# Comparative Study of NoSQL Databases

## Group 9


Muhammad Ramdhan Hidayat
P-COM0078/20
*School of CpmputerSciences*
*Universiti Sains Malaysia*
Pulau Pinang, Malaysia
ramdhan_hidayat@student.usm.my

Kie Omatsu
P-COM0082/19
*School of Computer Sciences*
*Universiti Sains Malaysia*
Pulau Pinang, Malaysia
kie.omatsu@student.usm.my

Atina Najahan Mohd Rashidi
P-COM0137/20
*School of Computer Sciences*
*Universiti Sains Malaysia*
Pulau Pinang, Malaysia
atinanajahan@student.usm.my

Maisarah Binti Mohamed Pauzi
P-COM0126/20
*School of Computer Sciences*
*Universiti Sains Malaysia*
Pulau Pinang, Malaysia
maisarah.mohd.pauzi@student.usm.my

Nur Alia Binti Anuar
P-COM0129/20
*School of Computer Sciences*
*Universiti Sains Malaysia*
Pulau Pinang, Malaysia
aliaanuar@student.usm.my



***Abstract*—As the age of technology progresses, so does the database systems of the world. In order to keep up with the rapid development of industry, utilizing database and storage management systems, advanced NoSQL databases such as Amazon DynamoDB, MongoDB, Apache Cassandra and Neo4j comes to light. In this paper, we compare the differences between these aforementioned databases and study their characteristics.**



***Keywords*— Database, NoSQL, DynamoDB, MongoDB, Cassandra, Neo4j**


## I. Introduction

Relational databases, represented by Oracle database and MySQL, have been playing a pivotal role in the last 40 years of database development. However, recent advancements in cloud computing and web applications leads to the explosion of data generation at the rate that challenge the innate limitations on the single nodes systems adopted by relational databases. Horizontal scalability is difficult to achieve while query performance struggle in handling massive amounts of data. These scalability and performance issues leads to a huge cost inflation in managing data. The NoSQL database system came to existence to address these drawbacks. In general, the way of which NoSQL stores data can be categorized in four categories: Key-Value Store, Document Store, Wide Column Store, and Graph Database.

- Key-Value Store

Key-Value store is a data storage system which is designed for storing, retrieving, and managing associative arrays and a data structure commonly known as hash table or dictionary. Associative array is basically an abstract datatype composed of a collection of pairs, such that each possible key appears at most once in the collection. Hash table is a data structure that implements an associative array abstract data type. In other words, a key is required every time data is retrieved and updated. Considering this, key management is a vital aspect for smooth operations in a key-value database. One of enterprise-ready database solutions that utilize the key-value paradigm is DynamoDB, which is developed by Amazon.

- Document Store

Document store is a document-based database where data are stored in form of documents. Documents are stored in form of standard formats such as JSON or XML. A document is almost similar to a record in a

relational database with unique ID and data is stored in key-values form. However, they provide more flexibility due to its schema less feature. A document can consist of different fields unlike a record in relational database. Thus, it is suitable to be applied for large data size with inconsistent fields such as text documents. One of the commonly used source-available document database is MongoDB.

- Wide Column Store

A column store database is a type of database that stores data using column-oriented database. In RDBMS, data is normally executed in row-major order whilst in column stored database, data is processed in column-major order [1]. Column store database use keyspace concept [2]. The keyspace contains all the column families. A column family consists of multiple rows. Each row contains number of columns. The number of columns can be different to other rows. Each column contains name or value along with timestamp. The columns are easy to scale since it can store big size of data. This allows users to spread data through many computing nodes and data stores. Examples of column family database are Apache Cassandra, HBase, MariaDB, Cloudata and Google BigTable. In this study, we will be discussing Cassandra as column stored database.

- Graph Database

Graph databases are databases that use graph structure to store data, which is motivated by real-life applications [3]. It is utilized especially when interconnectivity is a key feature in data. Graph based stores consist of nodes, properties, and relationships [4]. Nodes contain records of data in key-value pairs. They are tagged with labels, to represent different roles in the domain. Node labels attach metadata, such as index or constraint information, to specific nodes. Properties are the attributes that nodes hold including name or value pairs. In addition, relationships denote connection between nodes. It always has a direction in either one way or two ways, a type, a start node, and an end one. It can contain properties, especially quantitative ones.

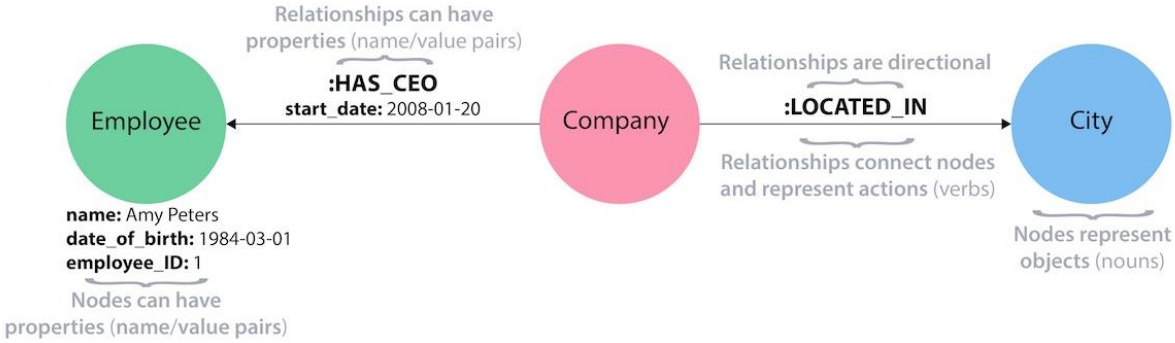


Fig. 1. Example of graph database structure [5]

The following sections will explore unique features of above NoSQL database enterprise solutions in more detail.

