Neo4j [6] is one of the promising graph databases that have been developed in Java. Neo4j uses nodes and relations (edges of the graph) to hold the data in a graph structure. Nodes and edges in Neo4j have sets of properties which add meaning to the data and distinguish this database from that of a simple data structure. Queries such as "return all of the nodes that are in relation with a specific node" are built-in queries in Neo4j and may be easily expressed by developers. Applying the same query in MySQL usually requires joining tables and sometime recursive function calls which is a time consuming process for the database.

2.3 MongoDB

Different categories of NoSQL databases were studied and in continuing to study MongoDB in detail, some insights about this type of document-oriented database will be offered. In addition, the differences between MongoDB and MySQL will be presented, and some query examples in both databases will be provided at the end of this chapter.

2.3.1 MongoDB Features

MongoDB is an open source database written in C++ and supported by 10gen under APGL[7] license. Craigslist, Foursquare, and numerous other companies use MongoDB in production [6] [7]. Each database in MongoDB has a set of collections. Collections in MongoDB are similar to that of tables in MySQL. Each collection holds documents which are similar to that of rows in MySQL. The main difference between MySQL and MongoDB is that while the way in which documents inside a MongoDB collection are not required to follow a particular schema, conversely, MySQL imposes this to the user. Documents inside the collection are stored as BSON objects.

The following are some features of MongoDB:

- Rich queries:

Unlike RDBMSes, MongoDB uses JavaScript programming language to provide interface with the database [7]. This decision has made MongoDB easy to use for web developers who are already familiar with JavaScript syntax. MongoDB provides a rich set of operations for CRUD[8] applications which are the core for database driven application development [7].

- Document-oriented storage:

BSON which is the core unit of every MongoDB document is designed to be lightweight, traversable, and efficient [20]. The following displays a sample document in MongoDB:

[7] Affero General Public License
[8] Create Read Update Delete

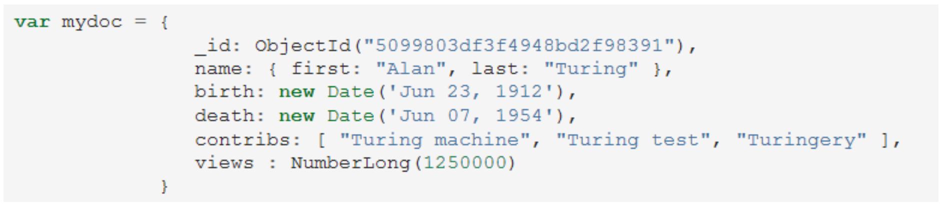

```
var mydoc = {
               _id: ObjectId("5099803df3f4948bd2f98391"),
               name: { first: "Alan", last: "Turing" },
               birth: new Date('Jun 23, 1912'),
               death: new Date('Jun 07, 1954'),
               contribs: [ "Turing machine", "Turing test", "Turingery" ],
               views : NumberLong(1250000)
            }
```

Figure 1 : BSON document **[7]**

Two documents within the same collection may have different fields. The only restriction that MongoDB has is that the **_id** field is reserved as primary key. The **_id** is generated by MongoDB unless it is replaced by a unique **_id.** As displayed in the next picture the documents in MongoDB may have embedded documents.

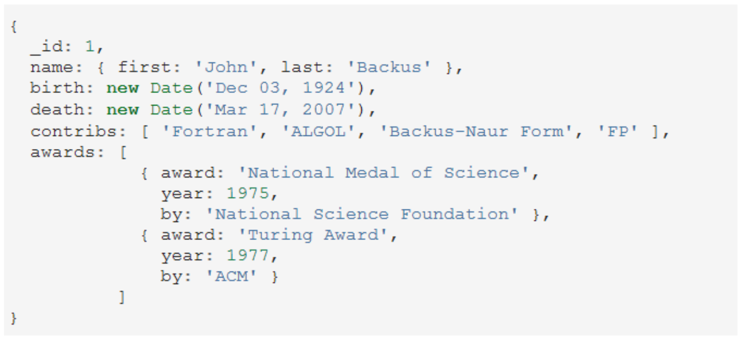

```
{
  _id: 1,
  name: { first: 'John', last: 'Backus' },
  birth: new Date('Dec 03, 1924'),
  death: new Date('Mar 17, 2007'),
  contribs: [ 'Fortran', 'ALGOL', 'Backus-Naur Form', 'FP' ],
  awards: [
            { award: 'National Medal of Science',
              year: 1975,
              by: 'National Science Foundation' },
            { award: 'Turing Award',
              year: 1977,
              by: 'ACM' }
          ]
}
```

Figure 2 : MongoDB embedded document **[7]**

In looking closely at the document it is clear that the “Awards” field displays a document of awards that John has received. In order to store this information in a normalized manner in MySQL developers have had to create a separate table for “Awards.” John may have as few as zero to as many as several numbers of awards in MongoDB. The only restriction is that the maximum document size in MongoDB is 16 megabytes, but that should be sufficient for storing massive textual data. This criterion ensures that a single document does not require an excessive amount of RAM or bandwidth. In order to store larger documents such as video files and pictures MongoDB provides GridFS [8] [7].

- Replication:

Database replication in MongoDB is a built-in functionality that ensures database backup, high-availability, and automatic failover. There are two replication mechanisms offered by MongoDB: master/slave replication and replica sets. Replica sets are favored in MongoDB for databases with less than 12 nodes [7]. Replica sets in MongoDB provide automatic failover. In automatic failover an instance of the database may automatically choose another primary if one fails. More information about replica sets is available on MongoDB's website, as this aspect of MongoDB is not the focus of this thesis.

- Sharding:

Sharding is what makes MongoDB scalable. MongoDB provides sharding by making the database horizontally scalable among several numbers of machine instances called nodes [17]. Automatic balancing and loads are performed by databases. Sharding distributes documents on several machines based upon their shard key. The Shard key is a field in the MongoDB collection that has been used to distribute data across clusters [7]. Sharding in MongoDB is based upon collections. If a collection expands, MongoDB automatically shards that collection, while other collections remain on a single machine as long as they do not require additional space. The following displays an example of a MongoDB sharding cluster.

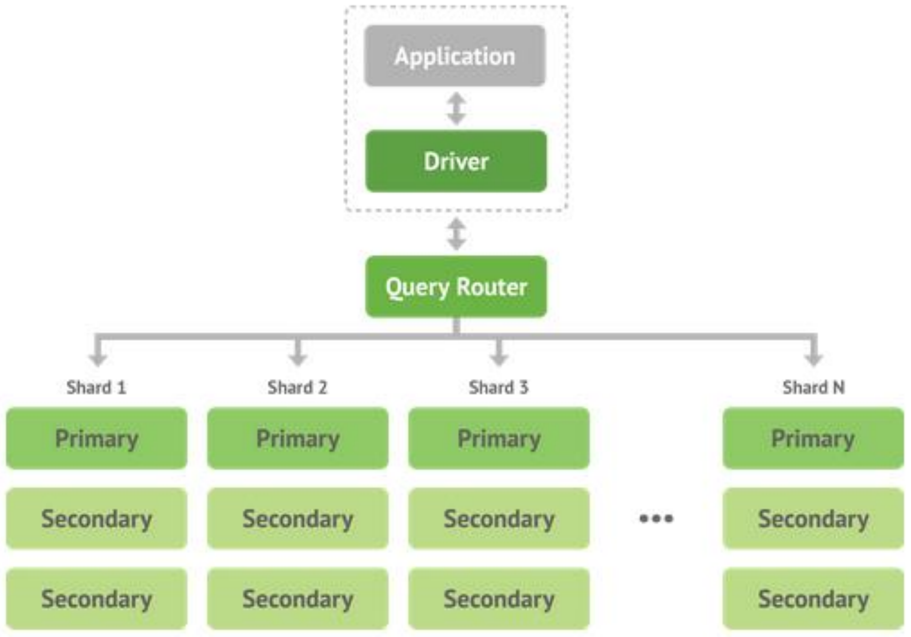


Figure 3 : MongoDB sharding **[8]**

- Full index support:

  Indexes in MongoDB are like indexes in many SQL databases. They provide high performance access to data especially when the size of a collection is bigger than RAM. MongoDB supports indexes for both fields and sub-fields. Even though indexes do enhance the query performance they often have a negative effect on the write performance since they add overhead to the write operation [7]. In addition to text indexing, MongoDB provides Geospatial indexing which is useful for querying geospatial data in applications such as GIS or Maps.

- Map/reduce:

Aggregation framework was introduced in MongoDB 2.2. Prior to that Mongo users chose map/reduce as a tool for aggregating content [7]. The Map/reduce mechanism in MongoDB is similar to that of Google's Map/Reduce paper [17]. Even though new aggregation key-words are introduced into the database, Map/reduce is still a more powerful tool for handling complex aggregation tasks.

- GridFS:

GridFS divides larger data files such as video or audio into several pieces called chunks. Each chunk is stored in a separate document. MongoDB also uses two collections to store GridFS data. One of the collections is used to store the metadata and the other to store chunks of data. By dividing the chunks MongoDB enables developers to access a particular piece of a video or an audio file and skip pieces as necessary [7] [17]. This feature is useful when developing web applications that need to transfer or load large data files for the sole purpose previewing parts of it to the user. Youtube and Vimeo are good examples of this feature.

