Swinburne University of Technology
Faculty of Science, Engineering, and Technology

COS200015: Fundamentals of Data Management

Research Report: Exploring NoSQL

Date of report submission	11/11/19
Lab Supervisor	Rusul Abduljabbar
Group	2:30pm Wednesday

Name	**Student ID**	**Signature**
Jimmy Trac	101624964	

This page is mostly blank.

The purpose of this page is to separate the cover page from the report when printed double-sided.

NoSQL Research Report

Jimmy Trac | 101624964

Abstract

Databases, or datastores, are a crucial component of Information Communication Technologies since time immemorial (circa 1970 for computer programmers). They allow computers to store information effectively and efficiently through scalable and logical means. This report aims to develop a comparison between NoSQL Databases and traditional SQL-based systems such as MySQL, such as the key limitations of MySQL and the

Table of Contents

I. Introduction

In the increasingly interconnected world that humans inhabit, the requirement to store, access, and manipulate data is a fundamental component of any modern service. The need for increasing data storage, processing, and delivery has been prevalent throughout the development of the early 21st Century, and along with it, the development of databases.

Traditional SQL and relational databases have lost popularity to NoSQL databases on the grounds of flexibility and speed. MongoDB is the #5 most popular and is NoSQL database management system whereas MySQL coming in at #2 with HazelCast, supporting both Key-Value and Document store at #45. (DB-Engines, October 2019)

II. Methodology

This report aims to explore the relational databases and their limitations, along with the differences inherent within NoSQL. The NoSQL databases explored within are MongoDB, HazelCast, and will be compared to MySQL. The objective of this report is to understand the differences of each and the reason for decline of relational databases in the modern environment.

2.1. Exploring Relational Databases

Relational Database Management Systems (RDBMS) store information in two-dimensional table form with 'relationships' describing both the context with rules that strictly define the structure. This form of Relational Database is widely used in SQL (Structured Query Language) databases, with one of the well-known modern products being MySQL. On the other hand, NoSQL (Not Only SQL) are commonly referred to as "non-relational" databases and come in various flavours, such as Document-Store or Key-Value Store. (Ceiko, 2011, pp. xvii) (Wei-Ping, Ming-Xin, Huan, 2011, pp. 303).

2.2. Limitations of Relational Databases

Relational databases are not without major limitations, which, in recent years, have lost ground to more flexible NoSQL products such as MongoDB or Redis (DB-engine Ranking Trend, October 2019). There are two main issues with the current implementation of relational databases: horizontal scaling; and the proliferation of unstructured data (Drake, 2019)

Horizontal Scaling refers to the addition of new machines to an existing system, increasing parallel capacity and computational power. Conversely, Vertical Scaling refers to upgrading the current, existing hardware to increase capacity and processing power with no changes to the software or the current solution. Figure 2.2.1 illustrates the difference between the two. Horizontal Scaling presents a unique issue with one of the primary properties of relational databases – notably the C in the 'ACID' principles, referring to the *Consistency* of the database system. Consistency roughly abstracts to the integrity of the system, where the database must remain consistent before and after any transaction within the database. (Kaur, 2017). Ensuring consistency is difficult when a database is spread across multiple machines – possibly in different geographic locations – thereby allowing the possibility that two different clients may access different values for the same data due to an update or delay. Furthermore, it requires that data be transferred across different nodes and updated, limiting either the performance of the system, or sacrificing database consistency.

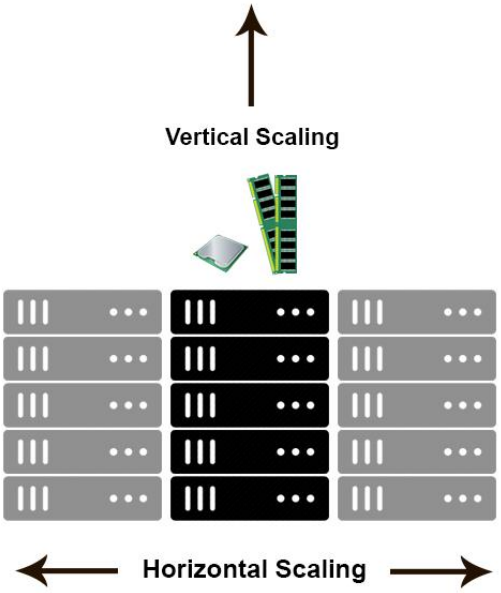

Figure 2.2.1 – Horizontal and Vertical Scaling illustrated (Korpal, 2019)

