

Regis University

## ePublications at Regis University

---

Regis University Student Publications  
(comprehensive collection)

Regis University Student Publications

---

Summer 2012

### Transitioning From Relational to Nosql: a Case Study

John McPhillips  
*Regis University*

Follow this and additional works at: <https://epublications.regis.edu/theses>



Part of the [Computer Sciences Commons](#)

---

#### Recommended Citation

McPhillips, John, "Transitioning From Relational to Nosql: a Case Study" (2012). *Regis University Student Publications (comprehensive collection)*. 224.  
<https://epublications.regis.edu/theses/224>

This Thesis - Open Access is brought to you for free and open access by the Regis University Student Publications at ePublications at Regis University. It has been accepted for inclusion in Regis University Student Publications (comprehensive collection) by an authorized administrator of ePublications at Regis University. For more information, please contact [epublications@regis.edu](mailto:epublications@regis.edu).

**Regis University**  
College for Professional Studies Graduate Programs  
**Final Project/Thesis**

# **Disclaimer**

Use of the materials available in the Regis University Thesis Collection ("Collection") is limited and restricted to those users who agree to comply with the following terms of use. Regis University reserves the right to deny access to the Collection to any person who violates these terms of use or who seeks to or does alter, avoid or supersede the functional conditions, restrictions and limitations of the Collection.

The site may be used only for lawful purposes. The user is solely responsible for knowing and adhering to any and all applicable laws, rules, and regulations relating or pertaining to use of the Collection.

All content in this Collection is owned by and subject to the exclusive control of Regis University and the authors of the materials. It is available only for research purposes and may not be used in violation of copyright laws or for unlawful purposes. The materials may not be downloaded in whole or in part without permission of the copyright holder or as otherwise authorized in the "fair use" standards of the U.S. copyright laws and regulations.

# TRANSITIONING FROM RELATIONAL TO NOSQL: A CASE STUDY

A THESIS,

SUBMITTED ON 24 OF AUGUST, 2012

TO THE DEPARTMENT OF INFORMATION SYSTEMS,

OF THE SCHOOL OF COMPUTER & INFORMATION SCIENCES

OF REGIS UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS OF MASTER OF  
SCIENCE IN SOFTWARE ENGINEERING AND DATABASE TECHNOLOGIES

BY



---

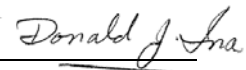
John McPhillips

APPROVALS



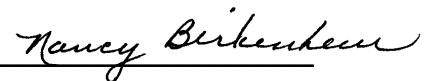
---

Darl Kuhn, Thesis Advisor



---

Donald J. Ina - Faculty of Record



---

Nancy Birkenheuer, Program Coordinator

### **Abstract**

Data storage requirements have increased dramatically in recent years due to the explosion in data volumes brought about by the Web 2.0 era. Changing priorities for database system requirements has seen NoSQL databases emerge as an alternative to relational database systems that have dominated this market for over 40 years. Web-enabled, always on applications mean availability of the database system is critically important as any downtime can translate in to unrecoverable financial loss. Cost is also hugely important in this era where credit is difficult to obtain and organizations look to get the maximum from their IT infrastructure from the least amount of investment. The purpose of this study is to evaluate the current NoSQL market and assess its suitability as an alternative to a relational database. The research will look at a case study of a bulletin board application that uses a relational database for data storage and evaluate how such an application can be converted to using a NoSQL database. This case study will also be used to assess the performance attributes of a NoSQL database when implemented on a low cost hardware platform. The findings will provide insight to those who are considering making the switch from a relational database system to a NoSQL database system.

### **Acknowledgements**

To Majka and Oisín, thank you for your love, support, encouragement, understanding and patience during the last two years.

To my parents, for always encouraging me to better myself and for your never ending support and assistance.

To my advisor Darl Kuhn, thank you for your time and the incredible advice and support you have given during this process.

To my employer, thank you for providing the financial assistance to complete this course.

To the staff at NUIG and Regis University, thank you for providing this tremendous learning experience.

## Table of Contents

### Table of Contents

<b>Abstract.....</b>	<b>ii</b>
<b>Acknowledgements .....</b>	<b>iii</b>
<b>Table of Contents .....</b>	<b>iv</b>
<b>List of Figures (If Applicable).....</b>	<b>viii</b>
<b>List of Tables (If Applicable) .....</b>	<b>x</b>
<b>Chapter 1 – Introduction .....</b>	<b>1</b>
Thesis Statement .....	1
Preface.....	1
Research Methodology.....	3
Thesis Scope.....	4
Success Criteria .....	4
Chapter Summary.....	5
<b>Chapter 2 – Literature Review .....</b>	<b>6</b>
Introduction .....	6
Classification of NoSQL Systems.....	6
Characteristics of NoSQL systems.....	9

Use Cases in Industry .....	11
Facebook.....	11
Nokia .....	11
Danish Department of Health.....	12
Guardian.co.uk.....	12
Scalability Concepts .....	13
The Cost Advantage of Scaling with NoSQL .....	19
NoSQL Performance Characteristics .....	20
Yahoo! Cloud Serving Benchmark .....	21
Challenges for NoSQL Adoption.....	26
Vendor Support.....	27
Data Querying.....	27
Immaturity of the technology .....	29
<b>Chapter 3 - Methodology.....</b>	<b>31</b>
Introduction .....	31
Use Case.....	31
Application Details.....	33
Database Description.....	35
Application Description .....	37

Choosing a NoSQL Database.....	40
MongoDB Comparison to Relational Database .....	41
Designing the Schema .....	43
Data Types in MongoDB.....	46
Test Data .....	47
Test data format. ....	47
Mapping the test data to the schema.....	49
Loading the Test Data.....	51
Deployment .....	52
Amazon EC2 Instances.....	52
Web Server Setup .....	53
MongoDB Setup .....	57
Chapter Summary.....	60
<b>Chapter 4 – Analysis.....</b>	<b>61</b>
Converting the PHP Code .....	61
Converting read queries.....	62
Converting the insert/update/delete queries. ....	71
Performance and Optimization.....	77
Hardware setup.....	77



Method used for measurement taking .....	77
Comparison of MySQL and MongoDB write performance .....	78
Comparison of MySQL and MongoDB read performance. ....	80
Using Map-Reduce for Aggregation. ....	87
Chapter Summary.....	92
<b>Chapter 5 – Conclusions.....</b>	<b>93</b>
Research Findings .....	93
Lessons Learned .....	95
Summary of Contributions .....	96
Future Research.....	96
<b>References .....</b>	<b>97</b>
<b>Appendix A .....</b>	<b>102</b>
<b>Appendix B .....</b>	<b>111</b>
<b>Appendix C .....</b>	<b>124</b>
<b>Appendix D .....</b>	<b>132</b>

### List of Figures (If Applicable)

Figure 1. System cost vs. No of users of RDBMS (Couchbase, 2012) .....	14
Figure 2. Cost effectiveness of horizontal scaling (Couchbase, 2012).....	15
Figure 3. ACID vs. BASE (Brewer, 2000) .....	16
Figure 4. Characterisation of DBMS applications (Stonebraker & Cattell, 2011) .....	20
Figure 5. YCSB Client Architecture (Cooper et al., 2010).....	23
Figure 6. YCSB Workloads (Cooper et al., 2010).....	24
Figure 7. Workload A results - (a) is read operations and (b) is update operations (Cooper et al., 2010).....	24
Figure 8. Workload B results - (a) read operations, (b) update operations (Cooper et al., 2010)	25
Figure 9. Scalability test (Cooper et al., 2010) .....	25
Figure 10. Elasticity results (Cooper et al., 2010) .....	26
Figure 11. Bulletin Board Relational Schema .....	35
Figure 12. Comparison between MyISAM and InnoDB Storage Engines.....	36
Figure 13. Launching EC2 Instance from AWS Marketplace.....	54
Figure 14. LAMP Instance running in EC2 .....	55
Figure 15. Accessing the LAMP Management Interface.....	56
Figure 16. Defining a hostname.....	56
Figure 17. Connecting through SSH.....	58
Figure 18. Security Group for MongoDB.....	59
Figure 19. MongoDB Web Interface .....	59
Figure 20. Converted PHP code for the getForum() function .....	64

Figure 21. Code for index.php query .....	66
Figure 22. Converted PHP code for viewforum.php .....	70
Figure 23. Converted PHP code for transact-post.php insert function .....	73
Figure 24. Converted PHP code for transact-post.php update function .....	75
Figure 25. Converted PHP code for transact-post.php delete function.....	76
Figure 26. PHP code to measure MySQL query time .....	78
Figure 27. Comparison of import times for users.xml files .....	79
Figure 28. Comparison of import times for posts.xml files .....	80
Figure 29. Create view statement.....	81
Figure 30. Revised SQL query for viewforum.php .....	82
Figure 31. Results of Test No. 1 .....	84
Figure 32. Results of Test No. 2 .....	84
Figure 33. Results of Test No. 3 .....	85
Figure 34. Results of Test No. 4 (Using the MyISAM engine) .....	87
Figure 35. Map function from viewforum.php .....	88
Figure 36. Reduce function from viewforum.php .....	89
Figure 37. Executing the Map-Reduce command.....	89
Figure 38. viewforum.php using Map-Reduce .....	90
Figure 39. Comparison of query times using Map-Reduce .....	91

### List of Tables (If Applicable)

Table 1. Comparison between MongoDB and RDBMS.....	43
Table 2. Mapping Tables to Collections.....	<b>Error! Bookmark not defined.</b>
Table 3. MongoDB users Schema .....	<b>Error! Bookmark not defined.</b>
Table 4. MongoDB posts Schema .....	46
Table 5. MongoDB forum Schema.....	<b>Error! Bookmark not defined.</b>
Table 6. MongoDB forum_admin Schema.....	<b>Error! Bookmark not defined.</b>
Table 7. users.xml format .....	48
Table 8. posts.xml format .....	<b>Error! Bookmark not defined.</b>
Table 9. Relational Schema Mapping: forum_users.....	<b>Error! Bookmark not defined.</b>
Table 10. Relational Schema Mapping: forum_posts.....	<b>Error! Bookmark not defined.</b>
Table 11. NoSQL Schema Mapping: users.....	51
Table 12. NoSQL Schema Mapping: posts.....	51
Table 13. SQL Analysis for functions.php .....	63
Table 14. Comparison of PHP code for the getForum function .....	64
Table 15. SQL Analysis for index.php .....	65
Table 16. SQL Analysis for viewforum.php.....	68
Table 17. SQL Analysis of transact-post.php insert function.....	72
Table 18. SQL Analysis for transact-post.php update function.....	74
Table 19. SQL Analysis of transact-post.php delete function .....	76
Table 20. Test specifications.....	83
Table 21. MongoDB read performance results summary .....	86

## Chapter 1 – Introduction

### Thesis Statement

*Web 2.0 applications are turning to NoSQL databases as a more scalable solution than a relational database for data storage. Transitioning an application from a relational environment to NoSQL environment presents many challenges in terms of schema design and data access methods.*

### Preface

Information explosion is a term that first appeared in the 1960's in reference to the increasing amounts of electronic data that was starting to appear at that time. It would have been impossible to envisage, however, how big that explosion would become once the Internet became mainstream over 40 years later. The amount of data that exists today is growing at an alarming rate year on year and this has put an increasing amount of pressure on the database systems that are tasked with storing and managing this data. In 2008, the number of devices connected to the Internet exceeded the population of the planet (Evans, 2011). And this number has continued to grow to the point where it is expected to be at 50 billion devices by 2020.

The vast majority of these devices produce data of some shape or form that needs to be persisted, communicated or otherwise processed in some way. We use our personal computers to create documents, write email, download music; smart phones are used to take pictures which are posted to social networking sites along with other user generated content. High speed mobile networks have enabled us to stay connected outside of the home or office, and once we are connected, we are generating data.

This unprecedented increase in data has put new demands on database systems. The more data contained in a database, the more difficult it becomes to manage this data and, more

importantly to retrieve this data so it can be used. The relational model has been the mainstay of data storage systems since its inception by Edgar F. Codd in 1969. Commercially, vendors such as IBM, Oracle and Microsoft have had tremendous success with their database products which are all based on the relational model. However, as successful as this model has been, recently, companies have found that when the volume of data reaches a certain level, maintaining a relational database becomes an increasingly difficult task. There are of course methods of implementing a relational database that can handle even the largest volumes of data that need to be stored, but the cost of implementing such systems is out of reach of many organizations.

This has created somewhat of a gap in the data storage market for a system that is capable of handling large volumes of data but does not carry a cost beyond the boundaries of a company's budget. NoSQL databases have emerged as the leading contender to fill this gap. These systems are designed to handle huge volumes of data through a scale-out model that allows the database to be spanned over hundreds, or even thousands, of low cost commodity servers. This distributed model is designed to be highly fault tolerant as each piece of data in the database is replicated a number of times on other participating nodes in the cluster. Although relational database systems also support distributed architectures, the difficulty and cost of implementing this with the relational model has been a barrier for many. NoSQL databases aim to make this type of architecture more accessible by automating many of the difficult aspects of data distribution. Features such as automatic data sharding and replication abstract much of the complexity of distribution, allowing an organisation to easily expand their data storage system. On top of all this, NoSQL databases provide a flexible schema for data storage, as opposed to the rigid schema required in the relational model. This makes them more suitable for unstructured

data and changing requirements which has become increasingly important when designing databases systems.

These benefits have seen an increasing uptake in NoSQL technology in recent years. Web industry giants such as Google, Facebook and Twitter have led the way in terms of adopting this technology and this has given confidence to others to move in the same direction. But what if an organisation is heavily invested in the relational world? Making the change to a new technology may seem like a daunting prospect for many, particularly as such a high value is placed on an organisations information. The goal of this study is to show that moving from a relational environment to a NoSQL environment can be achieved if the right approach is taken. Schema translation from relational to NoSQL is a key aspect of making the transition. Normalisation of data is the order of the day when designing relational databases, but a different thought process is required for NoSQL databases. Another important aspect is data retrieval. Although there is no standard query language for NoSQL, most SQL can be converted to equivalent commands in whatever language the NoSQL database uses for data access. This thesis will evaluate both schema design transfer and SQL to NoSQL translation.

## **Research Methodology**

The research conducted for this thesis will require both qualitative and quantitative research methodologies. I intend to research the usage of NoSQL databases in industry, in particular the reasons organisations decided to utilize NoSQL and the steps taken to come to this conclusion. A content analysis will be conducted on literature pertaining to the usage of NoSQL in industry to identify patterns in the thought process undertaken by an organisation deciding to

use a NoSQL database. Customer service in the NoSQL sector will also be examined to determine if it is comparable to the established relational database vendors.

To evaluate the process of transitioning from a relational database to a NoSQL database, a case study will be undertaken that takes an existing application based on a relational database and converts the application code to use a NoSQL database instead. This case study will also be used to assess some of the performance characteristics of the NoSQL database in comparison to the relational database.

### **Thesis Scope**

A definition of a NoSQL database will be established in Chapter 2 and the research will be confined to those databases classed as NoSQL. For reference, the relational database used in the case study will be MySQL and the NoSQL database will be restricted to a single vendor. Other aspects such as backup & recovery, performance tuning tools, high availability tools, replication, and the robustness of the query language will not be considered.

### **Success Criteria**

The following list makes up the success criteria for this study:

- The reasons for transitioning to a NoSQL database have been established through research;
- An analysis of the support provided by NoSQL vendors has been undertaken;
- A case study has been undertaken to convert an application from using a MySQL database to a NoSQL database;



- An analysis of the performance of the NoSQL database compared to the MySQL database has been completed.

## **Chapter Summary**

NoSQL databases are an emerging technology that provides an alternative to the more established relational databases that have traditionally dominated the data storage market. They aim to provide a more scalable and highly available database system with less cost and overhead, while sacrificing certain characteristics such as consistency and a simple query language.

The remainder of this thesis is structured as follows:

- Chapter 2 provides a review of the literature pertaining to the use of NoSQL databases in industry, the different classifications of NoSQL databases, performance characteristics of NoSQL database and the challenges they face in order to achieve wide spread adoption.
- Chapter 3 describes the methodology used to transfer an existing relational database in to a NoSQL environment.
- Chapter 4 provides an analysis of converting existing database access code from relational to NoSQL in a typical application and a comparative analysis of the performance differences between the existing application and the converted application.
- Chapter 5 summarises the findings of the research and details some further research areas to be explored.

## **Chapter 2 – Literature Review**

### **Introduction**

This literature review will research the current trends in the large data store industry as it relates to NoSQL databases and scalable relational databases. NoSQL databases are a recent addition to this industry and as such, the various vendors are constantly altering their offerings to provide continuous improvements to the underlying system. This creates somewhat of a moving target in terms of the correctness of the literature available. Articles and papers that make statements about a particular NoSQL system can quickly become outdated by revisions to that system. Bearing this in mind, this literature review will attempt to draw information from sources that are as recent as possible.

The purpose of the literature review is to gather and analyse information relating to NoSQL databases and scalable relational databases to support the decisions made in the practical element of the thesis. The first part will deal with NoSQL systems currently available and the classification of these systems. The next section will look at how NoSQL systems are currently being used in industry. This will be followed by a discussion of how NoSQL systems support scalability in comparison to scalable relational databases, the performance characteristics of NoSQL systems and the challenges facing NoSQL in terms of widespread adoption in the industry.

### **Classification of NoSQL Systems**

The NoSQL movement came about largely because of the increasing data storage needs of the Web 2.0 industry. The big players in this industry became frustrated with the difficulties of building distributed storage architectures based on traditional RDBMS systems. Because of this,

web application developers took matters in to their own hands and developed their own database technologies (Couchbase, 2012). Google and Amazon are two such companies that are now seen as pioneers of the NoSQL movement (Cattell, 2010; Couchbase, 2012; Neo Technology, 2011). Google's BigTable showed that it was possible to store simple data on a system that could scale to hundreds or thousands of nodes (Cattell, 2010). Amazon's Dynamo database pioneered the idea of sacrificing strong consistency in favour of high availability; data was not guaranteed to be up-to-date on every node but updates would be applied to each node eventually (Cattell, 2011).

Since then, there have been a many developments in the NoSQL industry with many more vendors now providing NoSQL systems. Of the systems currently available, most will fall in to one of three categories (Cattell, 2010):

- **Key-value stores:** All data is stored as a simple key-value index. The key is used to identify a value that is typically stored as a BLOB, but can contain other data types such as strings or pointers. These systems can be equated to a distributed index that was popularised by the memcached open source cache system which took advantage of the increasing availability of main memory to store in-memory indexes (Cattell, 2011). These systems are highly efficient, they can scale to a high number of nodes, but provide a very simple data model (Pokorny, 2011). Examples of key-value stores include Redis, Riak, Scalaris and Project Voldemort
- **Document Stores:** Data is stored in semi-structured documents that are indexed by a key. Documents can have a varying number of attributes of varying types and can also be queried to look for matching attributes contained within the document (Cattell, 2010). They offer additional functionality to key-value stores while

maintaining the ability to partition data over multiple nodes and provide support for replication and automatic recovery (Cattell, 2010). Examples of document stores are CouchDB, MongoDB and Dynamo.

- Extensible Record Stores: Also referred to as wide column stores due to the fact that each row in a related dataset can contain a varying amount of attributes (Cattell, 2010). Of the NoSQL systems, this data model is the closest to the relational data model. Data is stored in tables but each row has a dynamic number of attributes (Cattell, 2010). Both rows and columns can be distributed across nodes providing high scalability and availability (Cattell, 2011). Examples of extensible record stores are Google BigTable, Cassandra, HyperTable and HBase.

Other non-relational database systems in existence have been put into the NoSQL category that do not fit in to the three categories above. Graph databases, such as Neo4j, store data as relationships between key value pairs (Neo Technology, 2011). Object-oriented databases store data as collections of objects that can be easily materialized as programming language objects (Cattell, 2011). Both of these systems provide features such as horizontal scaling and the ability to store massive amounts of data, however as Cattell (2011) points out, these systems differ from those found in the three categories described above in that they generally provide ACID transactions and data querying involves complex object behaviour rather than simple key lookups (Cattell, 2011). This represents a significant characteristic difference to the key-value stores, document stores and extensive record stores. Therefore, this review will exclude these systems from the NoSQL category and any reference to NoSQL will assume a system from the three categories described above.

**Characteristics of NoSQL systems**

Besides placing each system in to one of the three categories described, all of the systems have common characteristics which allow them to be collectively described as NoSQL databases.

- **Horizontal Scaling:** This is a key feature of NoSQL. Data can be replicated and partitioned over many servers in a “shared nothing” architecture (Cattell, 2011), that is, all nodes are equal and none of the hardware is shared. This enables two important features of NoSQL – storage of large amounts of data and the ability to use cheaper commodity servers instead of more expensive enterprise class servers. The CouchDB system derives its name from this characteristic: Cluster Of Unreliable Commodity Hardware (Bhat & Jadhav, 2010).
- **Automatic Sharding:** Data is automatically spread across all servers in the cluster. Also referred to as “elasticity”, due to the fact that servers can be added or removed without any downtime. Any new server added immediately begins to receive data from the other servers in the cluster. Data is also replicated across the cluster (Couchbase, 2012).
- **No Schema:** Unlike a traditional RDBMS, NoSQL databases are confined to a rigid data schema. Any record that is inserted can have an arbitrary number of attributes associated with it and these attributes can be altered at any time (Couchbase, 2012). This provides excellent flexibility for applications whose data may not conform to a constant structure and is likely to change regularly.

- Simple query interface: As the name implies, NoSQL does not support the SQL query language<sup>1</sup>. Instead, querying is provided through various different mechanisms which varies from one distribution to another. For example, Amazon's SimpleDB uses a subset of SQL commands such as SELECT and DELETE along with operations like GetAttributes and PutAttributes. HBase uses another restricted SQL variant called HQL, and CouchDB uses a procedural approach to querying its document based records (Pokorny, 2011).
- Highly Available: Data is replicated across multiple servers (and even across multiple data centres) allowing for a highly available configuration that can handle multiple server failures and support disaster recovery (Cattell, 2010) (Couchbase, 2012).
- Weaker consistency model: Providing ACID semantics has been a staple feature of RDBMS databases since their inception in the 1970's. However, in a distributed architecture, the consistency property becomes more difficult to guarantee. Because the web has enabled 24x7x365 access to applications, availability has become a high priority for a database system. Because of this, developers are willing to sacrifice strong consistency in favour of higher availability. NoSQL systems provide this sacrifice, offering eventual consistency instead of strong consistency. This means that it may not be possible to get a consistent view of data across all nodes at any one time. Many developers are willing to live with this compromise (Cattell, 2010).

---

<sup>1</sup> Although lack of SQL support is a characteristic of NoSQL, the NoSQL name is commonly understood to mean 'Not Only' SQL.

### **Use Cases in Industry**

Despite the relative immaturity of NoSQL, there has been a surprisingly positive uptake of various implementations in industry. Unsurprisingly, a large percentage of organisations that are using NoSQL are Web 2.0 industry leaders such as Google and Facebook. However, there are also implementations from organisations in the areas of media, online entertainment and governmental departments.

#### **Facebook.**

In November 2010, Facebook launched their new messaging system which is based on HBase. Other platforms such as MySQL and Cassandra were considered for the project but HBase was chosen based on its ability to scale well, the replication system it employs and previous experience with the underlying technology within Facebook (DBPeditas.com, n.d.). MySQL was originally used to support this application but the volume of data involved (25TB per month) necessitated a system capable of supporting a high write throughput (Muthukkaruppan, 2011).

#### **Nokia.**

Like Facebook, Nokia have also converted a working relational database system to a proprietary NoSQL system developed in-house by Nokia (Farrell, 2011). The Nokia Places application is a POI registry used to support GPS applications. The system was originally built on a MySQL database and contained about 600GB of data (Farrell, 2011). However, due to a possible merger with Bing maps and the unwillingness to make a significant investment in upgrading the current RDBMS server, Nokia took the decision to move to a NoSQL system (Farrell, 2011). This enabled on demand scaling, which was a key requirement given the possible

merger. Another factor was the fact that the schema required for the system was not rigid. This is another area where NoSQL provided an advantage over the existing MySQL system.

#### **Danish Department of Health.**

The Department of Health in Denmark are using the key-value store Riak to store data for a medicine card system which controls the administration of drugs to medicine card holders (Thorub, 2011). This is another example of a system that was based on MySQL and converted to NoSQL. The main reasons cited for this were (Thorub, 2011):

- High availability: Because of the criticality of the system, high availability was a high priority requirement. Using Riak allowed the system to be configured in a distributed architecture across multiple data centres.
- Scalability: To prepare for expected growth by allowing dynamic scaling of the system.
- Operational improvements: Such as the ease of creating backups in comparison to the MySQL system.

This use case demonstrates that the use of NoSQL is not confined to the Web 2.0 industry; it is also being employed to support highly critical enterprise applications.

#### **Guardian.co.uk.**

Developers at the guardian.co.uk took the decision to move to a NoSQL environment when they began to hit continuous road blocks with their J2EE/RDBMS/ORM solution due to the complexity associated with every upgrade of their web site. Every change they made required a schema upgrade which changes had to be made to over 300 tables, 10,000 lines of hibernate XML configuration, 1000 domain objects mapped to the database and 70,000 lines of domain object code (Wall, 2011). The tight binding to the application prompted them to look for an



alternate solution and because they were using a JSON API, it made sense for them to store the data as JSON documents. This eventually led them to the document store MongoDB.

Where most of the NoSQL use cases in industry originate from the need for scalability and availability, the primary reasons for choosing MongoDB in this particular case were because of its data model, its ability to do complex queries, the flexible consistency modes, no schema and the fact that it works well at large and small scale deployments (Wall, 2011). This shows that NoSQL databases are being used for a range of reasons and not just for scalability and availability.

### **Scalability Concepts**

The ability to scale easily and on demand is seen as one of the most appealing characteristics of NoSQL systems amongst adopters in industry. This is worth discussing in the context of current scalability options user of relational databases are faced with. The relational database was designed on a centralised computing model. Notably, other tiers in an enterprise architecture, such as the web tier and the application tier, are typically built on a distributed model (Couchbase, 2012). The only option to increase the capacity of a standalone relational database system is to invest in more powerful hardware; increasing CPU, memory, I/O capacity and disk space. Pokorny (2011) refers to this as vertical scaling, or scale up. This form of scaling introduces a number of undesirable characteristics:

- Big servers are usually highly complex and expensive. The more users a system must support, the bigger the server required. This brings an exponential growth in cost for a linear increase in users (Figure 1);

- No matter how much a server is upgraded, there is a limit to the total capacity of a system (Figure 1);
- There is increased pressure on fault tolerance and high-availability strategies if there is only one database server and often these strategies involve highly complex hardware configurations (Couchbase, 2012).

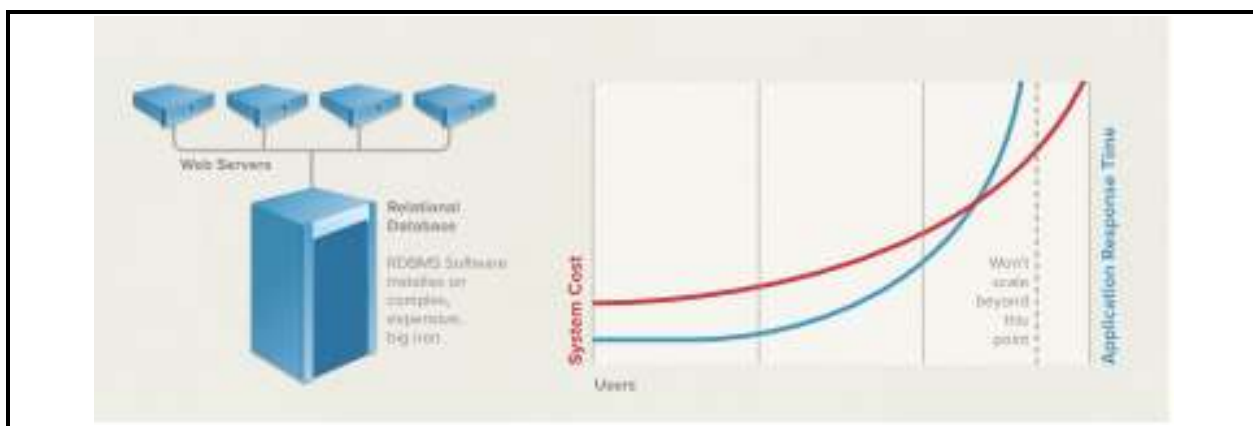


Figure 1. System cost vs. No of users of RDBMS (Couchbase, 2012)

In contrast, NoSQL systems scale horizontally, or scale out, using cheaper commodity servers. This mirrors more closely the distributed architectures found in web tiers and application tiers. Each server contributes equally in a shared-nothing architecture and there is no one single point of failure (Cattell, 2011). Horizontal scaling allows for a more agile and cost effective way to scale a database (Figure 2).