2.3.2 MongoDB Use Cases

MongoDB has been widely used by web application developers who employ the agile and iterative approach to application development. Start-ups are great fans of MongoDB since it enables the developers to have fast iterations and add functionalities on the fly. Customers of MongoDB may be categorized as following [8]:

- Big Data:

Auto-sharding is the feature in MongoDB which makes it a great fit for the cloud. Cloud providers usually charge customers on a scale and MongoDB's auto-sharding may dynamically allocate resources. This leads to an efficient use of servers and databases provided by PaaS[9] or

[9] Platform as a Service

IaaS[10] services. Deployment of MongoDB in organizations such as Craigslist and Disney is the successful evidence of Mongo being a great fit for handling big data operations [8].

- Content Management Delivery:

Web applications such as an online shop or a flash sale needs to deliver different types of content and products to users. The data for these products is not always structured. A massive amount of data that needs to be stored may be un-structured or hierarchal, or polymorphic [8], which explains why MongoDB has been a successful replacement for MySQL in some content management delivery systems. Integrated features such as GridFS and Full-text search reduce the complexity of content management delivery systems that use MongoDB as their database [17] [7].

- Mobile and Social Infrastructure:

MongoDB as a modern database supports Geospatial capabilities [17] that may be beneficial for mobile application development and social network. Because of the small form factor of the mobile displays and the noises in social media, companies are constantly looking for ways to add insights to the data that is provided to the user. MongoDB's map/reduce along with aggregation framework offers rich sets of features than may be used for fast and agile data mining, knowledge extraction, and deployment of recommendation systems [8].

- User Data Management:

MongoDB enables developers to store complex user data models in short development cycles. One winning factor for MongoDB is the commercial support that is provided by 10gen. Commercial support distinguishes MongoDB from its competitors. Organizations may start

[10] Infrastructure as a Service

small by using MongoDB as an open source project and continue to do so. Meanwhile they may collaborate with 10gen if the database needs additional care and attention.

- Data Hub:

Scalability and low cost of ownership is what encourages most organizations to use MongoDB as a central data hub that collects all of the data. Storing all of the data in one place provides the benefit of applying rich business intelligence practices to obtain valuable insights on the data. As previously mentioned, map/reduce is a first class concept in MongoDB which reduces additional overhead by using Hadoop for data analysis among several shards of the database [17] [8].

2.3.3 Cloud Providers Supporting MongoDB

Platform-as-a-Service cloud providers usually provide set of pre-configured databases for their customers. Amazon EC2, dotCloud, Joynet Cloud, Red Hat Openshift and many other cloud providers offer MongoDB as a choice of database. MongoDB features are designed to perform efficiently and smoothly in a cloud environment. Auto-sharding enables companies such as Openshift to provide pay-as-use payment system for the customers of MongoDB. If a single database instance expands enough, the additional shard will be added to the system and data will be distributed based upon the shad-key. After a while, if the database shrinks enough that it no longer needs an additional shard, MongoDB will automatically shrink the data back to smaller instances [17].

2.3.4 Full Text Search

Even though SQL databases such as MySQL and Oracle provide built-in text search features, most application developers rely on third party tools for searching the content of their applications. Open source tools such as Elasticsearch and Apache Solr are examples of third party APIs that provide searching facilities. Even though it is strongly recommended to use a mature searching API, sometimes adding additional code-base increases complexity and could become a point of failure. In order to enable searches for databases the particular field needs to be indexed which may reduce the write performance of the database. This is inevitable in both databases.

Full text search is added to the MongoDB version 2.4, which was released on March 19, 2013. Prior to the release an alpha version was available for developers on Github. MongoDB search matches on complete stemmed words similar to MySQL text search. The API offers sets of options to customize the search result query [7]. MongoDB full text search currently supports text search for texts written in European languages.

2.4 MongoDB and MySQL Comparison

Among NoSQL databases document-oriented databases are the ones that have the most similarities with SQL databases. In this research MySQL was chosen as the sample SQL database. In fact**,** MySQL is also an open-source database with an active developer community. MySQL is used in numerous companies and has its own custom made tools for distributed computing such as Hadoop. Feature comparison of different NoSQL databases with RDBMSes is shown in picture below.

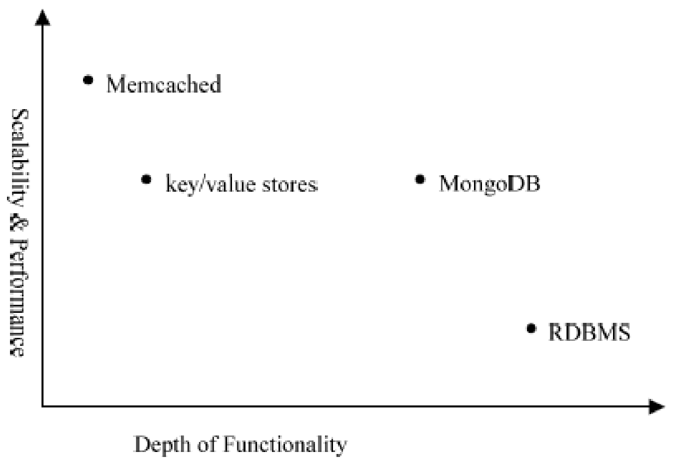


Figure 4 : Feature comparison **[10]**

Some differences between MongoDB and MySQL will follow:

2.4.1 Data Modeling Comparison

The following table maps the terminology of database elements between MySQL and MongoDB [21]. Most of the features within RDBMSes are offered in MongoDB databases. The notion that an embedded document is a replacement for Join may be misleading. In fact, it all depends upon the system architect's approach on how to aggregate and model the data. Overall "Joins" may be achieved by using embedded documents, aggregation framework, map/reduce, and references to other collections.

RDBMSES	MongoDB
Views and Tables	Collection
Row	Document

Index	Index
Join	Embedded Document
Foreign Key	Reference
Partition	Shard

Table 1 : MongoDB and RDBMSES terminology

Figure 5 shows the data model of MongoDB [11]. It is different from the table structure of a MySQL database. Consider a library application in MySQL. Database typically contains a table for the books and another table for the users of the library. If information is needed about which user has borrowed which book, an additional table is needed to store books that are borrowed by the user.

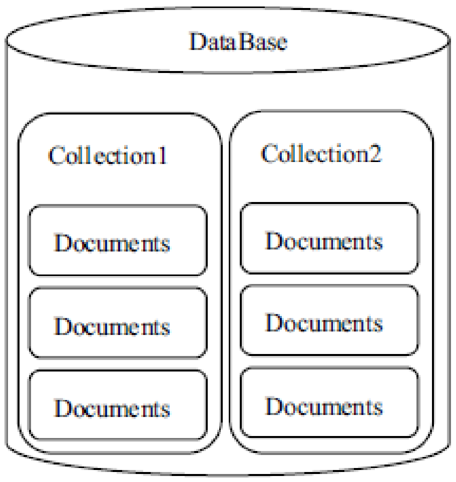


Figure 5 : MongoDB data model

Modeling the same structure in MongoDB, all of the information about the books needs to be stored in a separate collection but it is not necessary to have an additional collection for the books borrowed by a student. Borrowed books for a specific user may be stored as an embedded document. It is clear that finding books that are borrowed by a specific user may be faster in this case. There are different data modeling approaches that mostly rely on embedded documents. It is strongly advised to consider different methods and decide upon an optimized model that is tailored for the application.

2.4.2 Query Analysis

Creating a collection in MongoDB is as simple as giving it a name. There is no need to define a schema. The following is the initialization of a simple table of "users" in both MongoDB and MySQL [7].

MySQL	MongoDB
`CREATE TABLE users (`	`db.users.insert( {`
`    id MEDIUMINT NOT NULL`	`    user_id: "abc123",`
`        AUTO_INCREMENT,`	`    age: 55,`
`    user_id Varchar(30),`	`    status: "A"`
`    age Number,`	`} )`
`    status char(1),`	
`    PRIMARY KEY (id)`	
`)`	

Table 2 : Creation of user data model

As is displayed, the "users" collection will be created on the fly by adding a document to a collection by that name. Querying the data in MongoDB is quite similar to accessing object's attributes and methods in an object oriented programming language. The “Find” query in MongoDB replaces the “Select” statement for MySQL [7].

MySQL	MongoDB
`SELECT * FROM users WHERE status = "A" ORDER BY user_id ASC`	`db.users.find( { status: "A" } ).sort( { user_id: 1 } )`
`SELECT * FROM users WHERE age < 25`	`db.users.find( { age: { $lt: 25 } } )`

Table 3 : Find query

MongoDB uses both aggregation commands such as “sort” and aggregation operators such as “$lt” (less than) to achieve the results. A full comparison table of the MongoDB and MySQL may be found in the appendix.

CHAPTER 3: CONTENT MANAMENET SYSTEMS

Content Management Systems are a great replacement for static web html pages, since they aggregate the data in an organized form and deliver the textual and graphical content to the users in a maintainable manner. The purpose of Content Management Systems has been categorized by Pirtle as: 1) Manage content; 2) Organize navigation; 3) Dynamic search; and 4) Self-service management of content [22]. Most CMSes provide these basic functionalities for the users and all of these four purposes rely heavily upon the choice of database. It is clear that CMS refers to a wide range of applications that simplifies the process of managing and publishing web content. Web content could be anything, ranging from a picture to textual content or document files such as Pdfs or Xls. The features of a CMS are highly coupled to its architecture and also to the technologies used by CMS. Additionally, the community that uses the CMS also has a great influence on it.