## II. Part A

## 1. Highlights of four types of NoSQL databases

### a. DynamoDB

DynamoDB was officially released by Amazon in 2012 with its first initiation reach far back to 2007 [1] [6]. It is a commercially licensed product, which means that its free tier limits the amount of database operations that can be utilized. Like many other NoSQL databases, DynamoDB is a schema-less design [7]. This means that developers do not need to define a strict schema for tables but rather just required to define the primary key and index.

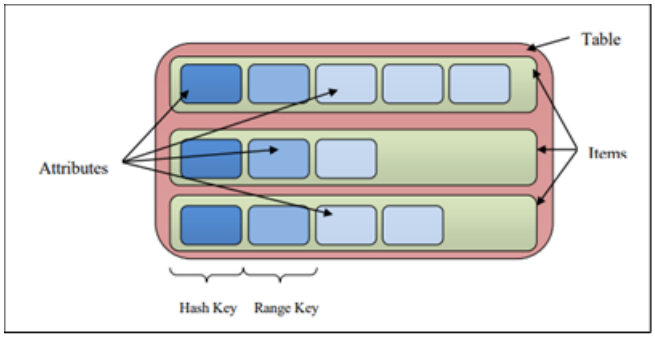

Fig. 2. Amazon DynamoDB data model

As mentioned in previous section, the key-value store paradigm utilizes hash table to organize and manage keys. The specific implementation of this data model in DynamoDB is shown in Figure 1. It illustrates that in DynamoDB, data are organized in Tables containing a collection of rows, which are called as items. Each item supports arbitrary numbers of attributes with maximum size is set to be 400 KB. For every attribute, a pair of Hash Key and Range Key represent a Name-Value pair. Furthermore, each attribute can accommodate several data types such as number, string, Boolean, set, binary, lists and map. An optimal use case for this schema design is a simple table containing a wide range of different keys and a sort key attached to them. A specific example of this would-be storing data of users that does not related to other tables. Figure 2 illustrates how this can be done in DynamoDB.

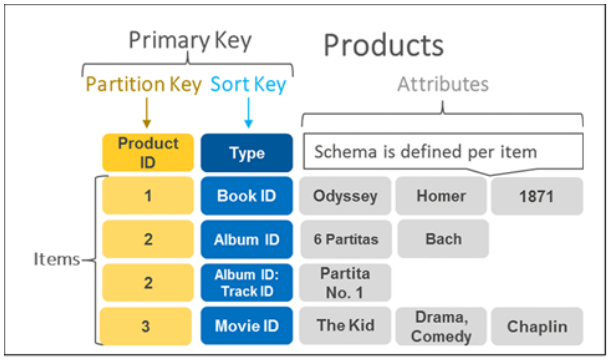

Fig. 3. DynamoDB data model application

On the left side of figure is Primary Key, which store the ID of products in the listing. There are two types of primary keys supported by DynamoDB, which are called Partition Key and Sort Key. The combination of these two keys is what DynamoDB named as Composite Primary Key. The right-side of the diagram depicts the list of attributes related to each product [8]. By default, there are no pre-defined limits regarding the maximum amount of data each hash table can store. Users can store and retrieve data in however amount they wish as DynamoDB will spread the large amount of data across their servers as the amount of data to be stored grows. This makes DynamoDB to be highly scalable. Scalability refers to a system's ability to deal with increasing workloads.

In terms of ease of use, DynamoDB can be considered as very easy to use [8]. This is thanks to the strong integration with the Amazon Web Service (AWS) ecosystem. Developers do not have to deal with setting up and maintaining any software or hardware resources. They only required to specify the rate of read and write throughput they desired for each hash table. The management of resources to fulfill the request will be automatically handled by DynamoDB.

At the moment, DynamoDB provides official drivers for up to eight programming languages including the likes of Java, Python, JavaScript, and PHP [9]. The complete list of supported languages is tabulated in Table 1.

### b. MongoDB

MongoDB was founded in 2009 is a quite a new concept of database with no tables, schemas, SQL or rows [10]. Other features such as foreign keys, joins and ACID compliance which normally present in traditional relational databases are also absent from the system. It is a non-relational JSON-based database based with dynamic schemas written in C++ and the query language is JavaScript [11]. MongoDB is great at storing various and more complex types of data with different set of attributes due to its schema-less feature [12]. MongoDB also runs on various operating systems such as Linux, Windows, Mac, and Solaris.

A downside to using MongoDB is the lack of support for transactions which does not offer a guarantee on data consistencies. This is the compensation design for having a simple, fast, and scalable database [10]. Thus, there are certain situation where it is not fit for MongoDB to be used such as being implemented on an accounting application. Besides that, MongoDB takes shortcuts to improve its speed causing it to be harder to recover in case of crashes.