The second limitation of relational databases is the increasing prevalence of unstructured data. Unstructured Data refers to any information that is not organised in any pre-determined manner (or *schema*). This is in contrast with Structured Data, where information is organised into a proper *schema* with each piece of information organised into tables, or similar data structures. A key feature of structured data is that information has a specific data type and some fields may have restrictions on what sort of data it may store; the quintessential structured data is a relational database.

Figure 2.2.2 – A diagram of structured to unstructured data.

The compromise between the two types is Semi-Structured data, which does not necessarily have a strict, pre-defined schema, but retains some organisation for ease of sorting and flexibility, hence the middle position as seen in figure 2.2.2. Examples of semi-structured data include XML or JSON formats, or even HTML webpages, which contain a mix of structured, unstructured, and semi-structured data (Vishwakarma, 2018). The issue with relational databases is that they require strict structured data in the same format (i.e. the same schema) of the relational database itself, which has become increasingly difficult as unstructured and semi-structured data becomes more common (Drake, 2019). Furthermore, altering an existing relational database is difficult – both in application and in risk – as any adjustments made to a schema may result in serious issues if implemented incorrectly or may cause compatibility issues, downtime, or similar disruptions of service. This is undesirable in applications such as social networks, online shopping, or even online games, which are expected – and almost required – to run continuously with little to no downtime, lest they lose clients and customers.

Due to the growing demand for services to expand and operate at ever-increasing speeds, the limitations imposed by relational databases have paved the way for the growth and development of NoSQL databases.

2.3. Exploring NoSQL Databases

Though defined earlier as 'Not Only SQL', the term NoSQL has different definitions and is not clearly defined. This report will refer to NoSQL as database management products that do not adhere to traditional SQL or relational data models (Ceiko, 2011, pp. xvii) (Drake, 2019). NoSQL databases come in various forms (i.e. *data models*), including, but not limited to:

- Key-Value Store, such as HazelCast;
- Columnar, such as Cassandra;
- Document Store, such as MongoDB; and

- Graph Databases, such as OrientDB (Drake, 2019) (Ceiko, 2011, pp. xvii).

This report will be exploring HazelCast and MongoDB, from both the perspective of Document and Key-Value stores.

Key use-cases for NoSQL centre around overall performance and processing (notably the processing of 'big data') through parallel models and the use of semi- or unstructured-data for both information service and information collection. Practices such as 'data mining', or the process of processing and analysing large swathes of data to extract information (Techopedia, 2019) and are used extensively for personalisation and service delivery (Stephan, 2015).

III. Results

This section will explore the aspects of NoSQL and how they compare to MySQL.

3.1. Key-Value Store

Also known as key-value databases, they operate in a similar manner to dictionaries or hash-tables consisting of key-value pairs. With the key acting as the unique identifier to access any value and as a result, has no predetermined structure or schema. One key difference between key-value and document-based stores is that all data within a key-value database, all information is treated as 'opaque blob'. Opaque refers to the system treating the stored data as unobservable, meaning that the application itself knows the structure of the datastore. There are genuine benefits to the opaque nature of key-value stores, notably in privacy- or security-sensitive applications such as web banking, where information about a user cannot be obtained or extrapolated from the database without directly accessing the database itself. In addition, 'blob' refers to the system requiring no upfront data modelling.

