

Bachelor's thesis
Information Technology
NINFOS13
2017

Xiaochuan He

NOSQL ANALYSIS AND A CASE STUDY OF MONGODB



BACHELOR'S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information Technology

2017 | 25 pages

Xiaochuan He

NOSQL ANALYSIS AND A CASE STUDY OF MONGODB

While the booming of NoSQL technology breaks the old pattern in data storage industry, experts start to pay more attention to its performance in response to market demands. The purpose of this thesis was to introduce NoSQL, analyze its performance based on current industry background as well as implement a case study of data replication in MongoDB. Firstly, this thesis has focused on elaborating its origin, features as well as pros and cons. Then by analyzing and summarizing its properties, it can be concluded that the NoSQL technology provides many benefits to solve the problems caused by RDBMS deficiencies, however the disadvantages of NoSQL have to be considered when implementing it. Starting a NoSQL database or converting current RDBMS to NoSQL requires deep understanding of NoSQL and consideration of specific scenarios. A case study of data replication by using MongoDB has been implemented to show how MongoDB achieves high data availability.

KEYWORDS:

NoSQL, MongoDB, RDBMS, databases, SQL

CONTENTS

LIST OF ABBREVIATIONS (OR) SYMBOLS	5
1 INTRODUCTION	6
2 NOSQL FUNDAMENTALS	7
2.1 Definition	7
2.2 History	7
2.3 NoSQL theory	8
2.3.1 CAP theorem	8
2.3.2 BASE	9
2.4 Characteristics	10
2.5 NoSQL data storage types	11
2.5.1 Document store	11
2.5.2 Key-value/ tuple store	12
2.5.3 Graph store	13
2.5.4 Column store	14
3 NOSQL ANALYSIS	15
3.1 RDBMS approach	15
3.2 NoSQL approach	18
4 CASE STUDY	22
4.1 Replication in MongoDB	22
4.2 Implementation	23
4.2.1 Setting up	23
4.2.2 Data replication	26
4.2.3 Failover in replication	27
4.3 Conclusion	29
5 CONCLUSION	30
REFERENCES	31

PICTURES

Picture 1. Three characteristics of NoSQL. [8]	10
Picture 2. Key-value store.	12
Picture 3. Graph database. [9]	13
Picture 4. Replication in MongoDB [14].	23
Picture 5. Environmental variable.	24
Picture 6. Replica sets status.	25
Picture 7. Data on secondary node.	26
Picture 8. Insert error on secondary node.	27
Picture 9. Shutting down primary node.	27
Picture 10. Secondary node becomes primary.	28
Picture 11. Insert data on primary node.	28
Picture 12. Reconnect to port 27017.	29

GRAPHS

Graph 1. ER diagram.	16
Graph 2. Class diagram	16

LIST OF ABBREVIATIONS

ACID	Atomicity, Consistency, Isolation, Durability
BASE	Basic Availability, Soft state, Eventual consistency
BSON	Binary JavaScript Object Notation
DBMS	Database Management System
RDBMS	Relational Database Management System
SQL	Structured Query Language
UUID	Universally Unique Identifier
JSON	JavaScript Object Notation
XML	Extensible Markup Language

1 INTRODUCTION

Relational databases have been used in data storage over the past 40 years while SQL has been an important language for RDBMS. In the last few years, WEB 2.0 era has come, a generation that websites emphasize user-generated content, usability, and interoperability for end users. As an impact to data storage market, dealing with user-generated data that is more than ever became a technical issue. In meantime, new market demands gradually appeared. Under this circumstance, experts realized that RDBMS was no longer the universal key for all cases. Consequently, NoSQL products has been developed to solve current database system problems and meet the new market demands. By the test of time, NoSQL has attracted attention because of its advantages such as flexible data model, elastic scalability as well as high performance. However, as a matter of fact, many of these new database management systems while providing great innovations and improvements have also sacrificed some critical properties that have made RDBMS become the gold standard for decades [1]. Thus, although NoSQL offers plenty reasons to be chosen in the current market, implementation of NoSQL databases requires deep understanding of NoSQL products and detailed analysis on practical application.

This thesis shows the significant features of NoSQL that make NoSQL differ from RDBMS and fit better than RDBMS in Web 2.0. Additionally, a case study using MongoDB [2] has been implemented to show how MongoDB achieves high data availability by data replication.

2 NOSQL FUNDAMENTALS

This chapter has introduced NoSQL fundamentals which give theoretical basis for NoSQL analysis.

2.1 Definition

In computing, NoSQL (mostly interpreted as “not only SQL”) is a broad class of database management systems identified by its non-adherence to the widely used relational databases management system model [3].

NoSQL is not an abbreviation for a certain technology or product. It refers to a wide variety of different database technologies by using a general term, for example, MongoDB (document-oriented databases), Cassandra (column store databases) and Redis (in-memory databases). Although NoSQL is a large family of those new technologies developed to meet the demands in different scenarios, the non-adherence to RDBMS is one significant property that they have in common. Thus, NoSQL can be regarded as “no more SQL” or “not only SQL” .

2.2 History

During the last decade, World Wide Web websites have stepped in to a new generation that emphasizes user-generated content, usability and interoperability for end users, called “Web 2.0”. This change means that a website can function well and, at the same time, deal with other products, systems and devices.

With Web 2.0, the amount of user-generated data has been rapidly increasing. As for the data storage market, relational databases have been used in industry over the past 40 years while SQL has been an important language for RDBMS. However, because of the shock brought by Web 2.0, new data technology demands appeared to meet the market needs. For RDBMS, “one size fits all” is unlikely to successfully continue under these circumstances [4].

RDBMS is based on a complex relational framework that is constructed by tables linked to each other. Additionally, traditional RDBMS applications have highly focused on ACID transactions [3]:

- Atomicity: Everything in a transaction succeeds lest it is rolled back.
- Consistency: A transaction cannot leave the database in an inconsistent state.
- Isolation: One transaction cannot interfere with another.
- Durability: A completed transaction persists, even after applications restart.

Thus, in other words, a company that owns relational databases should make a huge effort on redefining the previous schemas when the company expands business or integrates with other enterprises. In addition, due to the consistency that RDBMS strictly follows, data processing will slow down when the amount of data increases. Consequently, many new types of database management system have been developed as solutions to the issues mentioned above.