MongoDB stores data in collections which contains documents inside. Collections themselves are stored within related database. The documents are stored in BSON format that supports string, integer, float, and Boolean data types. BSON was developed to handle binary JSON form to improve the processing speed. It uses “collection” and “documents” which are similar to table and rows concepts in RDBMS to store data and schema information [13]. Documents store data in the form of keys (fields) and values. Field functions as the unique data identifier and values are data related to the particular identifier. They also being designed to store complex information such as lists, dictionaries, and lists of dictionaries [10]. Key is used to provide a reference to the value in a document. MongoDB automatically assigned a unique primary key (_id) to identify a newly created document unless user opts to manually generate the keys themselves. Below is an example of a document.

```
{
    "_id" : ObjectId("56d61033a378eccde8a83551")
    "first_name" : "Taylor",
    "last_name" : "Swift",
    "phone_number" : [
        "+601234567",
        "+601357986"
        ]
}
```

Fig. 4. Example of document collection

MongoDB supports sharding which allows data split across multiple nodes with each node managing different subset of the dataset. MongoDB does not provide master-master replication, but it provides master-slave replication. Only the master node is capable of writing task and the writes data are passed to the slave nodes to be used for reads and backups.

Craigslist has migrated over two billion documents to MongoDB in 2011 from their traditional relational database due to MongoDB scalability and flexible schema [14]. Beforehand, they were facing problems of migrating archive-ready data from live database which has experienced schema changes to the archive database. MongoDB with its flexible schema features has managed to accommodate to their problems without costly schema migrations. MongoDB has also reduced their operational expense with auto-sharding and high availability features.

### c. Cassandra

Cassandra was first developed by Avinash Laskhman and Prashant Malik from Facebook team [15]. It was first built to power Facebook’s inbox search in July 2008. Cassandra is available under the Apache License 2.0.

Cassandra comes with notable features such as peer-to-peer distributed system, no single point of failure and eventual consistency model. In Cassandra, every node has the same role and responsibility. Every node in Cassandra can serve client request thus eliminating the master-slave model [16]. This is known as peer-to-peer architecture. Failed node in Cassandra will be detected by gossip protocols [17]. Nodes will use gossip protocols to communicate between each other.

Below are the key components of Cassandra [18]:

Components of Cassandra

**Node** – The place where data is stored.

**Data center** – A collection of related nodes.

**Cluster** – One or more data center is placed here.

**Commit log –** Each write operation is written to commit log. It is also a mechanism for crash-recovery in Cassandra.

**Memtable** − It is a write back cache of data partition. Memtable stores write operation after commit log.

**SSTable** – It is an actual file on disk. A new SSTable will be created when data is flushed from the memtable.

**Bloom filter** – A space efficient probabilistic data structure and is used to test if the data is exist or not in the given file. Bloom filters are accessed after every query.

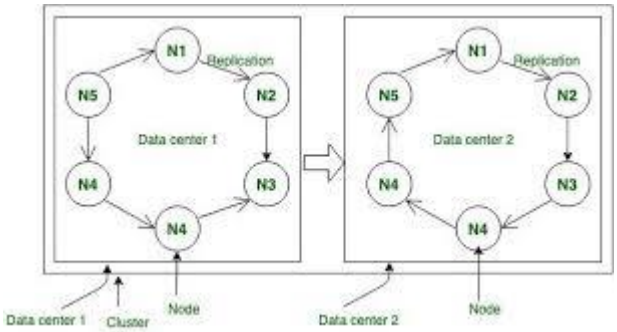


Fig. 5. Nodes, data center and cluster in Cassandra

Figure 4 shows how nodes, data centers and clusters are arranged in Cassandra. For the read-write operations, client will approach the nodes. Each node writing operation is captured by the commit logs written to the nodes. The data is later registered and stored in the memtable. Data will be written into the SStable data file once the memtable is completed. All of the the writes are partitioned and replicated automatically in the cluster. Cassandra consolidates the SSTables timely, discarding unnecessary data. Cassandra obtains values from the memtable during reading operations and tests the bloom filter to find the appropriate SSTable that holds the necessary data.

In Cassandara, users can access the database through its nodes using Cassandra Query Language (CQL), an alternative to Structured Query Language (SQL). Cassandra Query Language Programmers use cqlsh: a prompt to work with CQL or separate application language drivers. Cassandra was written in Java and it can be used in wide variety of language including C#, Scala, Python and Ruby [19]. Cassandra comes with many applications. One of the noteworthy applications of Cassandra is storage. Cassandra can store any type of data and this data is stored at nodes. Another major application of Cassandra is monitoring. Many applications are based on user activities. These activities can either be media, music or art. This feature allows developer to monitor user activities through Cassandra.

Cassandra was first built by Facebook to power Facebook's inbox search with 200 nodes deployed. Later, it was abandoned in year 2010 when they developed Facebook Messenger on HBase. However, Cassandra has been adopted by many companies as their database. It is also recorded that Netflix has also used Cassandra as their back-end database for the streaming services. Cisco's Webex also adopted Cassandra to store user feed and activity in near real-time [17].