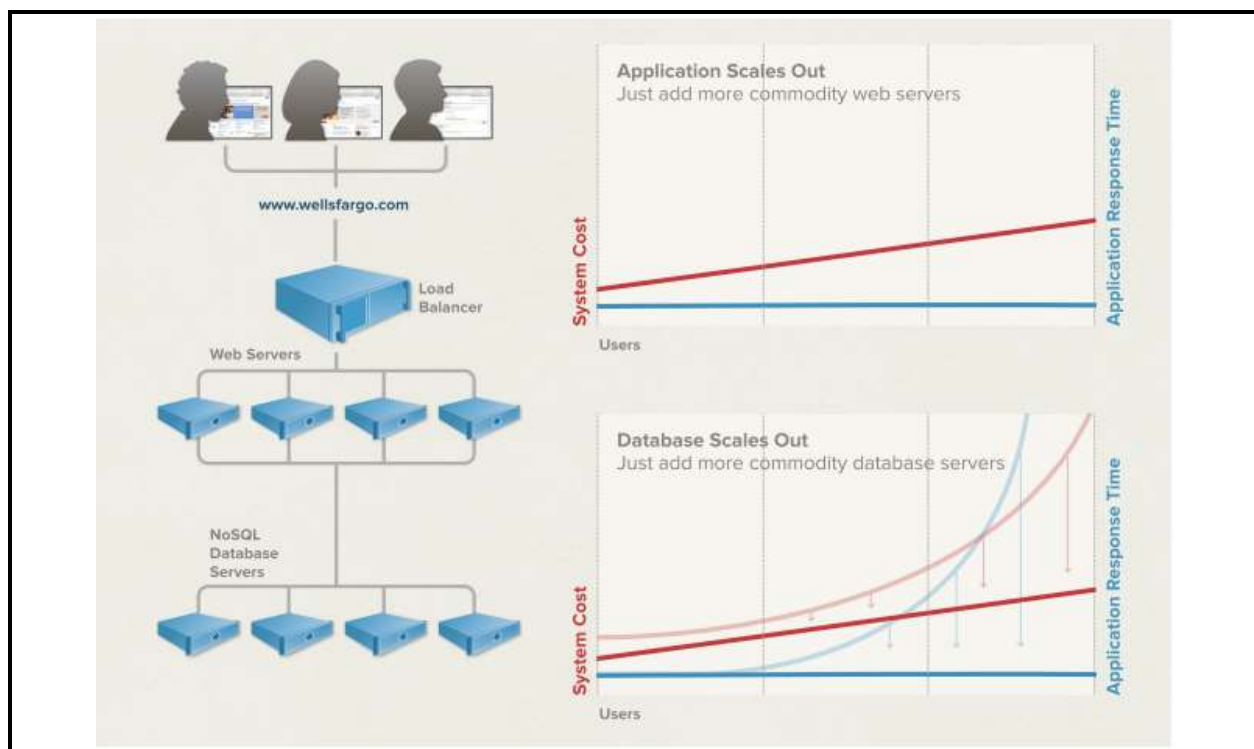


Figure 2. Cost effectiveness of horizontal scaling (Couchbase, 2012)

In order to provide horizontal scaling, NoSQL systems provide a weaker concurrency model than the ACID transactions found in relational database systems. The acronym BASE (Basically Available, Soft State, Eventually consistent) has been suggested in many sources as a more appropriate term for NoSQL (Cattell, 2011). What this acronym alludes to is that a NoSQL system will always be available, but the data may not be in a consistent state. Instead, most NoSQL systems provide eventual consistency, meaning all nodes will eventually be in a consistent state after updates have been propagated. The properties of an ACID system versus those of a BASE system are presented by Brewer (2000) and shown in Figure 3.



The image is a presentation slide titled "ACID vs. BASE" in yellow text on a dark blue background. In the top right corner, there is a logo consisting of a red hexagon with a yellow and blue geometric shape inside, and the text "inktomi" below it. The slide is divided into two columns by a vertical line. The left column is headed "ACID" in yellow, and the right column is headed "BASE" in yellow. Both headings are underlined. Each column contains a list of characteristics, each preceded by a yellow diamond symbol. The ACID list includes: Strong consistency, Isolation, Focus on "commit", Nested transactions, Availability?, Conservative (pessimistic), and Difficult evolution (e.g. schema). The BASE list includes: Weak consistency (with a sub-bullet "stale data OK"), Availability first, Best effort, Approximate answers OK, Aggressive (optimistic), Simpler!, Faster, and Easier evolution.

<u>ACID</u>	<u>BASE</u>
◆ Strong consistency	◆ Weak consistency — stale data OK
◆ Isolation	◆ Availability first
◆ Focus on “commit”	◆ Best effort
◆ Nested transactions	◆ Approximate answers OK
◆ Availability?	◆ Aggressive (optimistic)
◆ Conservative (pessimistic)	◆ Simpler!
◆ Difficult evolution (e.g. schema)	◆ Faster
	◆ Easier evolution

Figure 3. ACID vs. BASE (Brewer, 2000)

An alternative view is to compare scalability in a relational database and a NoSQL database in terms of the CAP theorem, also presented by Brewer (2000). This theorem states that of the three properties consistency, availability and partition tolerance, only two of these can be guaranteed in a shared data system. Pokorny (2011) explains these properties in more detail:

- *Consistency*: After data is written, all users will see the same version of the data
- *Availability*: Every operation on the database will terminate successfully.
- *Partition tolerance*: The database can still operate even when some nodes in the distribution are unavailable.

Distributed relational databases tend to forfeit the availability property in favour of consistency and partition tolerance. But in order to provide this strong consistency in a distributed system, any writes to the database must be committed on every node before a

transaction can complete. This is the critical factor that limits the scalability of relational systems.

Conversely, NoSQL systems forfeit the consistency property in favour of availability. The eventual consistency approach means that a transaction can complete once the data is written to one node and it is then up to the database engine to propagate the data to each of the other nodes in the distribution. If it is critical that an application be always available and an acceptable possibility that it may read stale data, then this compromise may be more desirable than a relational system. Furthermore, many NoSQL systems provide a level of control on the consistency property, allowing the developer to specify a number of nodes in the cluster that the write must propagate to before the transaction can be considered complete. Cassandra, for example, provides six different consistency levels for write operations. Specifying a consistency level of ONE or ANY requires a write to complete on only one node in a replica, ALL requires the write to complete on all replica nodes and QUORUM requires the write to complete on a quorum of nodes calculated by the formula (Black, 2009):

$$(\text{replication\_factor} / 2) + 1$$

Therefore, if the number of replicas is 3, then writes must succeed on at least two of the nodes before the transaction completes. The quorum can also be enforced on read operations. If the data returned from each node in the quorum does not match, the conflict must be dealt with before the data is returned. This ensures that no stale data is read because the write and read quorums overlap (Black, 2009).

Even though a traditional RDBMS does not provide simple scalability, many developers are left with no choice but to implement a distributed system when the demands of applications outgrow what is achievable with a single server system. The most common approach to scaling

an RDBMS is to shard data across several servers. This involves partitioning a table based on some pre-defined criteria and placing each partition on a separate node in the distribution.

Although this is a common technique, there are a number of drawbacks to this approach:

- A cross-shard filter or join must be performed in the application;
- If there are updates on multiple shards within a transaction, then the application is responsible for guaranteeing consistency across nodes;
- As the system scales, node failures become more common. Consistent replicas are difficult to maintain, it is difficult to detect failures, fail over to replicas and replace failed nodes in a running system;
- Making schema changes is very difficult without taking shards offline;
- Adding additional nodes or changing configuration is “extremely tedious” and “much more difficult if the shards cannot be taken offline” (Stonebraker & Cattell, 2011).

There are, however, a number of scalable relational systems available that claim to provide comparable scalability and availability to NoSQL systems, such as MySQL Cluster, VoltDB and Clustrix. Cattell (2011) notes that some RDBMS systems can provide scalability on a similar level to NoSQL systems if they can abide by two conditions:

1. Avoid operations that span many nodes, e.g. joins over many tables.
2. Avoid transactions that span many nodes. The communication and two-phase commit overhead will lead to inefficient performance.

It is worth noting that NoSQL systems inherently avoid these issues by making it difficult to perform these types of operations (Cattell, 2011). If these scalable RDBMS systems prove to

be comparable with NoSQL, then the fact that they retain ACID properties and the SQL query language could make them a more attractive option for a scalable data store.

### **The Cost Advantage of Scaling with NoSQL**

As well as being easier to scale with NoSQL, it can also be more cost effective than it is to scale out with a relational database system. Many NoSQL database vendors license the database engine under the GNU Free Software Foundation license, leaving the only costs with the hardware required to implement the cluster. Compared to a commercially licensed relational database system, the savings can be significant. For example, the cost for a typical 4 node Oracle RAC Cluster set up:

- Standard license:  $\$17,500 \times 2 \text{ processors} \times 4 \text{ nodes} = \$140,000$ ;
- Real Application Clusters option:  $\$23,000 \times 2 \text{ processors} \times 4 \text{ nodes} = \$184,000$ ;
- Partitioning option:  $\$11,500 \times 2 \text{ processors} \times 4 \text{ nodes} = \$92,000$ ;
- In-memory Database Cache:  $\$23,000 \times 2 \text{ processors} \times 4 \text{ nodes} = \$184,000$

(Oracle, 2012).

This gives a total licensing cost of \$600,000. MySQL Cluster is probably a more comparable option as, like the NoSQL databases, it is free to license. However, if support is required, this costs \$10,000 per year for each 1-4 socket server (Oracle, n.d.). Not all NoSQL vendors offer a support option, but two that do are Riak and MongoDB. They provide support at a cost of \$3,995 and \$4,000 per node per year respectively (10Gen, 2012) (Basho Technologies Inc., 2012).

### NoSQL Performance Characteristics

Besides from the scalability advantages, the other area in which NoSQL claims to provide an advantage over traditional RDBMS systems is in performance. However, performance of a database system can be affected by several different factors and performance requirements can vary from one application to another. A classic OLTP type system is generally more write intensive than read intensive, while a data warehouse type system will normally be more read intensive. A Web 2.0 application will normally fall at a varying point between these two. Stonebraker & Cattell (2011) take social networking as a typical Web 2.0 DBMS application and plot the graph in Figure 4 to depict where such an application will fall on a write-focus vs. read-focus scale. Different applications will have different performance requirements and therefore it is not possible to simply state that NoSQL databases provide better performance than distributed relational databases. The reality is likely to be that NoSQL can provide performance gains for *certain* types of applications.

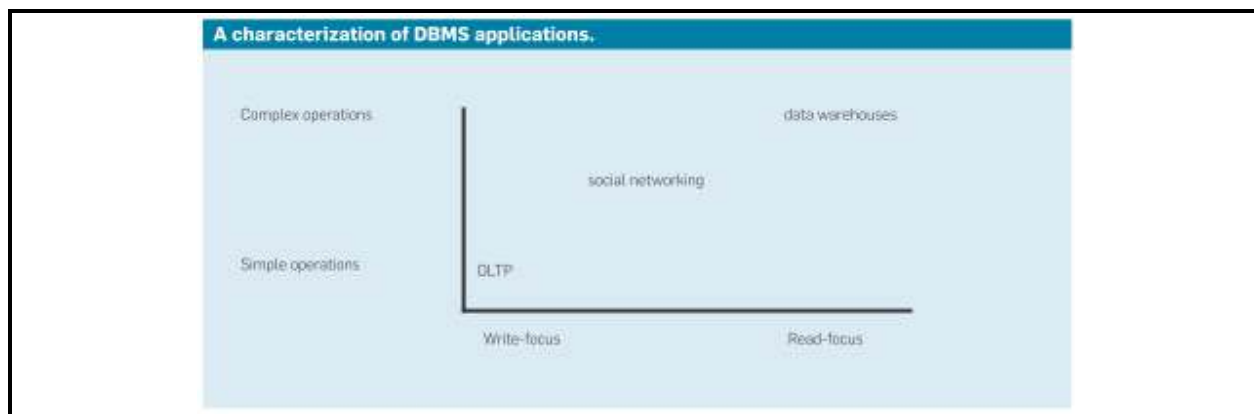


Figure 4. Characterisation of DBMS applications (Stonebraker & Cattell, 2011)

Although there are a number of recognised benchmarks for measuring the performance of a relational DBMS, such as TPC-C, which measures performance of a database processing a typical OLTP workload (Raab, Kohler, & Shah, n.d.), there has been a limited amount of



research in the area of performance measurement for NoSQL systems, and any systems that have been developed are in the early stages of maturity. The most widely recognised benchmark for NoSQL is the Yahoo! Cloud Serving Benchmark (YCSB), developed by Yahoo! to facilitate comparisons between “new generation cloud data serving systems”, which are analogous to NoSQL systems (Cooper, Silberstein, Tam, Ramakrishnan, & Sears, 2010). The YCSB is not confined however to NoSQL systems. It is the intention of the framework to provide evaluation of any cloud based storage systems, including sharded relational systems. The YCSB is a significant tool in terms of comparing sharded relational and NoSQL systems to assist in the selection of an appropriate data storage solution and merits further discussion.

### **Yahoo! Cloud Serving Benchmark**

The YCSB is an extensible open source framework and it is the intention of the developers that the benchmark be adapted for use with multiple database systems. However, the system is in the very early stages of maturity, having been released in 2010, and there is a limited amount of results of usage of the system at this point.

The benchmark currently evaluates systems on two tiers: Performance and Scalability<sup>2</sup>:

- Performance is evaluated in terms of latency of requests when the database is under load. There is a trade-off between latency with throughput. A system is evaluated by measuring latency as throughput is increased until a saturation point is reached (Cooper et al., 2010).

---

<sup>2</sup> Other aspects such as availability and replication can be supported through extension of the system

- The scalability tier examines the impact on performance as more servers are added.

The benchmark measures two metrics in this tier:

- Scale up – A workload is run on a given number of servers and performance is evaluated. More servers are then added and a larger workload is run on this configuration. Good scaling is indicated if performance remains constant as the workload and the number of servers are proportionally increased.
- Elastic Speedup – This metric evaluates how performance is impacted as the number of machines is increased in a running system. A system that provides good elasticity should show an increase in performance when a machine is added, after an initial period of reconfiguration.

The YCSB uses a set of predefined workloads to produce results. These workloads can be tweaked to perform various combinations of insert, update, read and scan operations. In order to decide which records should be operated on, the tool uses one of several distribution algorithms:

- Uniform: An item is chosen uniformly at random. All records are equally likely to be chosen;
- Zipfian: Choose an item according to the Zipfian distribution. In this mode, some records will be extremely popular while most will be unpopular. This is useful for modelling items whose popularity is independent of their newness, for example in a social networking application where some profiles are updated more often than others, even though the profile may be many years old;

- Latest: The most recent records are more likely to be chosen. This could be used to simulate something like a blogging application where the most recent posts are more likely to see activity;
- Multinomial: The probability of operations can be assigned a value. For example, a probability of 0.95 can be assigned to the write operation to simulate a write-heavy workload (Cooper et al., 2010).

Once the workload has been created, it is executed against the database using a Java program called the YCSB client. This client is responsible for generating both the data to load and the operations to be performed. The workload executor drives multiple threads which throttle the rate at which requests are generated to control the load on the database. They are also responsible for measuring latency and throughput and reporting the results to a statistics module (Cooper et al., 2010). The client architecture is shown in Figure 5.

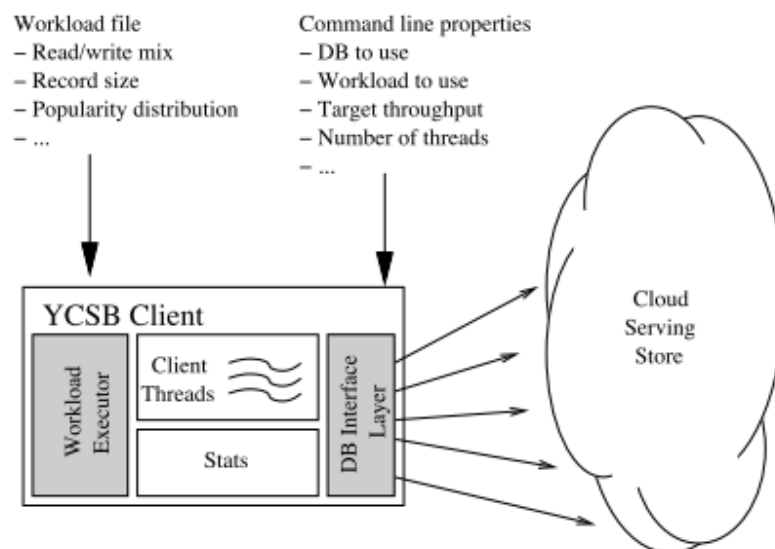


Figure 5. YCSB Client Architecture (Cooper et al., 2010)

In order to demonstrate the YCSB tool, Cooper et al. (2010) present benchmarking results for four distributed database systems: Cassandra, HBase, PNUTS and sharded MySQL. These results provide a valuable insight in to how a distributed relational system might compare against a NoSQL system under certain conditions. Five workloads were created and these are detailed in Figure 6.

Workload	Operations	Record selection	Application example
A—Update heavy	Read: 50% Update: 50%	Zipfian	Session store recording recent actions in a user session
B—Read heavy	Read: 95% Update: 5%	Zipfian	Photo tagging; add a tag is an update, but most operations are to read tags
C—Read only	Read: 100%	Zipfian	User profile cache, where profiles are constructed elsewhere (e.g., Hadoop)
D—Read latest	Read: 95% Insert: 5%	Latest	User status updates; people want to read the latest statuses
E—Short ranges	Scan: 95% Insert: 5%	Zipfian/Uniform*	Threaded conversations, where each scan is for the posts in a given thread (assumed to be clustered by thread id)

\*Workload E uses the Zipfian distribution to choose the first key in the range, and the Uniform distribution to choose the number of records to scan.

Figure 6. YCSB Workloads (Cooper et al., 2010)

Results for performance are presented in a graph of Read latency (measured in milliseconds) versus throughput (measured in total operations per second, reads and writes). Workload A results, shown in Figure 7, indicate that the NoSQL systems (Cassandra and HBase) perform better for update-heavy workloads consisting of an equal amount of reads and updates.

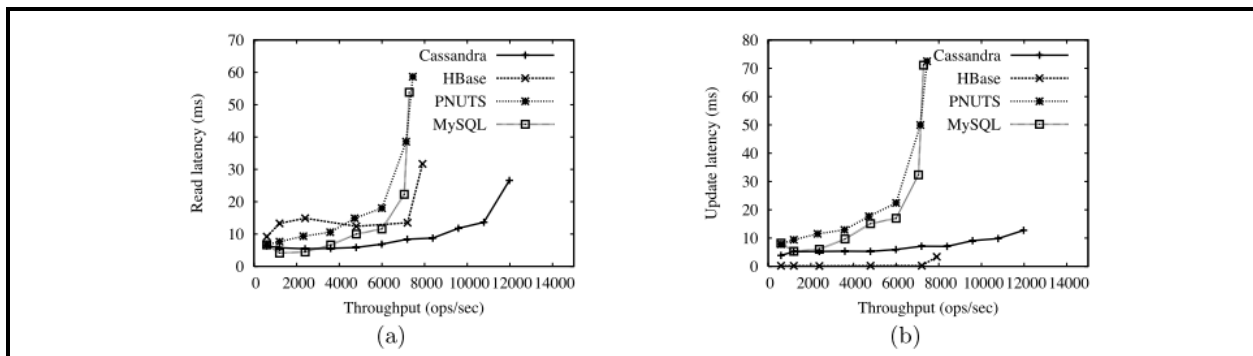


Figure 7. Workload A results - (a) is read operations and (b) is update operations (Cooper et al., 2010)

The read-heavy operations of workload B, however, show that the sharded relational system performs significantly better than the NoSQL systems (Figure 8), indicating again that NoSQL is a more suitable candidate for write-heavy or update-heavy types of applications. Workloads C (read only) and workload D (read latest) also indicated that the sharded MySQL system performed better for these operations.

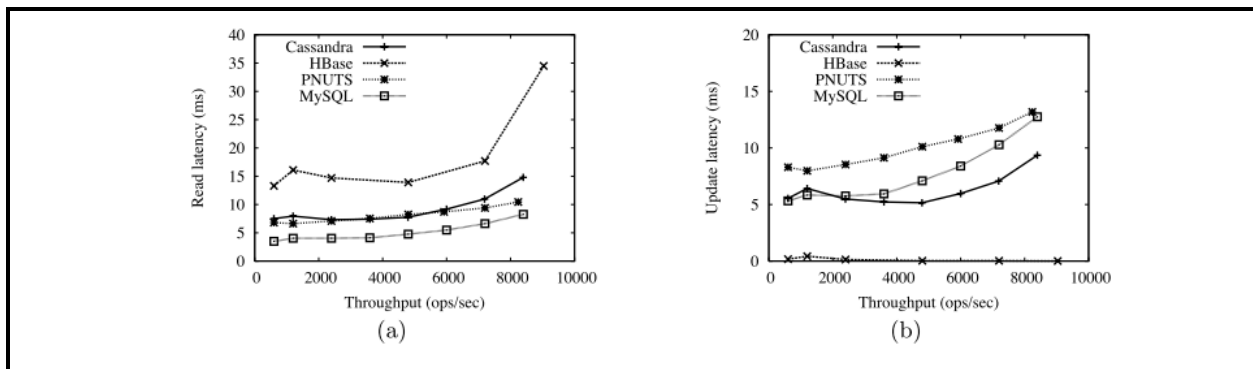


Figure 8. Workload B results - (a) read operations, (b) update operations (Cooper et al., 2010)

The remainder of the tests focus on the tier 2 benchmarks, scalability and elasticity. To test scalability, performance is measured with a varying number of servers. The load is increased proportionally with each server added, which should produce a straight line result. This is shown to be the case in Figure 9, apart from HBase which only begins to perform consistently at a higher number of servers. MySQL is omitted from this test which is unexplained by the authors.

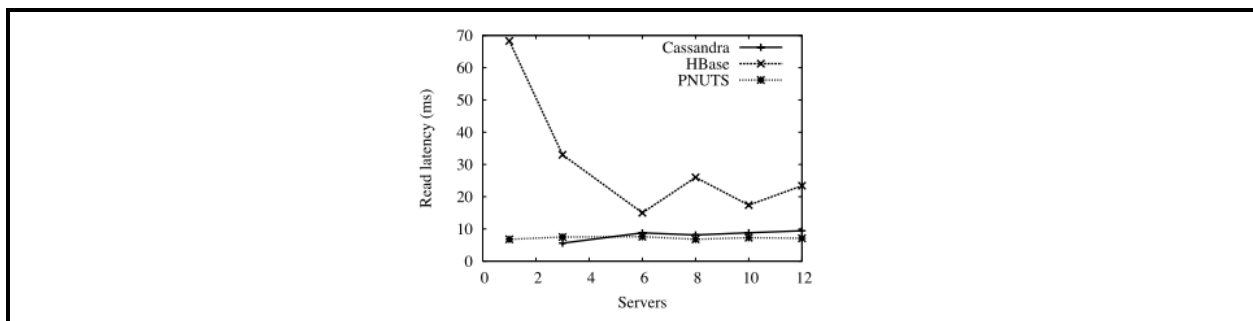


Figure 9. Scalability test (Cooper et al., 2010)

The final set of tests demonstrate elasticity by measuring performance as a single server is added to a cluster. Results were expected to show high latency at the point where the server is added before returning to a consistent level. However, as Figure 10 shows, this was not exactly the case, in particular with Cassandra which performed erratically after the server was added. The sharded MySQL system is automatically omitted from this test on account of being “inherently inelastic”, giving an instant win for NoSQL systems in this area.

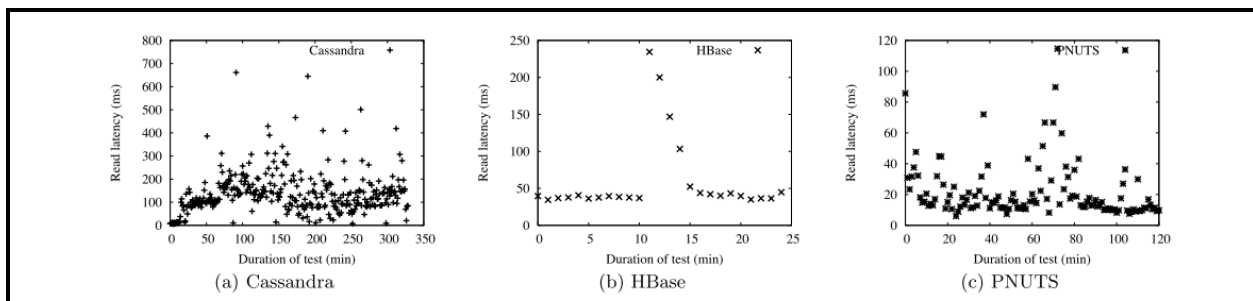


Figure 10. Elasticity results (Cooper et al., 2010)

The YCSB benchmark results presented by Cooper et al. (2010) provide a solid foundation for testing the performance of NoSQL systems in comparison with a distributed relational system. The results predominantly match the claims of the NoSQL systems, but there are some exceptions, particularly in the elasticity tests, indicating that vendor claims need to be fully tested before a system can be put to use. There are other factors, such as availability and replication which have not been addressed and this provides an area for further research.

### Challenges for NoSQL Adoption

Despite the advantages of NoSQL, there are a number of challenges facing vendors that must be addressed before widespread adoption can occur.

**Vendor Support.**

Although there are many different NoSQL vendors, many are community driven and do not provide any formal support structure. Most businesses will look for the assurances of a support contract when choosing a database system to prevent any potential data loss or unavailability of data. Some NoSQL vendors however do provide a support option for enterprise level applications. The initiator and sponsors of MongoDB, 10gen, offer two support packages with varying levels of support for an annual fee of either \$2,500 or \$4000 per cluster node (10Gen, 2012). Riak provides an Enterprise license for their database system which includes 24x7 customer support, developer support and implementation consultancy at an annual cost of \$3,995 per cluster node (Basho Technologies Inc., 2012). Other vendors will need to follow suit in order to penetrate the enterprise market.

**Data Querying.**

One of the main advantages of relational databases is the ability to query for data using SQL. Apart from some minor differences between vendors, the SQL language can be used on practically all relational database systems. Consequently, extracting data from relational databases is a standardized process that has been in existence for nearly 40 years. People are familiar with how to use SQL and it makes it easier to transfer from one relational DBMS to another. Because NoSQL is relatively new technology, a standardized query language that is capable of extracting data from all NoSQL database types has yet to be developed. There are efforts in existence that are attempting to achieve this, one of which is UnQL, an unstructured query language for JSON, semi-structured and document databases (Young, 2011). However, efforts on this project have slowed recently and it does not seem likely that it will be continued (Celler, 2012).

Relational databases are all based on the use of tables to store data and therefore a universal language is easier to create. With NoSQL, there are several different approaches to storing data, as discussed above, therefore making it more difficult to create a universal language. SQL succeeded because it was easy to learn, easy to read and easy to understand. With NoSQL, you either need to learn the query language that the vendor has created for each individual product or be proficient in a programming language that is compatible with the NoSQL system. For instance, Google has created its own query language called GQL which is compatible with its own data store products such as Google Big Table (Google, 2012). Riak bases its query language on Lucene, an open source search engine written in Java (Rowe, 2012). These are two vastly different approaches to querying data and in the case of Riak and other vendors that use Java based languages for querying, it is likely to deter people from transitioning from the familiarity of relational databases and SQL.

### ***Using Map-Reduce for data retrieval.***

One area of querying where NoSQL vendors do seem to be united on is the use of Map-Reduce for distributed queries. Map-Reduce is a model for processing large amounts of data that is distributed over a large number of nodes, in order to produce a set of derived data. Essentially, Map-Reduce is a method to abstract the complexity of parallelization, fault tolerance, data distribution and load balancing that is required when running computations across hundreds or thousands of nodes (Dean & Ghemawat, 2008). A Map-Reduce operation is capable of retrieving data at rates of up to 30GB/s by using its approach of assigning worker threads to each node in the cluster to process the data and then merging the output of these worker threads in to one final result (Dean & Ghemawat, 2008).