This is in stark contrast to relational databases, which aren't necessarily opaque but require a strict schema to set up. The benefit of Key-value stores is that speed of retrieval; like hash-tables, unique keys and pointers enable the retrieval of very precise information at very high speeds. (Drake, 2019) (Kulkarni, 2019) An example of code for HazelCast is given in figure 3.1.1, showing the key-value properties and the ability to replace values by key.

```
var hz = HazelcastClient.NewHazelcastClient();
// Get the Distributed Map from Cluster.
var map = hz.GetMap<string, string>("my-
distributed-map");

//Standard Put and Get.
map.Put("key", "value");

Console.WriteLine(map.Get("key"));
// Output: value

map.PutIfAbsent("somekey", "somevalue");
map.Replace("key", "value", "newvalue");
// Shutdown this Hazelcast Client
hz.Shutdown();
```

Figure 3.1.1 – Example Code for a C# HazelCast Client

Addressing the issue of horizontal scalability, HazelCast can be set up such that server members can be set up where increased demand must be met. Figure 3.1.2 shows the server/client set-up of HazelCast In-Memory Data Grid (IMDG). When the database service needs to be scaled up, the documentation recommends "just [adding] more HazelCast server members" (HazelCast Docs, 2019, Section 3.2.).

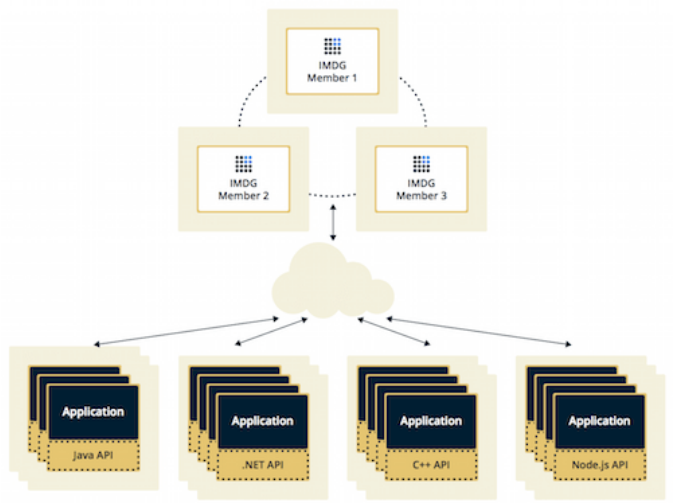

Figure 3.1.2 – Figure illustrating the HazelCast client/server model (HazelCast Docs, 2019, Section 3.2.)

3.2. Document Store

As the name suggests, document-store databases store information in documents and are a subtype of Key-Value stores, where each document contains a key and the document itself is the value (Drake, 2019).

Documents are often stored in JSON, XML, or similar formats where the information of an object is given by the structure of a document. An example of this data structure is given in figure 3.2.1, showing an object's recipe for *Factorio* (Wube Software, 2019) (kevinta893, 2019). This form of flexible database allows the game developers to edit, adjust, and include entirely new sections for any recipe as desired, whilst keeping the database in human-readable format.

```
[
    {
        "id": "accumulator",
        "name": "Accumulator",
        "type": "Machinery",
        "wiki_link":
"https://wiki.factorio.com/Accumulator",
        "category": "Production",
        "recipe": {
            "time": 10,
            "yield": 1,
            "ingredients": [
                {
                    "id": "battery",
                    "amount": 5
                },
                {
                    "id": "iron-plate",
                    "amount": 2
                }
            ]
        }
    },
...
```

```
]
```

Figure 3.2.1 – a snippet of the JSON file used to store recipes for Factorio, showing the Accumulator.

Note that figure 3.2.1 is from the *recipe* document and contains no other information about the object itself. This is one of the other features of document stores: the ability to spread information across multiple documents, tables, or even databases. This is important in a concept known as *Sharding*, allowing databases to be expanded horizontally at ease when compared to relational databases. Thus, due to the incredible flexibility and scalability of document-stores, they have garnered massive popularity with MongoDB currently ranked the #5 most popular database only behind keystone systems such as PostgreSQL and Oracle's offerings. (DB-Engines, October 2019)

3.2.1. MongoDB Data Model

MongoDB is a document store, saving information in JSON format. One important note is that, according to the documents, "the denormalized data model is optimal" (MongoDB Docs, 2019). An example document is given in figure 3.2.1.1.