### d. Neo4j

Neo4j is an open-source, disk based, and NoSQL native graph database. The development started in 2003, written in Java and Scala, and in 2007 its code became open source. The schema-optional feature allows you to add and remove label, nodes, properties, and relationships though you need to create them in advance. The structure is flexible unlike traditional databases that treat data in rows and columns, which often have constraints in terms of structures [4] [5]. It is an ACID-compliant transactional database.

Neo4j has three editions, Community, Enterprise, and Government. Community edition is a version where any users can try its service, and Enterprise Edition is more complete version of Community Edition. The difference between these two are online backup, performance of memory cache, monitoring system scalability and so on. In the Enterprise Edition, the scalability is horizontal while Community Edition has vertical scalability. Government Edition offers more upgraded services [20].

In managing data on a database, language plays an important role in terms of usability. Neo4j has its graph query language designed for connectedness called Cypher. It is based on JSON Syntax, and introduces ASCII art where codes represent drawing of a pattern of the graph in q2uery. Neo4j supports other languages such as JavaScript, Python, R, and C/C++ [5].

The examples of recommended usages of Neo4j are fraud-detection and e-commerce business [21]. In a scenario where fraudsters act as if legitimate customers, it is really difficult to detect the crime. Graph database offers solutions to uncover fraud rings to solve these issues. In traditional relational databases, to model graph from a set of tables and columns is much time consuming and computationally expensive. However, Neo4j enables entity link analysis, where layered models are created to detect criminal behaviors. In addition, Neo4j has changed e-commerce delivery service routing and recommendation engine, as seen in case study of Walmart and eBay [22]. Neo4j requires only a little codes, which enables them to be in time for the same-day delivery, and also it can deal with online users' behavior. Thus, thanks to the timeliness and ability of querying data, Neo4j plays a significant role in marketing too.

There are several highlights of using Neo4j. One of them is its intuitiveness of use thanks to its graph database design and its query language, Cypher. It excels in speeds for both execution of its transactions which the ability to obtain result in real time. Due to its ease of use, users of Node4j can easily integrate, debug and write their code, adopting to their desired needs. Neo4j also provides an API for developers and data scientists to create quick visualizations from the database that represents the real world accurately [4].

One of the drawbacks of Neo4j is that it does not perform fast on query transaction that involves a lot of data crunching. According to a study, it is proven that it takes longer if queries include calculations in the graph database, and an initial execution also requires much time because the graph database leverages storage engine [20]. In addition, when it comes to huge data, it takes almost a day to load files more than 700MB. Moreover, to find a specific data node with certain attribute is not easy because it is necessary to traverse through several tables [20].

## 2. Feature comparison

Based on the highlights covered in the previous section, the summary of the four databases is tabulated below. As we can see, Cassandra supports various languages the most among the four.

TABLE 1. OFFICIALLY SUPPORTED PROGRAMMING LANGUAGES

| | **DynamoDB** | **MongoDB** [23] | **Cassandra** [24] | **Neo4j** [25] |
|---|---|---|---|---|
| **Officially Supported Languages** | .Net<br>Java<br>JavaScript<br>Node.js<br>PHP<br>Python<br>Ruby<br>C++ | C#<br>C++<br>C<br>Go<br>Java<br>Node.js<br>PHP<br>Python<br>Ruby<br>Scala<br>Rust<br>Swift | C<br>C#<br>C++<br>Erlang<br>Go<br>Haskell<br>Java<br>JavaScript<br>Perl<br>PHP<br>Python<br>Ruby<br>Rust<br>Scala<br>Swift<br>Node.js | .Net<br>Go<br>Java<br>JavaScript<br>Python<br>Spring |

The table below shows the features of the four databases.

TABLE 2. FEATURE COMPARISONS

| | **DynamoDB** | **MongoDB** | **Cassandra** | **Neo4j** |
|---|---|---|---|---|
| **Storage Method** | Unique Key-Value | Documents | Columns | Graph |
| **License Type** | Commercial | Open Source and Commercial | Open Source (Apache V2) | Open Source (GPL V3) , Commercial & Government |
| **Format** | Schema-less | Schema-less | Schema-less | Dynamic Schema |
| **Applications** | Indexing | Programming objects storage | Sparse data | Social connections |
| **Complexity** | Low | Low | Low | High |
| **Flexibility of Schema** | High | Varies | High | High |
| **Scalability of data** | High | High | Average | High |

## 3. Discussion and recommendation

Various implementations of NoSQL file storage techniques are illustrated in Table 2. If the task on hand involves social connection analysis, then the graph-based solution offered by Neo4j would be the most suitable. In addition, tasks that heavily involved the storing of programming language objects would be well-accommodated by MongoDB since it supports JSON file format. Meanwhile, applications that heavily relies on indexing such as recording and managing transactions of multiple products can take advantage of hash table structure implemented by DynamoDB.

Another important parameter to consider when choosing a data management solution is scalability [26]. Among the NoSQL databases under discussion, Cassandra is the least desired when the project needs assurance on processing performance as the amount of data involved grows significantly. In addition, organizations needing the least complex database solution and not heavily involve graph-related task might want to avoid integrating Neo4j in their workflow as it scores the highest in terms of complexity. This is

because a highly complex database system could impact the magnitude of its administrative and development efforts [27].