The term NoSQL was used by Carlo Strozzi in 1998 to name his DBMS, Strozzi NoSQL open-source relational database which was still based on relational model [5]. Since then, the term NoSQL has been reintroduced by Johan Oskarsson of Last.fm when he organized an event to discuss "open source distributed, non-relational databases" [6]. After that NoSQL gradually became the label for non-relational, distributed data stores that increasingly emerged.

2.3 NoSQL theory

Unlike to RDBMS which is supported by ACID transactions, there are two basic theories applied in NoSQL concept. In this chapter, the CAP theorem and the BASE transaction are elaborated to help understand what tenets NoSQL are following.

2.3.1 CAP theorem

As Eric Brewer stated for distributed computer systems, "that though its desirable to have Consistency, High-Availability and Partition-tolerance in every system, unfortunately no system can achieve all three at the same time" [7]. The CAP theorem means that it is impossible for a distributed computer system to achieve all of the following guarantees:

- Consistency: A read process is able to receive the most recent write or error.
- Availability: Every request receives a (non-error) response – without guarantee that it contains the most recent write.
- Partition tolerance: the system continues to function when network partitions occur.

Because NoSQL databases are based on the distributed computer system and as a matter of fact that networks go down frequently and unexpectedly, partition tolerance is necessary when implementing NoSQL. Therefore, according to the CAP theorem that a distributed computer system cannot achieve consistency, availability and partition tolerance at the same time, NoSQL databases basically focus on consistency/partition tolerance or availability/partition tolerance.

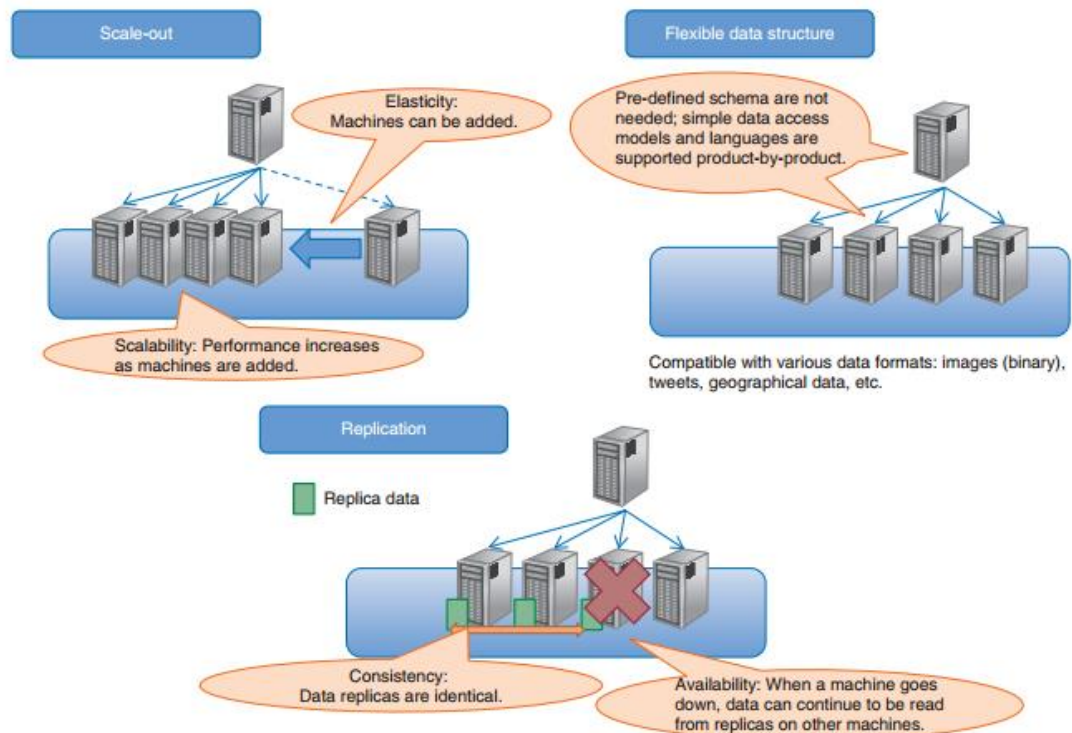
2.3.2 BASE

The ACID transaction has been mentioned in the previous chapter when introducing the advent of NoSQL. ACID is the prominent transaction that RDBMS strictly follows, but NoSQL focus more on BASE [3]:

- Basic Availability: every request will receive a response that if the request succeed or failed.
- Soft state: the system status changes over time without any input
- Eventual consistency: The database system may be temporarily inconsistent in soft state but will be consistent eventually.

2.4 Characteristics

There are 3 main characteristics of NoSQL: scale-out, replication, and flexible data structure [8].



Picture 1. Three characteristics of NoSQL. [8]

Scale-out means that NoSQL uses general-purpose machines in a distributed manner to achieve high performance. Distributing the data over a large number of machines enables scaling of the data set and distribution of the processing load.

Replication is to copy data to achieve data redundancy and load distribution. with the help of replication, even if data consistency has been lost temporarily among the replicas, consistency would be achieved eventually. When one machine goes down by accident or maintenance purpose, data can continually be read from replicas on other machines, which gives users availability for 24 hours.

Flexible data structure means that NoSQL databases do not have to define a complex database schema as what traditional RDBMS always requires. Therefore, NoSQL could allow users to store data with various structures in the same database table, for example, images, tweets and geographical data.

2.5 NoSQL data storage types

As mentioned above, NoSQL is compatible with various data types. Subsequently, 4 main storage types that NoSQL provides will be introduced.

2.5.1 Document store

Document store means that the database is designed to store data as documents. It allows data inserting, manipulating and retrieving. Document databases use XML, JSON or BSON. Three examples of documents by using JSON follow.

The first document contains the basic identity information for a student, however it is not detailed.

```
{
  "StudentID": "1300001"
  "Firstname": "Xiaochuan"
  "LastName" : "He"
}
```

Compared to the first one, more information is provided in the second example.

```
{
  "StudentID": "1300002"
  "Firstname": "San"
  "LastName" : "Zhang"
  "Age"      : "23"
  "Gender"   : "male"
}
```

The content in the third document is different from the two previous.