Drupal and WordPress are examples of widely used Content Management Systems that are open source. Drupal and WordPress are both written in PHP and use MySQL as their database to store data. Different CMSes have different levels of abstractions. Some, such as WordPress, put their focus on publishing textual or graphical content, while Drupal is a more generalized CMS which may be customized to serve different use cases. Currently**,** Drupal serves several e-commerce, web-forums, and even enterprise applications on the web [23].

CMSes, much as many software applications have several licensing categories. A considerable number of CMSes are developed and maintained by the open source community, which has contributed a lot of features to CMSes and as a result, Content Management Systems have matured.

CMSes, from a user perspective, may be separated into two parts: The admin user interface, which is the medium between the system and the webmaster or content managers, and the presentation of content to the users of the system. The admin interface provides simple functionalities for users to manage the content, and depending upon the type of content and the design of the CMS, there might be a learning curve for the users on the system at this level. The ideal goal for the admin user interface is to be simple enough for users to easily add or edit content. The presentation interface process involves data-visualization and content delivery. CMSes usually accomplish this by delivering the content in an HTML page format. The look and feel of the HTML page may be customized by the webmasters of the CMS. Some CMSes enable the users to interact with the databases as well. Features provided by some CMSes, such as “commenting” on a post or “liking” a post, are used mostly in Web 2.0.

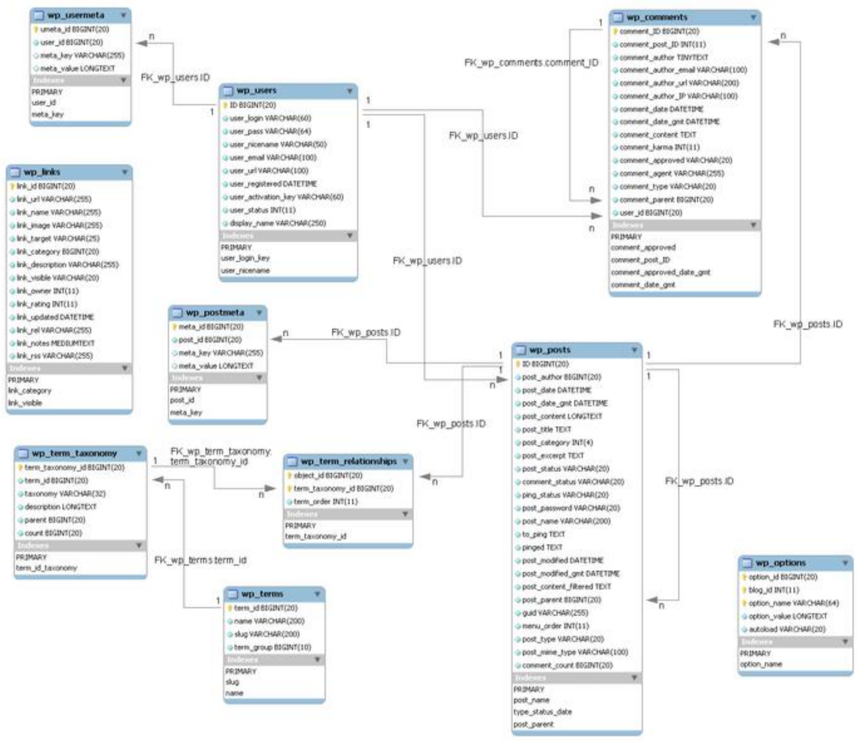


Figure 6 : WordPress CMS table relations

WordPress as a CMS is ideal for blogging and textual content, and focuses more on posts. To better study the connection between CMSes and relational databases, the relationships among tables of the WordPress CMS [24] is shown below in Figure 7. It shows that “wp-posts” table has more attributes than other tables within the database. Additionally, it is clear that the relationship between “wp-posts” and “wp-postmeta” identifies the additional table that WordPress uses to store the metadata. This restriction is imposed by MySQL or any other RDBMS. In a SQL database adding attributes to the “wp-posts” tables increases the complexity, while it also decreases the performance of the database. Given this structure, developers need to use SQL standard queries to retrieve metadata for specific posts, often creating problems in applications that require intense querying of metadata. To make RDBMSes fit for this use case, developers would de-normalize the data by adding a table that holds a joint view of both tables.

Pritle compares SQL and MongoDB's databases' schema for commenting and tagging an article, and as is shown in the display, the MongoDB version uses hierarchical data storage to simplify the collections. Querying the data in MongoDB will be managed by the use of either aggregation framework or Map/reduce [22].

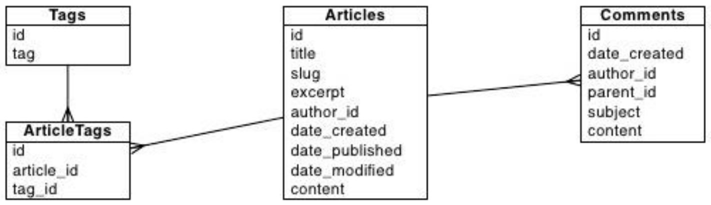


Figure 7 : SQL table relations

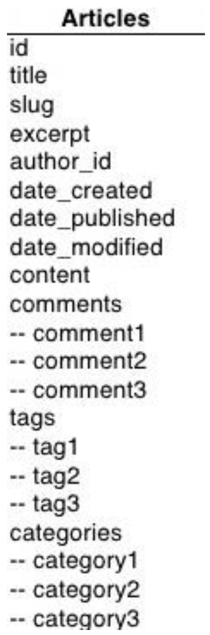


Figure 8 : MongoDB collection structure

Figure 7 and 8 show a general representation of the content management in both databases. These representations may be changed and optimized based upon the performance requirements of the system. Still, it is important to note that they both represent a mainstream design for MySQL and MongoDB and highlight the differences.

3.1 NoSQL CMSes

CMSes like many modern web applications are heavily reliant upon their communication with the databases, as it is undoubtedly necessary for CMSes to optimize their architecture to maximize their throughput. As a result, in most cases CMSes are tailored to serve a single database of their choice. This decision optimizes the code base of the CMS. Users are forced to install the database that the CMS imposes. Currently, RDBMSes hold the major responsibility in the database backend in CMSes. Even though this trend is shifting, many well-known CMSes are not built with the purpose of having a NoSQL database as backend. The efforts in moving to the NoSQL structure for the Content Management Systems may be categorized by the following groups:

1) Using the same traditional CMS structure of SQL and applying it to NoSQL;
2) Adding NoSQL as a separate entity to the current CMSes to support new functionalities; and
3) Reimagining CMSes by designing a CMS that incorporates NoSQL features.

Each of these methods has its own advantages and disadvantages that may be different based upon each use case. The decision to choose the right CMS among the massive pool of current SQL and NoSQL CMSes is often challenging and requires substantial research.

3.1.1 Traditional CMS Re-written in NoSQL

The goal in this category is to design a CMS that offers the same functionalities as a traditional CMS; yet uses a NoSQL database as opposed to SQL. These CMSes use almost the identical code as is found in a traditional CMS. Since the web-applications have been used through the years, their features have matured and systems have been proven to be reliable. The decision to design these new CMSes is largely due to the scalability issues of MySQL and the main challenge is to provide horizontal scalability. Lily and Daisy CMS are examples of such systems.

3.1.2 NoSQL as an Additional Entity

In this case NoSQL database acts as an extra power provider that works alongside of the SQL data storage units. The NoSQL module of the system is used for the development of new features that are more suitable for NoSQL database. Even some CMSes use NoSQL database as a replication system. An example of such work is an integration module that has been developed for Drupal and uses MongoDB. The module stores data for several fields such as cache, lock, or sessions in a MongoDB instance. MongoDB works and communicates with the application and the SQL database to provide insights and improve the architecture. Efforts in this category are proven to be reliable as they have the benefits of both systems. However, the negative effect is the increase of the complexity that is being introduced by adding an additional database abstraction layer. Teams who employ this method should have a good working knowledge of both NoSQL and SQL products.

3.1.3 Reimagining CMSes by Incorporating NoSQL Features

The third category tries to benefit from new features of NoSQL databases and create a more flexible and modern CMS. Prono and Locomotive are both open source CMSes that have pushed some boundaries of Content Management Systems. Locomotive uses MongoDB's document-oriented concept and enables users to easily create custom content types and manage them. Locomotive also provides cloud computing functionalities and scaling features for the CMS.

3.2 Simple CMS

The data migration process from a legacy web application to a Content Management System may be different depending upon the type of Content Management System being used. Most Content Management Systems provide API for the developer and users that enables them to migrate the data to their systems. Unfortunately, as was previously mentioned, they are not offering it for two different databases like MySQL and MongoDB. Even if it is available in some cases the underlying architecture of the CMS will be optimized based upon the functionality of one of the databases which creates bias within the benchmarking process.

In order to experience benchmarking in both MongoDB and MySQL, a Simple CMS was developed. Simple CMS is a minimal CMS that only stores html pages. Simple CMS has been developed on Python using Flask micro-framework, and because content delivery is the ultimate goal of any CMS, Simple CMS has been designed to efficiently deliver content based upon a REST API. Additionally, the application explores the full-text search feature that is built into the two databases.