Furthermore, Table 2 can also be used as reference when deciding which schema requirement is most suitable for related use cases. The dynamic nature of graph database like Neo4j allows adding and removing nodes and edges at any point during data's lifecycle. Other NoSQL databases in the list actually has a dynamic nature, but not to the extend of what graph-based databases are capable to achieve [28]. However, this unique capability comes at the price of high complexity as previously mentioned.

Last, but not least, programming languages officially supported by each NoSQL solution could be another important parameter to consider. Having a database system that has a plenty of option for programming languages would aid organizations to have a wider range of nets when scouring for talented potential employees. In addition, having a less variety of programming languages adopted across different functional operations could allow easier integration and communication between them. Table 1 shows that Neo4j has the least variety when it comes to officially supported languages. Note that community-supported programming languages also exists and allows those languages to be used for each NoSQL system. However, officially supported languages are warranted to constantly receive updates in the future. This is an important aspect to consider when the objective is to have a future-proof database system.

Thus, in selecting database, understanding both the feature of databases and the nature of the problem are important to draft the structure of the database on how data is stored, retrieved and crunched. Data needs to be modeled to determine accordingly.

## 4. Concluding remarks

There is not a single database management system that would be perfect for every use case situation. Each database management solution comes with its own trade-offs that organizations, institutions, and businesses need to weigh-in before making decision to integrate them to their day-to-day operational workflow. Picking a well-suited data solution could aid them in optimizing and improving their performance no matter the sector they are operating, since nowadays data is the lifeblood of any form of organizations and institutions. A brief feature comparison of four NoSQL database solution were covered in this section with that in mind. The following section will give a more in-depth performance analysis and comparison between MongoDB and Cassandra.

# III. PART B

In this part, we will make a comparison between Cassandra and MongoDB by experimentally running similar queries in different syntax.

## 1. Brief description of the given dataset

To begin with, we would like to describe the data used for the experiment. It is the records of crimes occurred in the City of Chicago, United States. Original data is derived from the Chicago Police Departments' CLEAR (Citizen Law Enforcement Analysis and Reporting) system from 2001 to present, excluding the most resent one week. The dataset used in this study is extracted from the database randomly [29]. It is 1.41 GB in csv format. The figure below is the sample of document collection shown by running **db.crimes.find().pretty()** in MongoDB. The data consists of 23 keys including auto-generated ID and 12,534,454 collection of documents. Each document is composed of keys such as location, date, and crime type to describe the crimes.

```
{
        "_id" : ObjectId("5fe1f4e4d4aaa3bb54ac6717"),
        "ID" : 3753798,
        "Case Number" : "HL122987",
        "Date" : "01/13/2005 03:20:00 PM",
        "Block" : "119XX S INDIANA AVE",
        "IUCR" : 460,
        "Primary Type" : "BATTERY",
        "Description" : "SIMPLE",
        "Location Description" : "RESIDENCE PORCH/HALLWAY",
        "Arrest" : "false",
        "Domestic" : "false",
        "Beat" : 532,
        "District" : 5,
        "Ward" : 9,
        "Community Area" : 53,
        "FBI Code" : "08B",
        "X Coordinate" : 1179720,
        "Y Coordinate" : 1825889,
        "Year" : 2005,
        "Updated On" : "04/15/2016 08:55:02 AM",
        "Latitude" : 41.677511887,
        "Longitude" : -87.617802135,
        "Location" : "(41.677511887, -87.617802135)"
}
```


Fig. 6. Preview snapshot of one document collection

## 2. Selection of implementation platform for MongoDB and Cassandra

In this study, the platform used is Google Cloud Platform. The cloud services that have been utilized in this study are:

- Google Cloud Storage
- Google Cloud Compute Engine

Initially, the dataset file (crimes.csv) have been import to the Google Cloud Storage. Then, by using Google Cloud Compute, a Virtual Machine (VM) is created. The Operating System (OS) of the VM is Ubuntu 18.04. Lastly, the dataset file is being copied from Google Cloud Storage into the file system of the VM. Google Cloud Storage is used as a backup storage as the VM might be down due to misconfiguration/errors and the data might be wiped out if the situation occurs.

## 3. Installation process, database construction and data entry

After updating OS to the latest version, MongoDB and Cassandra have been installed in the VM. For Cassandra, the installation process is as below:

a) Install OpenJDK package and check the Java version.
b) add the Apache Cassandra repository
c) Install Cassandra
d) Check successful installation by entering the Cassandra Query Language Shell (**cqlsh**)

For MongoDB, the installation steps are as below:

a) Install meta-package of MongoDB (**mongodb-org**)
b) Start MongoDB service
c) Check the service status
d) Enable the MongoDB service to start up at boot
e) Check successful installation by entering the MongoDB shell

In Cassandra, the database configuration is different from Mongo DB. Database in Cassandra are store in a table within a keyspace. Hence, before loading the dataset into the table, a keyspace must be created. Then only a table is created. The data is copied from the dataset file into the table. The queries are as below:

1) Create Table

```
CREATE TABLE crimes (ID int, CaseNumber varchar, Date varchar, Block varchar, IUCR varchar, PrimaryType varchar, Description varchar, LocationDescription varchar, Arrest boolean, Domestic boolean, Beat varchar, District varchar, Ward varchar, CommunityArea varchar, FBICode varchar, XCoordinate int, YCoordinate int, Year int, UpdatedOn varchar, Latitude varchar, Longitude varchar, Location varchar, PRIMARY KEY(ID));
```