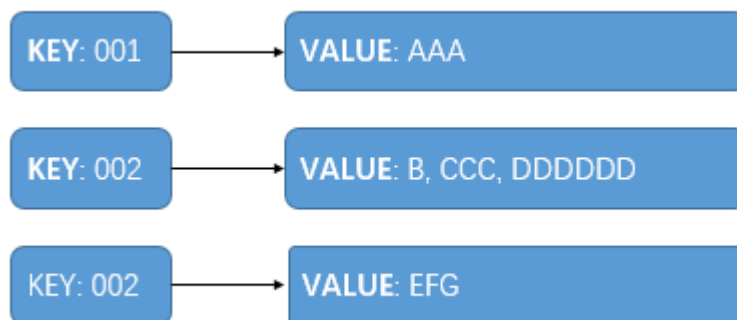
```
{
  "Degree Programme " : "Information Technology"
  "Location"          : "ICT building"
  "RegisteredStudentNumber": "200"
  "RegisteredTeacherNumber": "30"
}
```

From these examples, it is easy to see that the documents do not follow a strict schema as RDBMS databases always do. Furthermore, there is StudentID in first two examples, however, there is no ID information in the third one. As a matter of fact, StudentID is not the document ID, because document store databases embed the document ID automatically in the document somewhere. Thus, ID is not mandatory in document store.

In conclusion, the most prominent advantage of document store databases is that the content is schemaless. This is a very helpful property for web-scale application where DBMS is required to store different types of data that might evolve over time since redefining the strict schema for RDBMS became a huge cost.

2.5.2 Key-value/ tuple store

Key-value store functions are similar to document store. The data structure for a key-value database is commonly known as a dictionary or hash and it contains of a collection of records that has many different fields, also called, columns. The records are stored by using a unique key to identification. The structure is presented in Picture 2.



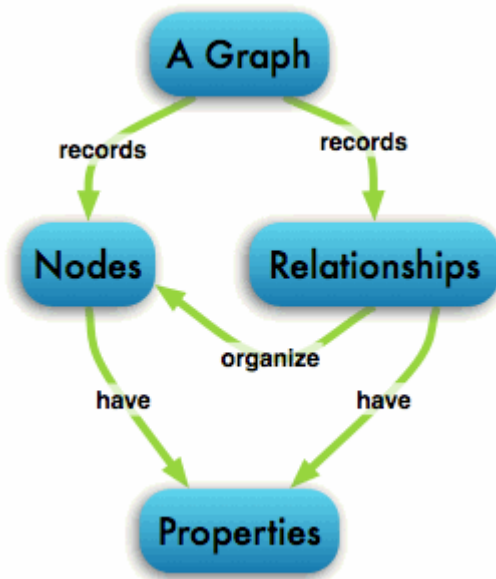
Picture 2. Key-value store.

Because of the key, it is quick to find specified data in key-value store. However, key-value also has few constraints that should be noticed. For the document store, manually setting UUID is not mandatory when storing a new document, but the key-value database requires specifying the key for every record when it is inserted. Besides that, values in records are opaque. In order to retrieve a value, the key must be known.

Key-value store has great performance on data storage, retrieval, and management. As such, it is a strong candidate for the situations that have heavy need for in-memory caches. Although a key-value database cannot directly query the values, it can still know the value type, which enables extra functionality. To be specific, since the value types are known, it is possible to keep the system performance when setting or updating multiple fields in some part of the document.

2.5.3 Graph store

A graph database, also called a graph-oriented database, is fairly new in the market. It refers to a database with an explicit graph structure that is constituted by nodes and links. Nodes represent the objects/entities in the data structure and each two nodes are connected by relationship. Picture 3 is a simple model of a graph database.



Picture 3. Graph database. [9]

One main advantage of graph databases is the powerful representation and manipulation of relationships. Because of the performance on presenting relationship that the majority of NoSQL products have sacrificed, a graph database is a good choice for analyzing interconnection and social media. However, due to their design, graph databases are essentially used to present and clarify collections of relationships. There is no place for them if relationships are not needed. Therefore, graph databases can be considered as an optimization of NoSQL especially for relation-heavy data. Besides, it is common that for certain scenarios, an enterprise chooses a graph database only for relationships and store data in a document-oriented database.

2.5.4 Column store

In traditional RDBMS, datasets are stored in tables which is constituted of columns and rows and, storing and retrieving data is processed one row at a time. However, the column-oriented databases use columns to manage data.

To illustrate the above, a group of data (order information that contains OrderID, OrderName, Quantity and OrderTime for this case) for a RDBMS table can be serialized as:

```
0001, fish, 50, 25.01  
0002, beef, 80, 08.02  
0003, pork, 30, 16.03
```

But in column-oriented database, this group of data will be stored as:

```
0001, 0002, 0003  
fish, beef, pork  
50, 80, 30  
25.01, 08.02, 16.03
```

Due to the difference in how the data has been stored, most column-oriented databases give flexibility to model and structure the data so that there is no restriction to set default values for existing rows when adding new columns. Therefore, one prominent advantage of column store is providing high flexibility for new columns. Additionally, because of the complex data process, the performance of RDBMS compromises when working with subset of columns, especially when dealing with large amount of data, to be more specific, calculating maximum, minimum, averages or sums. A column-oriented database provides reliable functionality for data computation due to its structure.

3 NOSQL ANALYSIS

NoSQL does provide solutions to many data storage problems. However, it is not a solution to every problem. NoSQL could be applied appropriately only when we have comprehended its pros and cons. This chapter is going to show when NoSQL is suitable for data storage and fits better than RDBMS.

In the following sections, firstly a simplified example is used to present RDBMS approach and the content focuses on analyzing RDBMS bottlenecks under the current data storage market. Subsequently, as a solution, the NoSQL approach is introduced. Pros and cons are summarized based on the information above. Finally, a summary of NoSQL is made.