3.2.1 Flask Framework

Flask is a micro framework that is inspired by Ruby's Sinatra. It is built on the Werkzeug, which is a utility library for Python [25] which developers may use to develop their own framework. Flask appears to keep the web application simple and flexible [26] and offers additional functionality by allowing extensions. Flask, unlike many other web frameworks, does not make the decision on which databases to use. In fact, Flask delegates the database abstraction layer to the developers, allowing them to choose different libraries that already exist for connecting to the database [26]. Flask's mission is to be used as a foundation for all types of web applications. NoSQL community has endorsed Flask because of its simplicity and because Flask makes it easy for developers to use new databases in the applications. Companies such as MemSQL use Flask to provide benchmarking to their customers [27]. Flask has been employed in this study to create a CMS application that connects to both MySQL and MongoDB.

3.2.2 Simple CMS Web Application

The overall architecture of the Simple CMS shows the straightforwardness of the application. Simple CMS delivers HTML documents and provides search functionality to users. The best way to deliver HTML content to users is subject to different applications. In Simple CMS a textual metadata referred to as "iamap" is employed to categorize the content. "iamap" is a substitute for "Information Architecture" of a web application. HTML pages are stored within the DB and users access that data using the "iamap" or the actual key of that object.

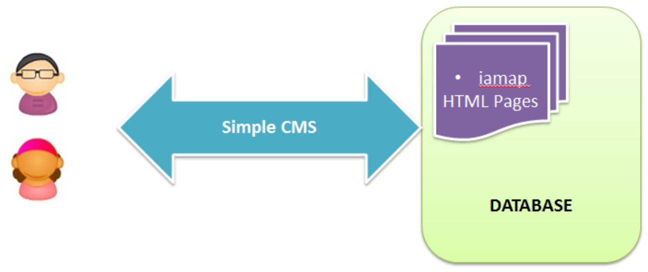


Figure 9 : Simple CMS abstraction form databases

Out of REST requests in any RESTfull CMS application, “Get” requests are the ones that are most frequently used and may have intense effects on the performance of the CMS. Simple CMS’s API provides five “Get” requests, and these requests have two main responsibilities: one is to provide data to the client and the other is to render templates. These sets of “Get” requests are identical in both MongoDB and MySQL versions of the CMS.

	Get Request	Response
1	@app.route('/')	Renders the “Index” Page
2	@app.route('/map/<iamap>')	Returns the pages that have the same “iamap” of the one in the URL
3	@app.route('/page/<int:pageid>')	Returns a page with its primary_key which is the “pageid”
4	@app.route('/search')	Renders the “Search” Page
5	@app.route('/search_query')	Returns a JSON object of the search query’s result.

Table 4 : Simple CMS GET queries

Both CMSes are running on Virtualenv, which is an isolated environment, guarantees that there will be no conflict between the installed packages within the system and also improves the process of debugging by tracing back directly to the package [28]. In this study, “pip” is used to install packages on Virtualenv and two CMSes are running on two separate, identical Virtualenv.

3.2.3 Simple CMS-MySQL

The MySQL version of the Simple CMS is similar to that of the MongoDB, with respect to displaying the content. The difference, as expected is the communication between the application and the database. The SQL based CMS uses MySQLDB which is a python library for connecting to the MySQL database. In order to enable full-text search html pages sources are stored and indexed in a MyISAM table. The following tables display a side by side comparison of the queries for MySQL and MongoDB.

Corresponding Queries for the “Get” Requests	
@app.route('/map/<iamap>')	
MongoDB: Articles.aggregate({"$group": {"_id" : "$iamap" }})	MySQL: Articles.select().group_by(Articles.iamap)
@app.route('/page/<int:pageid>')	
MongoDB: Articles.objects(pageid = pageid)	MySQL: Articles.select().where(Articles.pageid == pageid)

@app.route('/search_query')	

MongoDB:
 	
`result = db.command('text', 'articles',`
 	
`search = q , project = {`
 	
`"webpage": False, "created": False, "_id" : False})`
	MySQL:
	
	`statement = " select pageid,title`
	
	`from articles where match(title,webpage)`
	
	`against ('+%s'  IN BOOLEAN MODE) " % (q)`
	
	`cursor.execute("" + statement + "")`
	
	`result = cursor.fetchall()`

Table 5 : Side by side comparision of queries

3.2.4 Simple CMS-Mongo

MongoEngine has been used as an object relation mapper for connecting to MongoDB and developing the admin interface. Since full-text search was a new functionality in MongoDB at the time this study was conducted. PyMongo had to be used to implement the text search query. The comparison of the queries in Table 5 above shows that MongoDB ORMs are similar in concept and syntax to the object oriented programming languages. This simplifies the process of debugging an application. In order to enable full-text search html pages, the field is indexed, which in turn affects the write performance of the databases.

3.3 Data Migration to Simple CMS

Data migration from the legacy application will be described in detail in Chapter 5. For the purpose of benchmarking it was required to have two identical datasets in both applications. A migration module is written to copy the data from MySQL to MongoDB. The module migrates between MySQL and MongoDB, not from the legacy web application.

CHAPTER 4: BENCHMARKS FOR SIMPLE CMS

Speed is the ultimate user experience factor that distinguishes products amongst competitors. Interestingly, speed may be seen as a pure technical challenge with respect to user experience design. Over the years there has been an excessive huge increase in the speed of RAM, CPU, and bandwidth; yet, the application's communication with databases and file system has remained an issue. Web application architects are faced with the challenge of optimizing databases communications. In this chapter the benchmarking of the performance of both MySQL and MongoDB in Simple CMS will be explored.

4.1 Apache JMeter

Apache JMeter was used to test and measure the performance of the Simple CMS application. Apache JMeter is an open source project that is designed to test static and dynamic resources and may be used to test different load tests and massive concurrent loads. In this case, an environment was simulated with 1000 concurrent users sending the same "Get" request to the Simple CMS. Apache JMeter was able to perform this huge load test on a single machine, and the results of the performance test were displayed on graphs via the Apache's user interface.

4.2 Dataset Generation

The dataset that was used for this benchmarking consisted of ten different legacy web applications that were migrated to both versions of Simple CMS. These datasets were collections of HTML webpages with titles associated with them. In order to experience full-text search indexing in MySQL and MongoDB was ensured by using proper queries. The following table displays the statistics of each website that was migrated.

```
ALTER TABLE articles ADD FULLTEXT(title, webpage)
```

Figure 10 : MySQL index

```
db.articles.ensureIndex( {  title :'text' , webpage : 'text'})
```

Figure 11 : MongoDB index

Website Index	Number of HTML Pages	Size on MySQL			Size on MongoDB		
1.A1	958	Data	2602	Kb	Data	2841	Kb
		Index	1752	Kb	Index	4702	Kb
2.T2	248	Data	804.1	Kb	Data	891.9	Kb
		Index	488	Kb	Index	1245	Kb
3.D3	191	Data	534.1	Kb	Data	580	Kb
		Index	348	Kb	Index	950.1	Kb
4.G4	371	Data	972	Kb	Data	1061	Kb
		Index	639	Kb	Index	1152	Kb
5.M5	147	Data	406.2	Kb	Data	449.3	Kb
		Index	237	Kb	Index	646.7	Kb
6.MS6	122	Data	256.1	Kb	Data	283.1	Kb
		Index	174	Kb	Index	497	Kb
7.N7	365	Data	986.4	Kb	Data	1075	Kb
		Index	627	Kb	Index	1692	Kb
8.P8	175	Data	528.8	Kb	Data	577.3	Kb
		Index	346	Kb	Index	902.2	Kb
9.S9	185	Data	529.2	Kb	Data	576.9	Kb
		Index	348	Kb	Index	950.4	Kb
10.V10	111	Data	326.5	Kb	Data	357.6	Kb
		Index	417	Kb	Index	598.8	Kb

Table 6 : Dataset's statistics

It is clear that MongoDB data requires slightly more space than MySQL with respect to database storage, but in comparison, a MongoDB full-text index requires much more space than MySQL. Clearly, MySQL exceeds MongoDB in full-index text searching which may be problematic in some cases where full-text indexing is essential for the database. For instance, in studying the 1.A1 (Table 6) dataset, it is clear that MongoDB indexes requires 4702 Kb as compared to MySQL which requires 1702 Kb – nearly double the amount of space used for Indexing in MongoDB. Interestingly, the amount of storage for the content is not that different, as MongoDB requires 2841 Kb while MySQL requires 2602 Kb.

The number of indexes generated in MongoDB with MySQL was compared and the result was that on average, MongoDB indexes are 2.48 times higher than those of MySQL. Surprisingly, in 10.V10 the number of indexes is quite large in MySQL as compared to other datasets which shows that indexes are highly dependent upon the nature of the textual data.