2) Copy data from file

```
COPY project.crimes (ID, CaseNumber, Date, Block, IUCR, PrimaryType, Description, LocationDescription, Arrest, Domestic, Beat, District, Ward, CommunityArea, FBICode, XCoordinate, YCoordinate, Year, UpdatedOn, Latitude, Longitude, Location) FROM '/home/Dataset2.csv' WITH DELIMITER=',' AND HEADER=TRUE ;
```

For MongoDB, the database is stored in a collection within a database. A new database is created. Then, the collection is created with data copied from the dataset file. Both collection creation and data import happen in a single command. The command is as below:

```
mongoimport --type csv -d project -c crimes --type csv --file /home/Dataset2.csv --headerline
```

## 4. Meaningful queries in MongoDB and Cassandra

For the comparison, seven queries are executed. To set the number of data to show **limit()** is used, which enables to save space and prevent the CLI (Command Line Interface) from flooding. Firstly, we obtain a thousand data without any conditions. Next, we try to gain specific data by adding conditions, such as arrested cases, location, crime type, and combination of them. Finally, crimes happened in specific years are filtered out with comparison and logical operators. The summary of the queries as shown in the table below.

TABLE 3. FIVE MEANINGFUL QUERIES IN Cassandra AND MongoDB

| Query | Cassandra | MongoDB |
|---|---|---|
| **Snapshot of first 1000 data** | `SELECT * FROM crimes LIMIT 1000;` | `db.crimes.find().limit(1000)` |
| **List of first 1000 successful arrested crime** | `SELECT * FROM crimes WHERE arrest = true LIMIT 1000;` | `db.crimes.find({'Arrest':'true'}).limit(1000)` |
| **List of 1000 Theft cases** | `SELECT* FROM crimes WHERE primarytype = 'THEFT' LIMIT 1000;` | `db.crimes.find({"Primary Type":"THEFT"}).limit(1000)` |
| **List of 1000 crimes happen on the Street** | `SELECT * FROM crimes WHERE locationdescription = 'STREET' LIMIT 1000;` | `db.crimes.find({"Location Description":"STREET"}).limit(1000)` |
| **List of 1000 theft cases happen on the street in 2016** | `SELECT * FROM crimes WHERE primarytype = 'THEFT' AND locationdescription = 'STREET' AND year = 2016 LIMIT 1000;` | `db.crimes.find({"Year":2016,"Primary Type":"THEFT","Location Description":"STREET"}).limit(1000)` |

| | | |
|---|---|---|
| **List of 1000 crimes happened after 2010** | `SELECT * FROM crimes WHERE year > 2010 LIMIT 1000;` | `db.crimes.find({Year:{$gt:2010}}).limit(1000)` |
| **List of 1000 crimes happened in either 2012 or 2013** | `* OR operator unsupported in Cassandra` | `db.crimes.find({$or:[{Year:2012},{Year:2013}]})` |

## 5. Comparison, discussion and recommendation

We would like to compare Cassandra and MongoDB in terms of ease of use and processing speed.

a. Ease of use

TABLE 4. COMPARISON OF USABILITY

| **Cassandra** | **Mongo DB** |
|---|---|
| Readability:<br>• Cassandra display data in a table structure with first row showing the column names and the rest are the data rows.<br>• By default, the data display by default pagination (10 rows per page). If there are 1000 rows, then there are 100 pages of data displayed. | • MongoDB uses MongoDB Query Language (MQL) designed for easy use.<br>• Data shown in JSON (JavaScript Object Notation), and it is quite readable especially by executing `pretty()`. |

b. Data processing speed

For comparison of processing speed, we executed the queries used above. Overall, though it takes Cassandra longer to load data than MongoDB, Cassandra processed faster than MongoDB as shown in the table below.

TABLE 5. COMPARISON OF PROCESSING SPEED

| **Query** | **Time execution in Cassandra** | **Time execution in MongoDB** |
|---|---|---|
| **Loading Data from CSV file into Database** | 7 minutes and 49.195 seconds | 47.666 seconds |
| **Snapshot of 1000 data list** | 0.003 seconds | 0.037 seconds |
| **List of 1000 successfully arrested crime** | 0.009 seconds | 0.038 seconds |
| **List of 1000 Theft cases** | 0.011 seconds | 0.057 seconds |
| **List of 1000 crimes happen on the Street** | 0.008 seconds | 0.071 seconds |
| **List of 1000 theft cases happen on the street in 2016** | 0.114 seconds | 2.292 seconds |
| **List of 1000 crimes happened after 2010** | 0.009 seconds | 0.934 seconds |
| **List of 1000 crimes happened in either 2012 or 2013** | - | 10.885 seconds |

c. Discussion

- Cassandra

Though it takes longer to load dataset initially, execution is much faster than MongoDB. This is because Cassandra concurrently returns the query results by page. Each page will display maximum 100 results.

Although the query performance in Cassandra is about 10 times faster than MongoDB, there are some operators that are not supported in Cassandra such as 'OR' operator and 'Not Equal' operator. The only supported operators are 'Equal (=)', 'AND', 'Less than (<)', 'Greater than (>)','Less than and equal (<=)', 'Greater than and equal (>=)', 'CONTAINS' and 'CONTAINS KEY' [30].