Figure 3.2.1.1 – The MongoDB Data Model, given as an example (MongoDB Docs, 2019)

With no strict schema, MongoDB can still model one-to-one and one-to-many relationships. Figure 3.2.1.2 shows an object ID used to connect one document to another.

```
{
   _id: "joe",
   name: "Joe Bookreader"
}

{
   patron_id: "joe",
   street: "123 Fake Street",
   city: "Faketon",
   state: "MA",
   zip: "12345"
}
```

Figure 3.2.1.2 – Example one-to-one relationship in MongoDB (MongoDB Docs, 2019).

3.3. MapReduce

A key differentiator between NoSQL and relational databases is the MapReduce model, allowing for fast retrieval of astronomical amounts of data from file systems (Ceiko, 2011, pp. xviii). It should be noted that MapReduce is a programming framework (read: way of programming) and has open-source implementations such as Apache Hadoop (hadoop.apache.org).

Compared to SELECT statements from MySQL, MapReduce allows the parallelising of the data analysis over a distributed 'cluster' (group) of individual server nodes (sessions / shards / instances).

3.4. Comparison to MySQL

When it comes to the design, development, and overall documentation of the required DBMS, it is rather difficult to compare MySQL and relational systems to NoSQL products as in the fullest sense, it is comparing apples to oranges. Based on the situation, there are various considerations a developer should make when considering the right product for the problem set at hand.

3.5.1. Querying

Querying refers to the ability to search for certain entries, documents, or tuples, based on certain criteria within a database. HazelCast supports various query flavours, HazelCast also supports SQL-like syntax. On the other hand, MongoDB utilises a system called *CRUD*, referring to the ability to *Create*, *Remove*, *Update*, and *Delete* documents.

Despite the flexibility of HazelCast, the system's ability to query is not particularly intuitive. Figure 3.5.1.1 highlights the C# code require to query an employee that is currently active and is aged above 30.

```
IMap<String, Employee> map =
hazelcastInstance.getMap( "employee" );
Set<Employee> employees =
map.values( Predicates.sql( "active AND
age < 30" ) );
```

Figure 3.5.1.1 – SQL-like queries for HazelCast (HazelCast Docs, 2019)

This is compared to MySQL's SQL Statements in figure 3.5.1.2 and MongoDB's object querying notation in figure 3.5.1.3.

```
SELECT employeeID FROM Employees
WHERE active=true AND age > 30;
```

Figure 3.5.1.2 – SQL queries within MSQL

```
db.employees.find(
{ {active : "true"}, {age : { $gt 30} } } }
```

Figure 3.5.1.3 – A MongoDB 'Find' query using object notation

3.5.2. Data Input

An interesting but minor note to make here is the data model and the subsequent data input required for each database product. In the case of relational databases such as MySQL, the strict transaction-based ACID approach allows for multi-row transactions and strong database integrity.

In the case of the Distinction Task for Fundamentals of Data Management, the strong schema enforcement of MySQL meant

that table-based data was an effective solution for the organisation and input of data. That is, table-based data with relationships between then was relatively painless to modify and implement for the RDBMS. This is apparent in the *Kantai Collection Database* implemented for the Distinction Level task which converted table data relating to the statistics of the ships into a relational database. Additional examples of this can be seen in games such as PAYDAY 2 with its wide array of weapons which all conform to certain characteristics.

On the other hand, NoSQL databases such as HazelCast, but in particular, MongoDB, are more suited for unstructured data that can be horizontally scaled, and its schema adjusted down the line without major hassle. As noted earlier in figure 3.2.1, Wube Software's *Factorio* utilises a NoSQL document-based structure for their recipe system due to the flexibility needed and the highly complex interactions of the game and its components.

3.5.3. Summary

In the end, it is in the developer's interest to choose the correct database product for the problem at hand.