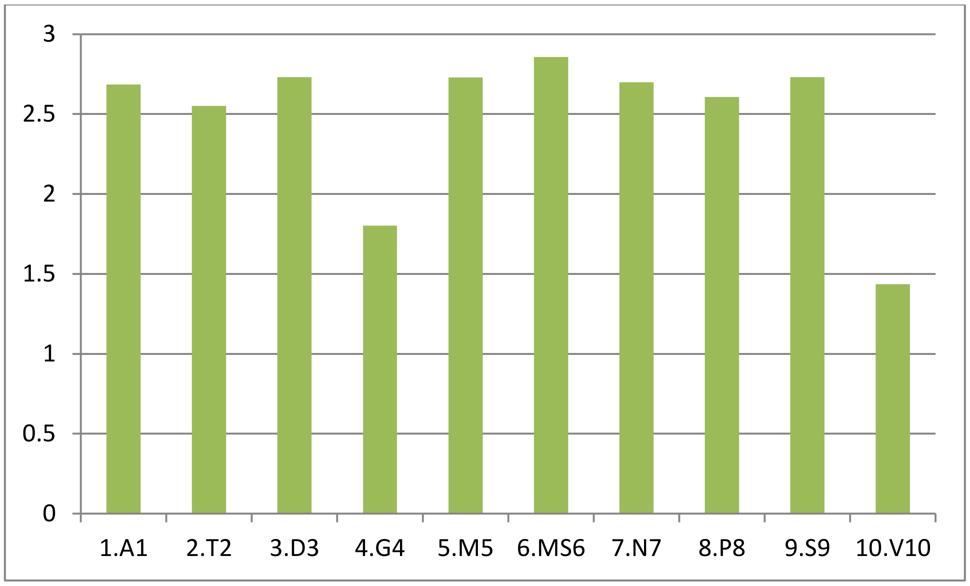


Figure 12 : Ratio of MongoDB/MySQL indexes

In studying the data it is clear that on average, the same data that is stored in MongoDB is nine percent higher than that of MySQL.

4.3 JMeter Get Request

In this section JMeter was used to simulate 1000 concurrent users sending requests to the Simple CMS. Two of the most frequently used requests were studied: “Get page” and “Search Query” which were explained in Table 4. JMeter’s output is visualized in a diagram that has six parameters: The *Black* line shows the latest sample from the experience, which is actually the maximum amount of time it required for the client to retrieve data from the CMS. The *Blue* line shows the average time of requests and the *Green* line represents the throughput of the system.

4.3.1 Get Query Benchmarks

In this experiment a get page request is sent to the 3.D3 (Table 6) dataset on both CMSes. MongoDB displays a better performance on the read query (See Figures 13 and 14).

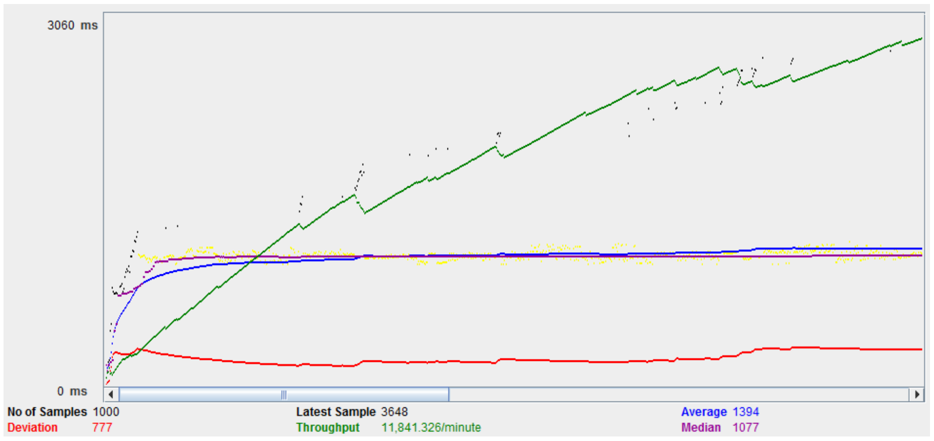


Figure 13 : Page 90 get request on 3.D3 MySQL

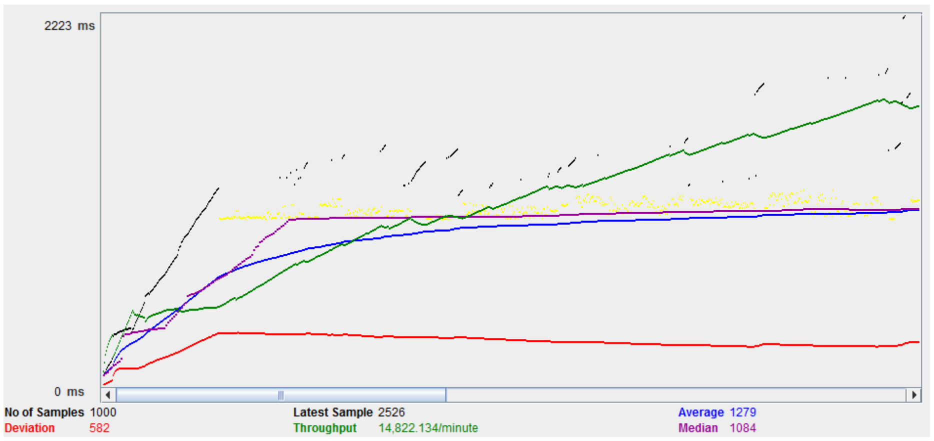


Figure 14 : Page 90 get request on 3.D3 MongoDB

In order to support the idea the same query was run for several pages in the 2.T2 (Table 6) dataset with the results shown below in Figure 15. MongoDB exceeded in the read performance from MySQL and had better throughput.

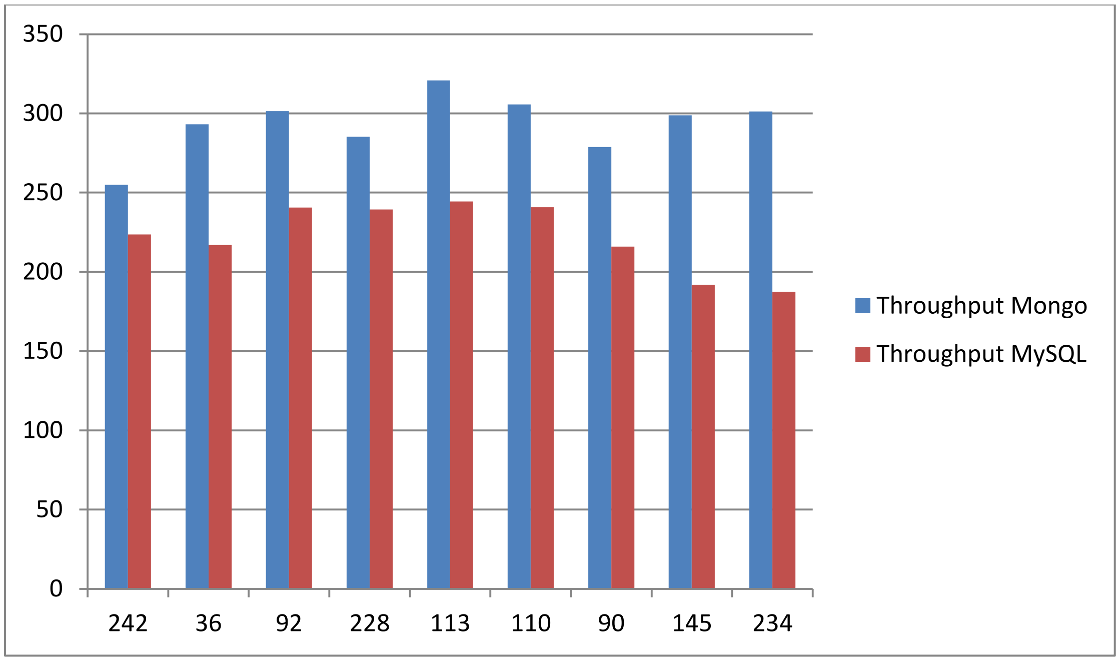


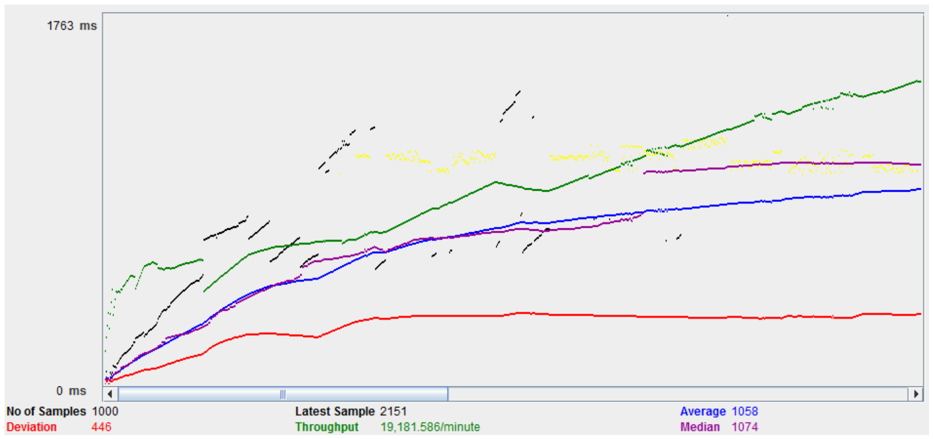
Figure 15 : Throughput for 2.T2 page requests

4.3.2 Get Search Query

To complete the experience a text search was explored in both databases. Searching the word “engineer” (Figure 16) in MySQL yielded three pages of results, while MongoDB retrieved thirteen pages for this search query. This clearly indicates a different approach to text searching in both databases. The performance of this query is displayed in Figures 17 and 18.