3.1 RDBMS approach

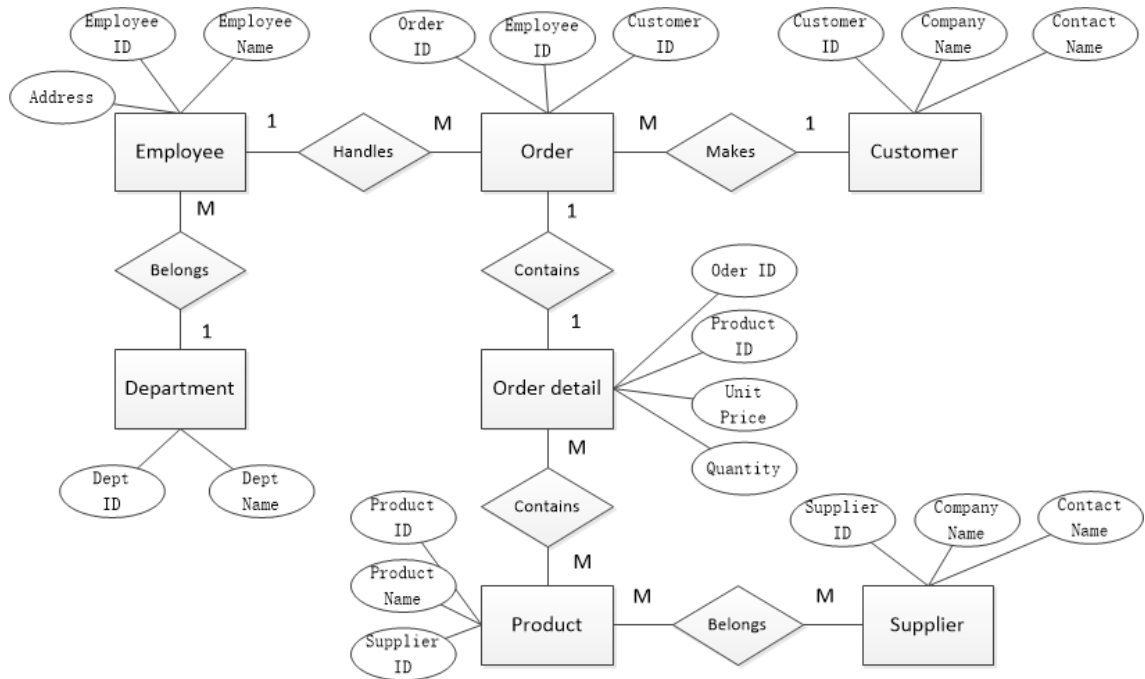
The example used here is a DBMS for a business. The company in this example deals with food delivery, receiving orders and delivering food. It should be noticed that in real life, it is far more complex than this one. This example is only used to give an idea how RDBMS works, which helps summarize what challenges traditional RDBMS is facing currently.

The RDBMS approach uses the following steps.

1. Define the actors and the objects in the transaction process.
2. Define entities, which means forming a table according to the objects for this case, defining columns and rows. Besides, column types and constraints also should be defined.
3. Define relationships from one table to another by using foreign keys which means that we need to find out the relationship between entities. The entity relationships include one-to-one, one-to-many, many-to-many and other object relationships.
4. Program database by using SQL and develop the application.

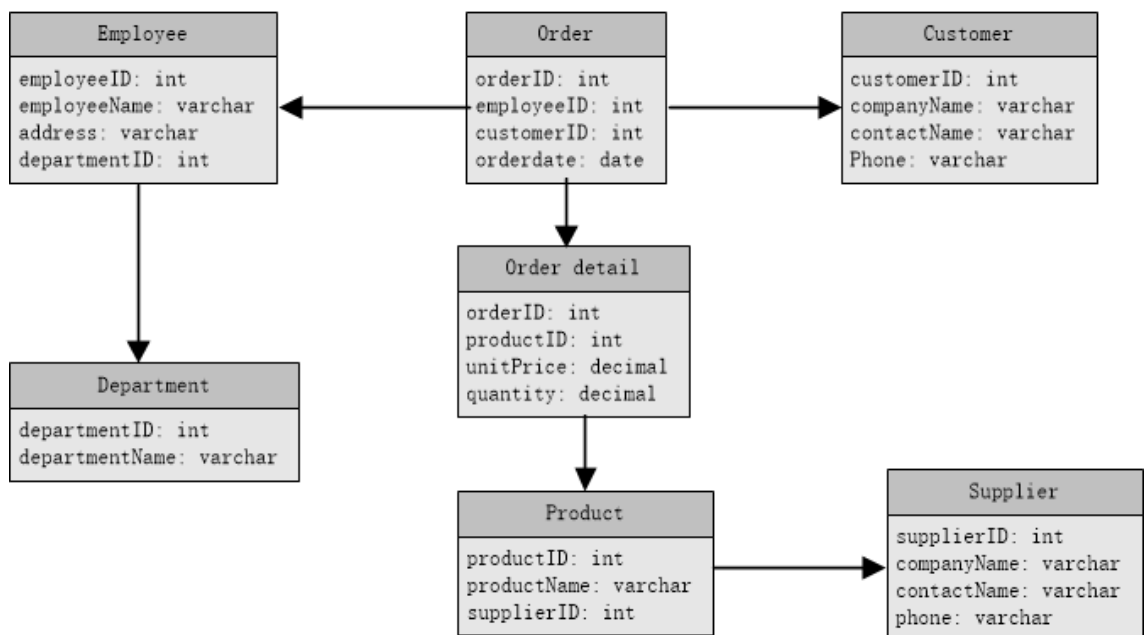
In this case, for a DBMS for a food company, firstly the actors include employee, customer and order. Then employeeID, lastname and firstname are the entities for actor employee. Foreign keys will be used to define relationships between tables, for example, it is one-to-many relationship between order and customer, many-to-many relationship

between product and order detail. The following picture is the entity-relationship diagram for this case.



Graph 1. ER diagram.

The following representational class diagram shows the value type of the data and presents relationships from the table point of view.



Graph 2. Class diagram

To illustrate, here is the code that forms the Order table:

```
CREATE TABLE Order(  
orderID INT NOT NULL AUTO_INCREMENT,  
employeeID INT NOT NULL,  
customerID INT NOT NULL,  
orderDate DATE NOT NULL,  
PRIMARY_KEY (orderID),  
FOREIGN_KEY (employeeID) REFERENCES Employee(employeeID),  
FOREIGN_KEY (customerID) REFERENCES Customer(customerID),  
);
```

The RDBMS approach has been presented above and it seems great and reasonable, however, it has challenges. The issues that the food company might have in future are listed.