The strengths of the systems are given in table 3.5.3.1 and the cons in table 3.5.3.2. (Eugeniya, 2019)

Table 3.5.3.1 – The strengths of the DBMS'

MySQL
• Strong and very strict transaction and isolation level support • JOIN support • Widely used and adopted ('mature')
MongoDB
• Document Validation • Flexible and non-strict schema • Community-focused with open development
HazelCast
• Straightforward key-value store • Normally used as a cache • Opaque database provides security

Table 3.5.3.2 – The weaknesses of the DBMS'

MySQL
• Difficult to scale • Strict schema making it difficult to modify • Stability concerns • Owned by Oracle, not open to community-driven development
MongoDB
• Does not operate well with complex transactions • Difficult to replace legacy solutions • Not 'mature'
HazelCast
• Is only a key-value store • Technically complex with limited support

IV. Conclusion

There is no one-size fits all solution due to the limitations of hardware and software. Depending on the analysis of the situation, a proper solution requires proper design, development, support documentation, and long-term implementation and troubleshooting support for the client. Based on the requirements identified in the analysis, the best solution for any business is the solution that best conforms to the requirements set out by the client. Accounting systems and organisations that require strict adherence to transactions see stronger usage of relational systems such as MySQL. Other applications such as real-time analytics and content management, where there is no strict definition or there is a requirement to be flexible and schema-free, NoSQL systems such as MongoDB and HazelCast provide strong support for dealing with huge sets of data. (Eugeniya, 2019) (Sarig, 2017).

V. References

Ceiko, K, 2011, *Joe Celko's Complete Guide to NoSQL: What Every SQL Professional Needs to Know About Non-Relational Databases*, Waltham, MA: Morgan Kaufmann.

DB-Engines, *the most popular database management systems*, viewed 14 October 2019, < https://db-engines.com/en/>

DB-Engines Ranking Trend, *DB-Engines Ranking - Trend Popularity*, *viewed* 14 October 2019, < https://db-engines.com/en/ranking_trend>

Drake, M, 2019, *A Comparison of NoSQL Database Management Systems and Models*, DigitalOcean LLC, viewed 14 October 2019, <https://www.digitalocean.com/community/tutorials/a-comparison-of-nosql-database-management-systems-and-models>

HazelCast IMDG Docs, 2019, last viewed 17 October 2019, < https://docs.hazelcast.org/docs/latest-dev/manual/html-single/>

Eugeniya, 2019, 'MongoDB vs MySQL Comparison: Which Database is Better?', *Hackernoon*, last viewed 26 October 2019 < https://hackernoon.com/mongodb-vs-mysql-comparison-which-database-is-better-e714b699c38b >

Kaur, A, 2017, 'ACID Properties in DBMS', *Geeks for Geeks*, viewed 14 October 2019, < https://www.geeksforgeeks.org/acid-properties-in-dbms/>

kevinta893, 2019, 'factorio-recipes-json' on Github, < https://github.com/kevinta893/factorio-recipes-json>

Korpal, A, 2019, 'Scaling Horizontally and Vertically for Databases', *Medium*, viewed 14 October 2019, < https://medium.com/@abhinavkorpal/scaling-horizontally-and-vertically-for-databases-a2aef778610c>

Kulkarni, C, 2019, 'NoSQL Databases – Key-Value Store', *Core View Systems*, viewed 15 October 2019, < https://coreviewsystems.com/nosql-databases-key-value-store/>

Technopedia, 2019, 'Data Mining', viewed 15 October 2019, <https://www.techopedia.com/definition/1181/data-mining>

Sarig, M, 2017, 'MongoDB Vs MySQL: The Differences Explained', *Panoply*, viewed 26 October 2019, < https://blog.panoply.io/mongodb-and-mysql>

Stephan, Tim, 2015, *10 use cases where NoSQL will outperform SQL*, Network World, viewed 15 October 2019, <https://www.networkworld.com/article/2999856/10-use-cases-where-nosql-will-outperform-sql.html>

Vishwakarma, A, 2018, 'Difference between Structured, Semi-structured and Unstructured data', *Geeks for Geeks*, viewed 15 October 2019, <https://www.geeksforgeeks.org/difference-between-structured-semi-structured-and-unstructured-data/>

Wei-Ping, Z, Ming-Xin, L, Huan, C, 2011, 'Using MongoDB to Implement Textbook Management System instead of MySQL', *IEEE 3rd International Conference on Communication Software and Networks*, Xi'an, pp. 303-305.

Wube Software, 2019, *Factorio*, <https://factorio.com/>