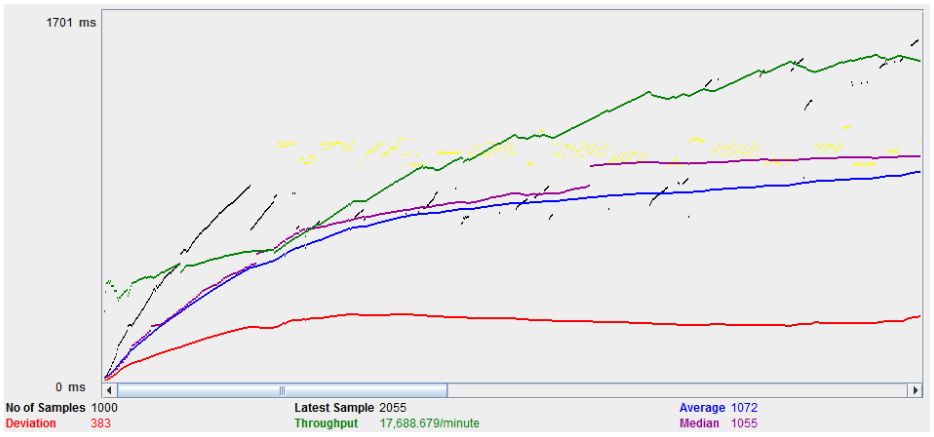
```
search_query/?query=engineer
```

Figure 16 URL for Search Query



No of Samples 1000
Deviation 446
Latest Sample 2151
Throughput 19,181.586/minute
Average 1058
Median 1074

Figure 17 : MySQL's search performance



No of Samples 1000
Deviation 383
Latest Sample 2055
Throughput 17,688.679/minute
Average 1072
Median 1055

Figure 18 : MongoDB's search performance

CHAPTER 5: MIGRATION OF LEGACY WEB APPLICATION

In this chapter the migration of legacy web applications to a Content Management System will be studied. In order to have a successful migration, the legacy data needs to be studied and proper data extraction methods that suit that data need to be incorporated. The process of importing data to CMS relies heavily upon the mapping decision of the data. Current migration processes utilize the capabilities of the business teams to achieve the proper mapping of data from the legacy application to the Content Management System. To understand this concept, an explanation follows about current migration processes, followed by the introduction of NoSQL into the process.

5.1 Legacy Migration Workflow

Migration of any legacy web application starts with identifying and categorizing legacy data and finding the proper representation of the data in its new incarnation. The migration team needs to study different HTML pages within the legacy web application and decide upon a destination for each HTML page or piece of data in the CMS. These activities may be broken down into the following steps (Figure 19).

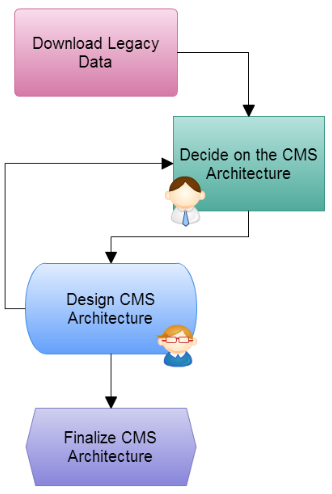


Figure 19 : Data mapping activities

As shown in Figure 19, the migration team, which includes both business and engineering team, needs to study the legacy data and decide which data should be migrated and where to import the migrated data. This activity has several risks associated with it, such as what if there is a type of content that is not being considered and is left behind or how should CMS architectural changes be incorporated into the migration process? In this method, once the CMS architecture has been finalized, applying changes to the CMS requires a migration within itself, doubling the amount of effort for the team.

5.1.1 Extracting Content

In order to obtain the latest data, HTTrack [28] was used. HTTrack is web crawler designed to extract the very latest data from the legacy website, and that data will then be stored on the local file system of a machine. Since the legacy application maintains content in plain HTML, the extraction and mining of data is often quite challenging. Researchers have devoted their efforts on using intelligent and dynamic methods in order to extract the useful information from web pages [1]. Finding and querying information from web pages are useful not only for web applications migration, but also for the collection of search engine data, data mining, and web scraping.

Different methodologies have been proposed to maximize the results of data extraction from HTML pages. Introduced methods generally are heuristic and many rely upon time consuming machine learning algorithms. A simpler form of data extraction occurs when the user is familiar with the structure of the HTML page and tagging that has been incorporated. Developers may extract data by parsing HTML code and facilitate HTML tags to obtain the necessary information [29]. During the migration of legacy web applications, it is vital to be certain that all of the data is captured and that none of that data is lost; therefore, it is more beneficial to rely upon on HTML parsing methods as opposed to heuristic models. In order to achieve a fast and clean parsing method, some unnecessary information was removed from the web pages.

Web pages that are created by proprietary software, such as Front Page, have different templates. Those pages usually contain collections of HTML tags with inline styling. It is clear that some of the abstract templates encountered during the process of migration have been displayed in Figure 20 below. Since the migration goal is to extract the content of the pages, unnecessary tags from the HTML were stripped with the use of a parser Java library named jsoup, which also was used for the removal of inline style tags. This had a direct impact upon the performance of specific tag selection of an HTML.

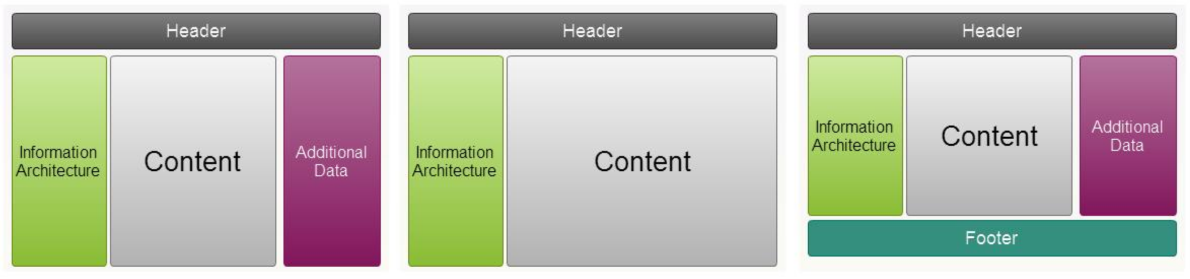
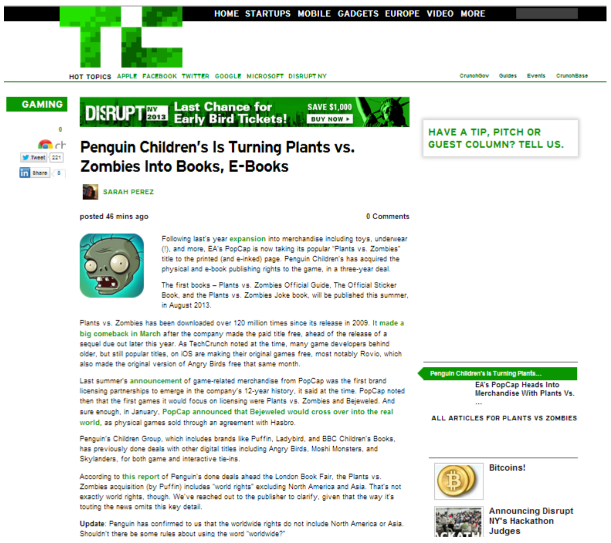


Figure 20 : Sample template pages from legacy application

In order to access a specific attribute from an HTML page, XPath or any other selector may be used. Initally, XPath was designed to access parts of an XML document [30]. XPath may be used to traverse through an HTML or XML document and to retrieve information from those documents. Google chrome or any other HTML editor tool may be used to extract XPath of a specific tag. Other available selectors are CSS selectors [31].

In this example, as shown in Figure 21, the content extraction practice for a sample web page is applied. A specific web page from a website is downloaded using HTTrack.


HOME STARTUPS MOBILE GADGETS EUROPE VIDEO MORE
HOT TOPICS APPLE FACEBOOK TWITTER GOOGLE MICROSOFT DISRUPT NY
CrunchGov Guides Events CrunchBase
GAMING
DISRUPT NY 2013 Last Chance for Early Bird Tickets! SAVE $1,000 BUY NOW
Tweet 221
Share 8
HAVE A TIP, PITCH OR GUEST COLUMN? TELL US.

Penguin Children's Is Turning Plants vs. Zombies Into Books, E-Books

SARAH PEREZ

posted 46 mins ago 0 Comments

Following last's year expansion into merchandise including toys, underwear (!), and more, EA's PopCap is now taking its popular "Plants vs. Zombies" title to the printed (and e-inked) page. Penguin Children's has acquired the physical and e-book publishing rights to the game, in a three-year deal.

The first books – Plants vs. Zombies Official Guide, The Official Sticker Book, and the Plants vs. Zombies Joke book, will be published this summer, in August 2013.

Plants vs. Zombies has been downloaded over 120 million times since its release in 2009. It made a big comeback in March after the company made the paid title free, ahead of the release of a sequel due out later this year. As TechCrunch noted at the time, many game developers behind older, but still popular titles, on iOS are making their original games free, most notably Rovio, which also made the original version of Angry Birds free that same month.