Map-Reduce is a concept that transfers well to NoSQL environments. If the data is sharded across several nodes, the database engine can dispatch the Map-Reduce job to all of the nodes and run them in parallel. This allows even the largest databases to return the results of queries very quickly in a “divide and conquer” approach. Because of the way Map-Reduce iterates over data in a collection, it is possible to aggregate data which have fields with matching values and store these computations in secondary collections. As a result, Map-Reduce functions are an ideal replacement for SQL GROUP BY statements that would be found in a relational database.

**Immaturity of the technology.**

NoSQL databases have only been in existence for a matter of years and many vendors are still in beta stage or are releasing updates continuously. This could be unsettling for potential adopters of the technology as most are looking for ultra reliability in the system that is managing their critical data. Software often needs to go through many minor and major revisions before bugs are discovered and patched.

The immaturity of the technology also brings a shortage of expertise in the field. Relational database systems have built up sizeable knowledge bases and technical papers to assist users in the deployment and use of their system. With NoSQL, many adopters are taking a “learn as we go” approach, which may not be to everyone’s liking. For example, Foursquare, the location based social network were one of the first major users of MongoDB and they suffered an 11 hour outage early in their NoSQL adoption because of a flaw in their design which led to uneven growth of their partitions in the database which eventually resulted in the database having to be taken offline (Bodkin, 2010). As a result, the creators of MongoDB used this incident to improve the reliability of their system and the learning’s were passed on to other

users through the MongoDB community, but, this episode does indicate that there may be potential issues with the software that the vendors have not yet encountered.

## **Chapter 3 - Methodology**

### **Introduction**

This section will outline the methodology used to convert an existing PHP/MySQL application to a NoSQL application. The relational model used for the bulletin board application will be discussed first and subsequently outline how this model can be converted to a NoSQL system. By comparing the chosen NoSQL database to a relational database, the existing relational schema can be converted to a NoSQL schema. This model will then form the basis for a new version of the PHP application which will need to be converted to work with the NoSQL schema.

This chapter will also describe how test data was loaded to both the relational database and the NoSQL database for the purposes of performance analysis. The last section of the chapter describes how both databases were set up and deployed.

### **Use Case**

Choosing a data storage platform for an application is no longer the straightforward choice that it once was. The relational database was, by and large, the only option available and usually the decisions at this level revolved around what vendor to choose and what version of DBMS from that vendor. But with new data storage technologies emerging, such as NoSQL, designers are starting to evaluate data storage needs from a different viewpoint. Estimating the volume of data that an application is likely to produce is becoming an increasingly difficult task, particularly if the application is web enabled and accessible by an arbitrary number of users that could potentially grow in to the millions. Therefore, scalability is now a much higher priority than it may have been 15 to 20 years ago. The database must be able to grow in line with

potential demand and, as alluded to in the literature review, the cost of scaling up a relational database can be a significant barrier. Increasing volumes of data also bring performance considerations which must be addressed. Read and write operations can be adversely affected if the database is not designed to cope with high volumes of data.

These are all problems that many users are facing with relational databases and for this reason are starting to look at other options. NoSQL databases can potentially provide the solution. They have better scaling capabilities because they can be horizontally scaled, they can support large volumes of data with minimal impact to performance through Map-Reduce, and they can provide a simpler method of implementing a distributed database. Some of the use cases where this may be particularly beneficial include Content Management Systems, Ecommerce, Online Games, Real-Time Analytics, Event Logging and the Operational Data Store for a website (Merriman & Francia, 2011).

To evaluate these potential benefits, a single use case will be presented in this chapter to provide a basis for comparison between a NoSQL system and a relational system. Online bulletin board applications are widespread on the web today and are among the most popular destinations for internet users. Many of these sites must be able to support thousands of concurrent users on a 24x7 basis, who are constantly producing new data and interacting with the database. These are exactly the types of requirements to which a NoSQL system is potentially well suited.

However, many of these bulletin boards are PHP applications which use a relational database for data storage. This can be attributed to the fact that sites like phpBB and vBulletin offer free open source online bulletin board applications with this configuration. PHP and MySQL are two closely knit technologies, but this does not necessarily mean that MySQL offers the best solution for a PHP application such as this one. The use case presented in this chapter

will include a PHP application that connects to both a MySQL database and a comparable NoSQL database. The intent is to show that a NoSQL database can work just as well as a MySQL database as the operational data store for a PHP application and in particular situations can actually provide a better solution.

### **Application Details**

There are many options available in terms of creating a bulletin board application. The aforementioned vBulletin and phpBB applications provide “off the shelf” frameworks which allow anyone to create a customized bulletin board application instantly. These frameworks provide a rich set of features such as search optimization, security add-ons like captcha, email verification, granular privileges etc., user management functions, customizable styles and many more (phpBB, 2007; vBulletin Solutions, 2012).

Although either of these systems would be ideal to use as the basis for the use case in this methodology, converting either to use a NoSQL back-end would be a considerable project. For this reason, an application with a subset of this functionality was chosen as the basis for the use case. This application is presented in (Naramore & Glass, 2005). It is a basic PHP/MySQL bulletin board application. As well as post and forum creation, the application has the following additional functions:

- User Authentication
- Search Engine
- Board Administration
- Regular Expressions for post formatting
- Pagination to limit the number of posts per page

The application interacts with the relational database for all of these functions and therefore affects the amount of time required to load a page. At small amounts of data and users, this time will be unnoticeable to the end user. But as the number of concurrent users grows and the database size increases, this time will eventually reach a point where performance is negatively affected to the point where the application becomes unusable. At this point, a course of action must be taken to ensure that the application can continue to operate in a usable manner.

Assuming that this application is running on a single database server, there are a number of options available to improve the performance of this application. One is to upgrade the hardware of the database server to increase transaction performance. This might include the addition of memory, a faster CPU, quicker hard disks in a striped configuration etc. One would need to analyze the system to determine where the bottlenecks are in the system and what particular hardware can alleviate this bottleneck. This approach can provide a “quick fix” to performance problems. In the long term however, this approach will not prove successful. A further increase in user traffic in the future will require further upgrades leading to exponential cost increases for hardware and eventually reaching a point where upgrading is no longer possible.

A second option is to add a second database server and implement a partitioning strategy so that data is divided between the two servers thereby halving the workload of the single server system. This approach will prove effective at improving performance but it also adds a layer of complexity to the application layer which may prove to be overwhelming for developers and unsustainable if the application requires additional servers in the future.

Replication is also a common methodology used to improve the performance of a database application. This involves copying the data from the main database server to one or

more replicas. The replicas can then be used for read operations allowing the writes to happen exclusively on the main server.

This study proposes that a NoSQL database provides a further option for improving application performance. MongoDB, which will be used in this case study, provides automatic replication and sharding, thereby combining both of the options discussed above without the complexity overhead. This methodology will describe how an application can be converted to using MongoDB to take advantage of these features.

### Database Description

The application uses a MySQL relational database for storage of all persistent data. This includes user registration details, user posts, forum metadata and configuration data such as the board title and pagination limits. The schema for the database is shown in Figure 11.

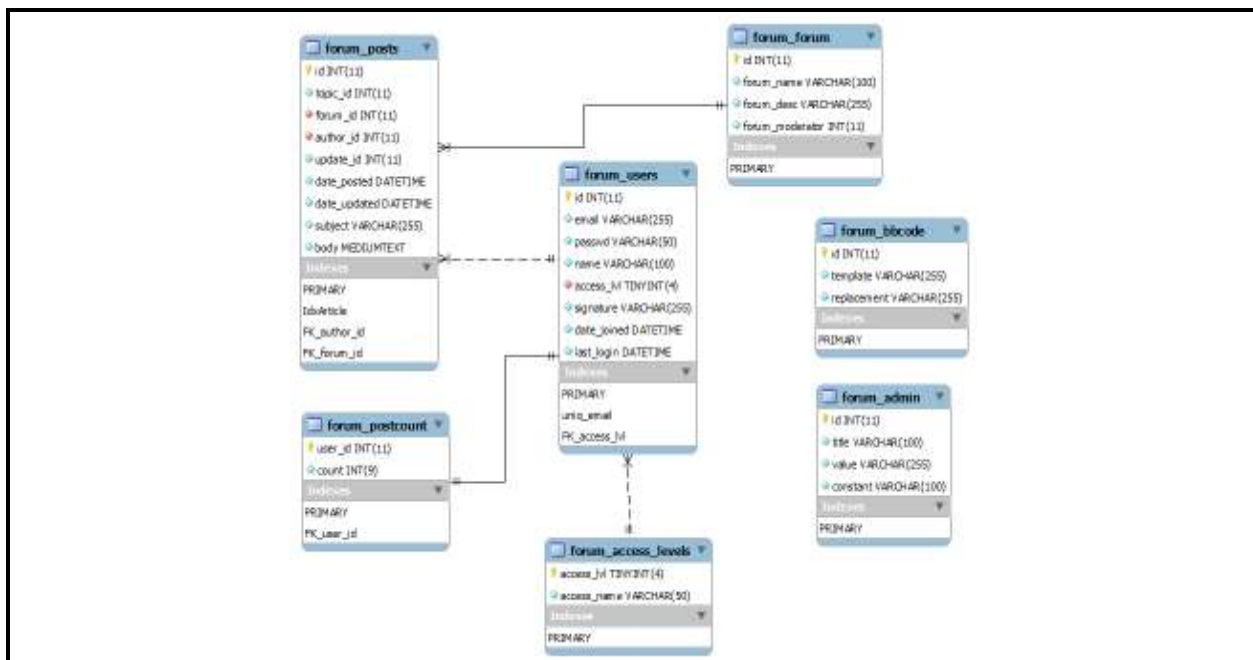


Figure 11. Bulletin Board Relational Schema

MySQL offers pluggable data storage engines, the two most common of which are MyISAM and InnoDB. The original schema for this application uses the MyISAM engine for all tables, however, as of MySQL 5.5, the default engine is now InnoDB. The main feature differences between MyISAM and InnoDB are outlined in Figure 12. Traditionally, MyISAM was seen as the better performing option, but recent benchmark tests have shown that with more relaxed ACID constraints, InnoDB can perform comparably with MyISAM while providing additional benefits such as crash recovery, referential integrity and scalable performance increases (Oracle, 2011). For this reason, the InnoDB engine was chosen as the default for this application.

Feature	InnoDB	MyISAM
ACID Transactions	Yes	No
Configurable ACID Properties	Yes	No
Crash Safe	Yes	No
Foreign Key Support	Yes	No
Row-Level Locking Granularity	Yes	No (Table)
MVCC	Yes	No
Geospatial Data Type Support	Yes	Yes
Geospatial (R-Tree) Indexing Support	No	Yes
B-tree Indexes	Yes	Yes
Full-text search Indexes	No	Yes
Clustered Index	Yes	No
Data Caches	Yes	No
Index Caches	Yes	Yes
Compressed Data	Yes [b]	Yes [a]
Read & Write to Compressed Tables	Yes	No (Read-Only)
Encrypted Data [c]	Yes	Yes
Replication Support [d]	Yes	Yes
Backup / Point-in-Time Recovery [d]	Yes	Yes
Query Cache Support	Yes	Yes
Update Statistics for Data Dictionary	Yes	Yes
Storage Limits (Table Size)	64TB	256TB

Figure 12. Comparison between MyISAM and InnoDB Storage Engines

Most of the activity in this database will occur in the forum\_posts table as users create and update posts through the web interface. The majority of activity will be inserts as users



create new posts as either the initial post in a new topic or a reply to a post in an existing topic.

There will also be a significant amount of updates as users and moderators edit existing posts.

Frequently accessed columns are also indexed in this table; `idxArticle` is a multi-column index on the fields `forum_id`, `topic_id`, `author_id` and `date_posted`.

The original application from Naramore & Glass (2005) provides a full-text index on the subject and body fields. However, the InnoDB engine that will be used in this application does not support full-text searches<sup>3</sup> and therefore will be excluded.

The next largest table to this will be the `forum_users` table which holds the data for all registered users. A new user registration will produce an insert while any changes to user profiles will produce an update. There is also a separate table for storing the number of posts a user has. Transactions on this table will be predominantly updates. The other tables in the schema are mainly used for site metadata and configuration data and therefore the majority of transactions will be reads.

## Application Description

The application consists of 21 PHP files, some of which contain the embedded HTML required to display the content in a web browser, and the remaining of which are pure PHP used for scripting and database interaction. The files that are of most interest in the context of database performance are those that initiate a connection and subsequent transaction on the MySQL server. These files will be described in this section.

---

<sup>3</sup> Full-text search is due to be added to InnoDB in the upcoming 5.6.4 release of MySQL

Starting with the home page, `index.php`, a request to this page alone creates several transactions in the database. Initially, there are three other files included with this file: `conn.php` which creates the connection to the database, `functions.php` is a helper file containing useful functions and `header.php` which displays the header for the site. Cascading from here, `header.php` includes the file `config.php` which queries the database for the site administration data contained in the `forum_admin` and `forum_bbcode` tables. The `index.php` page then queries the `forum_forum` table to get a list of the forum names and the `forum_posts` table to get a count of the number of threads in each forum.

Progressing from the `index.php` page, a user will either choose to log in, register or view forum. The log in and registration functions both post to the file `transact-user.php` which contains the functionality to create and modify user information in the database. This file reads an “action” parameter sent by the posting page and interacts with the `forum_users` table accordingly. The actions are as follows:

- Login Action: Reads from `forum_users` to check the given email and password combination and then updates this table with the last login time for this user.
- Create Account: Writes the new user information to the `forum_users` table.
- Modify Account: Updates the supplied information in the `forum_users` table.
- Edit Account: Reads the users existing password from the `forum_users` table for verification and then updates the supplied user information.
- Send Reminder: Reads the users existing password from the `forum_users` table.

Choosing the option to view a forum brings the user to the viewforum.php page. This page retrieves a list of the threads in the selected forum, the originating author of the thread, the number of posts in the thread and the date of the last post.

Clicking on any thread will load the viewtopic.php file. This file calls the showTopic function, which is defined in the file functions.php, along with many other functions that are used throughout the application. The showTopic function retrieves all of the post details from the database to be displayed in the browser. This includes the post subject, body, author and date, as well as the number of posts each user has made and the users signature. This function triggers a series of read transactions in the database involving the forum\_posts, forum\_forum, forum\_users and forum\_postcount tables.

Logged in users have a number of additional options in using the application. In terms of database transactions, this mainly involves updating the user profile, which has been discussed already, and creating and editing posts, which will be the predominant activity during normal application operation. Transactions related to forum posts are contained within the file transact-posts.php. This file operates in a similar manner to the transact-user.php file, reading an action parameter from the posting file and responding accordingly. The actions it performs are as follows:

- Submit new post: Writes a new post to the forum\_posts table and upserts the forum\_postcount table
- Create a new topic: Performs the same action as submitting a new post but with a new topic id.
- Update existing post: Updates the specified post in the forum\_posts table.
- Delete a post: Deletes the specified post from the forum\_posts table.

A user logged in with elevated privileges can also carry out a number of admin functions which will also initiate transactions in the database. The `transact-admin.php` and `transact-affirm.php` encapsulate this functionality. The type of transactions carried out by these files are CRUD operations on the forum definitions and the application data (`forum_forum` and `forum_admin`) tables. Admin users can also modify and delete posts, but this is just an extension of the transactions already described in the `transact-posts.php` file. The level of database activity generated by admin users will be fractional compared to normal users however it should be noted that deleting a forum will cause a cascading delete of all the posts in that forum and this function is only available to an admin user.

### **Choosing a NoSQL Database**

MongoDB was selected as the NoSQL database to use for this use case. The reasons for choosing MongoDB were:

- A reasonably simple evolution from a relational database can be achieved;
- It works at small and large scale, unlike other NoSQL databases, such as Cassandra and HBase, which are only suited to very large scale deployments.
- It has the capability to replicate any complex queries that may be encountered in a complex relational database.

MongoDB is a document oriented database where data is stored in schema-less documents which have a unique identifier (the document id). Documents are grouped in to collections and each document within a collection can have an arbitrary number of fields. It supports indexing of fields in the same way as a relational database, however there are no joins

between collections. It provides easy scalability through automatic sharding of data across servers and high availability through replicated servers with automatic failover.

Documents in MongoDB are stored as BSON documents, which is short for Binary JSON (JavaScript Object Notation). JSON is a language independent “lightweight data-interchange format” (“Introducing JSON,” n.d.). It is a basic human readable format that is built around a collection of key/value pairs or an ordered list of values. A value can be a string, number, object, array, Boolean type or null. Below is a sample JSON document:

```
{
  "Car": {
    "Doors": 4,
    "Instructor": 600,
    "Title": "View from 15th Floor",
    "Engine": {
      "CC": "2.0",
      "Fuel": "Petrol"
    },
    "Colours": ["Blue", "Black", "Red", "Silver"]
  }
}
```

BSON then, is a “binary-encoded serialization of JSON-like documents” (Creative Commons, n.d.). It expands on the JSON format by adding data types such as Date and BinData. BSON documents can be traversed easily, are lightweight, and can be easily encoded and decoded in most languages (Creative Commons, n.d.).

### **MongoDB Comparison to Relational Database.**

When converting a relational database to MongoDB, it is important to take a comparative look at the objects in MongoDB and how they relate to a relational database. At the top level, a database in MongoDB is conceptually equivalent to a relational database. The major difference to note is that the database does not need to be explicitly created in MongoDB, it is implicitly created once the first document is inserted.

A collection in MongoDB is analogous to a table in a relational database (Murphy & Chodorow, 2012a). Instead of containing rows, a collection contains documents which are in BSON format. Every document has an “\_id” field, but apart from this, a document can have a varying number of fields and all documents in a collection do not have to contain the same fields, unlike in a relational table where each row will have the exact same set of fields. Even though the document structure is flexible, the design from the outset should identify the specific fields required in each document to allow the application to be built around it.

Indexes are supported in MongoDB and are conceptually similar to indexes in relational databases (Murphy & Chodorow, 2012b). The basic storage format for a MongoDB index is the B-Tree and they can be created on any field in a document. The \_id field automatically has an index created on it, much like the primary key of a relational table.

Another important concept in MongoDB is sharding. This allows a collection to be partitioned across several nodes in a cluster if a collection becomes too large to perform sufficiently on a single node. MongoDB supports automatic load balancing of shards so that the volume of data on any particular shard does not become proportionally greater than any other shard. Sharding is conceptually similar to the partitioning in a relational database. . below summarizes the comparisons between MongoDB and relational databases.

Table 1. Comparison between MongoDB and RDBMS

<b>RDBMS</b>		<b>MongoDB</b>	
Table		Collection	
Row		BSON Document	
Index		Index	
Join		Embedding & Linking	
Partition		Shard	

### **Designing the Schema.**

In order to convert the PHP application to use MongoDB, the first task is to design a schema for the database. To create the schema for the MongoDB database, I analysed the relational schema to determine what data points are required and how this data is structured. This information can then be used to organize the data in to the format that fits the MongoDB engine. This schema will then be used as the basis for the analysis chapter where a number of changes will be implemented to evaluate the effect this has on the application code and functionality.

Using the concepts and the table from the previous section, the MongoDB schema was designed by converting the artefacts from the MySQL schema in to their equivalent MongoDB artefacts. Tables will map to collections, as shown in Table 2.

Table 2. Mapping Tables to Collections

Table	Collection
forum_users	users
forum_posts	posts
forum_forum	forum
forum_admin	forum_admin
forum_postcount	postcount
forum_access_levels	access_lvls

One of the main differences in NoSQL databases is that joins are not supported in the same way as in relational databases. Collections can be linked through fields at the application level but this will require an extra lookup for every link in the query. With this in mind, one of the goals when converting from relational to NoSQL should be to minimize linking where possible. Applying this concept to the proposed schema above, and referring to the schema diagram in Figure 11, a possible area where links can be eliminated immediately is the one-to-many relationship between forum\_users and forum\_postcounts, and forum\_users and forum\_access\_levels. The “one” side in these relationships contain only a minimal amount of data (one field in both cases) and can be easily absorbed in the users collection. This reduces the list of collections to:

- users
- posts
- forum
- forum\_admin



To assess the impact of this change, we must first deduce why the schema was designed like this in the relational database. In the case of the `forum_access_levels` table, a logical explanation for keeping this separate to the `forum_users` table is to prevent insert anomalies – new access levels can be added without the need to add a row to the users table. Furthermore, an application can easily retrieve a list of access levels (to populate a control for example) by issuing a `SELECT` query on the `forum_access_levels` table. Otherwise, retrieving such a list would require issuing a `SELECT DISTINCT` query on the `forum_users` table. De-normalizing these tables in the MongoDB schema raises the same concerns, however as will be demonstrated in the Analysis chapter, it is preferable to design the schema in this way than to have three collections with links.

When the collections have been identified, the next task is to add in the field names that will be used in the JSON document. Again, the field names from the relational schema were used as the basis for this. Tables 3 to 6 below list the details for each collection, including the data type, an example value and the purpose of the field.

Table 3. MongoDB users Schema

users			
Key	Data Type	Example	Purpose
<code>_id</code>	integer	234	Default ID Field
<code>name</code>	string	John Doe	Users full name
<code>email</code>	string	<a href="mailto:johndoe@foobar.com">johndoe@foobar.com</a>	Users email address
<code>date_joined</code>	date	2012-01-01T13:15:41	Date the user joined
<code>last_login</code>	date	2012-07-12T14:56:43	Date of last login
<code>signature</code>	string		Signature appended to each post
<code>post_count</code>	integer	1025	Number of posts
<code>access_lvl</code>	integer	3	Users access level

Table 4. MongoDB posts Schema

<b>posts</b>			
<b>Key</b>	<b>Data Type</b>	<b>Example</b>	<b>Purpose</b>
_id	integer	221	Default ID Field
author_id	integer	234	Link to the id of the poster (users)
forum_id	integer	4	Link to the id of the forum (forum)
subject	string	Recipe for success	The subject of the post
body	string	Work hard every day...	The body of the post
date_posted	date	2012-04-12T12:13:32	Date the post was entered
date_updated	date	2012-04-12T12:13:32	Date the post was updated
update_id	integer	65	Link to the id of the updater (users)
topic_id	integer	21	Identifier for the thread

Table 5. MongoDB forum Schema

<b>forum</b>			
<b>Key</b>	<b>Data Type</b>	<b>Example</b>	<b>Purpose</b>
_id	integer	2	Default ID Field
name	string	StackApps	Name of the forum
description	string	Posts from Stack Apps	A description of the forum
moderator_id	integer	-11	Link to the id of the moderator (users)

Table 6. MongoDB forum\_admin Schema

<b>forum_admin</b>			
<b>Key</b>	<b>Data Type</b>	<b>Example</b>	<b>Purpose</b>
_id	integer	2	Default ID Field
constant	string	titlebar	A reference name for the code
title	string	description	Key
value	string	StackApps	Value

### Data Types in MongoDB.

As MongoDB uses BSON documents, which is a “binary-encoded” serialization of JSON-like documents” (Creative Commons, n.d.), all of the standard JSON data types are

available. This includes string, integer, Boolean, double, null, array and object. There are also a number of additional types – date, object id, binary data, regular expression and code – which are each implemented in language specific ways depending on the driver being used (Chodorow & Merriman, 2011). In this case, the PHP driver supports saving and querying of data in all basic PHP data types, compound arrays and a number of other classes such as Date, Regex, Timestamp and DBRef, as well others listed at (The PHP Group, 2012).

### **Test Data**

For the purposes of testing the PHP application, both the relational database and the MongoDB database must be loaded with test data. Stack Exchange is a popular network of forums where users can get answers to questions on a range of topics from their peers. With over 87 sites in the network, each site produces a large amount of user generated data every day. This data is made publically available under the Creative Commons cc-wiki license (Atwood, 2009) every three months as XML “data dumps”. It is ideally suited for testing the PHP application in this study for the following reasons:

- It fits well with both the relational schema and the NoSQL schema, mainly because the Stack Exchange sites are bulletin board type applications themselves.
- Each three-monthly data dump is 4-7GB which provides a significant volume of data for performance testing purposes.
- The data is in XML format which can be easily parsed by the PHP application.

### **Test data format.**

The data from each site is packaged in a zip file containing six XML files:

- badges.xml
- comments.xml
- posts.xml
- posthistory.xml
- users.xml
- votes.xml

Initially, the posts.xml file and the users.xml file were used as the source data to load.

Each of these files has a defined list of fields which are described in **Error! Reference source not found.** and **Error! Reference source not found.** **Error! Reference source not found.**

Table 7. users.xml format

users.xml	
Field	Description
Id	Unique Identifier
Reputation	Users reputation score
CreationDate	Date the account was created
DisplayName	Users display name
EmailHash	A hash of the users email
LastAccessDate	Date of last login
WebsiteURL	URL of the users website
Location	Users location
Age	Age
AboutMe	A short profile
Views	Number of times user has viewed a post
UpVotes	The number of times user has up voted a post
DownVotes	The number of times user has down voted a post

Table 8. posts.xml format

posts.xml	
Field	Description
Id	Unique Identifier
PostTypeId	Integer identifying post type (1=Question, 2=Answer)
ParentID	The ID of the parent post (Only if post is an answer)
AcceptedAnswerId	The ID of the post accepted as answer (Only if post is a question)
CreationDate	Date the post was created
Score	Community score of the post
ViewCount	Number of times post has been viewed
Body	The body of the post
OwnerUserId	The user id of the poster
LastEditorUserId	The user id of the last editor
LastEditorDisplayName	The display name of the last editor
LastEditDate	The date the post was last edited
LastActivityDate	The date of the last activity on the post
CommunityOwnedDate	
ClosedDate	The date the post was closed
Title	The title of the post
Tags	A list of tags describing the post
AnswerCount	The number of answers (if post is a question)
CommentCount	The number of comments on the post
FavouriteCount	The number of users who have marked the post as a favourite

### Mapping the test data to the schema.

Having described the fields available, the next step is to map these fields to both the relational schema and NoSQL schema. Some fields will match exactly to those in each of these schemas while others require a level of improvisation. There are also other fields that will not be used in the mapping, but will have a role to play in the import code. **Error! Reference source not found.** and **Error! Reference source not found.** **Error! Reference source not found.** lists the mappings for the relational schema.

Table 9. Relational Schema Mapping: forum\_users

<b>forum_users</b>	
<b>Table field</b>	<b>users.xml field</b>
id	Id
date_joined	CreationDate
name	DisplayName
email	EmailHash
last_login	LastAccessDate
signature	WebsiteUrl
passwd	N/A (password will be randomly generated)
access_lvl	N/A (Access level will be randomly assigned)

Table 10. Relational Schema Mapping: forum\_posts

<b>forum_posts</b>	
<b>Table field</b>	<b>posts.xml field</b>
id	Id
topic_id	ParentID (if post type is 2, otherwise set to 0)
forum_id	N/A (Set manually depending on the forum posts are being imported for)
author_id	OwnerUserId
update_id	LastEditorUserId
date_posted	CreationDate
date_updated	LastEditDate
subject	Title
body	Body

The NoSQL schema will be mapped in a similar fashion. Each field in the specified collection will have a corresponding field in the xml files. The details are listed in

and **Error! Reference source not found.** below.

Table 11. NoSQL Schema Mapping: users

<b>users</b>	
<b>Collection field</b>	<b>users.xml field</b>
id	Id
date_joined	CreationDate
name	DisplayName
email	EmailHash
last_login	LastAccessDate
signature	WebsiteUrl
passwd	N/A (password will be randomly generated)
access_lvl	N/A (Access level will be randomly assigned)
post_count	Set to 0 for initial import

Table 12. NoSQL Schema Mapping: posts

<b>posts</b>	
<b>Table field</b>	<b>posts.xml field</b>
id	Id
topic_id	ParentID (if post type is 2, otherwise set to 0)
forum_id	N/A (Set manually depending on the forum posts are being imported for)
author_id	OwnerUserId
update_id	LastEditorUserId
date_posted	CreationDate
date_updated	LastEditDate
subject	Title
body	Body

### Loading the Test Data.

The PHP code for loading the test data to both the relational database and the MongoDB database is straightforward. The XML file is loaded in to the PHP function `simplexml_load_file`

which returns an object of class SimpleXMLElement, an array like structure which can be used to iterate through the individual elements of the XML file. As the code iterates through the object, the elements needed from the array are assigned to individual variables. In the case of the relational schema, I then executed an INSERT statement using the previously set variables as the parameters for the insert statement. For MongoDB, all the variables are added to an array and I used the MongoDB insert function provided by the PHP driver which takes the array as a parameter and adds the data to the specified collection as a BSON document. Appendix C and Appendix D contains the source code required for loading the test data to the MySQL database and the MongoDB database.