- The company wants to implement online orders to expand business, which requires modifying the current entities or creating new entities.
- The company decides to integrate with another company for expansion. It means that two database systems need to be integrated somehow. It is possible that they use different table structure because of their different business model. Therefore, redefining models, entities and table relationships are required.
- The company grows and has few branches in other cities. This means that remote access to database from branches should be enabled. Additionally, the database system will receive a huge amount of order per day across the country.

The cases above can be converted into technical demands for RDBMS.

- Schema flexibility: it is easy to see that because of its complex table design, redefining relationships is hard to be implemented once a RDBMS has been set up. As for entities modification, according to the diagram and code for table presented above, we can tell that data constraint/type and default value are mandatory for every column. Thus, adding or modifying entities is a complex work especially when adding multiple columns for large number of row. Additionally, besides the huge work of adding or creating entities itself, the schema will become more inflexible. It is an endless loop.
- High-level queries: Due to the table structure, retrieving data, for example, JOIN queries requires implementing many database resources. The following codes are an example for JOIN queries.

```
SELECT b.branchNo, b.city, s.staffNo, fName, lName,propertyNo
FROM Branch b, Staff s, PropertyForOrder p
WHERE b.branchNo = s.branchNo AND s.staffNo = p.staffNo
ORDER BY b.branchNo, s.staffNo, propertyNo;
```

- Data update: it is another technical issue that updating data across multiple tables is complex especially if it is in part of transaction, because keeping the transaction in process for a long time impact to performance negatively.
- Scalability: When data is stored locally, scalability means adding more capacity to a single machine. For this case, distributed systems should be implemented to enable remote access from branches, scalability means add capacity by adding more machines to cluster. However, according to the CAP theorem, it is impossible to guarantee consistency, availability, and partition tolerance at the same time in a distributed system. Achieving ACID transactions in distributed systems makes it hard to build distributed database systems based on the relational model and it will hinder low latency and high availability [10].
- Data availability: distributed relational databases use replication to provide data availability. Data on the main server will be copied to other servers and it automatically copies the latest change. However, due to the restrictions from the CAP theorem, ACID transactions and the complex table structure, data integrity is complex to be maintained from multiple machines. Furthermore, because of the low tolerance to partition, there is a high possibility that the whole database system shuts down when a single machine crashes.

3.2 NoSQL approach

Here is the NoSQL approach that provides solutions to the technical issue listed in the RDBMS approach.

- Schema flexibility: compared to RDBMS that stores data in complex table connected by relationships, there is no such constraint in NoSQL databases. Column-oriented databases give adequate convenience and flexibility for adding columns. Additionally, document databases are strong candidates due to their support for semi-structured data.
- High-level queries: There are no relationships or foreign keys in NoSQL structure, thus, there are not complicated queries that are executed across multiple tables. To some extent, NoSQL maintains the high performance of database system.

- High scalability: NoSQL is ACID non-compliant as there is no complexity to add machines to current cluster. As such, NoSQL products provide better scalability than RDBMS. It is easier to scale out for the sake of future development. Besides, high scalability is the prerequisite for achieving high availability of data, data safety and some other purposes.
- Data update: data update and synchronization are always hard to balance within a datacenter or across multiple datacenters. For NoSQL databases, the system may be temporarily inconsistent in soft state but will be consistent eventually. Every transaction does not need to strictly follow consistency. It gives NoSQL databases strong ability on data update especially when dealing with huge amount of data, e.g. MongoDB implements concurrency control which allows concurrent updates to be synchronized and eventually be consistent across nodes in few milliseconds [12].
- High data availability: NoSQL provide replication among multiple datacenters to achieve data redundancy and load distribution. Compared to distributed relational databases, NoSQL has sacrificed consistency but in return, it ensures high partition tolerance which helps achieve data availability for 24 hours. When one machine goes down, data can still be read from replica sets on other machines without the risk of downtime.
- Massive writing performance: NoSQL provides outstanding write performance which is able to easily handle the massive user-generated data from different applications at the same time. For example, Netflix has switched their databases from RDBMS to Cassandra (one product of NoSQL) and the write performance could reach over a million times per second [13].

It is worthwhile to mention that the RDBMS license price for enterprise is hard to afford for small companies, and it rises rapidly with the increase of scale. Comparably, there is no high expenditure for database license and maintenance for NoSQL databases since most of NoSQL database software is open source.

The drawbacks of NoSQL cannot be ignored:

- No support for high-level query languages. Most NoSQL databases do not support high-level query languages such as SQL, which means there is no JOIN and cross-entity query for data manipulating. Although the complexity of SQL in RDBMS is an issue that can compromise the system performance, no support for these queries in NoSQL is still a drawback. To fix this, many databases focus on the integration of queries simplification, data cache, and complex operations in application layer.

Additionally, due to the varieties of different NoSQL database, learning and immigration would cost more than expected.

- No ACID transactions: ACID transactions provide the strongest guarantees for data consistency and make it much easier for developers to build reliable abstractions. However, NoSQL focuses more on BASE transactions. Thus, this property makes NoSQL a weak choice for transactional applications.
- Less support: all the RDBMS vendors have made a great effort to keep data reassurance service available at any time, which gives enterprises strong support. In contrast, because NoSQL databases tend to be open source, many of them lacks the resources to fund support on a global scale, as well as the credibility that the RDBMS vendors have established for years, for example, Oracle, IBM and Microsoft.

Indeed, NoSQL does bring a lot of benefits to current data storage industry, but it still has obvious deficiencies. Which one are we supposed to choose, traditional RDBMS or NoSQL? There is no necessity to give an absolute answer because the final decision depends on the type and scale of application.

To illustrate the above point, for transactional application, generally speaking, the consistency and integrity of data and ACID transaction are extremely important properties. NoSQL cannot replace RDBMS for this case. For web-scale application, however, it would generate massive user-generated data all day and most of enterprises do not want to compromise the database performance. Additionally, in the long term for business, it requires high scalability and flexible schema. Consequently, NoSQL perform a better implementation for web-scale application. Additionally, as for computational application, both of RDBMS and NoSQL can be implemented. RDBMS support high-level queries and the column-oriented database can reach high speed for data computation. Theoretically it depends only on the size of data. Besides, a composite choice of RDBMS and NoSQL are possibly needed in certain situation, having said that, relationships can be designed by using graph database and data sets can be stored in RDBMS, for example. On the one hand, the application type is an important factor to make a decision. On the other hand, the scale of application also should be taken into account. RDBMS is able to handle most of small-scale application. When scale grows, there is an increasing need on scalability and data availability. NoSQL is a good choice for this case if there is no special requirement on ACID transactions and high-level query.