Last summer's announcement of game-related merchandise from PopCap was the first brand licensing partnerships to emerge in the company's 12-year history, it said at the time. PopCap noted then that the first games it would focus on licensing were Plants vs. Zombies and Bejeweled. And sure enough, in January, PopCap announced that Bejeweled would cross over into the real world, as physical games sold through an agreement with Hasbro.

Penguin's Children Group, which includes brands like Puffin, Ladybird, and BBC Children's Books, has previously done deals with other digital titles including Angry Birds, Moshi Monsters, and Skylanders, for both game and interactive tie-ins.

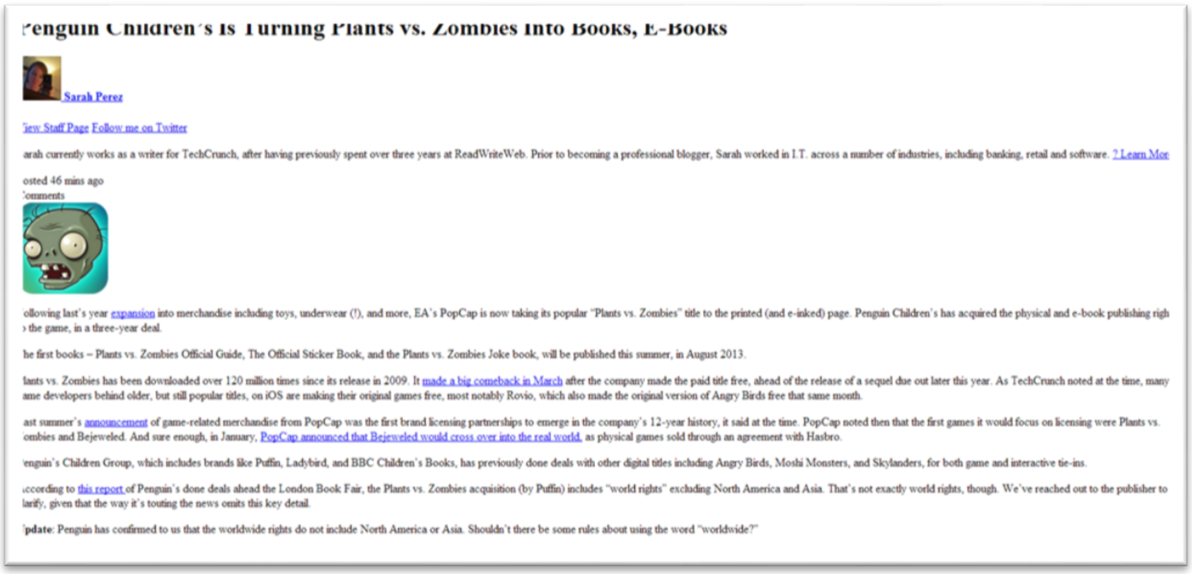
According to this report of Penguin's done deals ahead the London Book Fair, the Plants vs. Zombies acquisition (by Puffin) includes "world rights" excluding North America and Asia. That's not exactly world rights, though. We've reached out to the publisher to clarify, given that the way it's touting the news omits this key detail.

Update: Penguin has confirmed to us that the worldwide rights do not include North America or Asia. Shouldn't there be some rules about using the word "worldwide?"

Penguin Children's Is Turning Plants...
EA's PopCap Heads Into Merchandise With Plants Vs. ...
ALL ARTICLES FOR PLANTS VS ZOMBIES
Bitcoins!
Announcing Disrupt NY's Hackathon Judges

Figure 21 : HTTrack result for a sample webpage

In order to extract the content from this webpage, the HTML page was parsed and the "id" of the article was used to extract the content (Figure 22).



'enguin Children's Is Turning Plants vs. Zombies Into Books, E-Books

Sarah Perez

'iew Staff Page Follow me on Twitter

arah currently works as a writer for TechCrunch, after having previously spent over three years at ReadWriteWeb. Prior to becoming a professional blogger, Sarah worked in I.T. across a number of industries, including banking, retail and software. ? Learn Mor

osted 46 mins ago
'omments

ollowing last's year expansion into merchandise including toys, underwear (!), and more, EA's PopCap is now taking its popular "Plants vs. Zombies" title to the printed (and e-inked) page. Penguin Children's has acquired the physical and e-book publishing righ
› the game, in a three-year deal.

he first books – Plants vs. Zombies Official Guide, The Official Sticker Book, and the Plants vs. Zombies Joke book, will be published this summer, in August 2013.

lants vs. Zombies has been downloaded over 120 million times since its release in 2009. It made a big comeback in March after the company made the paid title free, ahead of the release of a sequel due out later this year. As TechCrunch noted at the time, many
ame developers behind older, but still popular titles, on iOS are making their original games free, most notably Rovio, which also made the original version of Angry Birds free that same month.

ast summer's announcement of game-related merchandise from PopCap was the first brand licensing partnerships to emerge in the company's 12-year history, it said at the time. PopCap noted then that the first games it would focus on licensing were Plants vs.
ombies and Bejeweled. And sure enough, in January, PopCap announced that Bejeweled would cross over into the real world, as physical games sold through an agreement with Hasbro.

'enguin's Children Group, which includes brands like Puffin, Ladybird, and BBC Children's Books, has previously done deals with other digital titles including Angry Birds, Moshi Monsters, and Skylanders, for both game and interactive tie-ins.

ccording to this report of Penguin's done deals ahead the London Book Fair, the Plants vs. Zombies acquisition (by Puffin) includes "world rights" excluding North America and Asia. That's not exactly world rights, though. We've reached out to the publisher to
larify, given that the way it's touting the news omits this key detail.

'**pdate**: Penguin has confirmed to us that the worldwide rights do not include North America or Asia. Shouldn't there be some rules about using the word "worldwide?"

Figure 22 : Extracted content

This simple method may be applied to different pages with different methods.

5.1.2 Importing Data to Database

In order to perform migration on an entire website, the output of HTTrack will be imported to the database using a script which is referred to as “migration script.” MySQL forces the migration process to design databases with predefined tables. Figure 23 shows the process of migration after deciding on page destinations. In an abstract model, it is clear that applying changes to the data model is almost inevitable, especially in migrations with huge amount of data.

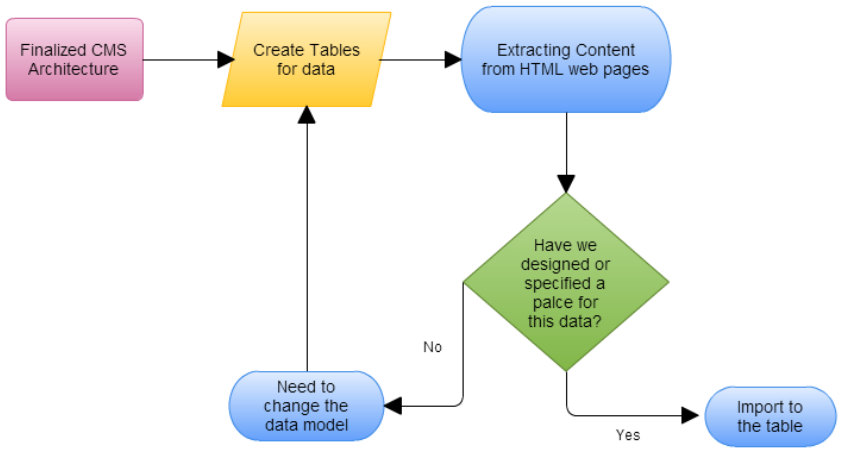

Finalized CMS Architecture
Create Tables for data
Extracting Content from HTML web pages
Have we designed or specified a palce for this data?
No
Need to change the data model
Yes
Import to the table

Figure 23 : Traditional migration process

5.2 MongoDB Migration

MongoDB’s migration practice is an iterative approach to migrating data, and it will break the migration into two main steps: 1) Importing data to a MongoDB collection and storing it into a single collection; and 2) Querying the mentioned collection to categorize data and to apply abstraction data layers.

5.2.1 MongoDB’s Migration Concept

Since there is no data modeling practice for MongoDB, developers are able store several attributes and metadata per specific pages within the database. In these process actual keys of the data, which are the same as those attributed to the data in the MySQL are stored with their values. In a collection of data in MongoDB, it is possible to store several pages with both the same or different Key, making the migration process dynamic and agile.

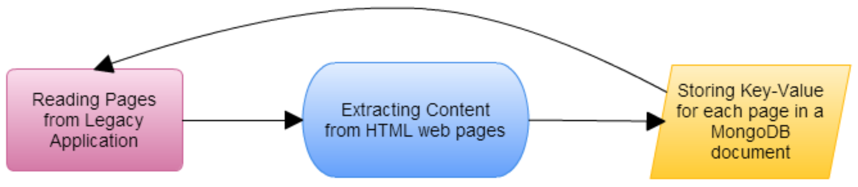

Reading Pages from Legacy Application
Extracting Content from HTML web pages
Storing Key-Value for each page in a MongoDB document

Figure 24 : MongoDB's migration steps

The result of this migration is a dynamic database that has all of the content necessary to be migrated within a single collection (Figure 24). The fact that database documents are different than one another can be seen as a performance challenge for the CMS. Some CMSes are able to work with unstructured content, but in order to complete the migration and export this data and import it into a SQL CMS, the aggregation process must be introduced.

5.2.2 Data Aggregation

As was explained in the previous section, data is stored in a MongoDB collection. In order to group similar documents and import them to their documents or tables the following feature of MongoDB was used, enabling the access to specific attributes and the ability to mine the data.