## **Deployment**

In this section, will describe how the application was set up from an infrastructure point of view. In order to deploy this application a web server is required to serve the PHP web pages and a database server is required to hold the data for the application. This server will initially host a MySQL database and subsequently a MongoDB database. An additional number of servers will be required to facilitate replication and failover.

### **Amazon EC2 Instances.**

Amazon Elastic Cloud Compute (EC2) was chosen as the platform to deploy all of the required servers for the application. EC2 is a pay as you go web service that allows new server instances to be deployed in minutes to Amazon's computing environment, making it an ideal service for scaling up or down as required (Amazon, 2012a). Instances are deployed to the cloud as preconfigured Amazon Machine Images (AMIs). There are a range of instance types available



offering varying amounts of computing power. In this study, all instances used were Micro instances which provide:

- 613MB of memory;
- Up to 2 ECUs;
- 32-bit or 64-bit platform;
- Elastic Block Store (EBS) storage which is off-instance storage that persists independently from the life of the instance itself (Amazon, 2012a).

### **Web Server Setup.**

One of the easiest ways to create a web server is to use a LAMP stack. This is an acronym for a server running a Linux operating system with Apache web server, MySQL Database and PHP components (or Python or Perl). To create a LAMP stack on Amazon EC2, one option would be to use the standard Linux instance and install each of the other three components individually. A more efficient way to create the LAMP stack is to use a preconfigured image from the Amazon Web Services Marketplace

The AWS Marketplace is an online shop which allows third parties to sell software to AWS users in the form of pre-configured Amazon Machine Images (AMIs) which can be deployed directly to the cloud as an EC2 instance (Amazon, n.d.-a). This allows software to be deployed and made publicly available in a matter of minutes. Furthermore, the image will be pre-tested and will be more reliable than a manually configured server. Rather than paying a license fee for the software, users are charged per hour for use of the software in addition to the standard EC2 charges.

There are many LAMP applications on AWS Marketplace. For this study I have chosen one provided by TurnKey Linux. There is no charge to use the software and it has an easy to use web management interface for configuration. To set up on EC2, the first task is to create an EC2 account. Once an account has been created, the instance can be launched from the AWS Marketplace web page (Amazon, n.d.-b), as shown in Figure 13 below.

The screenshot shows the AWS Marketplace interface for launching a TurnKey Linux LAMP Stack EC2 instance. The page is titled "Launch on EC2: LAMP Stack - Web Stack (MySQL) provided by TurnKey Linux". It features a "1-Click Launch" button and a "Launch with EC2 Console" button. The "1-Click Launch" section includes a "Click 'Accept Terms & Launch with 1-Click' to launch this software with the settings below" instruction. Below this, there are configuration settings for Version (11.3, released 04/03/2012), Region (EU West (Ireland)), EC2 Instance Type (Standard Micro (t1.micro)), Firewall Settings (turnkey-lamp-6283), and Key Pair (lamp). To the right, there is a "Monthly Estimate" of \$14.40, assuming 24x7 use over 30 days. Below this, a "Pricing Details" section shows hourly fees for different EC2 instance types in the EU West (Ireland) region. The pricing table is as follows:

EC2 Instance Type	Software	EC2	Total*
Standard Micro (t1.micro)	\$0.00/hr	\$0.02/hr	\$0.02/hr
Standard Small (m1.small)	\$0.015/hr	\$0.085/hr	\$0.10/hr
Standard Medium (m1.medium)	N/A	N/A	N/A
High-CPU Medium (c1.medium)	\$0.02/hr	\$0.186/hr	\$0.206/hr

Figure 13. Launching EC2 Instance from AWS Marketplace

This screen requests a number of configuration settings for the instance:

- Version: If there are different versions of the application a specific one can be chosen. In this case there is only one option.
- Region: The region that the instance is to run.

- **EC2 Instance Type:** The type of the instance. A micro instance was chosen in this case. The instance type can be changed at a later date if volume increases on the web server.
- **Firewall Settings:** This is a list of firewall rules (termed security groups in AWS) that define what traffic is allowed to connect to the server. This application provides a default security group which allows traffic through ports for its web management interface (12320-12322), MySQL connections (3306), SSH connections (22) and web traffic (80, 443).
- **Key pair:** This is the SSH public key which allows secure connections to be made to the server. This key must be created within the EC2 management console. A private key can then be downloaded to any clients connecting to the server to allow secure connections.

Once the configuration has been completed, clicking the “Accept Terms & Launch with 1-click” button will create the instance in EC2. After a few minutes the instance will be available and can be managed further from the EC2 interface, shown in Figure 14 below:

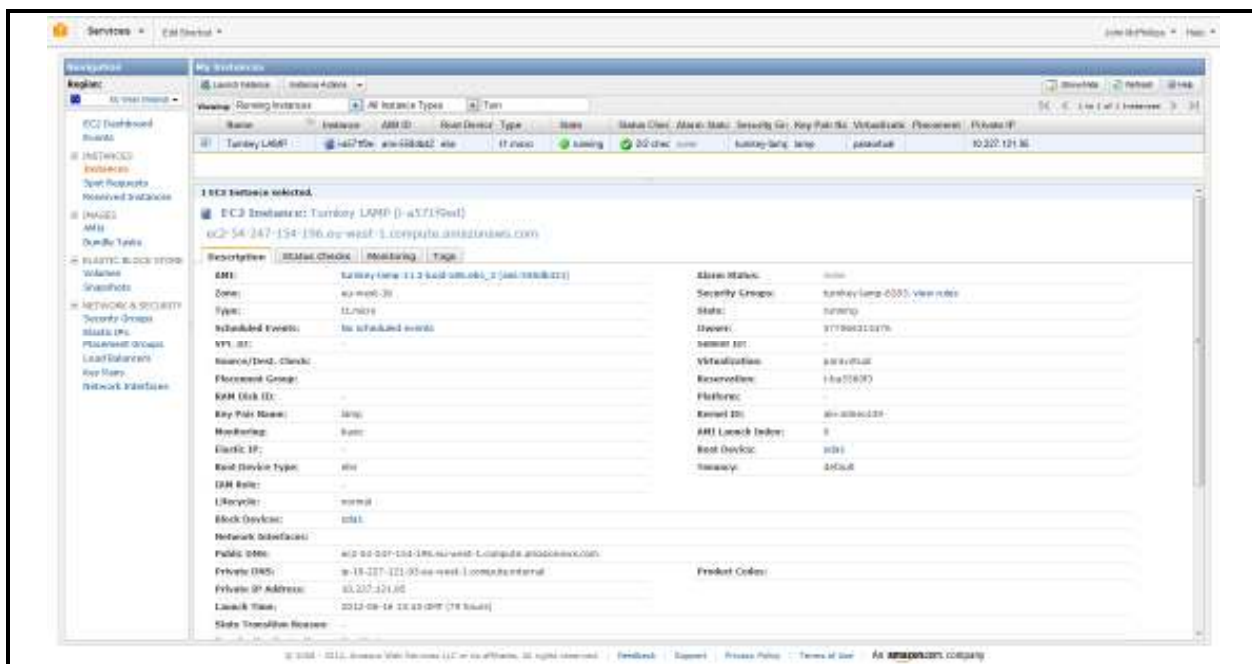


Figure 14. LAMP Instance running in EC2

From here, the various instance parameters can be assessed, such as the public DNS name of the server. This is needed for accessing the server through the management interface and through SSH. Figure 15 below shows the management interface provided by Turnkey which is accessed through port 12321.

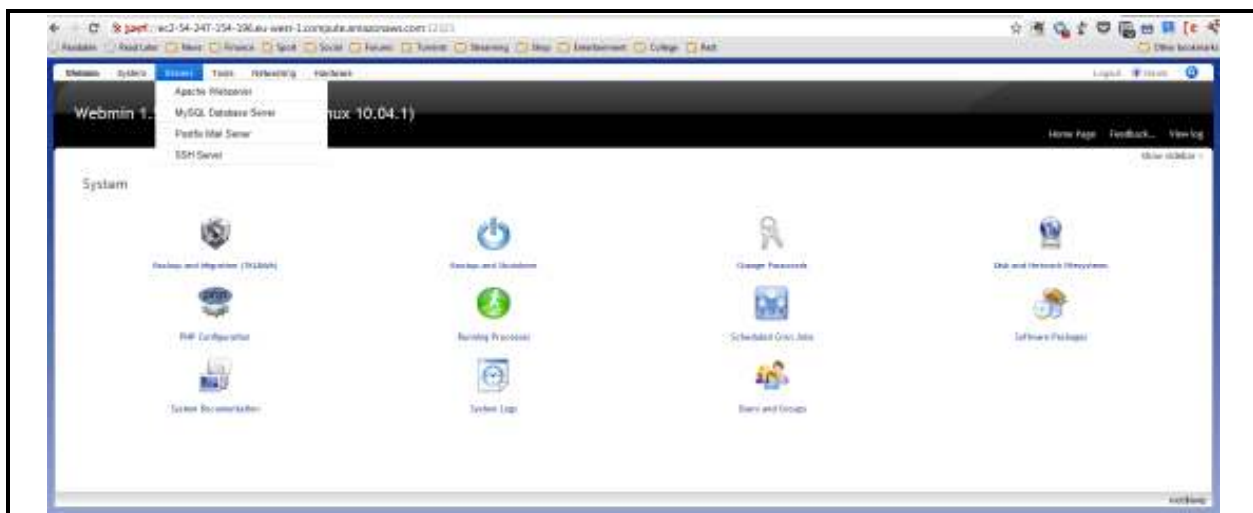


Figure 15. Accessing the LAMP Management Interface

This DNS name is fine for accessing management features, but it is not ideal for accessing any applications hosted on the server. A useful feature offered by Turnkey is the ability to create a hostname on their website which is linked to the instance running on EC2. To do this, details need to be registered with Turnkey. A hostname can then be entered for the instance. In this case, the hostname is `jmcp-lamp.tklapp.com`, highlighted in Figure 16 below. This interface can also be used to create an instance as an alternative to using the AWS marketplace.

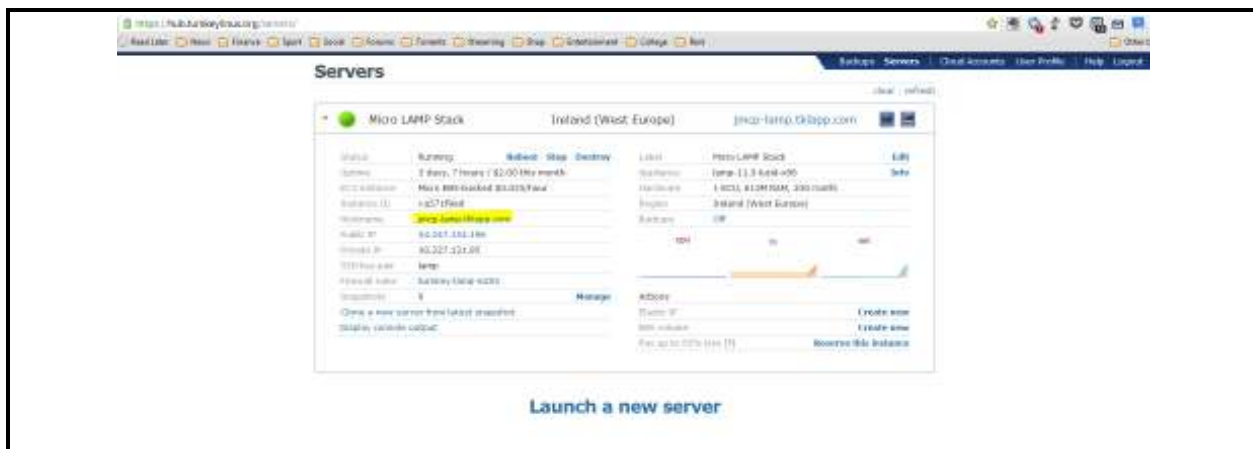


Figure 16. Defining a hostname

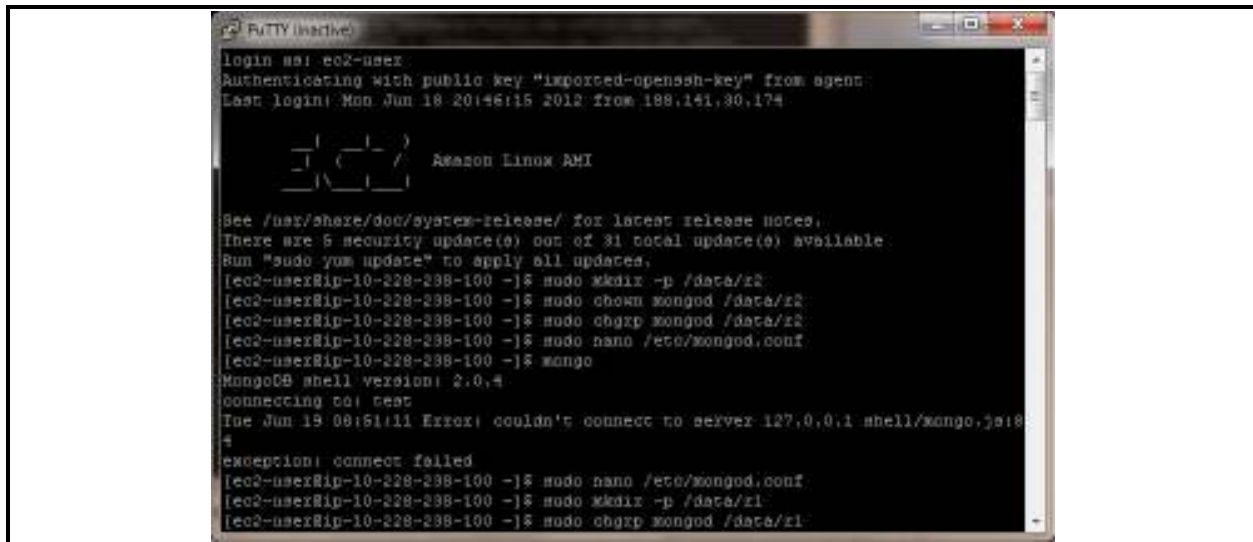
### MongoDB Setup.

To create the MongoDB instances, I went again to the AWS Marketplace. The creators of MongoDB, 10Gen, provide an AMI in the marketplace to allow consumers to easily get started with using MongoDB. The process is the same as what was used earlier to create the LAMP instance except this time it will be repeated two more times to make three instances in total. Once the instances are launched they can be managed from the EC2 management interface.

Because the option is not available to assign a hostname in this case, three Elastic IP addresses will be assigned to each of the MongoDB instances. Elastic IP addresses are static IP addresses which can be assigned to any EC2 instance (Amazon, 2012b). If an instance is restarted in EC2, the public DNS name will more than likely not be the same as it was previously. Using an Elastic IP address provides a more consistent means to reference an instance.

Once the instances are up and running, some additional configuration is required to start the process on each instance and to ensure the process starts when the server is restarted (10Gen,

n.d.-a). This can be done through an SSH connection, as shown in Figure 17 below. The same SSH key pair created earlier was used for authentication.



```
login as: ec2-user
Authenticating with public key "imported-openssh-key" from agent:
Last login: Mon Jun 18 20:46:15 2012 from 188.141.30.174

 _ _ _ _ _
| | | | |
|_|_|_|_|_|_ Amazon Linux AMI

See /usr/share/doc/system-release/ for latest release notes.
There are 5 security update(s) out of 31 total update(s) available.
Run "sudo yum update" to apply all updates.
[ec2-user@ip-10-228-238-100 ~]$ sudo mkdir -p /data/r2
[ec2-user@ip-10-228-238-100 ~]$ sudo chown mongod /data/r2
[ec2-user@ip-10-228-238-100 ~]$ sudo chgrp mongod /data/r2
[ec2-user@ip-10-228-238-100 ~]$ sudo nano /etc/mongod.conf
[ec2-user@ip-10-228-238-100 ~]$ mongo
MongoDB shell version: 2.0.4
connecting to: test
Tue Jun 19 08:51:11 Error: couldn't connect to server 127.0.0.1 shell/mongo.js:18
4
exception: connect failed
[ec2-user@ip-10-228-238-100 ~]$ sudo nano /etc/mongod.conf
[ec2-user@ip-10-228-238-100 ~]$ sudo mkdir -p /data/r1
[ec2-user@ip-10-228-238-100 ~]$ sudo chgrp mongod /data/r1
```

Figure 17. Connecting through SSH

To set up the MongoDB instances in a replica set, they must be able to communicate with each other. Amazon EC2 instances are all given a private IP address for internal communication between instances. To simplify configuration, an entry for each private IP address was added to the `/etc/hosts` file. This allows communication to be set up through a host name rather than using the IP address.

The default security group which was defined when the instance was created only allows SSH connections through port 22. MongoDB instances communicate through node 27017 by default so this was added to the security group. There is also a web interface for MongoDB which runs on port 28017, this was also added to the security group. The complete security group details are shown in Figure 18 below.

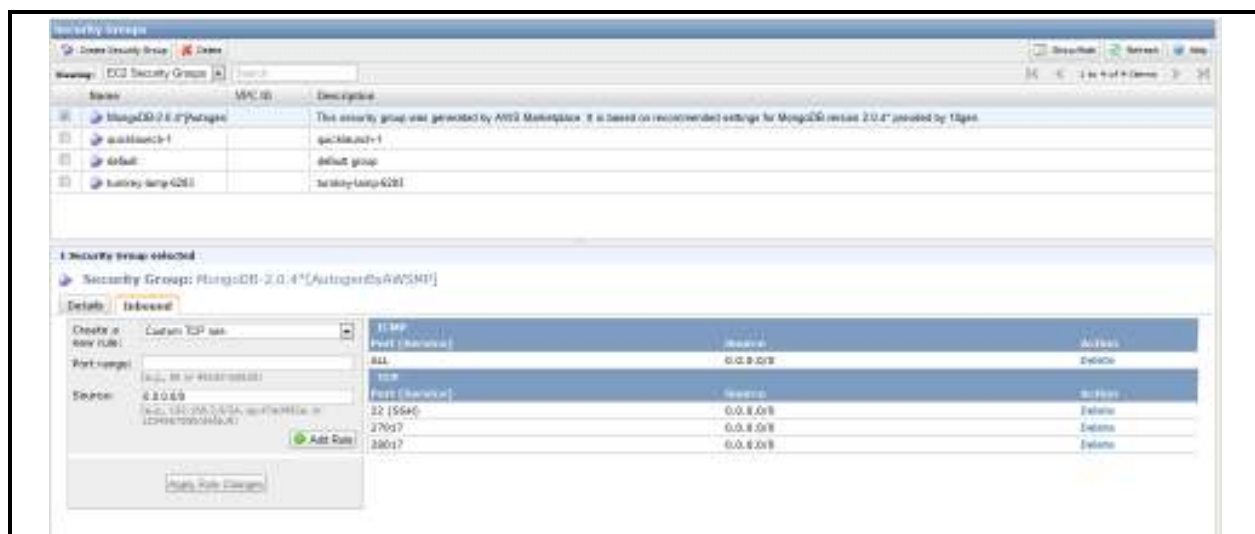


Figure 18. Security Group for MongoDB

The replica set can then be created using the process outlined in the MongoDB documentation (10Gen, n.d.-b) and can be verified through the web interface, as shown in Figure 19 below.



Figure 19. MongoDB Web Interface

**Chapter Summary**

In this chapter, I have presented a PHP bulletin board application which uses a MySQL database at its back end for data storage. The schema of this database was described in detail, with a view to converting the schema to a NoSQL system. I have also presented MongoDB as a candidate NoSQL system to facilitate this conversion. By making a comparison between MongoDB and RDBMS artefacts, a MongoDB schema was designed to replace the relational schema already in existence. This schema will now form the basis for our analysis in the next chapter where the PHP application will be converted to use our newly created MongoDB schema. I have also demonstrated how the application and the MongoDB database was deployed to the Amazon EC2 web service.



## **Chapter 4 – Analysis**

In this chapter, I will begin by analyzing the level of effort required to convert the PHP application presented in the previous chapter from using a MySQL back end to a MongoDB back end. Throughout the conversion process, I will analyze the differences between the relational and non-relational systems in use. I will not discuss converting the entire application as many aspects of the conversion are similar. Instead, a subset of pages from the PHP code will be chosen to try to cover all of the MongoDB concepts required to carry out a complete conversion. For reference, the source code for all pages discussed in this chapter can be found in Appendix A and Appendix B. Once the conversion process is complete, a number of performance tests will be carried out on both the MySQL and MongoDB versions of the application in order to compare and contrast the performance characteristics of both systems. The chapter will conclude with an implementation of a Map-Reduce function – the NoSQL approach to distributed queries and aggregation.

### **Converting the PHP Code**

The initial approach taken to convert the application is to analyze the database access code in each of the pages of the application and try to create equivalent code to access the MongoDB database. The steps taken were:

- Check each PHP page for SQL statements;
- Record each SQL statement in the page;
- Write down a logical sentence of what function the SQL statement is carrying out;
- Try to write data access code for MongoDB that produces the same result as the SQL statement;

- If it is not possible to create equivalent MongoDB code, re-evaluate the MongoDB schema design.

This will produce a first revision of our new PHP application which will be used for further analysis and testing later. As each individual SQL statement is converted, I will begin to develop a standard methodology in order to apply the approach to other relational based applications.

To demonstrate the methodology in use, several of the PHP pages in our sample bulletin board application will be converted. There are many candidate pages in this application that require conversion. A subset of pages was chosen for this study. These pages have been carefully selected in order to demonstrate a number of different query options available in MongoDB.

#### **Converting read queries.**

The first candidate page in the application that was converted is functions.php. This file is a helper file containing functions which other pages in the application can use. Many of those functions also contain SQL statements to retrieve information from the database. For each SQL statement found in the file a table of information should be constructed, such as Table 13, which details which PHP page the query is found in, the function within that page, the return type of the function (if applicable) and an explanation of the purpose of the query.

Table 13. SQL Analysis for functions.php

<b>File:</b>	functions.php
<b>Function:</b>	getForum
<b>Function Return Type:</b>	Integer
<b>SQL Statement:</b>	SELECT forum_name as name, forum_desc as description, forum_moderator as forum_mod FROM forum_forum WHERE id = \$id;
<b>SQL Explanation:</b>	Find the forum name, forum description and forum moderator id for a given forum id

From a first pass analysis of this statement it is clear that this is a read operation. There are two functions available in MongoDB for read operations: `find()` and `findOne()`. A filter can be included by passing an array to either of these functions to retrieve only the data that matches the array. This is the equivalent of the `WHERE` clause in the SQL statement. By default, `find()` and `findOne()` will return all fields in the document or documents that match the criteria. To retrieve only a number of specified fields, a second array can be passed to the function containing the fields that are required. The `find()` function returns a cursor to the data which can be iterated over, while the `findOne()` function will retrieve only one document from the collection, regardless of how many documents satisfy the criteria, and the result is automatically returned as an array in the PHP code. In this case `findOne` can be used since the search is for only one document and also because it eliminates the extra step of transferring the result to an array that would be required if `find()` were used. We can be sure that only one document will be found because we are filtering on the `_id` field which must be unique. The resultant PHP code is given in Figure 20.

```
function getForum($id) {
    global $m;
    global $db;

    $coll = new MongoClient($db, 'forum');
    $doc = $coll->findOne(array("_id" => $id));

    return $doc;
}
```

Figure 20. Converted PHP code for the getForum() function

This first conversion highlights one of the benefits of using MongoDB. The `findOne` method returns an array of one row of information (to use relational terminology) from the specified collection. In the MySQL code, the result of the SQL statement is returned as a resource object and a further processing step is required to transpose this result in to an array, using the function `mysql_fetch_array`. Table 14 below displays a comparison of the code required for this operation with MySQL and MongoDB. Clearly, the MongoDB code is more straightforward in this particular case.

Table 14. Comparison of PHP code for the getForum function

MySQL	MongoDB
<pre>\$sql = "SELECT forum_name as name, forum_desc as description, forum_moderator as forum_mod FROM forum_forum WHERE id = " . \$id;  \$result = mysql_query(\$sql)     or die(mysql_error() . "&lt;br&gt;" . \$sql); \$row = mysql_fetch_array(\$result);  return \$row;</pre>	<pre>\$coll = new MongoClient(\$db, 'forum'); \$doc = \$coll-&gt;findOne(array("_id" =&gt; \$id));  return \$doc;</pre>

The next page that requires conversion is the `index.php` page, the default page for the application. This page displays a list of the forums and some related information that are

currently available for browsing. There is only one SQL statement on this page, its details are captured in Table 15.

Table 15. SQL Analysis for index.php

<b>File:</b>	index.php
<b>Function:</b>	N/A
<b>Function Return Type:</b>	N/A
<b>SQL Statement:</b>	SELECT f.id as id, f.forum_name as forum, f.forum_desc as description, count(forum_id) as threads, u.name as forum_mod FROM forum_forum f LEFT JOIN forum_posts p ON f.id = p.forum_id AND p.topic_id=0 LEFT JOIN forum_users u ON f.forum_moderator = u.id GROUP BY f.id
<b>SQL Explanation:</b>	Find each forum id, forum name, a count of threads and the forum moderator

Replicating this statement in MongoDB presents a number of challenges. The first element that jumps out from this statement is the presence of two joins. It was noted earlier that joins are not supported in MongoDB. This does not mean that an equivalent query cannot be generated for this statement, but it does mean that the data cannot be retrieved with one query. Instead an initial query must be created with the fields from one collection and then loop through this result and retrieve the data from each collection that needs to be joined. This is known as client-side linking, owing to the fact that collections are joined by the client rather than on the server as is the case with relational database systems.

The second point to note with this query is the aggregation required to determine the number of threads in the forum. The query retrieves this by getting a count of the number of posts with a distinct topic id for each forum id. In MongoDB, this can be achieved by running the distinct command with a given key and search criteria. In this case, the key is the topic\_id and the filter criteria is the forum\_id.



```

$users_coll = new MongoClient($db, 'users');
$forum_coll = new MongoClient($db, 'forum');

// Find all forums
$cursor = $forum_coll->find();

// For each forum in the collection, get the number of threads
// from the posts collection and the moderator from the users collection.
// Construct a new array with the required fields.
foreach($cursor as $forum){
    // Get count of threads in the forum
    $threads = $db->command(
        array(
            "distinct" => "posts",
            "key" => "topic_id",
            "query" => array("forum_id"=>$forum['_id'])
        )
    );

    // Get the moderator name
    $moderator = $users_coll->findOne(array("_id"=>(int)$forum['moderator']));

    // Construct the row
    $row = array("forum_id"=>$forum['_id'], "forum_name"=>$forum['forum_name'],
        "forum_description"=>$forum['description'],
        "forum_moderator"=>$moderator['name'],
        "threads"=>count($threads['values']));

    // Push the row on to an array
    array_push($forum_row, $row);
}

```

Annotations in the image:

- Set up references to each collection required (points to the collection initialization lines)
- Find all forums (points to the `$cursor = $forum_coll->find();` line)
- Aggregate command to count the number of threads (points to the `$db->command()` block)
- Build an array with the required result set (points to the `array_push()` line)

Figure 21. Code for index.php query

The PHP code required for this is shown in Figure 21. We can clearly see here the initial query which finds all of the forums, followed by the iteration through this collection to retrieve the number of threads in each forum and the moderator of the forum. This is in contrast to the SQL query where the data is joined on the server and returned to the client. The SQL statement

requires, at worst, one scan of each table involved in the query to join the data. Analysing the code for the MongoDB query, we can see that for each forum that exists, one scan is required of both the posts collection and the users collection. This can have a significant impact on performance and must be considered carefully in the schema design. To determine the impact a join type query will have on the design, the number of collection lookups required can be determined using the formula below:

$$N_s = 1 + (N_{irf} \times N_{jc})$$

where:

$N_s$  = Number of collection scans

$N_{irf}$  = Number of initial records found

$N_{jc}$  = Number of collections the initial collection is joined to

Applying this to the index.php page, if our bulletin board has 12 forums, this query will require 25 find commands in total to retrieve the information required: 1 initial scan of the forum collection plus 12 scans each of the users collections and the posts collection. This particular example results in a small amount of find commands and the query time will be negligible. However, if the forum collection became very large, or if it was necessary to add another collection to the query, this figure could grow rapidly and have a detrimental effect on application performance. This situation arises in the next page that will be converted.