In conclusion, NoSQL provides huge benefits that are able to remedy RDBMS deficiencies but the disadvantages do exist. However, it is not the solution to all database problems. Starting a NoSQL database or switching a current RDBMS requires more considerations depending on actual situation.

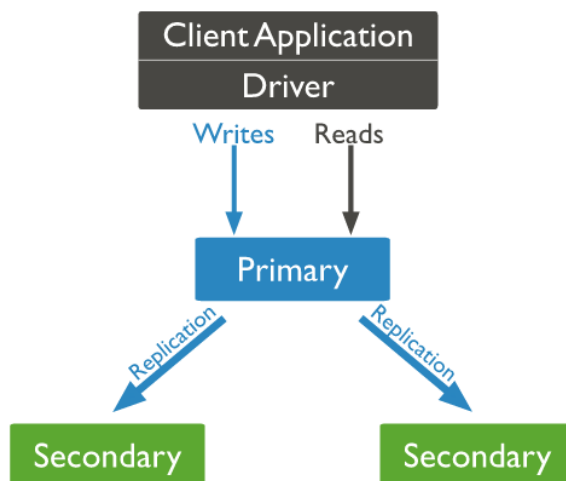
4 CASE STUDY

In the previous sections, the pros and cons of NoSQL have been analyzed and summarized at a macro level. To narrow it down, NoSQL products provide different technical solution to meet different market demands. This case study is to show how MongoDB achieves high availability and some other purposes by implementing data replication.

4.1 Replication in MongoDB

MongoDB is a document store database system encoded in BSON. A MongoDB database is set up by multiple collections and each one collection contains documents that carries data. MongoDB features automatic sharding, replication, support to rich queries and full indexes, and more.

According to the documents in MongoDB official website [14], a replica set is a group of instances that carry the same data set. A replica set constitutes of multiple data bearing nodes, one and only one node is assigned to be primary node, the other nodes are assigned as secondary nodes. The primary node has all write operations. The secondary nodes will replicate the operations of primary node and apply them to its own data sets. When the primary node is unavailable, one eligible secondary node will automatically become a new primary node which ensures that there is always a primary node available for all write operations.



Picture 4. Replication in MongoDB [14].

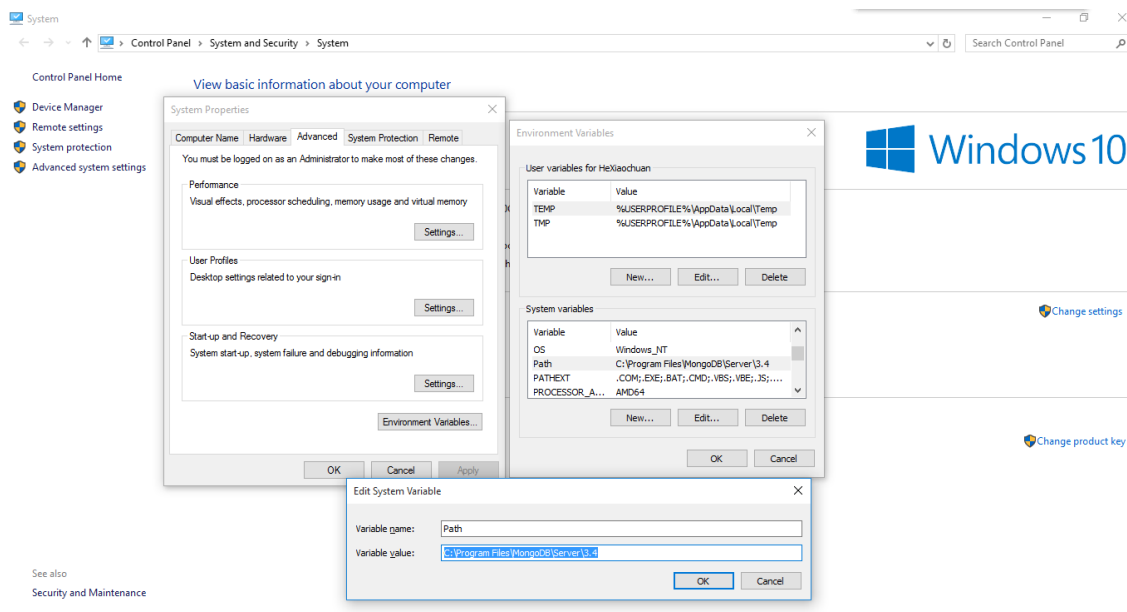
Because of the replication function, MongoDB provides high availability of data. Furthermore, replication ensures the possibility for data recovery. No downtime for maintenance is needed. Last but not the least, the replica set is transparent to the application.

4.2 Implementation

For this case study, MongoDB 3.4.2 downloaded from MongoDB official website is used. To implement data replication in MongoDB, 3 nodes as replica sets will be configured first. Then one of them will be deemed to primary and the others will be deemed as secondary. Subsequently, a set of data will be inserted to test the write operation access on those nodes. The final step is to shut down the primary node and check how secondary nodes react.

4.2.1 Setting up

It is necessary to set environment variable first.



Picture 5. Environmental variable.

Firstly, three directories were created to store three instances data (three replica sets).

```
mkdir \data\rs1 \data\rs2 \data\rs3
```

Subsequently, start three Mongod instances by following command on command prompt.

```
start mongod --replSet Xiaochuan --logpath \data\rs1\1.log --dbpath \data\rs1 --port 27017 --smallfiles --oplogSize 64
```

```
start mongod --replSet Xiaochuan --logpath \data\rs1\2.log --dbpath \data\rs2 --port 27018 --smallfiles --oplogSize 64
```

```
start mongod --replSet Xiaochuan --logpath \data\rs1\3.log --dbpath \data\rs3 --port 27019 --smallfiles --oplogSize 64
```

Three processes of Mongod.exe have started which are rs1, rs2 and rs3. Note that port 27017 is the default port for mongod instance and the replica sets are named as Xiaochuan.