Syntax: `{ field: { $exists: <boolean> } }`

Figure 25 : Finding documents that have a specific field

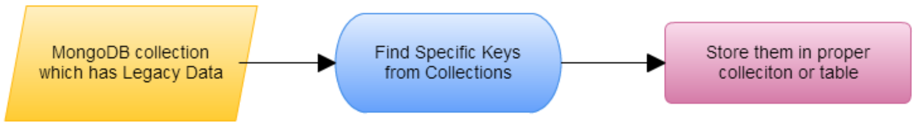

MongoDB collection which has Legacy Data
Find Specific Keys from Collections
Store them in proper colleciton or table

Figure 26 : MongoDB's export to CMS steps

Exporting to CMS and categorizing data is enabled by using MongoDB's "$exist" syntax. This syntax retrieves specific data from a collection (Figures 25 and 26).

5.3 Comparison of Two Methods

In order to compare both methods, complexity equations for both solutions were introduced. It is assumed that there are N numbers of HTML pages that need to be migrated from the legacy web application. "S" was specified as the pool of unknown page types. Members of S may be seen as tables in MySQL or collections in MongoDB. All of the pages from the legacy web application will map to one and only one category inside S. $P(i)$ shows the probability of knowing the data type and metadata for the page "i" within the legacy application. "S_{Tables}" is the process for making the migration tables. "S_1" refers to the effort of changing or creating one table or collection.

The amount of effort to migrate each page is shown as "M_i." The migration effort with "E" which is a parameter that determines the complexity of the migration is shown. For traditional migrations the complexity is:

$$E_{Traditional} = S_{Tables} + \sum_{i=0}^{n} (M_i + ((1 - P(i)) * S_1))$$

The reason that low "$P(i)$" has a negative effect is when there are pages in which their metadata or a table for them to map to has not yet been created; the tables that have not been created before have to be created or if they have been created they will need to be changed.

Now the complexity for the approach to the migration needs to be studied. With document-oriented databases, not knowing the structure of the data will not have any negative effect on the migration. As previously mentioned and shown in the equation, the migration starts by migrating pages to the databases (the first part of the equation) and then concludes with aggregating data on separate tables or collections.

$$E_{New\ Migration} = \sum_{i=0}^{n} (M_i) + S_{Tables}$$

Comparing these equations, it is clear that even if every detail is known about the legacy data using document-oriented databases, it will not add any additional benefit to the migration. On the other hand, if the structure of the data is not known, then it is strongly advised that the new migration method to be used for the migration.

CHAPTER 6: CONCLUSION

The establishment of MySQL and other RDBMSes as the database backend relies heavily upon the use cases of financial and enterprise applications. The requirements of RDBMSes are tailored for ACID transaction and the support of a defined schema. The limits imposed by RDBMSes cause them to be an inferior technology for the data migration practice, especially from proprietary applications. In this research study, a new approach was proposed for migrating data from legacy web applications to Content Management Systems (CMSes).

Through the creation of two similar CMSes, the performance characteristics of MySQL and MongoDB were explored. MongoDB displayed an ideal performance for querying the data but still had some shortcomings in the indexing. Another important factor in the consideration of a NoSQL database is the ease of use. MongoDB's syntax is intuitive in comparison to the structured query language employed by MySQL. In addition, MongoDB in a CMS space stores content in a format similar to that of JSON. JSON appears to be the new industry standard and is being used in most modern web applications. Applications that employ MySQL require the conversion of the MySQL's results to the JSON format by using additional libraries.

Both MongoDB and MySQL have a great community of supporters that provide third party ORMs for the databases. Yet, since MongoDB is a new database some of the features and third party tools may not be as mature as their MySQL counterparts. In this study we faced with the issue of, MongoDB's full-text search was not being supported in MongoEngine. Additionally, it is time consuming for developers to implement the new native features of the databases, but this is not a huge issue with MongoDB as its command control interface is written in JavaScript.

From the migration’s perspective, current migration methods require that a business team and developers are to be knowledgeable about the details of the legacy web application, which is unfeasible as legacy web applications have undergone several iterations over the years with numerous developers and are unstructured. This premature assumption about the structure of data leads to a less accurate migration and loss of information.

In this study, a new approach to the migration of legacy web applications was introduced through the exploration and questioning of the various restrictions imposed by RDBMSes. In this proposed migration process, one collection of data was used to import the HTML content for the web pages with their proper “fields.” Therefore, if an HTML page consists of news, a MongoDB document with the Key called “news” may be created, along with other pages with different Keys. Then, those Keys are aggregated and their proper structures are determined. Another advantage in using MongoDB is that it enables developers to add metadata to a specific document if necessary. In comparing both migrations, it was discovered that document-oriented databases are significantly a better fit for migrating HTML pages with no specific structure.

REFERENCES

[1] J. C. P. C. R Rodrigue-Echeverria, "Modernization of Legacy Web Applications into Rich Internet Applications," *7th Model-Driven Web Engineering Workshop (MDWE'2011)*, 2011.

[2] S. A. Walberg, "Tuning LAMP systems, Part 1: Understanding the LAMP architecture," 2007. [Online]. Available: http://www.ibm.com/developerworks/linux/library/l-tune-lamp-1/.

[3] H.M.Sheth, Scraper- a program to instantly convert a static website to a dynamic website, MI, 2010.

[4] C. Saathoff, "Lazy Migration in MongoDB with Scala Salat," 18 March 2013. [Online]. Available: http://kodemaniak.de/2013/03/lazy-migrations-in-mongodb-with-scala-salat/.

[5] S. A. A. S. C. Tauro, "Comparative study of the new generation, agile, scalable, high performance NoSQL databases," *International Journal of Computer Arpplications (0975-888)*, vol. 48, no. 20, 2012.

[6] P. Nasholm, "Extracting data from NoSQL databases," University of Gothenburg, Gothunburg, 2012.

[7] 10gen, "MongoDB," 10gen, 1 March 2013. [Online]. Available: www.mongodb.org.

[8] 10gen, "10gen," 1 March 2013. [Online]. Available: ww.10gen.com.

[9] D. Goans and G. Leach, "Developing and re-imagining library web," November 2005. [Online].

[10] G. Yunhua, S. Shu and Z. Guansheng, "Application of NoSQL Databases in Web Crawling," *International Journal of Digital Content Technology and its Applications,* vol. 5, no. 6, 2011.

[11] Z. Wei-ping, L. Ming-xin and C. Haun, "Using MongoDB to Implement Textbook Management System instead of MySQL," *IEEE,* 2011.

[12] D. Chamberlin, "Early History of SQL," *Annals of History of Computing,* vol. 34, no. 4, pp. 78-82, 2012.

[13] IBM, "CICS Transaction Server for z/OS," [Online]. Available: http://publib.boulder.ibm.com/infocenter/cicsts/v3r2/index.jsp?topic=%2Fcom.ibm.cics.ts.productoverview.doc%2Fconcepts%2Facid.html. [Accessed 1 March 2013].

[14] S. N. R. Elmasri, Foundamentals of Database Systems, Addison-Wesley , 2011.

[15] N. Leavitt, "Will NoSQL databases Live up to Their Promise?," *IEEE*, pp. 12-14, 2010.

[16] J. Scholz, "Coping with Dynamic, Unstructured Data Sets – NoSQL a Buzzword or a Savior?," *REAL CORP*, pp. 27-29, 2011.

[17] C. Strauch, "NoSQL Databases," Ultra-Large Scale Sites.

[18] Apache, "Cassandra," [Online]. Available: http://cassandra.apache.org/. [Accessed 1 March 2013].

[19] J. Han, H. Haihong, G. Le and J. Du, "Survey on NoSQL Databases," *IEEE*, pp. 363-366, 2011.

[20] "BSON," [Online]. Available: http://bsonspec.org/. [Accessed 1 March 2013].

[21] T. Rueckstiess, "Building Your First App with MongoDB," 10gen, 2013.

[22] M. Pirtle, "Content Management Systems and MongoDB," 2010.

[23] "Drupal," [Online]. Available: www.drupal.org. [Accessed 3 2013].

[24] "Codex," [Online]. Available: http://codex.wordpress.org/File:WP_27_dbsERD.png#file. [Accessed 15 3 2013].

[25] "Werkzeug," [Online]. Available: http://werkzeug.pocoo.org/. [Accessed 20 3 2013].

[26] "Flask," [Online]. Available: http://flask.pocoo.org/docs/foreword/#what-does-micro-mean. [Accessed 20 3 2013].

[27] "MemSQL Workload," [Online]. Available: https://github.com/memsql/workload-simulator. [Accessed 14 3 2013].

[28] "Virtualenv," [Online]. Available: http://jontourage.com/2011/02/09/virtualenv-pip-basics/. [Accessed 20 3 2013].

[29] "Jsoup," [Online]. Available: http://jsoup.org/. [Accessed 5 2 2013].

[30] "W3C-XPath," [Online]. Available: http://www.w3.org/TR/xpath/. [Accessed 12 3 2013].

[31] "CSS Selector," [Online]. Available: http://www.w3.org/TR/css3-selectors/. [Accessed 10 3 2013].

[32] "HTTrack," [Online]. Available: http://www.httrack.com/. [Accessed 4 3 2013].