The viewforum.php page is responsible for retrieving all the starting posts for each topic and displaying the topic subject, along with the number of replies and the date of the last post. The SQL analysis is given in Table 16.

Table 16. SQL Analysis for viewforum.php

<b>File:</b>	viewforum.php
<b>Function:</b>	N/A
<b>Function Return Type:</b>	N/A
<b>SQL Statement:</b>	<pre> CREATE TEMPORARY TABLE tmp (   topic_id INT(11) NOT NULL DEFAULT 0,   postdate datetime NOT NULL default '0000-00-00 00:00:00'  INSERT INTO tmp SELECT topic_id, MAX(date_posted)   FROM forum_posts   WHERE forum_id = \$forumid   AND topic_id &gt; 0   GROUP BY topic_id  SELECT  t.id as topic_id, t.subject as t_subject,         u.name as t_author, count(p.id) as numreplies,         t.date_posted as t_posted, tmp.postdate as re_posted   FROM forum_users u   JOIN forum_posts t     ON t.author_id = u.id   LEFT JOIN tmp     ON t.id = tmp.topic_id   LEFT JOIN forum_posts p     ON p.topic_id = t.id   WHERE t.forum_id = \$forumid   AND t.topic_id = 0   GROUP BY t.id   ORDER BY re_posted DESC </pre>
<b>SQL Explanation:</b>	Find the topic_id, subject, author name, number of replies, date posted and date of last post of each starting post for a given forum id

This is a quite complex query that involves the use of a temporary table in order to first determine the date of the most recent post for a thread. This temporary table is then joined with



the forum\_posts table, which in turn is joined with the forum\_users table. This enables all of the required fields to be retrieved for the given forum id. There is also some aggregation in this query: a count of the number of posts for each topic\_id (i.e. the number of replies) and the maximum post date for each topic\_id (i.e. the date of the most recent post).

To convert this query to use MongoDB the same methodology defined in the previous example was used. The main table involved in the query is forum\_posts as this is the table that contains all of the posts that comprise the forum. Note that there are three joins in the query. The forum\_users table is joined with the author\_id field in order to get the thread author, the temporary table is joined on the id field to include the date of the last post, and there is a self join in order to count the number of replies. This means that one initial query of the posts collection is required and then three subsequent queries for each record found. The initial query will find the subject and the post date, along with the join fields required to retrieve the rest of the fields, namely, topic\_id and author\_id. Retrieving the post author and the number of replies is trivial and requires one line of code to query the users collection and the posts collection respectively. Retrieving the date of the last post however is not so straightforward.

In order to find the last post date, we must first determine if there are any replies to the post. If there are no replies, then the last post date is the post date of the parent post. If there is only one reply, then the last post date is the post date of this reply. If there are multiple replies, then the last post date is the most recent date of this collection of replies. Represented in pseudo-code:

*retrieve all date\_posted records for the given topic\_id*

*if there are no records*

*last\_post\_date = date\_posted of parent post*

*else if there is one record*

*last\_post\_date = date\_posted of this record*

*else there are multiple records*

*last\_post\_date = maxium(date\_posted) of the records found*

The corresponding PHP implementation is shown in Figure 22.

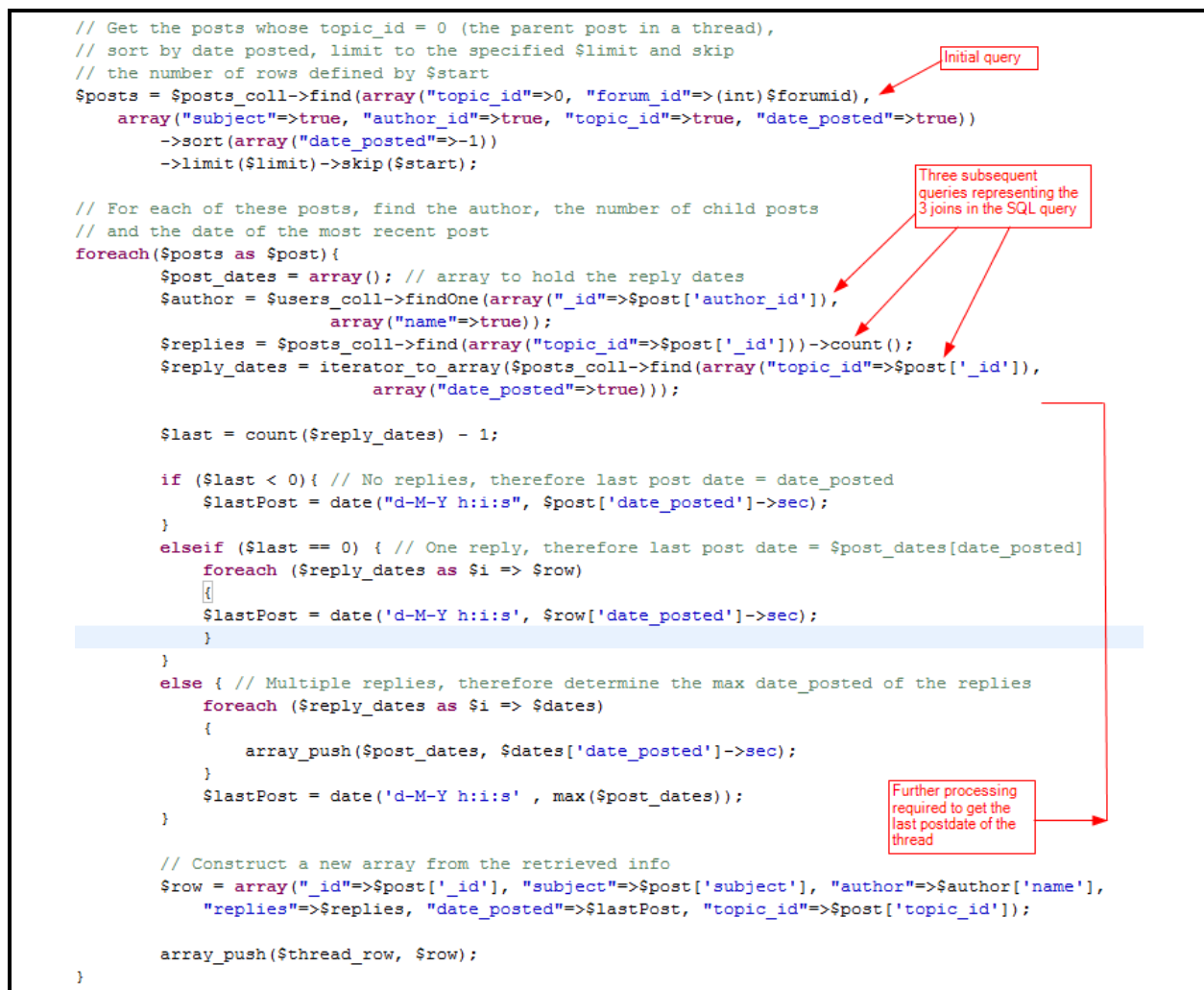


Figure 22. Converted PHP code for viewforum.php

On examination of this code, the first thing that is apparent is the complexity. Although the SQL query in the relational example was by no means straightforward, once the data had been retrieved from the database there was very little processing required at the code level to display this information. The other point to note is the number of total queries required to retrieve the data. The initial query requires one scan of the posts collection, but then to retrieve the number of replies and the last post date the posts table must be scanned twice for each record found, as well one scan of the users table for each record. Using the formula defined in the previous example, the number of scans required can be determined if a forum had 10,000 posts.

$$N_s = 1 + (10,000 \times 3) = 30,001$$

It must also be considered here that two of the collection scans within the for loop are also of the posts collection, which is the collection that will contain the most amount of documents and therefore the longest scan times.

### **Converting the insert/update/delete queries.**

Converting the sections of the application that deal with inserting, updating and deleting data is relatively straightforward when compared to the read operations. The bulletin board application encapsulates all update/insert operations in 4 files:

- transact-admin.php,
- transact-affirm.php,
- transact-post.php,
- transact-user.php.

The transact-post.php file will be examined further in this section, as this file contains all interactions relating to inserting/updating posts on the forum, however, the methodology can be applied to all SQL statements carrying out insert/update/delete functions. The transact-post.php

file performs a number of different SQL operations depending on the parameter sent to the page.

This analysis will deal with each of these queries separately.

### *Inserting a new post.*

The first operation is to insert a new post to the database. The analysis for the SQL statement is shown in Table 17.

Table 17. SQL Analysis of transact-post.php insert function

<b>File:</b>	transact-post.php
<b>Function:</b>	Insert new post
<b>Function Return Type:</b>	N/A
<b>SQL Statement:</b>	<pre> INSERT INTO forum_posts VALUES (     NULL, \$_POST['topic_id'],     \$_POST['forum_id'] ,     \$_SESSION['user_id'],     0, date("Y-m-d H:i:s",time()),     0, \$_POST['subject'],     \$_POST['body'];  INSERT IGNORE INTO forum_postcount VALUES ( \$_SESSION['user_id'], 0);  UPDATE forum_postcount SET count = count + 1 WHERE user_id = \$_SESSION['user_id']; </pre>
<b>SQL Explanation:</b>	<p>Insert a new row in to the forum_posts table with the given parameters and update the post count for the user.</p>

This query performs a straightforward insert to the forum\_posts table and then updates the forum\_postcount table. By using INSERT IGNORE, the query handles the scenario of the user's first post. If the user exists, the INSERT statement is ignored and the UPDATE statement proceeds to increment the count field. If the user does not exist, the INSERT statement will

succeed and enter 0 for the post count and then proceed to increment this to 1 with the following UPDATE statement.

To convert this functionality for the MongoDB database, an array is required that has all the required variables and then this array is passed to the MongoDB insert function, as shown in Figure 23. Converted PHP code for transact-post.php insert function. The second parameter passed to this command specifies that this command should be executed in safe mode, which tells the PHP script to wait for a response from the database. If the insert fails, the script will throw an exception which can be handled by the code. If this parameter is not explicitly set to true, the PHP code will continue regardless of the result of the insert command. Unless insert speed is critically important for the application, this parameter should always be set to true. Otherwise any issues with the database will not be captured and the command will appear to execute successfully.

```

$newPost = array("date_posted"=>new MongoDBDate(strtotime(date("Y-m-d H:i:s", time()))),
    "topic_id"=>$_POST['topic_id'],
    "forum_id"=>$_POST['forum_id'],
    "date_updated"=>"",
    "body"=>$_POST['body'],
    "subject"=>$_POST['subject'],
    "author_id"=>$_SESSION['user_id'] );

// insert the new post
$db->posts->insert($newPost, true);

// 'upsert' the post count for the user
$db->users->update(array("id"=>$_SESSION['user_id']),
    array('$inc'=>array("post_count"=>true)), true);

```

Figure 23. Converted PHP code for transact-post.php insert function

Recall that in the schema design for MongoDB, the post count was moved to the users collection, therefore this field needs to be updated with the MongoDB update command. The first parameter for this command corresponds to the WHERE clause in the SQL statement. The

second parameter contains the operation to perform, in this case an increment of the post count field. Passing 'true' as the third parameter in this command specifies that the statement is an "upsert" statement. This means that if the user exists, perform an update, and if the user does not exist, perform an insert. This functionality means that the INSERT IGNORE statement in the MySQL code does not have to be replicated and where two statements were required with SQL, only one is required with MongoDB.

Another point to note here is that with the SQL statement, a number of dummy values are required for fields that are not relevant to a new post such as the update\_id and date\_updated fields. Because there is no schema required in the MongoDB database, the document can be inserted without these fields at this point and added later if required.

### *Updating an existing post.*

Table 18 details the SQL statement from transact-post.php that implements the post update functionality. This statement simply updates one post in the forum\_posts table with the provided subject, update user id, post body and the current date.

Table 18. SQL Analysis for transact-post.php update function

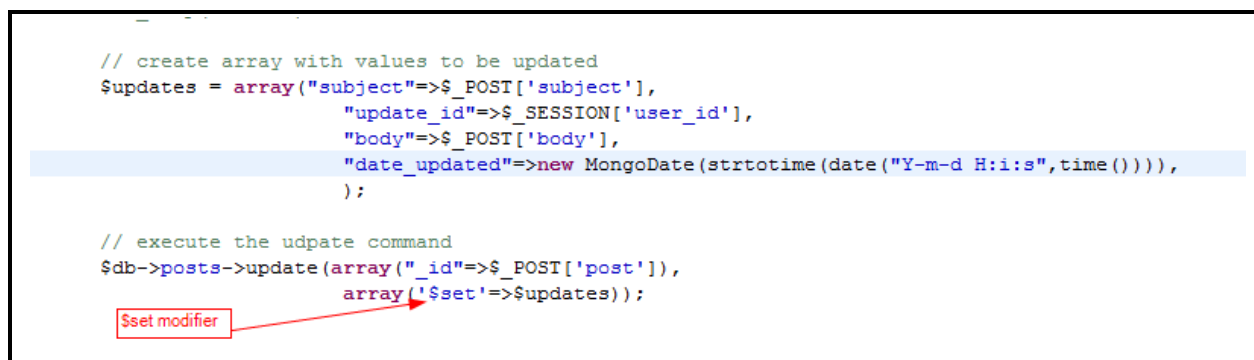
<b>File:</b>	transact-post.php
<b>Function:</b>	Update existing post
<b>Function Return Type:</b>	N/A
<b>SQL Statement:</b>	<pre>UPDATE forum_posts SET subject= \$_POST['subject'] ,     update_id= \$_SESSION['user_id'] ,     body= \$_POST['body'] ,     date_updated=date("Y-m-d H:i:s",time()) WHERE id= \$_POST['post'];</pre>
<b>SQL Explanation:</b>	Update the subject, user id, body and date updated fields for the

---

given post id.

---

The procedure for converting this statement to use the MongoDB database is similar to the update operation to the post count that has been discussed already. Again, the MongoDB update command is used, but on this occasion, the \$set modifier is used rather than the \$inc modifier. The fields that need to be updated are added to an array and this array is passed to the \$set modifier in the update command. The first parameter again corresponds to the where clause in the SQL statement. The converted code is shown in Figure 24.



```
// create array with values to be updated
$update = array("subject"=>$_POST['subject'],
               "update_id"=>$_SESSION['user_id'],
               "body"=>$_POST['body'],
               "date_updated"=>new MongoClient(date("Y-m-d H:i:s", time())));

// execute the update command
$db->posts->update(array("_id"=>$_POST['post']),
                  array('$set'=>$update));
```

Figure 24. Converted PHP code for transact-post.php update function

Because of the flexible schema provided by MongoDB, care must be taken when using the update command. For instance, if the 'date\_updated' field in the array above was misspelled as 'date\_update', the update command will still succeed and the document will either have no 'date\_updated' field or it will have both a 'date\_updated' field and a 'date\_update' field. Any queries that use the 'date\_updated' field would then give unexpected results.

Another caveat to the update command is that, by default, only the first record that matches the criteria gets updated. This is something that is likely to catch out SQL developers.

For example, to update the subject field for all posts with topic\_id 39037, you expect the command below to do this:

```
$db->posts->update(array("topic_id"=>39037),
    array('$set'=>array("subject"=>"New subject")));
```

However, this will only update one record. To update all records, a value of true must be given as a fourth parameter to the update command:

```
$db->posts->update(array("topic_id"=>39037),
    array('$set'=>array("subject"=>"New subject")), , true);
```

### *Deleting a post.*

Table 19 outlines the SQL required to delete a post. The MongoDB implementation of the delete operation is very similar to that required for an insert. The id of the post required to be deleted is passed to the MongoDB remove operation, as shown in Figure 25.

Table 19. SQL Analysis of transact-post.php delete function

<b>File:</b>	transact-post.php
<b>Function:</b>	Delete post
<b>Function Return Type:</b>	N/A
<b>SQL Statement:</b>	DELETE FROM forum_posts WHERE id= \$_REQUEST['post'];
<b>SQL Explanation:</b>	Delete from the forum_posts table where the id matches the given id.

```
$db->posts->remove(array("_id"=>$_REQUEST['post']));
```

Figure 25. Converted PHP code for transact-post.php delete function



## **Performance and Optimization**

One of the most important aspects of a database system is the speed with which data can be retrieved relative to the hardware used to host the databases. This section will present a number of performance measurements that have been taken to assess how the NoSQL system performs relative to the relational system. These measurements will also be used to try to optimize the data retrieval code in the PHP/MongoDB application.

### **Hardware setup.**

In order to compare the performance of two or more database systems, it is important to ensure that all systems are operating on the same hardware setup and that there is minimal external influence on the database that may affect any results taken. Both the MySQL and the MongoDB databases were set up on Amazon Elastic Cloud Compute instances with no other applications competing for resources. As discussed in Chapter 2, one of the major claims of NoSQL databases is their ability to run on basic hardware. To test this claim, a micro instance was chosen as the instance type for both the MongoDB server and the MySQL server. This is the lowest spec instance type available in Amazon EC2. Its specifications were detailed in the previous chapter.

### **Method used for measurement taking.**

In order to capture the execution times of the queries in the PHP application, additional code was added to each PHP page containing either an SQL or a MongoDB query. At the start and end point of each query, a variable was assigned with the current time. The total execution time for the query is then the difference between these two variables. An example of this implementation is shown in Figure 26. This calculation will capture both the query duration and the fetch time required to return the result to the web server. This fetch time is negligible in this

case as the information is being transferred across the internal Amazon EC2 high speed network. Furthermore, the accuracy of the calculation was verified against results in the MySQL long running query log and the MongoDB query log, both of which report the execution time of queries in milliseconds.

```
$time_start = microtime(true);  
$result = mysql_query($sql)  
    or die(mysql_error() . "<br>" . $sql);  
  
$time_end = microtime(true);  
$time = $time_end - $time_start;  
echo "Data Retrieved. Took " . $time . "seconds <br>"
```

Figure 26. PHP code to measure MySQL query time

A threshold of 30 seconds was chosen as the maximum acceptable time for a query to execute. This number was chosen because it is a common timeout limit for web requests. If the data is not returned within this timeframe, the web server will normally return a timeout error. However, in a real life situation, query times of much less than 30 seconds would be required for an application such as this.

Furthermore, these tests do not consider concurrent requests for data as there is only one statement executing at a time on the database. In a real world scenario, the database would be handling many requests at the same time and the execution times of these tests will likely be much longer (or potentially shorter in MySQL as the same query may have been executed by a different connection which would leave the query result in the query cache).

### **Comparison of MySQL and MongoDB write performance.**

This section will provide a comparison of the write performance of MySQL and MongoDB by bulk loading data to both database systems from XML files and measuring the execution time for each file.

As discussed in Chapter 3, test data was loaded to both the MySQL database and the MongoDB database from XML files containing data from the StackExchange network of question and answer forums. Data was loaded to the `forum_users` and `forum_posts` tables in MySQL and to the `users` and `posts` collection in MongoDB. All foreign key references were disabled in the MySQL database. In order to effectively load the data, the respective XML files had to be divided into smaller files of equal size. The `posts.xml` file was divided into 336 files of 25.5MB each, containing approximately 20,000 rows per file. The `users.xml` file was divided into 15 files of 14.5MB each, containing approximately 60,000 rows per file.

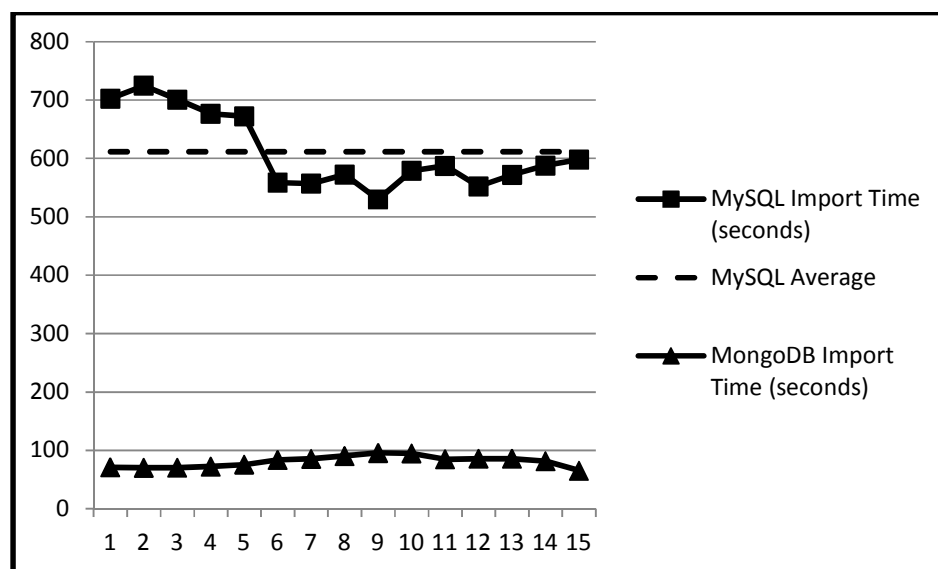


Figure 27. Comparison of import times for `users.xml` files

The import times for both MySQL and MongoDB are shown in Figure 27. The average time for MySQL was 611.24 seconds, while the MongoDB average time was 80.72 seconds, almost 8 times faster. This test highlights the write advantage that MongoDB has over MySQL. For the `posts.xml` files, the results are even more in favour of MongoDB which gave an average

time of only 8.93 seconds per file compared to 248.15 seconds for MySQL. These results are shown in Figure 28.

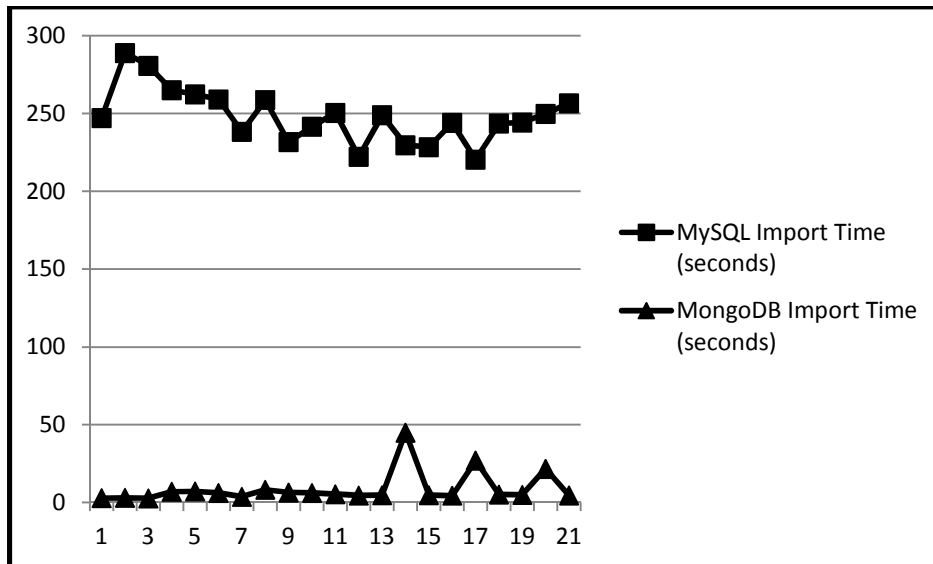


Figure 28. Comparison of import times for posts.xml files

### **Comparison of MySQL and MongoDB read performance.**

After converting the PHP application, it may be useful to compare the performance of the MySQL version of the application to the new version using MongoDB. Of the pages converted, the viewforum.php page is the most resource intensive, in terms of retrieving data from the database. Therefore this page will be used as the basis for the performance comparison.

#### ***Optimizing the MySQL query in viewforum.php.***

In order to provide a fair reflection on the performance of the MySQL database in viewforum.php, a number of optimizations were made with the intention of obtaining the best possible query times. Recalling from the SQL Analysis in Table 16, there are two tables involved in the query: forum\_posts and forum\_users. The first step taken to optimize the query is to create

indexes on the fields used in the WHERE clause and any fields used for joins, resulting in the following indexes:

- forum\_posts: id (PRIMARY KEY), topic\_id, forum\_id, author\_id, date\_posted
- forum\_users: id(PRIMARY KEY)

The original query for this page uses a temporary table to store the last post date for each topic id. This table is then joined to the forum\_posts table in order to generate the required result set. Joining the temporary table here will be a slow operation unless an index is created on the topic\_id field in the temporary table. However, creating this index is also a time consuming operation and will still impact the overall query time for this result set.

A better approach to this is to use a view to retrieve the last post date portion of the query and then join this view to the forum\_posts table instead. The view can utilize the existing indexes on the forum\_posts table. The definition for this view is given in Figure 29.

```
CREATE VIEW v_replies AS
SELECT topic_id, MAX(date_posted) postdate
FROM forum_posts
WHERE forum_id = $forumid
AND topic_id > 0
GROUP BY topic_id;
```

Figure 29. Create view statement

The complete SQL statement is then revised to that shown in Figure 30.

```
$sql = "SELECT SQL_CALC_FOUND_ROWS
        t.id as topic_id, t.subject as t_subject,
        u.name as t_author, count(p.id) as numreplies,
        t.date_posted as t_posted, v.postdate as re_posted
    FROM
        forum_users u
    JOIN forum_posts t
    ON t.author_id = u.id
    LEFT JOIN v_replies v
    ON t.id = v.topic_id
    LEFT JOIN forum_posts p
    ON p.topic_id = t.id
    WHERE t.forum_id = 4
    AND t.topic_id = 0
    GROUP BY t.id
    ORDER BY re_posted DESC
    LIMIT $start, $limit";
```

Figure 30. Revised SQL query for viewforum.php

### ***Test specifications.***

In order to compare the read performance of MySQL and MongoDB, a number of test specifications were devised. Each test involved measuring the execution time of the query from the viewforum.php page, varying the number of rows in the users and posts tables/collections. For each test, the query was executed 10 times and the result was recorded each time. Table 20 lists the parameters for each test devised.

Table 20. Test specifications

Test No.	Count of rows/documents in users table/collection	Count of rows/documents in posts table/collection
1	59998	27253
2	300000	27253
3	59998	55661
4	300000	55661
5	887373	55661
6	887373	112594
7	887373	139799
8	887373	543553

***Performance test results.***

Figure 31 shows the results of the performances tests for Test 1. It can be clearly seen here that MongoDB has a lower query time for this read operation with an average time of 1.18 seconds, compared to 3.23 seconds for MySQL. This is contrary to the read performance results presented by Cooper et al (2010) which were generated by the YCSB client. The key difference here is the hardware in use; the setup presented here uses significantly lower hardware that used in (Cooper et al., 2010). This suggests that MongoDB performs better on lower spec hardware.

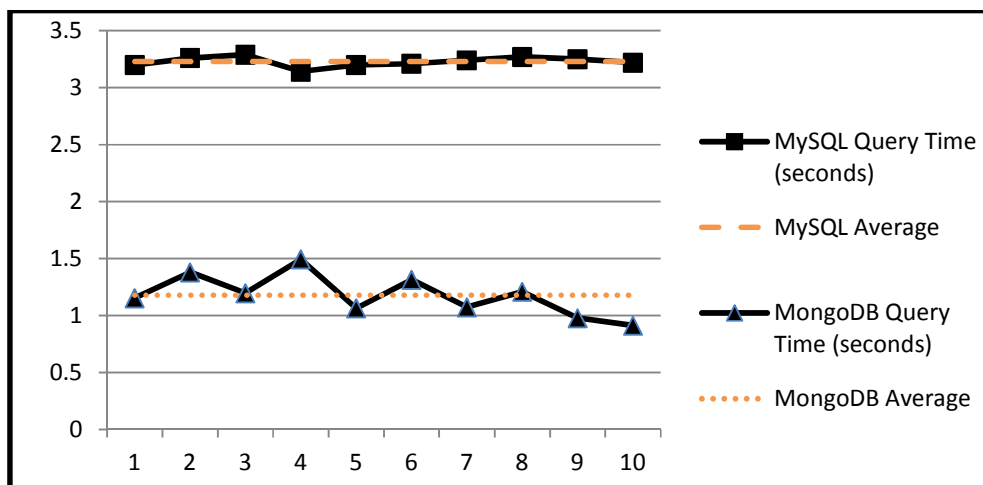


Figure 31. Results of Test No. 1

Test no. 2 increases the row count on one side of the join between the users and posts table/collection (the users side) but as shown in Figure 32. Results of Test No. 2 this has very little effect on query times.