Now three mongod servers are running, however, they are not configured to interconnect with each other. The next step is to configure the interconnection for all three nodes on port 27017 which means port 27017 will be deemed to be primary nodes and the other two will be secondary.

```
mongo --port 27017
```



```

config = { _id: "Xiaochuan", members: [
  { _id: 0, host: "localhost:27017" },
  { _id: 1, host: "localhost:27018" },
  { _id: 2, host: "localhost:27019" } ]
};

```

```
rs.initiate(config)
```

The command **rs.status()** is used to check the status of replica sets. As it shown from the following picture, local host 27017, 27018 and 27019 have been on the list, besides that, port 27017 become primary.



```

Command Prompt - mongo --port 27017
Xiaochuan:PRIMARY> rs.status()
{
  "set" : "Xiaochuan",
  "date" : ISODate("2017-03-31T19:32:34.693Z"),
  "myState" : 1,
  "term" : NumberLong(1),
  "heartbeatIntervalMillis" : NumberLong(2000),
  "optimes" : {
    "lastCommittedOpTime" : {
      "ts" : Timestamp(1490988750, 1),
      "t" : NumberLong(1)
    },
    "appliedOpTime" : {
      "ts" : Timestamp(1490988750, 1),
      "t" : NumberLong(1)
    },
    "durableOpTime" : {
      "ts" : Timestamp(1490988750, 1),
      "t" : NumberLong(1)
    }
  },
  "members" : [
    {
      "_id" : 0,
      "name" : "localhost:27017",
      "health" : 1,
      "state" : 1,
      "stateStr" : "PRIMARY",
      "uptime" : 2145,
      "optime" : {
        "ts" : Timestamp(1490988750, 1),
        "t" : NumberLong(1)
      },
      "optimeDate" : ISODate("2017-03-31T19:32:30Z"),
      "electionTime" : Timestamp(1490988529, 1),
      "electionDate" : ISODate("2017-03-31T19:28:49Z"),
      "configVersion" : 1,
      "self" : true
    },
    {
      "_id" : 1,
      "name" : "localhost:27018",
      "health" : 1,
      "state" : 2,
      "stateStr" : "SECONDARY",
      "uptime" : 236,
      "optime" : {
        "ts" : Timestamp(1490988750, 1),
        "t" : NumberLong(1)
      }
    }
  ]
}

```

Picture 6. Replica sets status.

4.2.2 Data replication

Now three nodes are properly configured and initiated. The next step is to insert data on primary node and check the operation authority on the other two secondary nodes.

Insert data on primary node.

```
db.test.insert({name:"He Xiaochuan",age:"23",address:"Turku"})
```

As it has been mentioned in previous chapter, there is no need to set data constraint, default value and identity in document type NoSQL database. The **test** is the name of collection. MongoDB will create the collection automatically by inserting query if it has not been created beforehand.

Connect to port 27018.

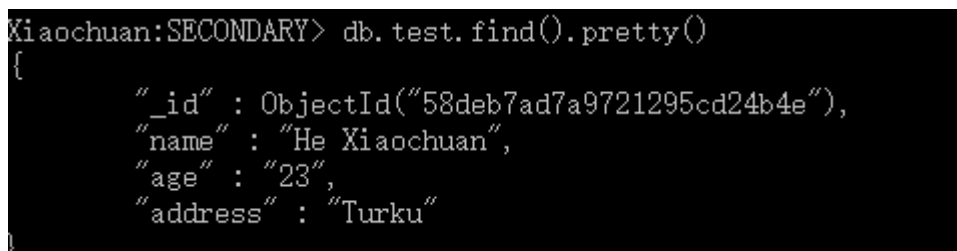
```
mongo --port 27018
```

Although port 27018 has already configured as secondary, it will report error once the node receives any read operation because by default the data on secondary node cannot be read. One more command is needed to set the secondary node as a slave to primary node.

```
rs.slaveOk();
```

After setting up, use following command to read data on secondary node. The result is shown as follows

```
db.test.find().pretty()
```



```
Xiaochuan:SECONDARY> db.test.find().pretty()
{
  "_id" : ObjectId("58deb7ad7a9721295cd24b4e"),
  "name" : "He Xiaochuan",
  "age" : "23",
  "address" : "Turku"
}
```

Picture 7. Data on secondary node.

Thus, it means the data sets on primary node has been copied to secondary nodes which is also readable. However users are not allowed to insert any data to secondary node.

As it shown in the screenshot, inserting data is not allowed because the node is not a master.

```
Xiaochuan:SECONDARY> db.test.insert({name:"Zed",age:"23",address:"Turku"})
WriteResult({ "writeError" : { "code" : 10107, "errmsg" : "not master" } })
```

Picture 8. Insert error on secondary node.

The following command is used to check the log for previous operations.

```
db.oplog.rs.find().pretty()
```

The similarity of the oplog.rs files in three replica sets shows that data sets on primary node have been copied to secondary nodes

4.2.3 Failover in replication

To see how MongoDB react to system failover, firstly the primary node should be shut down.

```
use admin
db.shutdownServer()
```

```
Xiaochuan:PRIMARY> use admin
switched to db admin
Xiaochuan:PRIMARY> db.shutdownServer()
server should be down..
2017-04-01T05:22:09.199+0800 I NETWORK [thread1] trying reconnect to 127.0.0.1:27017 (127.0.0.1) failed
2017-04-01T05:22:14.200+0800 W NETWORK [thread1] Failed to connect to 127.0.0.1:27017 after 5000ms milliseconds, giving up.
2017-04-01T05:22:14.200+0800 I NETWORK [thread1] reconnect 127.0.0.1:27017 (127.0.0.1) failed failed
```

Picture 9. Shutting down primary node.

After that, it is needed to check the status of other two secondary nodes by using **rs.status()** command.

```

},
{
  "_id" : 1,
  "name" : "localhost:27018",
  "health" : 1,
  "state" : 1,
  "stateStr" : "PRIMARY",
  "uptime" : 8846,
  "optime" : {
    "ts" : Timestamp(1490995610, 1),
    "t" : NumberLong(2)
  },
  "optimeDate" : ISODate("2017-03-31T21:26:50Z"),
  "electionTime" : Timestamp(1490995337, 1),
  "electionDate" : ISODate("2017-03-31T21:22:17Z"),
  "configVersion" : 1,
  "self" : true
},
}

```

Picture 10. Secondary node becomes primary.