Based on the result obtained from the experiment, the difference between the execution time between each query ranged from 0.003 seconds to 0.114 seconds. Since the queries only did read operation, hence there are no obvious difference except the conditon in each query. The more condition added to the query, the longer the time taken to execute the query. As a comparison, 'Snapshot of 1000 data list' only takes 0.003 seconds while the query of 'List of 1000 theft cases happen on the street in 2016' takes 0.114 seconds to execute. The difference of the execution time are 0.111 seconds.

- MongoDB

It takes much less time to load data initially though processing speed is slower than Cassandra. It is easy to read by using **pretty()**. There are more various ways to filter data because unlike Cassandra, many logical operators are supported.

Based on the result obtained from the experiment, the difference between the execution time between each query ranged from 0.037 seconds to 10.885 seconds. The fastest query executed in MongoDB is the 'Snapshot of 1000 data list' and the slowest query executed is the 'List of 1000 crimes happened in either 2012 or 2013'. The difference of the execution time are 10.848 seconds.

Thus, it turned out that MongoDB performs faster than Cassandra except loading data, though there are still possibilities that queries that are even more complex might result in different result.

d. Recommendation

As discussed in the previous chapter, it is no doubt that Cassandra performs better than MongoDB. However, the performance alone should not be the only requirement for a good database system. Although Cassandra is faster to execute the query, however the query language itself have some limitations. In this study, it is found that Cassandra does not support 'OR' operator.

Meanwhile, even if MongoDB is a little bit slower than Cassandra, it is more flexible and convenient to use. Not only it returns the query result in JSON format, which is easier to encode in the front end, MongoDB provides more filtering options, such that it supports common operators like 'OR' and 'Not Equal'.

One of the challenges in handling big data is the variety of the data types -- structured and unstructured data. In most cases, raw data like crimes records might be unstructured, hence, MongoDB is more suitable to be used as it is based on Rich Data Model. Since Cassandra is based on Big Table model, it is more suitable for large, structured data. In this case, Cassandra might be convenient for dataset that stored in a large web system such as social media and e-commerce site as they are automatically structured, unlike raw data from various resources.

## 6. Concluding remarks

In this study, we implemented several queries for comparison between Cassandra and MongoDB. Throughout the experiment, we found out that MongoDB is splendid because of convenience and flexibility though it inferior to Cassandra regarding processing speed. Consequently, we conclude that MongoDB is more convenient and versatile of the two.

# IV. Lessons learned

This project is an eye-opener for us to understand NoSQL databases better since a lot of reading and hands-on learning occur during the process of completing the project. In this project, we work as a group where teaching and learning takes place as everyone is working on different part. Working in team make a lot thing easier since the workload is distributed making it less to work on. However, it can also be challenging as it demands set of skills from team member. The amount of communication skills involve is also important in completing the project. We also learned to stay organize, finish the task before the deadline and rise up any issue if there is any.

In brief, this project has given us a new insight regarding NoSQL databases and developed sense of obligation among team members toward the completion of the task.

## V. Clear division of group members' roles

- Report

  Abstraction and keywords: Kie Omatsu

  1. Introduction: Muhammad Ramdhan Hidayat
  2. Part A
     1. Highlights of four types of NoSQL databases
        a. DynamoDB: Muhammad Ramdhan Hidayat
        b. MongoDB: Nur Alia Binti Anuar
        c. Cassandra: Atina Najahan Mohd Rashidi
        d. Neo4j: Kie Omatsu
     2. Comparison: Muhammad Ramdhan Hidayat
     3. Discussion and recommendation: Muhammad Ramdhan Hidayat
     4. Concluding remarks: Muhammad Ramdhan Hidayat
  3. Part B
     1. Brief description of the given dataset: Kie Omatsu
     2. Selection of implementation platform: Maisarah Binti Mohamed Pauzi
     3. Installation process, database construction and data entry: Maisarah Binti Mohamed Pauzi
     4. Meaningful queries in MongoDB and Cassandra: Maisarah Binti Mohamed Pauzi
        - Cassandra queries: Maisarah Binti Mohamed Pauzi
        - MongoDB queries: Kie Omatsu
     5. Comparison, discussion and recommendation: Maisarah Binti Mohamed Pauzi and Kie Omatsu
     6. Concluding remarks: Kie Omatsu
  4. Lesson learned: Atina Najahan Mohd Rashidi
  6. Conclusion: Atina Najahan Mohd Rashidi, Kie Omatsu

  Formatting: Kie Omatsu

- Presentation

  Part A:
  - DynamoDB: Muhammad Ramdhan Hidayat
  - MongoDB: Nur Alia Binti Anuar
  - Cassandra: Atina Najahan Mohd Rashidi
  - Neo4j: Kie Omatsu

  Part B:

  Maisarah Binti Mohamed Pauzi

  Presenters: Atina Najahan Mohd Rashidi, Nur Alia Binti Anuar, and Maisarah Binti Mohamed Pauzi

## VI. Conclusion

NoSQL databases are non-relational database in which data are not stored in traditional ways. The databases were developed in order to process unstructured data. In this project, several databases were studied to observe the highlights and components for each service, the similarities and differences between databases. We have compared four NoSQL databases, Amazon DynamoDB, Apache Cassandra, MongoDB and Neo4j. No database would be absolute for every usage situation because each of them has own benefits and detriments. Each solution is optimized for different workloads and use cases.