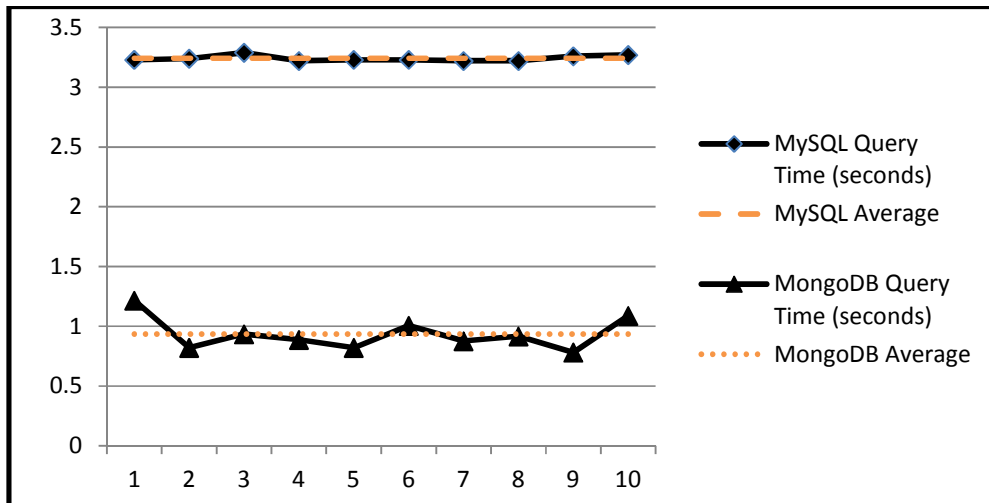


Figure 32. Results of Test No. 2

However, increasing the number of rows in the posts table/collection has a significant negative effect on the performance of the MySQL database, as shown in Figure 33. The limited I/O performance and available CPU cycles of the EC2 micro instance are beginning to have a



severe effect on the MySQL instance. The average query time for Test No. 3 is 14.63 seconds, while the MongoDB instance is largely unaffected by the increase in data volume, with an average query time of 1.1 seconds.

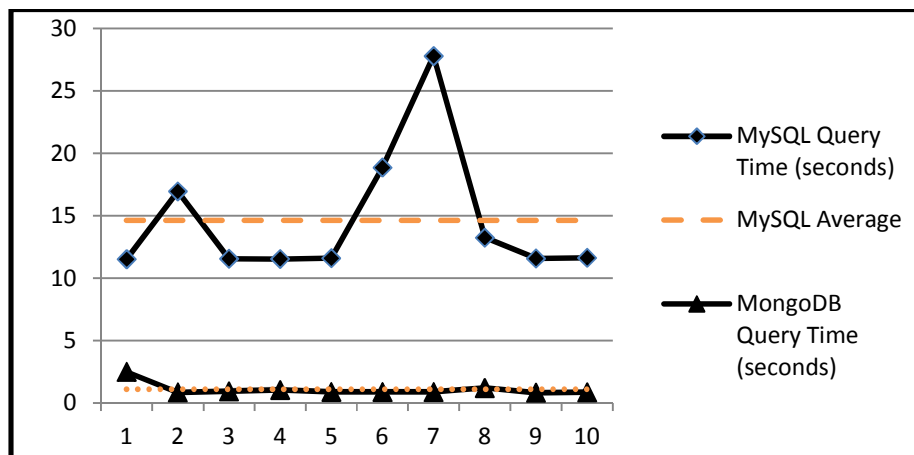


Figure 33. Results of Test No. 3

Moving on to Test No. 4, the results for MySQL start to go beyond the 30 second threshold that have been defined for this test. Results also become extremely inconsistent with query times ranging from 12 seconds to 120 seconds. At this point, either a hardware upgrade or a schema change is required for the MySQL instance. On the other hand, MongoDB is still capable of supporting this volume of data with results remaining consistent and only a slight increase in the average query time to 1.28 seconds. In fact, even with further increases to the row count in both the users and posts collection, the average read time for MongoDB stays below the threshold of 30 seconds for tests 5 through 8. The results are summarised in Table 21.

Table 21. MongoDB read performance results summary

Test No	Average Query Time (seconds)
4	1.28
5	1.9
6	3.5
7	5.6
8	7.15

The read performance only begins to degrade when the document count for the posts collection reaches 763,768 documents. At this point the query regularly returned the exception “too much data for sort() with no index. add an index or specify a smaller limit”. Therefore, this will be considered to be the breaking point for the MongoDB database. There is, however, a way to optimise this query to handle this volume of data which will be discussed in the next section.

### ***MySQL database engine.***

For reference, the MySQL database engine for the forum\_users and forum\_posts table was altered from using InnoDB to MyISAM to assess the effect on the test results. This engine has traditionally performed better than InnoDB and this proved to be the case for this particular test also. Re-running Test 4 with MyISAM, the MySQL database was able to return data within the 30 second timeout limit. However, as Figure 34 shows, the average time was still significantly lower than MongoDB. Couple this with the fact that you no longer have foreign key support with MyISAM and the trade off does not seem worthwhile.

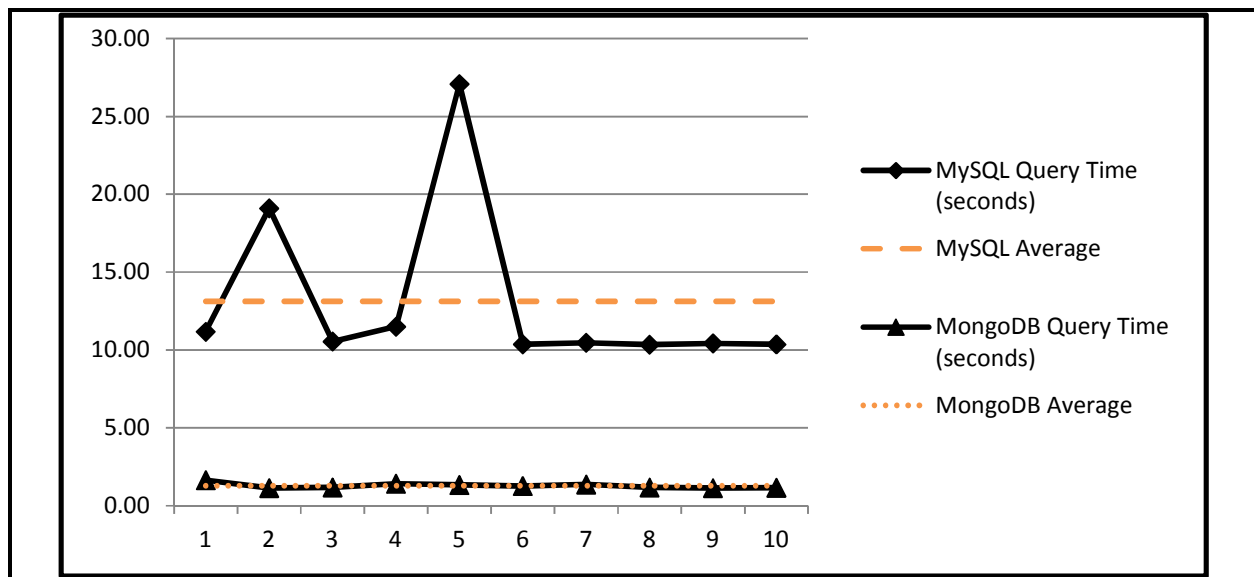


Figure 34. Results of Test No. 4 (Using the MyISAM engine)

### Using Map-Reduce for Aggregation.

As shown in the previous section, SQL queries that use joins and aggregation do not transfer well to NoSQL environments in terms of implementing the query. As discussed in Chapter 2, many NoSQL vendors support Map-Reduce for queries that require aggregations. In this section I will return to the viewforum.php page and implement the data access code with Map-Reduce to compare the different approaches and to measure any performance benefits.

To use Map-Reduce, a map function, a reduce function and an optional finalize function if there are further processing steps required is required. The map function will iterate over a specified collection and output its data as a set of key-value pairs. The key part will be the field that we are grouping on. The value part will be a list of the fields that are required in the aggregation operation. Looking again at the SQL query from viewforum.php in Table 16, the group by field is topic\_id, therefore this will be the key for the map function. For the aggregation

a count of the posts for each topic\_id and the maximum date\_posted are required, therefore we need the date\_posted field in the value part of the map function. Any of the fields can be used to count so no additional fields are needed for this. The map function gives its output through the emit statement and the format for this output must be specified. The code for the map function is shown in Figure 35. It can be seen here that the value part consists of an array that contains a date\_posted element to hold each of the dates for the given topic\_id and a count element which will be used to hold the number of replies. A filter can also be included in this function if required. The original SQL query looks for all posts where the topic\_id is not equal to zero (i.e. replies only), so this can be included in the map function also.

```
$map = "function () {".  
    "if (this.topic_id != 0) " .  
    "{" .  
    "emit(this.topic_id, {date_posted:this.date_posted, count: 1});" .  
    "} " .  
    "};"
```

Figure 35. Map function from viewforum.php

Next, the reduce function was created. This function takes the output from the map function as its input. It performs the necessary calculations on the data and returns it in the same format specified in the emit statement of the map function. The reduce function runs once for each row of data it receives from the map function. The values from each row can be accessed through the values input parameter. In this particular case, we are looking for the maximum date\_posted value and a count of the number of values in the date\_posted element. To get the maximum date, each value is examined through a for loop and some simple logic is implemented to find the most recent date. Getting the count is simply a case of incrementing a counter for each value found. The implementation of this reduce function can be seen in Figure 36.

```
$reduce = "function (key, values) {" .  
    "var re_date = ISODate('1970-01-01T00:00:00Z');" .  
    "var count = 0;" .  
    "values.forEach(function (val) {" .  
        "if ( re_date < val.date_posted ) re_date = val.date_posted;" .  
        "count += val.count;" .  
    "});" .  
    "return {date_posted : re_date, count: count}; " .  
"}";
```

Figure 36. Reduce function from viewforum.php

To run the map-reduce functions, a command is executed on the MongoDB server, specifying the collection on which to run the command, the map and reduce functions, how the output will be stored and an optional query parameter to provide further filtering. This command is shown in Figure 37. In this case, the output is stored as a collection called 'lastPosts'. The merge command specifies that if the collection already exists, the output will be merged with this existing collection, i.e. if the same key exists the value will be updated, if the key does not exist the value will be added to the collection. This command should be executed regularly to ensure that the data in the lastPosts collection is as current as possible. Ideally, it should be executed every time a new post is entered, or it can be executed on a schedule by a server side task.

```
$mapFunc = new MongoClient($map);  
$reduceFunc = new MongoClient($reduce);  
  
$result = $db->command( array(  
    "mapreduce" => "posts",  
    "map" => $mapFunc,  
    "reduce" => $reduceFunc,  
    "out" => array( "merge" => "lastPosts")  
)  
);
```

Figure 37. Executing the Map-Reduce command

The collection will now remain available in our database until it is removed explicitly<sup>4</sup> and can be accessed in the same way as any other collection. Now that the required information is contained in a new collection, the viewforum.php code can be modified. The changes are shown in Figure 38. Most notably, the two scans of the posts collection have been eliminated and replaced with a lookup of the collection produced as a result of implementing the Map-Reduce function. As this collection is an aggregation of the data from the posts collection, it will be a factor smaller, depending on the ratio of questions to replies. Therefore, the performance of this code in theory should be much better than previously.

```
// Get the threads whose topic_id = 0 (the parent post in a thread)
$postts = $posts_coll->find(array("topic_id"=>0, "forum_id"=>(int)$forumid),
                             array("subject"=>true, "author_id"=>true,
                                    "topic_id"=>true, "date_posted"=>true));

// Get the replies count and the most recent date posted
$collection = threadStats();

// For each of these posts, find the author and the number of child posts
foreach($postts as $post){
    $author = $users_coll->findOne(array("_id"=>$post['author_id']),
                                   array("name"=>true));

    $threadInfo = $collection->findOne(array("_id"=>$post['_id']));

    if (is_null($threadInfo)){
        $replies = 0;
        $lastPost = date("d-M-Y h:i:s", $post['date_posted']->sec);
    }
    else {
        $replies = $threadInfo['value']['count'];
        $lastPost = date("d-M-Y h:i:s", $threadInfo['value']['date_posted']->sec);
    }

    // Construct a new array from the retrieved info
    $row = array("subject"=>$post['subject'], "author"=>$author['name'],
                "replies"=>$replies, "date_posted"=>$lastPost, "topic_id"=>$post['topic_id']);

    array_push($thread_row, $row);
}
```

This function calls the Map-Reduce operation which returns a MongoClient object

Figure 38. viewforum.php using Map-Reduce

<sup>4</sup>MongoDB Version 1.8+

The drawback with using this approach however is that the collection may not always be current, depending on how frequently the Map-Reduce function is executed. One other point to note is that the Map-Reduce output collection can be indexed in the same way as any other collection. This is analogous to an indexed view in a relational database system. The next section will provide a comparative analysis of the performance gains as a result of these changes

### ***Performance benefits of using Map-Reduce***

The results for Test No. 8 earlier in this chapter produced an average read time of 7.15 seconds. Figure 39 presents the results of this test with and without using Map-Reduce. The average query time reduces to 3.82 seconds with the Map-Reduce approach.

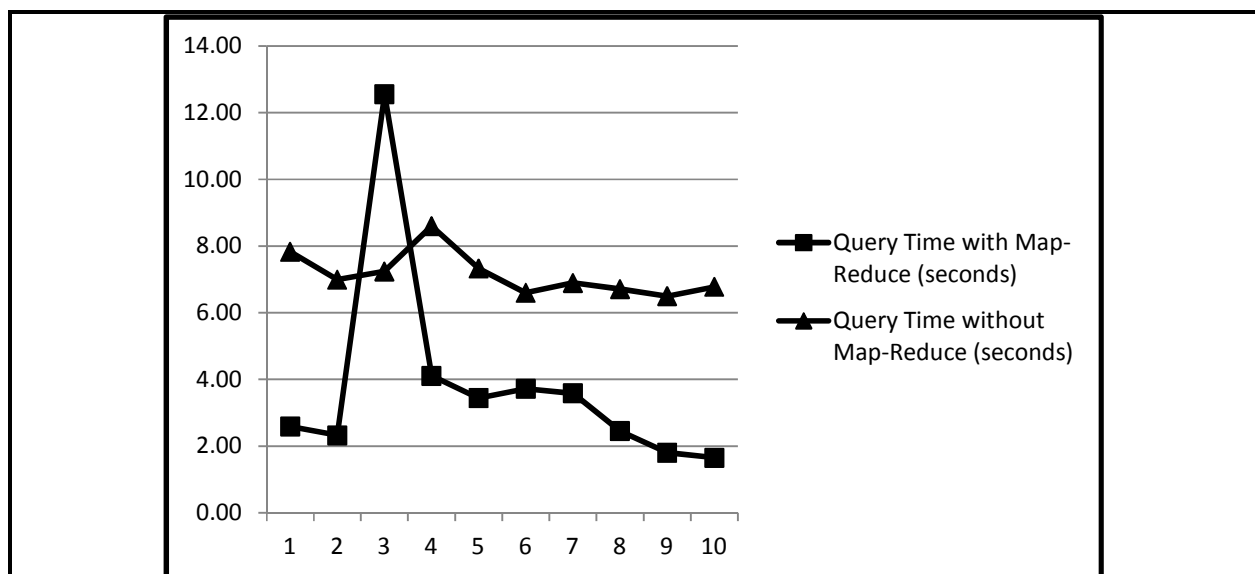


Figure 39. Comparison of query times using Map-Reduce

The key issue here however, is that the output of the Map-Reduce function must be updated whenever a new post is entered. With this volume of data, the length of time to execute the Map-Reduce function took an average of 537 seconds, calculated over 10 executions of the function. This is unlikely to be an acceptable length of time for an application such as this.

However, because of the Map-Reduce can deploy functions in parallel, this time would dramatically reduce if the data was sharded over several nodes in the cluster. The more nodes containing shards of the data, the quicker the execution time of the Map-Reduce function. This is where the real benefit of Map-Reduce can be realised.

### **Chapter Summary**

In this chapter, I have presented how an existing PHP application based on MySQL can be converted to use a MongoDB backend. By extracting each of the SQL queries from the existing application, each one can be analysed and converted to equivalent PHP code. Complex queries involving joins and aggregations are more difficult to create in the MongoDB environment and require having to iterate several times over each collection in the query. Map-Reduce can be used to replicate complex SQL queries, but requires a considerable amount of time to execute when hardware resources are limited.

In contrast, insert, update or delete functions are more straightforward to implement in MongoDB and are more susceptible to changes in the schema of the database. In terms of performance, I have shown that MongoDB can perform better than MySQL for both write and read operations when the capabilities of the hardware are limited.



## Chapter 5 – Conclusions

### Research Findings

Transitioning from a relational to a NoSQL environment can result in many benefits for an organisation. It is possible to take an existing relational schema and translate the design in to an effective NoSQL schema. Document stores are especially well suited for this as their data model allows for an almost direct translation: tables map to collections, rows map to documents and columns map to fields. Indexes are used in the same way as they would in a relational database.

Some of the benefits that can be realised that this study has shown include:

- Lower cost;
- Shorter development and deployment times;
- Better performance on lower cost hardware;
- The flexibility to change the schema without affecting client systems.

The majority of NoSQL databases carry no license fee and can be installed for free, which would give a significant cost saving if transitioning from a commercial relational system. They also perform well on low cost, low specification hardware. This is a particularly useful trait if a company is also transitioning to cloud based services such as Amazon EC2.

Setting up a NoSQL database in a cluster is also much simpler than deploying an equivalent distributed relational database which can reduce overall deployment time for the database system. In the case of the MongoDB database used in this study, a total of 3 servers were configured in a replica set and this was achieved with the minimal of effort. A simple configuration file containing the host name and the replica set name for each member server is

all that is required. This file can also be updated on the fly to include new replicas which begin replicating data as soon as they are joined to the set.

The thought process for designing a NoSQL schema needs to be adjusted slightly in order to maximise the strengths of the NoSQL model. Where data would have been normalised in a relational system to reduce redundancy, often it requires de-normalisation in a NoSQL system which will require extra care when dealing with data that may exist in more than one location in the database. The reason for this is the difficulty in writing the data access code to return data from two or more collections with related fields. The conversion process needs to look at the most common SQL queries with table joins that execute on the database and see where tables can be consolidated in to one collection to reduce the need for joins in the NoSQL database. An example of this was seen in Chapter 3 where three relational tables were consolidated in to one collection in the MongoDB schema.

Adjusting to a new query language instead of SQL can be challenging at first, however, in the case of MongoDB, it was not necessarily more difficult to query the database than it would be to write an SQL statement to query the relational database. Insert, update and delete operations in particular are just as straightforward as their relational counterparts. It was only when complex SQL queries were encountered that it became difficult to transfer to a MongoDB query. When this situation occurs, it should be seen as a trigger to re-evaluate the schema design to look for possible improvements that would reduce the complexity of the query.

In terms of performance, the NoSQL database returned surprisingly better results than the equivalent MySQL design. Despite the limited resources available, it is perfectly viable to deploy a large database on the lowest spec hardware available on Amazon's EC cloud

infrastructure. This can result in huge cost reductions to the overall annual cost of a data storage system.

### **Lessons Learned**

One of the most well stated characteristics of NoSQL databases is that they are not ACID compliant, in particular that they do not support consistency in the same way as relational databases. This actually a misconception. It is true that NoSQL databases predominantly only support eventual consistency, however certain NoSQL databases (such as MongoDB and Cassandra) can be made to be fully ACID compliant if necessary. Furthermore, this can be set at the query level, providing complete customization over the consistency property.

The more striking difference between NoSQL and relational is in the area of foreign key support. Referential integrity is one of the key characteristics of a relational database and this an area where one needs to take the most care when transitioning to a NoSQL database. Because of the differences in the way in which data is queried, it is often the case where data must be de-normalised in the NoSQL database in order to create an efficient query. This can lead to issues with data redundancy which may not have arisen in a relational system.

On a related point, converting to NoSQL may not be a viable option if the application contains complex SQL queries. Although Map-Reduce provides a means to create complex queries on NoSQL databases, the actual implementation can be significantly more difficult than the equivalent SQL query.

Infrastructure-as-a-service and the cloud is becoming more and more prevalent and it is therefore important that a database system is able to perform well in this environment. It was for this reason that Amazon EC2 was chosen as the platform for the use case. The NoSQL database

performed very well in this environment and was surprisingly more efficient for data retrieval than the relational database deployed on the same platform.

### **Summary of Contributions**

This study has attempted to give an insight in to the different types of NoSQL technologies that exist in the market today and the types of applications that these systems are currently being used for. It is the intention of the research to enlighten the reader to the possibility of transitioning from an existing relational environment to a NoSQL environment and to show that this can be particularly beneficial in certain areas. The use case outlined in Chapter 3 and Chapter 4 is intended to prove that an application based on a relational back-end can be converted to use a NoSQL back-end if the right approach is taken.

### **Future Research**

NoSQL is a very new and exciting technology that can have many different applications. This study has looked at only one particular use case and this research could be extended to take into account other types of applications that are built on relational back-ends. Any high volume OLTP type systems could potentially transfer to a NoSQL environment to provide better scalability, availability and at a lower cost.

The area of performance testing that was touched on in the analysis chapter is an area that warrants an entire study in itself. This paper compared the performance of a NoSQL database with a relational database under very specific conditions. All tests were made in isolation and did not take into account concurrency issues. Different types of workloads could also be considered to give a better picture of the performance characteristics of each system.

### References

- 10Gen. (2012). 10gen - MongoDB Subscription. Retrieved August 8, 2012, from <http://www.10gen.com/subscription>
- 10Gen. (n.d.-a). MongoDB: AWS Marketplace. Retrieved from <http://www.mongodb.org/display/DOCS/AWS+Marketplace>
- 10Gen. (n.d.-b). MongoDB: Replica Set Configuration. Retrieved from <http://www.mongodb.org/display/DOCS/Replica+Set+Configuration>
- Amazon. (2012a). Amazon Elastic Cloud Compute. Retrieved from <http://aws.amazon.com/ec2/>
- Amazon. (2012b). Amazon EC Elastic IP Addresses. Retrieved from <http://aws.amazon.com/articles/1346>
- Amazon. (2012c). Amazon EC2 Instance Types. Retrieved August 13, 2012, from <http://aws.amazon.com/ec2/instance-types/>
- Amazon. (n.d.-a). AWS Marketplace. Retrieved from [https://aws.amazon.com/marketplace/help/200899830/ref=gtw\\_r1](https://aws.amazon.com/marketplace/help/200899830/ref=gtw_r1)
- Amazon. (n.d.-b). LAMP Stack - Web Stack (MySQL) provided by TurnKey Linux.
- Atwood, J. (2009, April 6). Stack Overflow Creative Commons Data Dump. Retrieved July 18, 2012, from <http://blog.stackoverflow.com/2009/06/stack-overflow-creative-commons-data-dump/>
- Basho Technologies Inc. (2012). Basho | Riak Enterprise. Retrieved August 8, 2012, from <http://basho.com/products/riak-enterprise/>
- Bhat, U., & Jadhav, S. (2010). Moving Towards Non-Relational Databases. *International Journal of Computer Applications IJCA*, 1(13), 40–46.

- Black, B. (2009, April 12). Introduction to Cassandra: Replication and Consistency. Retrieved August 11, 2012, from <http://www.slideshare.net/benjaminblack/introduction-to-cassandra-replication-and-consistency>
- Bodkin, R. (2010, October 15). InfoQ: Foursquare's MongoDB Outage. Retrieved March 10, 2012, from [http://www.infoq.com/news/2010/10/4square\\_mongodb\\_outage](http://www.infoq.com/news/2010/10/4square_mongodb_outage)
- Brewer, E. A. (2000). Towards robust distributed systems. *Proceedings of the Annual ACM Symposium on Principles of Distributed Computing* (Vol. 19, pp. 7–10).
- Cattell, R. (2010, December). Relational Databases, Object Databases, Key-Value Stores, Document Stores, and Extensible Record Stores: A Comparison. Retrieved from <http://www.odbms.org/download/Cattell.Dec10.pdf>
- Cattell, R. (2011). Scalable SQL and NoSQL data stores. *SIGMOD Rec.*, 39(4), 12–27. doi:10.1145/1978915.1978919
- Celler, F. (2012, April 7). Is UNQL Dead? Retrieved August 18, 2012, from [http://www.arangodb.org/2012/04/07/is\\_unql\\_dead](http://www.arangodb.org/2012/04/07/is_unql_dead)
- Chodorow, K., & Merriman, D. (2011, May 13). Data Types and Conventions - MongoDB. Retrieved July 17, 2012, from <http://www.mongodb.org/display/DOCS/Data+Types+and+Conventions>
- Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R., & Sears, R. (2010). Benchmarking cloud serving systems with YCSB. *Proceedings of the 1st ACM symposium on Cloud computing* (pp. 143–154).
- Couchbase. (2012). NoSQL Database Technology: Post-relational data management for interactive software systems. Retrieved from

<http://www.couchbase.com/sites/default/files/uploads/all/whitepapers/NoSQL-Whitepaper.pdf>

Creative Commons. (n.d.). BSON - Binary JSON. Retrieved July 17, 2012, from

<http://bsonspec.org/#/>

DBPeditas.com. (n.d.). Facebook Messaging - HBase Comes of Age. Retrieved February 4, 2012,

from [http://dbpeditas.com/wiki/HBase:Facebook\\_Messaging\\_-\\_HBase\\_Comes\\_of\\_Age](http://dbpeditas.com/wiki/HBase:Facebook_Messaging_-_HBase_Comes_of_Age)

Dean, J., & Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters.