It could be easily seen that port 27018 now is deemed to be primary node and data can be inserted successfully into new primary node.

```

Xiaochuan:PRIMARY> db.test.insert({name:"Teemo",age:"22",address:"Helsinki"})
WriteResult({"nInserted" : 1 })
Xiaochuan:PRIMARY>

```

Picture 11. Insert data on primary node.

When reconnecting port 27017, it functions as secondary node, only read operation is allowed.

```

C:\Users\HeXiaochuan>start mongod --replSet Xiaochuan --logpath \data\rs1\l.log --dbpath \data\rs1 --port 27017 --smallfiles --oplogSize 64

C:\Users\HeXiaochuan>mongo --port 27017
'mongo--port' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\HeXiaochuan>mongo --port 27017
MongoDB shell version v3.4.2
connecting to: mongod://127.0.0.1:27017/
MongoDB server version: 3.4.2
Server has startup warnings:
2017-04-01T05:43:50.206+0800 I CONTROL [initandlisten]
2017-04-01T05:43:50.206+0800 I CONTROL [initandlisten] ** WARNING: Access control is not enabled for the database.
2017-04-01T05:43:50.206+0800 I CONTROL [initandlisten] **          Read and write access to data and configuration is unrestricted.
2017-04-01T05:43:50.206+0800 I CONTROL [initandlisten]
Xiaochuan:SECONDARY> rs.slaveOk();
Xiaochuan:SECONDARY> db.test.find().pretty()
{
  "_id" : ObjectId("58deb7ad7a9721295cd24b4e"),
  "name" : "He Xiaochuan",
  "age" : "23",
  "address" : "Turku"
}
{
  "_id" : ObjectId("58decbf24554131ac4383821"),
  "name" : "Teemo",
  "age" : "22",
  "address" : "Helsinki"
}
Xiaochuan:SECONDARY> db.test.insert({name:"Martin",age:"23",address:"Helsinki"})
WriteResult({"writeError" : { "code" : 10107, "errmsg" : "not master" } })
Xiaochuan:SECONDARY>

```

Picture 12. Reconnect to port 27017.

From the picture above, it can be noticed that port 27017 has automatically update the replica set as soon as it has been configured as slave to primary node because it contains the data(Teemo) which was inserted on port 27018 when port 27017 was down.

4.3 Conclusion

By implementing replication in MongoDB, how replication actually works has been presented. On the one hand, multiple secondary nodes guarantees the high data availability. On the other hand, one and the only one primary node keeps the balance between system performance and the access to write operation. Both of the advatages are welcome in Web 2.0 era.

5 CONCLUSION

This thesis has analyzed and summarized the pros and cons of NoSQL, and a case study of data replication has implemented to present how MongoDB achieves high data availability.

NoSQL products provide solutions to most of technical problems that RDBMS cannot deal with in current market, namely, flexible schema lets users no longer struggle with complicated relational model and complex query, high scalability gives higher possibility to optimize transmission and synchronization when dealing with massive data. For developing enterprises, NoSQL databases can easily evolve over time. Besides that, dissimilar to RDBMS, the diversity of NoSQL products gives enterprises more options so that they can choose one that fits their own applications. However, there are few points that cannot be ignored. Non-adherence to ACID transaction and high-level query makes NoSQL a weak choice for transactional applications. Additionally, it costs more on learning immigration for NoSQL technology because NoSQL products differ from each other and they are encoded in different language. As for RDBMS products, SQL is the universal language. It is always difficult to balance the cost and benefit. Therefore, implementing NoSQL database or changing current RDBMS to NoSQL requires deep consideration of NoSQL products and the actual scenario.

REFERENCES

- [1] NoSQL Databases Pros and Cons. [online] Available at <https://www.mongodb.com/scale/nosql-databases-pros-and-cons> [Accessed 25 February 2017]
- [2] MongoDB. [online] Available at <https://www.mongodb.com/> [Accessed 25 February 2017]
- [3] Vaish, G. (2013). Getting started with NoSQL. 1st ed. Birmingham: Packet publishing
- [4] Stonebraker, M., Cetintemel, U. (2005). "One Size Fits All": An Idea Whose Time Has Come and Gone. the 21st International Conference on Data. [online] P. 10. Available at https://cs.brown.edu/~ugur/fits_all.pdf [Accessed 25 February 2017]
- [5] Lith, A., Mattson, J (2010). Investigating storage solutions for large data. Göteborg: Department of Computer Science and Engineering, Chalmers University of Technology. [online] p. 70. Available at <http://publications.lib.chalmers.se/records/fulltext/123839.pdf> [Accessed 25 February 2017]
- [6] NoSQL 2009. [online] Available at http://blog.sym-link.com/2009/05/12/nosql_2009.html . [Accessed 25 February 2017]
- [7] Brewers CAP Theorem on distributed systems (2010) [online] Available at <http://www.royans.net/wp/2010/02/14/brewers-cap-theorem-on-distributed-systems/> [Accessed 25 February 2017]
- [8] Tsuyuzaki, K., Onizuka, M. (2012). NoSQL Database Characteristics and Benchmark System. NTT Technical Review. [online] Available at https://www.ntt-review.jp/archive/ntttechnical.php?contents=ntr201212fa3.pdf&mode=show_pdf [Accessed 25 February 2017].
- [9] What is a Graph Database? [online] Available at <https://neo4j.com/developer/graph-database/> [Accessed 25 February 2017]
- [10] Alvaro, P., Conway, N., Hellerstein, J., Marczak, W. Consistency Analysis in Bloom: a CALM and Collected Approach [online] Available at http://www.neilconway.org/docs/bloom_calm_cidr11.pdf [Accessed 12 April 2017].

[11] What is Cassandra. [online] Available at <http://cassandra.apache.org/> [Accessed 23 March 2017]

[12] FAQ: Concurrency [online] Available at <https://docs.mongodb.com/manual/faq/concurrency/#what-type-of-locking-does-mongodb-use> [Accessed 23 March 2017]

[13] Cockcroft, A., Sheahan, D. (2011). Benchmarking Cassandra Scalability on AWS - Over a million writes per second. The Netflix Tech Blog. [online] Available at <http://techblog.netflix.com/2011/11/benchmarking-cassandra-scalability-on.html> [Accessed 23 March 2017]

[14] Replication [online] Available at <https://docs.mongodb.com/manual/replication/> [Accessed 30 March 2017]