*Commun. ACM*, 51(1), 107–113. doi:10.1145/1327452.1327492

Evans, D. (2011, July 15). The Internet of Things. *blogs@Cisco - Cisco Blogs*. Retrieved August

19, 2012, from <http://blogs.cisco.com/news/the-internet-of-things-infographic/>

Farrell, E. (2011, September 22). *Nokia: Lessons Learnt Migrating a Very Large and Highly*

*Relational Database into a “Classic” NoSQL*. Presented at the QCon London 2011,

London. Retrieved from [http://www.infoq.com/presentations/Nokia-Lessons-Learnt-](http://www.infoq.com/presentations/Nokia-Lessons-Learnt-Migrating-into-a-Classic-NoSQL)

[Migrating-into-a-Classic-NoSQL](http://www.infoq.com/presentations/Nokia-Lessons-Learnt-Migrating-into-a-Classic-NoSQL)

Google. (2012, April 24). GQL Reference. Retrieved August 18, 2012, from

<https://developers.google.com/appengine/docs/python/datastore/gqlreference>

Introducing JSON. (n.d.). Retrieved July 23, 2012, from <http://www.json.org/>

Merriman, D., & Francia, S. (2011, December 12). Use Cases - MongoDB. Retrieved July 4,

2012, from <http://www.mongodb.org/display/DOCS/Use+Cases>

Murphy, R., & Chodorow, K. (2012a, May 15). Schema Design - MongoDB. Retrieved July 15,

2012, from <http://www.mongodb.org/display/DOCS/Schema+Design>

Murphy, R., & Chodorow, K. (2012b, May 31). Indexes - MongoDB. Retrieved July 15, 2012,

from <http://www.mongodb.org/display/DOCS/Indexes>

- Muthukkaruppan, K. (2011, September 29). *HBase @ Facebook*. Presented at the QCon 2011, San Francisco. Retrieved from <http://www.infoq.com/presentations/HBase-at-Facebook>
- Namore, E., & Glass, M. K. (2005). *Beginning PHP5, Apache, and MySQL Web Development*. John Wiley and Sons.
- Neo Technology. (2011, November). NoSQL For The Enterprise. Retrieved from <http://www.infoq.com/resource/vcr/1706/file/NOSQLfortheEnterprise.pdf>
- Oracle. (2011). MySQL 5.5: Storage Engine Performance Benchmark for MyISAM and InnoDB. Retrieved from [http://www.mysql.com/why-mysql/white-papers/mysql\\_5.5\\_perf\\_myisam\\_innodb.php](http://www.mysql.com/why-mysql/white-papers/mysql_5.5_perf_myisam_innodb.php)
- Oracle. (2012, February 10). Oracle Technology Global Price List. Retrieved August 20, 2012, from <http://www.oracle.com/us/corporate/pricing/technology-price-list-070617.pdf>
- Oracle. (n.d.). Product Details - MySQL Cluster Carrier Grade Edition Subscription (1-4 socket server). Retrieved August 20, 2012, from [https://shop.oracle.com/pls/ostore/f?p=dstore:product:1061138348116637::NO:RP,6:P6\\_LPI,P6\\_PROD\\_HIER\\_ID:60723802201480530690577,58095029061520477171389&tz=1:00](https://shop.oracle.com/pls/ostore/f?p=dstore:product:1061138348116637::NO:RP,6:P6_LPI,P6_PROD_HIER_ID:60723802201480530690577,58095029061520477171389&tz=1:00)
- phpBB. (2007). Features of phpBB. Retrieved July 9, 2012, from <http://www.phpbb.com/about/features/?from=submenu&sid=e21f7d440df8c9691aabdf33db2059c1>
- Pokorny, J. (2011). NoSQL databases: a step to database scalability in web environment. *Proceedings of the 13th International Conference on Information Integration and Web-based Applications and Services, iiWAS '11* (pp. 278–283). New York, NY, USA: ACM. doi:10.1145/2095536.2095583



- Raab, F., Kohler, W., & Shah, A. (n.d.). TPC-C - Overview of the TPC-C Benchmark. Retrieved February 15, 2012, from <http://www.tpc.org/tpcc/detail.asp>
- Rowe, S. (2012, April 23). Lucene Wiki. Retrieved August 18, 2012, from <http://wiki.apache.org/lucene-java/FrontPage?action=show&redirect=FrontPageEN>
- Stonebraker, M., & Cattell, R. (2011). 10 rules for scalable performance in “simple operation” datastores. *Commun. ACM*, 54(6), 72–80. doi:10.1145/1953122.1953144
- The PHP Group. (2012, July 20). PHP Manual: Mongo Types. Retrieved from <http://us2.php.net/manual/en/mongo.types.php>
- Thorub, K. K. (2011, February 12). *Case Study: Riak on Drugs (and the Other Way Around)*. Presented at the GOTO Conference 2011, Aarhus. Retrieved from <http://www.infoq.com/presentations/Case-Study-Riak-on-Drugs>
- vBulletin Solutions. (2012). vBulletin Features. Retrieved July 9, 2012, from <https://www.vbulletin.com/index.php?do=features>
- Wall, M. (2011, August 11). Why I Chose MongoDB for guardian.co.uk. Retrieved August 20, 2012, from <http://www.infoq.com/presentations/Why-I-Chose-MongoDB-for-Guardian>
- Young, B. (2011, August 16). UnQL Specification. Retrieved August 18, 2012, from <http://www.unqlspec.org/display/UnQL/Home>

## Appendix A

### Selected PHP Source Code for MySQL database access

#### index.php

```

<?php
require_once 'conn.php';
require_once 'functions.php';
require_once 'header.php';

$sql = <<<EOS
    SELECT f.id as id, f.forum_name as forum,
           f.forum_desc as description,
           count(forum_id) as threads, u.name as forum_mod
    FROM forum_forum f
    LEFT JOIN forum_posts p
    ON f.id = p.forum_id
    AND p.topic_id=0
    LEFT JOIN forum_users u
    ON f.forum_moderator = u.id
    GROUP BY f.id
EOS;
$result = mysql_query($sql)
    or die(mysql_error());
if (mysql_num_rows($result) == 0) {
    echo "<br>\n";
    echo "    There are currently no forums to view.\n";
} else {
    echo "<table class=\"forumtable\" cellspacing=\"0\" ";
    echo "cellspacing=\"0\"><tr>";
    echo "<th class=\"forum\">Forum</th>";
    echo "<th class=\"threadcount\">Threads</th>";
    echo "<th class=\"moderator\">Moderator</th>";
    echo "</tr>";
    $rowclass = "";
    while ($row = mysql_fetch_array($result)) {
        $rowclass = ($rowclass == "row1"? "row2": "row1");
        echo "<tr class=\"$rowclass\">";
        echo "<td class=\"firstcolumn\"><a href=\"viewforum.php?f=" .
            $row['id'] . "\">";
        echo $row['forum'] . "</a><br>";
        echo "<span class=\"forumdesc\"> " . $row['description'];
        echo "</span></td>";
        echo "<td class=\"center\"> " . $row['threads'] . "</td>";
        echo "<td class=\"center\"> " . $row['forum_mod'] . "</td>";
        echo "</tr>\n";
    }
    echo "</table>";
}

require_once 'footer.php';
?>

```

**viewforum.php**

```

require_once 'conn.php';
require_once 'functions.php';
require_once 'http.php';
if (!isset($_GET['f'])) redirect('index.php');
require_once 'header.php';

$forumid = $_GET['f'];
$forum = getForum($forumid);

echo breadcrumb($forumid, "F");
if (isset($_GET['page'])) {
    $page = $_GET['page'];
} else {
    $page = 1;
}
$limit = $admin['pageLimit']['value'];
if ($limit == "") $limit = 25;
$start = ($page - 1) * $admin['pageLimit']['value'];

$sql = "SELECT SQL_CALC_FOUND_ROWS
        t.id as topic_id, t.subject as t_subject,
        u.name as t_author, count(p.id) as numreplies,
        t.date_posted as t_posted, v.postdate as re_posted
        FROM
        forum_users u
        JOIN forum_posts t
        ON t.author_id = u.id
        LEFT JOIN v_replies v
        ON t.id = v.topic_id
        LEFT JOIN forum_posts p
        ON p.topic_id = t.id
        WHERE t.forum_id = 4
        AND t.topic_id = 0
        GROUP BY t.id
        ORDER BY re_posted DESC
        LIMIT $start, $limit";

$time_start = microtime(true);
$result = mysql_query($sql)
    or die(mysql_error() . "<br>" . $sql);

$time_end = microtime(true);
$time = $time_end - $time_start;
echo "Data Retrieved. Took " . $time . "seconds <br>";

$numrows = mysql_num_rows($result);
echo "Numrows: " . $numrows . "<br>";
if ($numrows == 0) {
    $msg = "There are currently no posts. Would you " .
        "like to be the first person to create a thread?";

```

```

$title = "Welcome to " . $forum['name'];
$dest = "compose.php?forumid=" . $forumid;
$sev = "Info";
$message = msgBox($msg,$title,$dest,$sev);
echo $message;
} else {
    if (isset($_SESSION['user_id'])) {
        echo topicReplyBar(0, $_GET['f'], "right");
    }
    echo "<table class=\"forumtable\" cellspacing=\"0\" ";
    echo "cellpadding=\"2\"><tr>";
    echo "<th class=\"thread\">Thread</th>";
    echo "<th class=\"author\">Author</th>";
    echo "<th class=\"replies\">Replies</th>";
    echo "<th class=\"lastpost\">Last Post</th>";
    echo "</tr>";
    $rowclass = "";
    while ($row = mysql_fetch_array($result)) {
        $rowclass = ($rowclass == "row1"? "row2": "row1");
        if ($row['re_posted'] == "") {
            $lastpost = $row['t_posted'];
        } else {
            $lastpost = $row['re_posted'];
        }
        if ((isset($_SESSION['user_id'])) and
            ($_SESSION['last_login'] < $lastpost)) {
            $newpost = true;
        } else {
            $newpost = false;
        }
        echo "<tr class=\"{$rowclass}\">";
        echo "<td class=\"thread\">" . ($newpost?NEWPOST."&nbsp;";:"");
        echo "<a href=\"viewtopic.php?t=";
        echo $row['topic_id'] . "\">" . $row['t_subject'] . "</a></td>";
        echo "<td class=\"author\">" . $row['t_author'] . "</td>";
        echo "<td class=\"replies\">" . $row['numreplies'] . "</td>";
        echo "<td class=\"lastpost\">" . $lastpost . "</td>";
        echo "</tr>\n";
    }
    echo "</table>";
    echo paginate($limit);
    echo "<p>" . NEWPOST . " = New Post(s)</p>";
}

```

```
require_once 'footer.php';
```

## functions.php

```
<?php
```

```

function getForum($id) {
    $sql = "SELECT forum_name as name, forum_desc as description, " .
        "forum_moderator as forum_mod " .
        "FROM forum_forum ";
}

```

```

        "WHERE id = " . $id;
$result = mysql_query($sql)
    or die(mysql_error() . "<br>" . $sql);
$row = mysql_fetch_array($result);
return $row;
}

function getForumID($topicid) {
    $sql = "SELECT forum_id FROM forum_posts WHERE id=$topicid";
    $result = mysql_query($sql)
        or die(mysql_error() . "<br>" . $sql);
    $row = mysql_fetch_array($result);
    return $row['forum_id'];
}

function breadcrumb($id, $getfrom="F") {
    $sep = "<span class=\"bcsep\">";
    $sep .= " &middot; ";
    $sep .= "</span>";
    if ($getfrom == "P") {
        $sql = "SELECT forum_id, subject FROM forum_posts " .
            "WHERE id = " . $id;
        $result = mysql_query($sql)
            or die(mysql_error() . "<br>" . $sql);
        $row = mysql_fetch_array($result);
        $id = $row['forum_id'];
        $topic = $row['subject'];
    }
    $row = getForum($id);
    $bc = "<a href=\"index.php\">Home</a>$sep";
    switch ($getfrom) {
        case "P":
            $bc .= "<a href=\"viewforum.php?f=$id\">".$row['name'] .
                "</a>$sep" . $topic;
            break;

        case "F":
            $bc .= $row['name'];
            break;
    }
    return "<h4 class=\"breadcrumb\">" . $bc . "</h4>";
}

function showTopic($topicid, $showfull=TRUE) {
    global $conn;
    global $userid;
    global $limit;

    echo breadcrumb($topicid, "P");
    if (isset($_GET['page'])) {
        $page = $_GET['page'];
    } else {
        $page = 1;
    }
    if ($limit == "") $limit = 25;

```

```

$start = ($page - 1) * $limit;
if (isset($_SESSION['user_id'])) {
    echo topicReplyBar($topicid, getForumID($topicid), "right");
}
$sql = "SELECT SQL_CALC_FOUND_ROWS " .
    "p.id, p.subject, p.body, p.date_posted, " .
    "p.date_updated, u.name as author, u.id as author_id, " .
    "u.signature as sig, c.count as postcount, " .
    "p.forum_id as forum_id, f.forum_moderator as forum_mod, " .
    "p.update_id, u2.name as updated_by " .
    "FROM forum_forum f " .
    "JOIN forum_posts p " .
    "ON f.id = p.forum_id " .
    "JOIN forum_users u " .
    "ON u.id = p.author_id " .
    "LEFT JOIN forum_users u2 " .
    "ON u2.id = p.update_id " .
    "LEFT JOIN forum_postcount c " .
    "ON u.id = c.user_id " .
    "WHERE (p.topic_id = $topicid OR p.id = $topicid) " .
    "ORDER BY p.topic_id, p.date_posted " .
    "LIMIT $start,$limit";
$result = mysql_query($sql, $conn)
    or die(mysql_error() . "<br>" . $sql);
$pagelinks = paginate($limit);
if (mysql_num_rows($result) == 0) {
    $msg = "There are currently no posts.  Would you " .
        "like to be the first person to create a thread?";
    $title = "No Posts...";
    $dest = "compose.php?forumid=" . $forumid;
    $sev = "Info";
    $message = msgBox($msg, $title, $dest, $sev);
    echo $message;
} else {
    echo "<table class=\"forumtable\" cellspacing=\"0\" ";
    echo "cellpadding=\"2\"><tr>";
    echo "<th class=\"author\">Author</th>";
    echo "<th class=\"post\">Post</th>";
    echo "</tr>";
    $rowclass = "";
    while ($row = mysql_fetch_array($result)) {
        $lastupdate = "";
        $editlink = "";
        $delink = "";
        $replylink = "&nbsp;";
        $pcount = "";
        $pdate = "";
        $sig = "";
        if ($showfull) {
            $body = $row['body'];
            if (isset($_SESSION['user_id'])) {
                $replylink = "<a href=\"compose.php?forumid=" .
                    $row['forum_id'] . "&topicid=$topicid&reid=" . $row['id'] .
                    "\" class=\"buttonlink\">REPLY</a>&nbsp;";
            } else {

```

```

        $replylink = "";
    }
    if ($row['update_id'] > 0) {
        $lastupdate = "<p class=\"smallNote\">Last updated: " .
            $row['date_updated'] . " by " .
            $row['updated_by'] . "</p>";
    }
    if (($userid == $row['author_id']) or
        ($userid == $row['forum_mod']) or
        ($SESSION['access_lvl'] > 2)) {
        $editlink = "<a href=\"compose.php?a=edit&post=".$row['id'].
            "\" class=\"buttonlink\">EDIT</a>&nbsp;";
        $dellink = "<a href=\"transact-affirm.php?action=deletepost&".
            "id=" . $row['id'] .
            "\" class=\"buttonlink\">DELETE</a>&nbsp;";
    }
    $pcount = "<br><span class=\"textsmall\">Posts: " .
        ($row['postcount']==""?"0":$row['postcount']) . "</span>";
    $date = $row['date_posted'];
    $sig = ($row['sig'] != ""?"<p class=\"sig\">".
        bbcode(nl2br($row['sig'])):"")."</p>";
} else {
    $body = trimBody($body);
}
$rowclass = ($rowclass == "row1"? "row2": "row1");
echo "<tr class=\"".$rowclass.\">";
echo "<td class=\"author\">" . $row['author'];
echo $pcount;
echo "</td><td class=\"post\"><p>";
if (isset($SESSION['user_id'])
    and ($SESSION['last_login'] < $row['date_posted'])) {
    echo NEWPOST . " ";
}
if (isset($_GET['page'])) {
    $pagelink = "&page=" . $_GET['page'];
} else {
    $pagelink = "";
}
echo "<a name=\"post\" . $row['id'] .
    \"\" href=\"viewtopic.php?t=" . $topicid . $pagelink . "#post".
    $row['id'] . "\">".POSTLINK."</a>";
if (isset($row['subject'])) {
    echo " <strong>" . $row['subject'] . "</strong>";
}
echo "</p><p>" . bbcode(nl2br(htmlspecialchars($body))) . "</p>";
echo $sig;
echo $lastupdate;
echo "</td></tr>";
echo "<tr class=\"".$rowclass.\"><td class=\"authorfooter\">";
echo $date . "</td><td class=\"threadfooter\">";
echo $replylink;
echo $editlink;
echo $dellink;
echo "</td></tr>\n";
}

```

[illegible]



```

if(isset($_GET['page'])){
    $page = $_GET['page'];
} else {
    $page = 1;
}
$currepage = $_SERVER['PHP_SELF'] . "?" . $_SERVER['QUERY_STRING'];
$currepage = str_replace("&page=".$page,"",$currepage);

if($page == 1){
    $pagelinks .= "<span class=\"pageprevdead\">&lt; PREV</span>";
} else{
    $pageprev = $page - 1;
    $pagelinks .= "<a class=\"pageprevlink\" href=\"\" . $currepage .
        \"&page=\" . $pageprev . \"\">&lt; PREV</a>";
}

$numofpages = ceil($numrows / $limit);
$range = $admin['pageRange']['value'];
if ($range == "" or $range == 0) $range = 7;
$lrange = max(1,$page-(($range-1)/2));
$rrange = min($numofpages,$page+(($range-1)/2));
if (($rrange - $lrange) < ($range - 1)) {
    if ($lrange == 1) {
        $rrange = min($lrange + ($range-1), $numofpages);
    } else {
        $lrange = max($rrange - ($range-1), 0);
    }
}

if ($lrange > 1) {
    $pagelinks .= "..";
} else {
    $pagelinks .= "&nbsp;&nbsp;";
}
for($i = 1; $i <= $numofpages; $i++){
    if ($i == $page) {
        $pagelinks .= "<span class=\"pagenumdead\">$i</span>";
    } else {
        if ($lrange <= $i and $i <= $rrange) {
            $pagelinks .= "<a class=\"pagenumlink\" \" \" .
                \"href=\"\" . $currepage . \"&page=\" . $i .
                \"\">\" . $i . "</a>";
        }
    }
}
if ($rrange < $numofpages) {
    $pagelinks .= "..";
} else {
    $pagelinks .= "&nbsp;&nbsp;";
}

if (($numrows - ($limit * $page)) > 0){
    $pagenext = $page + 1;
    $pagelinks .= "<a class=\"pagenextlink\" href=\"\" . $currepage .
        \"&page=\" . $pagenext . \"\">NEXT &gt;</a>";
}

```

```
    } else {  
        $pagelinks .= "<span class=\"pagenextdead\">NEXT &gt;</span>";  
    }  
    } else {  
        $pagelinks .= "<span class=\"pageprevdead\">&lt; \" .  
            "PREV</span>&nbsp;&nbsp;&nbsp;";  
        $pagelinks .= "<span class=\"pagenextdead\"> \" .  
            "NEXT &gt;</span>&nbsp;&nbsp;&nbsp;";  
    }  
    $pagelinks .= "</div>";  
    return $pagelinks;  
}
```

## Appendix B

### Selected PHP Source Code for MongoDB Database Access

#### index.php

```

<?php
require_once 'mongo_conn.php';
require_once 'functions.php';
require_once 'header.php';

global $m;
global $db;

$forum_row = array();

$users_coll = new MongoClient($db, 'users');
$forum_coll = new MongoClient($db, 'forum');

// Find all forums
$cursor = $forum_coll->find();

// For each forum in the collection, get the number of threads
// from the posts collection and the moderator from the users collection.
// Construct a new array with the required fields.
foreach($cursor as $forum){
    // Get count of threads in the forum
    $threads = $db->command(
        array(
            "distinct" => "posts",
            "key" => "topic_id",
            "query" => array("forum_id"=>$forum['_id'])
        )
    );

    // Get the moderator name
    $moderator = $users_coll-
>findOne(array("_id"=>(int)$forum['moderator']));

    // Construct the row
    $row = array("forum_id"=>$forum['_id'], "forum"=>$forum['forum'],
        "forum_description"=>$forum['description'],
        "forum_moderator"=>$moderator['name'],
        "threads"=>count($threads['values']));

    // Push the row on to an array
    array_push($forum_row, $row);
}

if (empty($cursor)) {
    echo "    <br>\n";
    echo "    There are currently no forums to view.\n";
} else {

```

```

echo "<table class=\"forumtable\" cellspacing=\"0\" ";
echo "cellspacing=\"0\"><tr>";
echo "<th class=\"forum\">Forum</th>";
echo "<th class=\"threadcount\">Threads</th>";
echo "<th class=\"moderator\">Moderator</th>";
echo "</tr>";
$rowclass = "";
foreach ($forum_row as $rows){
    $rowclass = ($rowclass == "row1"? "row2": "row1");
    echo "<tr class=\"{$rowclass}\">";

    echo "<td class=\"firstcolumn\"><a href=\"viewforum.php?f=" .
        $rows['forum_id'] . "\">";
    echo $rows['forum'] . "</a><br>";
    echo "<span class=\"forumdesc\">" . $rows['forum_description'];
    echo "</span></td>";
    echo "<td class=\"center\">" . $rows['threads'] . "</td>";
    echo "<td class=\"center\">" . "" . "</td>";
    echo "<td class=\"center\">" . $rows['forum_moderator'] . "</td>";
    echo "<td class=\"center\">" . "" . "</td>";
}
echo "</table>";
}

require_once 'footer.php';
?>

```

### viewforum.php (not using Map-Reduce)

```

<?php
require_once 'mongo_conn.php';
require_once 'functions.php';
require_once 'http.php';
if (!isset($_GET['f'])) redirect('index.php');
require_once 'header.php';
global $m;
global $db;

$forumid = $_GET['f'];
$forum = getForum($forumid);

set_time_limit(600);

echo breadcrumb($forumid, "F");
if (isset($_GET['page'])) {
    $page = $_GET['page'];
} else {
    $page = 1;
}
$limit = $admin['pageLimit']['value'];
if ($limit == "") $limit = 25;
$start = ($page - 1) * $admin['pageLimit']['value'];

```

```

$posts_coll = new MongoClient($db, 'posts');
$users_coll = new MongoClient($db, 'users');

// Build the array with the required information
$thread_row = array();

$time_start = microtime(true);

$posts = $posts_coll->find(array("topic_id"=>0, "forum_id"=>(int)$forumid),
    array("subject"=>true, "author_id"=>true, "topic_id"=>true,
    "date_posted"=>true))
    ->sort(array("date_posted"=>-1))
    ->limit($limit)->skip($start);

// For each of these posts, find the author, the number of child posts
// and the date of the most recent post
foreach($posts as $post){
    $post_dates = array(); // array to hold the reply dates
    $author = $users_coll-
>findOne(array("_id"=>$post['author_id']),
    array("name"=>true));
    $replies = $posts_coll->find(array("topic_id"=>$post['_id']))-
>count();
    $reply_dates = iterator_to_array($posts_coll-
>find(array("topic_id"=>$post['_id'],
    array("date_posted"=>true)));

    $last = count($reply_dates) - 1;

    if ($last < 0){ // No replies, therefore last post date =
date_posted
        $lastPost = date("d-M-Y h:i:s", $post['date_posted']-
>sec);
    }
    elseif ($last == 0) { // One reply, therefore last post date =
$reply_dates[0]
        $lastPost = date('d-M-Y h:i:s', $reply_dates[0]['date_posted']-
>sec);
    }
    else { // Multiple replies, therefore determine the max
date_posted of the replies
        foreach ($reply_dates as $i => $dates)
        {
            array_push($post_dates, $dates['date_posted']-
>sec);
        }
        $lastPost = date('d-M-Y h:i:s' , max($post_dates));
    }
}

```

```

        // Construct a new array from the retrieved info
        $row = array("_id"=>$post['_id'], "subject"=>$post['subject'],
"author"=>$author['name'],
        "replies"=>$replies, "date_posted"=>$lastPost,
"topic_id"=>$post['topic_id']);

        array_push($thread_row, $row);
    }

$time_end = microtime(true);
$time = $time_end - $time_start;
echo "Data Retrieved. Took " . $time . "seconds <br>";

if ($numrows == 0) {
    $msg = "There are currently no posts. Would you " .
        "like to be the first person to create a thread?";
    $title = "Welcome to " . $forum['name'];
    $dest = "compose.php?forumid=" . $forumid;
    $sev = "Info";
    $message = msgBox($msg,$title,$dest,$sev);
    echo $message;
} else {
    if (isset($_SESSION['user_id'])) {
        echo topicReplyBar(0, $_GET['f'], "right");
    }
    echo "<table class=\"forumtable\" cellspacing=\"0\" ";
    echo "cellpadding=\"2\"><tr>";
    echo "<th class=\"thread\">Thread</th>";
    echo "<th class=\"author\">Author</th>";
    echo "<th class=\"replies\">Replies</th>";
    echo "<th class=\"lastpost\">Last Post</th>";
    echo "</tr>";
    $rowclass = "";

    foreach ($thread_row as $row) {
        $rowclass = ($rowclass == "row1"? "row2": "row1");
        if ((isset($_SESSION['user_id'])) and
            ($_SESSION['last_login'] < $lastpost)) {
            $newpost = true;
        } else {
            $newpost = false;
        }
        echo "<tr class=\"$rowclass\">";
        echo "<td class=\"thread\">" . ($newpost?NEWPOST." ":"");
        echo "<a href=\"viewtopic.php?t=";
            echo $row['_id'] . "\">" . $row['subject'] . "</a></td>";
        echo "<td class=\"author\">" . $row['author'] . "</td>";
        echo "<td class=\"replies\">" . $row['replies'] . "</td>";
        echo "<td class=\"lastpost\">" . $row['date_posted'] . "</td>";
        echo "</tr>\n";
    }
    echo "</table>";
    echo paginate($limit, $numrows);
    echo "<p>" . NEWPOST . " = New Post(s)</p>";
}

```

```
require_once 'footer.php';
?>
```

### viewforum.php (using Map-Reduce)

```
<?php
require_once 'mongo_conn.php';
require_once 'functions.php';
require_once 'map-reduce_functions.php';
require_once 'http.php';
if (!isset($_GET['f'])) redirect('index.php');
require_once 'header.php';
global $m;
global $db;

$forumid = $_GET['f'];
$forum = getForum($forumid);

echo breadcrumb($forumid, "F");
if (isset($_GET['page'])) {
    $page = $_GET['page'];
} else {
    $page = 1;
}
$limit = $admin['pageLimit']['value'];
if ($limit == "") $limit = 25;
$start = ($page - 1) * $admin['pageLimit']['value'];

set_time_limit(600);

$postes_coll = new MongoClient($db, 'posts');
$users_coll = new MongoClient($db, 'users');

// Build the array with the required information
$thread_row = array();

$time_start = microtime(true);

// Get the threads whose topic_id = 0 (the parent post in a thread)
$postes = $postes_coll->find(array("topic_id"=>0, "forum_id"=>(int)$forumid),

    array("subject"=>true, "author_id"=>true,

        "topic_id"=>true, "date_posted"=>true))
-
>sort(array("date_posted"=>-1))
-
>limit($limit)->skip($start);

// For each of these posts, find the author and the number of child posts
foreach($postes as $post){
    $author = $users_coll-
>findOne(array("_id"=>$post['author_id']),
```

```

        array("name"=>true));

        // Get the last post date and the number of replies
information
        // from the lastPosts collection which is produced as a result
        // of running the MapReduce function
        $threadInfo = $db->lastPosts-
>findOne(array("_id"=>$post['_id']));

        if (is_null($threadInfo)){
            $replies = 0;
            $lastPost = date("d-M-Y h:i:s", $post['date_posted']-
>sec);
        }
        else {
            $replies = $threadInfo['value']['count'];
            $lastPost = date("d-M-Y h:i:s",
$threadInfo['value']['date_posted']->sec);
        }

        // Construct a new array from the retrieved info
        $row = array("subject"=>$post['subject'],
"author"=>$author['name'],
            "replies"=>$replies, "date_posted"=>$lastPost,
"topic_id"=>$post['topic_id']);

        array_push($thread_row, $row);
    }

$time_end = microtime(true);
$time = $time_end - $time_start;
echo "Data Retrieved. Took " . $time . "seconds <br>";

if ($numrows == 0) {
    $msg = "There are currently no posts. Would you " .
        "like to be the first person to create a thread?";
    $title = "Welcome to " . $forum['name'];
    $dest = "compose.php?forumid=" . $forumid;
    $sev = "Info";
    $message = msgBox($msg,$title,$dest,$sev);
    echo $message;
} else {
    if (isset($_SESSION['user_id'])) {
        echo topicReplyBar(0, $_GET['f'], "right");
    }
    echo "<table class=\"forumtable\" cellpadding=\"0\" ";
    echo "cellpadding=\"2\"><tr>";
    echo "<th class=\"thread\">Thread</th>";
    echo "<th class=\"author\">Author</th>";
    echo "<th class=\"replies\">Replies</th>";
    echo "<th class=\"lastpost\">Last Post</th>";
    echo "</tr>";
    $rowclass = "";

```



```

foreach($thread_row as $row) {
    $rowclass = ($rowclass == "row1"? "row2": "row1");
    if ((isset($_SESSION['user_id'])) and
        ($_SESSION['last_login'] < $lastpost)) {
        $newpost = true;
    } else {
        $newpost = false;
    }
    echo "<tr class=\"\$rowclass\">";
    echo "<td class=\"thread\">" . ($newpost?NEWPOST."&nbsp;"; "");
    echo "<a href=\"viewtopic.php?t=";
        echo $_row['_id'] . "\">" . $row['subject'] . "</a></td>";
    echo "<td class=\"author\">" . $row['author'] . "</td>";
    echo "<td class=\"replies\">" . $row['replies'] . "</td>";
    echo "<td class=\"lastpost\">" . $row['date_posted'] . "</td>";
    echo "</tr>\n";
}
echo "</table>";
echo paginate($limit, $numrows);
echo "<p>" . NEWPOST . " = New Post(s)</p>";
}

require_once 'footer.php';
?>

```

## functions.php

```

function getForum($id) {
    global $m;
    global $db;

    $coll = new MongoClient($db, 'forum');
    $doc = $coll->findOne(array("_id" => $id));

    return $doc;
}

function getForumID($topicid) {
    // Description: Returns the forum id for a given topic id
    // Parameters:
    // $topicid - the id of the thread

    // $sql = "SELECT forum_id FROM forum_posts WHERE id=$topicid";
    // $result = mysql_query($sql)
    // or die(mysql_error() . "<br>" . $sql);
    // $row = mysql_fetch_array($result);
    // return $row['forum_id'];

    global $m;
    global $db;

    $db->resetError();
}

```

```

    $coll = new MongoClient($db, 'posts');
    $doc = $coll->findOne(array("_id" => $topicid));

    $result = $db->lastError();

    if ($result['err'] == null ) {
        return $doc['forum_id'];
    }
    else {
        var_dump($result);
        die("MongoDB Error: " . $result['err']);
    }
}

function breadcrumb($id, $getfrom="F") {
// Description: Returns a breadcrumb list of previous pages
// starting from the Home page down to the post level.
// e.g. Home . My Fourm . Thread Title
// Parameters:
// $m - the mongodb connection
// $id - the id of the current post
// $getfrom - the breadcrumb level, either F for Forum or P for Post

global $m;
global $db;

// Define the seperator
$sep = "<span class=\"bcsep\">";
$sep .= " &middot; ";
$sep .= "</span>";

$db->resetError();

// if we are at the post level then we need to get the thread name and link
to it
if ($getfrom == "P") {
    $coll = new MongoClient($db, 'posts');
    $doc = $coll->findOne(array("_id" => (string)$id));

    $id = $doc['forum_id'];
    $topic = $doc['subject'];
}
$row = getForum($id);

$bc = "<a href=\"index.php\">Home</a>$sep";
switch ($getfrom) {
    case "P":
        $bc .= "<a href=\"viewforum.php?f=$id\">".$row['forum_name'] .
            "</a>$sep" . $topic;
        break;

```

```

        case "F":
            $bc .= $row['forum'];
            break;
    }
    return "<h4 class=\"breadcrumb\">" . $bc . "</h4>";
}

function showTopic($topicid, $showfull=TRUE) {
    global $conn;
    global $userid;
    global $limit;
    global $m;
    global $db;

    echo breadcrumb($topicid, "P");
    if (isset($_GET['page'])) {
        $page = $_GET['page'];
    } else {
        $page = 1;
    }
    if ($limit == "") $limit = 25;
    $start = ($page - 1) * $limit;
    if (isset($_SESSION['user_id'])) {
        echo topicReplyBar($topicid, getForumID($topicid), "right");
    }

    $coll_posts = new MongoClient($db, 'posts');
    $query_1 = array("_id" => $topicid);
    $posts_doc_cursor = $coll_posts->find($query_1);
    $posts_result = iterator_to_array($posts_doc_cursor);

    $numrows = $coll_posts->find($query_1)->count();

    $coll_forum = new MongoClient($db, 'forum');
    $query_2 = array("_id" => getForumID($topicid));
    $forum_doc = $coll_forum->find($query_2);

    $pagelinks = paginate($limit, $numrows);
    if ($posts_doc_cursor->count() == 0) {
        $msg = "There are currently no posts. Would you " .
            "like to be the first person to create a thread?";
        $title = "No Posts...";
        $dest = "compose.php?forumid=" . $forumid;
        $sev = "Info";
        $message = msgBox($msg,$title,$dest,$sev);
        echo $message;
    } else {
        echo "<table class=\"forumtable\" cellspacing=\"0\" ";
        echo "cellpadding=\"2\"><tr>";
        echo "<th class=\"author\">Author</th>";
        echo "<th class=\"post\">Post</th>";
        echo "</tr>";
        $rowclass = "";
        while ($posts_result) {

```

```

$lastupdate = "";
$editlink = "";
$dellink = "";
$replylink = "&nbsp;";
$count = "";
$date = "";
$signature = "";
if ($showfull) {
    $body = $posts_result['body'];
    if (isset($_SESSION['user_id'])) {
        $replylink = "<a href=\"compose.php?forumid=\" .
            $posts_result['forum_id'] . "&topicid=$topicid&reid=\" .
$posts_result['id'] .
            \"\" class=\"buttonlink\">REPLY</a>&nbsp;";
    } else {
        $replylink = "";
    }
    if (($userid == $posts_result['author_id']) or
        ($userid == $posts_result['forum_mod']) or
        ($_SESSION['access_lvl'] > 2)) {
        $editlink = "<a
href=\"compose.php?a=edit&post=\".$posts_result['id'].
            \"\" class=\"buttonlink\">EDIT</a>&nbsp;";
        $dellink = "<a href=\"transact-affirm.php?action=deletepost&\".
            "id=\" . $posts_result['id'] .
            \"\" class=\"buttonlink\">DELETE</a>&nbsp;";
    }
    $count = "<br><span class=\"textsmall\">Posts: " .
        ($posts_result['postcount']==""?"0":$row['postcount']) . "</span>";
    $date = $posts_result['date_posted'];
} else {
    $body = trimBody($body);
}
$rowclass = ($rowclass == "row1"? "row2": "row1");
echo "<tr class=\"".$rowclass.\">";
echo "<td class=\"author\">" . $posts_result['author'];
echo $count;
echo "</td><td class=\"post\"><p>";
if (isset($_SESSION['user_id'])
    and ($_SESSION['last_login'] < $row['date_posted'])) {
    echo NEWPOST . " ";
}
if (isset($_GET['page'])) {
    $pagelink = "&page=" . $_GET['page'];
} else {
    $pagelink = "";
}
echo "<a name=\"post\" . $posts_result['id'] .
    \"\" href=\"viewtopic.php?t=" . $topicid . $pagelink . "#post\".
    $posts_result['id'] . \"\">".POSTLINK."</a>";
if (isset($row['subject'])) {
    echo " <strong>" . $row['subject'] . "</strong>";
}
echo "</p><p>" . bbcode(nl2br(htmlspecialchars($body))) . "</p>";
echo $signature;

```

```
echo $lastupdate;
echo "</td></tr>";
echo "<tr class=\"\$rowclass\"><td class=\"authorfooter\">";
echo $pdate . "</td><td class=\"threadfooter\">";
echo $replylink;
echo $editlink;
echo $dellink;
echo "</td></tr>\n";
}
echo "</table>";
echo $pagelinks;
echo "<p>.NEWPOST." = New Post&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&~";
echo POSTLINK." = Post link (use to bookmark)</p>";
}
}

function isParent($page) {
    $currentpage = $_SERVER['PHP_SELF'];
    if (strpos($currentpage, $page) === false) {
        return FALSE;
    } else {
        return TRUE;
    }
}

function topicReplyBar($topicid,$forumid,$pos="right") {
    $html = "<p class=\"buttonBar\" . $pos . \">";
    if ($topicid > 0) {
        $html .= "<a href=\"compose.php?forumid=$forumid\" .
            \"&topicid=$topicid&reid=$topicid\" \" \" .
            \"class=\"buttonlink\">Reply to Thread</a>";
    }
    if ($forumid > 0) {
        $html .= "<a href=\"compose.php?forumid=$forumid\" \" \" .
            \"class=\"buttonlink\">New Thread</a>";
    }
    $html .= "</p>";
    return $html;
}

function userOptionList($level) {
    $sql = "SELECT id, name, access_lvl " .
        "FROM forum_users " .
        "WHERE access_lvl=\"" . $level . "\" " .
        "ORDER BY name";
    $result = mysql_query($sql)
    or die(mysql_error());

    while ($row = mysql_fetch_array($result)) {
        echo "<option value=\"\". $row['id'] . \">" .
            htmlspecialchars($row['name']) . "</option>";
    }
}

function paginate($limit=10, $numrows) {
```

```

global $admin;

$pagelinks = "<div class=\"pagelinks\">";
if ($numrows > $limit) {
    if(isset($_GET['page'])){
        $page = $_GET['page'];
    } else {
        $page = 1;
    }
    $currpage = $_SERVER['PHP_SELF'] . "?" . $_SERVER['QUERY_STRING'];
    $currpage = str_replace("&page=", "", $currpage);

    if($page == 1){
        $pagelinks .= "<span class=\"pageprevdead\">&lt; PREV</span>";
    }else{
        $pageprev = $page - 1;
        $pagelinks .= "<a class=\"pageprevlink\" href=\"\" . $currpage . \"&page=\" . $pageprev . \"\">&lt; PREV</a>";
    }

    $numofpages = ceil($numrows / $limit);
    $range = $admin['pageRange']['value'];
    if ($range == "" or $range == 0) $range = 7;
    $lrange = max(1, $page - (($range-1)/2));
    $rrange = min($numofpages, $page + (($range-1)/2));
    if (($rrange - $lrange) < ($range - 1)) {
        if ($lrange == 1) {
            $rrange = min($lrange + ($range-1), $numofpages);
        } else {
            $lrange = max($rrange - ($range-1), 0);
        }
    }

    if ($lrange > 1) {
        $pagelinks .= "..";
    } else {
        $pagelinks .= "&nbsp;&nbsp;";
    }
    for($i = 1; $i <= $numofpages; $i++){
        if ($i == $page) {
            $pagelinks .= "<span class=\"pagenumdead\">$i</span>";
        } else {
            if ($lrange <= $i and $i <= $rrange) {
                $pagelinks .= "<a class=\"pagenumlink\" \" \" . \"href=\"\" . $currpage . \"&page=\" . $i . \"\">\" . $i . \"</a>";
            }
        }
    }
    if ($rrange < $numofpages) {
        $pagelinks .= "..";
    } else {
        $pagelinks .= "&nbsp;&nbsp;";
    }
}

```

```

if(($numrows - ($limit * $page)) > 0){
    $pagenext = $page + 1;
    $pagelinks .= "<a class=\"pagenextlink\" href=\"\" . $currpage
                  \"&page=\" . $pagenext . \"\">NEXT &gt;</a>";
} else {
    $pagelinks .= "<span class=\"pagenextdead\">NEXT &gt;</span>";
}
} else {
    $pagelinks .= "<span class=\"pageprevdead\">&lt; \" .
                  \"PREV</span>&nbsp;&nbsp;&nbsp;";
    $pagelinks .= "<span class=\"pagenextdead\"> \" .
                  \"NEXT &gt;</span>&nbsp;&nbsp;&nbsp;";
}
$pagelinks .= "</div>";
return $pagelinks;
}

```

## Appendix C

### PHP Source Code for MySQL Data Import

#### import\_users.php

```

function getForum($id) {
    global $m;
    global $db;

    $coll = new MongoClient($db, 'forum');
    $doc = $coll->findOne(array("_id" => $id));

    return $doc;
}

function getForumID($topicid) {
    // Description: Returns the forum id for a given topic id
    // Parameters:
    // $topicid - the id of the thread

    // $sql = "SELECT forum_id FROM forum_posts WHERE id=$topicid";
    // $result = mysql_query($sql)
    // or die(mysql_error() . "<br>" . $sql);
    // $row = mysql_fetch_array($result);
    // return $row['forum_id'];

    global $m;
    global $db;

    $db->resetError();

    $coll = new MongoClient($db, 'posts');
    $doc = $coll->findOne(array("_id" => $topicid));

    $result = $db->lastError();

    if ($result['err'] == null ) {
        return $doc['forum_id'];
    }
    else {
        var_dump($result);
        die("MongoDB Error: " . $result['err']);
    }
}

function breadcrumb($id, $getfrom="F") {
    // Description: Returns a breadcrumb list of previous pages
    // starting from the Home page down to the post level.
    // e.g. Home . My Fourm . Thread Title
    // Parameters:

```



```

// $m - the mongodb connection
// $id - the id of the current post
// $getfrom - the breadcrumb level, either F for Forum or P for Post

global $m;
global $db;

// Define the seperator
$sep = "<span class=\"bcsep\">";
$sep .= " &middot; ";
$sep .= "</span>";

$db->resetError();

// if we are at the post level then we need to get the thread name and link
to it
if ($getfrom == "P") {
    $coll = new MongoClient($db, 'posts');
    $doc = $coll->findOne(array("_id" => (string)$id));

    $sid = $doc['forum_id'];
    $topic = $doc['subject'];
}
$row = getForum($id);

$bc = "<a href=\"index.php\">Home</a>$sep";
switch ($getfrom) {
    case "P":
        $bc .= "<a href=\"viewforum.php?f=$id\">".$row['forum_name'] .
            "</a>$sep" . $topic;
        break;

    case "F":
        $bc .= $row['forum'];
        break;
}
return "<h4 class=\"breadcrumb\">" . $bc . "</h4>";
}

function showTopic($topicid, $showfull=TRUE) {
    global $conn;
    global $userid;
    global $limit;
    global $m;
    global $db;

    echo breadcrumb($topicid, "P");
    if (isset($_GET['page'])) {
        $page = $_GET['page'];
    } else {
        $page = 1;
    }
    if ($limit == "") $limit = 25;
}

```

```

$start = ($page - 1) * $limit;
if (isset($_SESSION['user_id'])) {
    echo topicReplyBar($topicid, getForumID($topicid), "right");
}

$coll_posts = new MongoClient($db, 'posts');
$query_1 = array("_id" => $topicid);
$posts_doc_cursor = $coll_posts->find($query_1);
$posts_result = iterator_to_array($posts_doc_cursor);

$numrows = $coll_posts->find($query_1)->count();

$coll_forum = new MongoClient($db, 'forum');
$query_2 = array("_id" => getForumID($topicid));
$forum_doc = $coll_forum->find($query_2);

$pagelinks = paginate($limit, $numrows);
if ($posts_doc_cursor->count() == 0) {
    $msg = "There are currently no posts. Would you " .
        "like to be the first person to create a thread?";
    $title = "No Posts...";
    $dest = "compose.php?forumid=" . $forumid;
    $sev = "Info";
    $message = msgBox($msg, $title, $dest, $sev);
    echo $message;
} else {
    echo "<table class=\"forumtable\" cellspacing=\"0\" ";
    echo "cellpadding=\"2\"><tr>";
    echo "<th class=\"author\">Author</th>";
    echo "<th class=\"post\">Post</th>";
    echo "</tr>";
    $rowclass = "";
    while ($posts_result) {
        $lastupdate = "";
        $editlink = "";
        $delink = "";
        $replylink = "&nbsp;";
        $pcount = "";
        $pdate = "";
        $sig = "";
        if ($showfull) {
            $body = $posts_result['body'];
            if (isset($_SESSION['user_id'])) {
                $replylink = "<a href=\"compose.php?forumid=" .
                    $posts_result['forum_id'] . "&topicid=$topicid&reid=" .
                    $posts_result['_id'] .
                    "\" class=\"buttonlink\">REPLY</a>&nbsp;";
            } else {
                $replylink = "";
            }
            if (($userid == $posts_result['author_id']) or
                ($userid == $posts_result['forum_mod']) or
                ($_SESSION['access_lvl'] > 2)) {
                $editlink = "<a
href=\"compose.php?a=edit&post=".$posts_result['id'].

```

```

        "\ " class="\buttonlink\"">EDIT</a>&nbsp;" ;
        $dellink = "<a href=\"transact-affirm.php?action=deletepost&".
            "id=" . $posts_result['id'] .
            "\" class=\buttonlink\">DELETE</a>&nbsp;" ;
    }
    $pcount = "<br><span class=\"textsmall\">Posts: " .
        ($posts_result['postcount']==""?"0":$row['postcount']) . "</span>" ;
    $pdate = $posts_result['date_posted'];
} else {
    $body = trimBody($body);
}
$rowclass = ($rowclass == "row1"? "row2": "row1");
echo "<tr class=\"$rowclass\">";
echo "<td class=\"author\">" . $posts_result['author'];
echo $pcount;
echo "</td><td class=\"post\"><p>";
if (isset($_SESSION['user_id'])
    and ($_SESSION['last_login'] < $row['date_posted'])) {
    echo NEWPOST . " ";
}
if (isset($_GET['page'])) {
    $pagelink = "&page=" . $_GET['page'];
} else {
    $pagelink = "";
}
echo "<a name=\"post" . $posts_result['id'] .
    "\" href=\"viewtopic.php?t=" . $topicid . $pagelink . "#post".
    $posts_result['id'] . "\">.POSTLINK.</a>";
if (isset($row['subject'])) {
    echo " <strong>" . $row['subject'] . "</strong>";
}
echo "</p><p>" . bbcode(nl2br(htmlspecialchars($body))) . "</p>";
echo $sig;
echo $lastupdate;
echo "</td></tr>";
echo "<tr class=\"$rowclass\"><td class=\"authorfooter\">";
echo $pdate . "</td><td class=\"threadfooter\">";
echo $replylink;
echo $editlink;
echo $dellink;
echo "</td></tr>\n";
}
echo "</table>";
echo $pagelinks;
echo "<p>.NEWPOST." = New Post&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&~";
echo POSTLINK." = Post link (use to bookmark)</p>";
}
}

```

---

```

function isParent($page) {
    $currentpage = $_SERVER['PHP_SELF'];
    if (strpos($currentpage, $page) === false) {
        return FALSE;
    } else {
        return TRUE;
    }
}

```

```

    }
}

function topicReplyBar($topicid,$forumid,$pos="right") {
    $html = "<p class=\"buttonBar\" . $pos . \">";
    if ($topicid > 0) {
        $html .= "<a href=\"compose.php?forumid=$forumid\" .
            "&topicid=$topicid&reid=$topicid\" \" \" .
            \"class=\"buttonlink\">Reply to Thread</a>";
    }
    if ($forumid > 0) {
        $html .= "<a href=\"compose.php?forumid=$forumid\" \" \" .
            \"class=\"buttonlink\">New Thread</a>";
    }
    $html .= "</p>";
    return $html;
}

function userOptionList($level) {
    $sql = "SELECT id, name, access_lvl " .
        "FROM forum_users " .
        "WHERE access_lvl=" . $level . " " .
        "ORDER BY name";
    $result = mysql_query($sql)
        or die(mysql_error());

    while ($row = mysql_fetch_array($result)) {
        echo "<option value=\"\". $row['id'] . \">\" .
            htmlspecialchars($row['name']) . "</option>";
    }
}

function paginate($limit=10, $numrows) {
    global $admin;

    $pagelinks = "<div class=\"pagelinks\">";
    if ($numrows > $limit) {
        if(isset($_GET['page'])) {
            $page = $_GET['page'];
        } else {
            $page = 1;
        }
        $currpage = $_SERVER['PHP_SELF'] . "?" . $_SERVER['QUERY_STRING'];
        $currpage = str_replace("&page=".$page,"",$currpage);

        if($page == 1){
            $pagelinks .= "<span class=\"pageprevdead\">&lt; PREV</span>";
        } else {
            $pageprev = $page - 1;
            $pagelinks .= "<a class=\"pageprevlink\" href=\"\" . $currpage .
                "&page=" . $pageprev . "\">&lt; PREV</a>";
        }

        $numofpages = ceil($numrows / $limit);
        $range = $admin['pageRange']['value'];
    }
}

```

```

if ($range == "" or $range == 0) $range = 7;
$lrangle = max(1,$page-(($range-1)/2));
$rrangle = min($numofpages,$page+(($range-1)/2));
if (($rrangle - $lrangle) < ($range - 1)) {
    if ($lrangle == 1) {
        $rrangle = min($lrangle + ($range-1), $numofpages);
    } else {
        $lrangle = max($rrangle - ($range-1), 0);
    }
}

if ($lrangle > 1) {
    $pagelinks .= "..";
} else {
    $pagelinks .= "&nbsp;&nbsp;&nbsp;";
}
for($i = 1; $i <= $numofpages; $i++){
    if ($i == $page) {
        $pagelinks .= "<span class=\"pagenumdead\">$i</span>";
    } else {
        if ($lrangle <= $i and $i <= $rrangle) {
            $pagelinks .= "<a class=\"pagenumlink\" \" \" .
                \"href=\"\" . $currpage . \"&page=\" . $i .
                \"\">\" . $i . \"</a>\";
        }
    }
}
if ($rrangle < $numofpages) {
    $pagelinks .= "..";
} else {
    $pagelinks .= "&nbsp;&nbsp;&nbsp;";
}

if(($numrows - ($limit * $page)) > 0){
    $pagenext = $page + 1;
    $pagelinks .= "<a class=\"pagenextlink\" href=\"\" . $currpage .
        \"&page=\" . $pagenext . \"\">NEXT &gt;</a>\";
} else {
    $pagelinks .= "<span class=\"pagenextdead\">NEXT &gt;</span>\";
}
} else {
    $pagelinks .= "<span class=\"pageprevdead\">&lt; \" \" .
        \"PREV</span>&nbsp;&nbsp;&nbsp;\";
    $pagelinks .= "<span class=\"pagenextdead\"> \" \" .
        \"NEXT &gt;</span>&nbsp;&nbsp;&nbsp;\";
}
$pagelinks .= "</div>\";
return $pagelinks;
}

```

## import\_posts.php

```

<?php
//require_once 'conn-ec2.php';

```

```

require_once 'conn.php';
require_once 'http.php';
require_once 'functions.php';
require_once 'ChromePhp.php';

set_time_limit(900);
$time_start = microtime(true);

for ($counter=1;$counter<5;$counter+=1) {
    $url = "Stack_Overflow_122011/posts" . $counter . ".xml";
    $xml = simplexml_load_file($url); //puts the xml contents in to an
    array like structure

    //to loop through values
    foreach($xml->row as $row)
    {
        //////////////////////////////////////
/*      **posts**.xml
        - Id
        - PostTypeId
          - 1: Question (?)
          - 2: Answer (?)
        - ParentId (only present if PostTypeId is 2)
        - AcceptedAnswerId (only present if PostTypeId is 1)
        - CreationDate *
        - Score
        - ViewCount
        - Body *
        - OwnerUserId *
        - LastEditorUserId *
        - LastEditorDisplayName="Rich B"
        - LastEditDate="2009-03-05T22:28:34.823"
        - LastActivityDate="2009-03-11T12:51:01.480"
        - CommunityOwnedDate="2009-03-11T12:51:01.480"
        - ClosedDate="2009-03-11T12:51:01.480"
        - Title= *
        - Tags=
        - AnswerCount
        - CommentCount
        - FavoriteCount

        */
        //////////////////////////////////////

        $id = $row["Id"];
        $date_posted = $row["CreationDate"];
        $date_updated = $row["LastEditDate"];
        $body = mysql_real_escape_string($row["Body"]);
        $author_id = $row["OwnerUserId"];
        $update_id = $row["LastEditorUserId"];
        if ($row["Title"]=="") {
            $subject = "No Subject";
        }
    }
}

```

```

        else{ $subject = mysql_real_escape_string($row["Title"]);}
        if ($row["PostTypeId"]==1){
            $topic_id = 0;
        }
        else {
            $topic_id = $row["ParentId"];
        }

        $forum_id = 4;

        $sql = "INSERT INTO forum_posts " .

"(id,topic_id,forum_id,author_id,update_id,date_posted,date_updated,subject,b
ody) " .
                                "VALUES ('" . $id . "','" . $topic_id . "','" .
$forum_id . "','" .
                                $author_id . "','" . $update_id . "','" .
$date_posted . "','" .
                                $date_posted . "','" . $subject . "','" . $body
. '');"

        if (!mysql_query($sql, $conn)){
            if (mysql_errno() == 1064){
                ChromePhp::error('Syntax Error');
                ChromePhp::log("STRING", $row["Body"]);
                ChromePhp::log("FIXED_STRING",
mysql_real_escape_string($row["Body"]));
            }
            die(mysql_error() . "<br>" . $sql);
        }

    }
    $time_end = microtime(true);
    $time = $time_end - $time_start;
    echo "Import of file " . $url . " to MySQL complete. Took " . $time .
"seconds <br>";
}
?>

```

## Appendix D

### PHP Source Code for MongoDB Data Import

#### import\_users.php

```

<?php
require_once 'http.php';
require_once 'functions.php';
require_once 'ChromePhp.php';
require_once 'mongo_conn-ec2.php';

global $m;
global $db;

set_time_limit(600);

$collection = $db->users;

for ($counter=1;$counter<2;$counter+=1) {
    $url = "Stack_Overflow_122011/users_" . $counter . ".xml";
    $xml = simplexml_load_file($url); //puts the xml contents in to an
    array like structure

    $time_start = microtime(true);

    //to loop through values
    foreach($xml->row as $row) {
        /*
            - **users**.xml
            - Id *
            - Reputation
            - CreationDate *
            - DisplayName *
            - EmailHash *
            - LastAccessDate *
            - WebsiteUrl *
            - Location
            - Age
            - AboutMe
            - Views
            - UpVotes
            - DownVotes

            DB Field
            id
            date_joined
            name
            email
            last_login
            signature

            passwd =
        */
        autoGenerate(randomNumber)

        access_lvl = (1 | 2 | 3)

    }
}

```



```

        $sid = (int)$row["Id"];
        $date_joined = new MongoDBDate(strtotime($row["CreationDate"]));
        $name = (string)$row["DisplayName"];
        $email = (string)$row["EmailHash"];
        $last_login = new
MongoDate(strtotime($row["LastAccessDate"]));
        $signature = (string)$row["WebsiteUrl"];
        $passwd = generatePassword(rand(5,20), 8);
        $access_lvl = rand(1,3);

        try {
            $arr = array('_id' => $sid, 'name' => $name, 'date_joined' =>
$date_joined, 'email' => $email,
                                'last_login' => $last_login,
            'signature' => $signature, 'password' => $passwd,
                                'access_lvl' => $access_lvl,
            'post_count' => 0);

            $db->users->insert($arr, true);

        }
        catch (MongoCursorException $mce){
            echo "error message: ".$mce->getMessage()."\n";
            echo "error code: ".$mce->getCode()."\n";
        }
        catch (Exception $e) {
            var_dump($e->getMessage());
            var_dump($arr);
            logToFile($logfile, $e->getMessage());
        }

    }

    $time_end = microtime(true);
    $time = $time_end - $time_start;
    echo "Import of file " . $url . " complete. Took " . $time . "seconds
<br>";
}
?>

```

### import\_posts.php

```

<?php
require_once 'http.php';
require_once 'functions.php';
require_once 'ChromePhp.php';
include 'mongo_conn-ec2.php';

global $m;
global $db;

$collection = $db->posts;

for ($counter=3;$counter<5;$counter+=1) {

```

```

$url = "Stack_Overflow_122011/posts" . $counter . ".xml";
$xml = simplexml_load_file($url); //puts the xml contents in to an
array like structure
$logfile = "log/import_posts-mongo_" . date("Ymdhis", mktime()) .
"log";

$time_start = microtime(true);
set_time_limit(600);

foreach($xml->row as $row) {
    ////////////////////////////////////////////
/*    **posts**.xml
    - Id *
    - PostTypeId *
      - 1: Question (?)
      - 2: Answer (?)
    - ParentId (only present if PostTypeId is 2)
    - AcceptedAnswerId (only present if PostTypeId is 1)
    - CreationDate *
    - Score
    - ViewCount
    - Body *
    - OwnerUserId *
    - LastEditorUserId *
    - LastEditorDisplayName="Rich B"
    - LastEditDate="2009-03-05T22:28:34.823"
    - LastActivityDate="2009-03-11T12:51:01.480"
    - CommunityOwnedDate="2009-03-11T12:51:01.480"
    - ClosedDate="2009-03-11T12:51:01.480"
    - Title= *
    - Tags=
    - AnswerCount
    - CommentCount
    - FavoriteCount

    */
    ////////////////////////////////////////////

    $id = (string)$row["Id"];
    $date_posted = new MongoDBDate(strtotime($row["CreationDate"]));
    $date_updated = new
MongoDate(strtotime($row["LastEditDate"]));
    $body = (string)$row["Body"];
    $author_id = (string)$row["OwnerUserId"];
    $update_id = (string)$row["LastEditorUserId"];
    if ($row["Title"]=="") {
        $subject = "No Subject";
    }
    else{ $subject = (string)$row["Title"];}
    if ($row["PostTypeId"]==1){
        $topic_id = 0;
    }
    elseif ($row["PostTypeId"]==2) {

```

```

        $topic_id = (string)$row["ParentId"];
    }
    $forum_id = 4;

    try {

        $arr = array('_id' => $id, 'date_posted' =>
$date_posted, 'date_updated' => $date_updated, 'body' => $body,
                                                                    'author_id' =>
$author_id, 'update_id' => $update_id, 'subject' => $subject,
                                                                    'topic_id' => $topic_id,
'forum_id' => $forum_id);

        ($db->posts->insert($arr));

    }
    catch (MongoCursorException $mce){
        echo "error message: ".$mce->getMessage()."\n";
        echo "error code: ".$mce->getCode()."\n";
    }
    catch (Exception $e) {
        var_dump($e->getMessage());
        var_dump($arr);
        logToFile($logfile, $e->getMessage());
    }

}

$time_end = microtime(true);
$time = $time_end - $time_start;
echo "Import of file " . $url . " complete. Took " . $time . "seconds
<br>";
}
?>

```