## VII. References


[1] R. Zafar, E. Yafi, M. F. Zuhairi, and H. Dao, "Big Data: The NoSQL and RDBMS review," *International Conference on Information and Communication Technology (ICICTM),* pp. 120-126, 2016.

[2] M. V., "Comparative Study of NoSQL Document, Column Store Databases and Evaluation of Cassandra," *International Journal of Database Management Systems,* vol. 6, no. 4, pp. 11-26, 2014.

[3] R. Angels and C. Gutierrez, "Survey of graph database models," *ACM Computing Surveys,* vol. 40, no. 1, pp. 1:1-1:39, 2008.

[4] A. Brigit Mathew, and S. D. Madhu Kumar, "An Efficient Index based Query handling model for Neo4j," *International Journal of Advances in Computer Science and Technology (IJACST),* vol. 3, no. 2, pp. 12-18, 2014.

[5] "What is a Graph Database?," Neo4j, [Online]. Available: https://neo4j.com/developer/graph-database/. [Accessed 9 1 2021].

[6] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: amazon's highly available key-value store," *ACM SIGOPS,* vol. 41, no. 6, p. 205–220, 10 2007.

[7] G. Bathla, R. Rani, and H. Aggarwal, "Comparative study of NoSQL databases for big data storage," *International Journal of Engineering & Technology,* vol. 7, no. 2.6, pp. 83-87, 2018.

[8] G. Antoniou, "Evaluation of Major NoSQL Databases,," p. 88.

[9] "Running the Code Examples in This Developer Guide," Amazon DynamoDB, [Online]. Available: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/CodeSamples.html. [Accessed 14 1 2021].

[10] E. Plugge, P. Membrey and T. Hawkins, The definitive guide to MongoDB, New York: Apress, 2010.

[11] C. Gyorödi, R. Gyorödi, G. Pecherle and A. Olah, "A Comparative Study: MongoDB vs. MySQL," *13th International Conference on Engineering of Modern Electric Systems (EMES),* 2015.

[12] R. Arora and R. Aggarwal, "Modeling and Querying Data in MongoDB," *International Journal of Scientific & Engineering Research,* vol. 4, no. 7, 2013.

[13] Z. Parker, S. Poe and S. Vrbsky, "Comparing NoSQL MongoDB to an SQL DB," in *Proceedings of the 51st ACM Southeast Conference on - ACMSE '13,* 2013.

[14] "MongoDB Case Study," MongoDB, 2012. [Online]. Available: https://www.mongodb.com/post/15781260117/mongodb-case-study-craigslist. [Accessed 8 1 2021].

[15] "What is Cassandra?," DataStax, 2014. [Online]. Available: http://planetcassandra.org/what-is-apache-cassandra. [Accessed 12 1 2021].

[16] Y. Kishore, N. V. Datta, K. Subramaniam, and D. Sitaram, "QoS Aware Resource Management for Apache Cassandra," *2016 IEEE 23rd International Conference on High Performance Computing Workshops (HiPCW),* 2016.

[17] A. Lakshman and P. Malik, "ACM SIGOPS Operating Systems Review," vol. 42, no. 3, 2008.

[18] P. Varalakshmi, S. Hima, and S. M. Varghese, "Cassandra a distributed NoSQL database for Hotel Management System," *International Journal on Cybernetics & Informatics*, vol. 5, no. 2, pp. 109-116, 2016.

[19] E. Hewitt, Cassandra: the Definitive Guide, O'Reilly Media, Inc., 2010 .

[20] J. Guia, V. Gonçalves Soares and J. Bernardino, "Graph Databases: Neo4j Analysis," *Proceedings of the 19th International Conference on Enterprise Information Systems (ICEIS 2017)*, vol. 1, pp. 351-356.

[21] G. Sadowksi and P. Rathle, "Fraud Detection: Discovering Connections with Graph Databases," 2015.

[22] "eBay and Walmart Adopt Neo4j: The Graph is Transforming Retail," Neo4j, [Online]. Available: https://neo4j.com/press-releases/ebay-walmart-adopt-neo4j-graph-transforming-retail/. [Accessed 10 1 2021].

[23] "Start Developing with MongoDB," MongoDB, [Online]. Available: https://docs.mongodb.com/drivers/. [Accessed 14 1 2021].

[24] "Client drivers," Apache Cassandra, [Online]. Available: https://cassandra.apache.org/doc/latest/getting_started/drivers.html. [Accessed 15 1 2021].

[25] "Drivers & Language Guides," Neo4j, [Online]. Available: https://neo4j.com/developer/language-guides/#:~:text=Neo4j%20officially%20supports%20the%20drivers,for%20all%20protocols%20and%20APIs.. [Accessed 14 1 2021].

[26] J. Ricardo Lourenço, B. Cabral, P. Carreiro, M. Vieira, and J. Bernardino, "Choosing the right NoSQL database for the job: a quality attribute evaluation," *J. Big Data*, vol. 2, no. 1, p. 18, 2015.

[27] B. R. Sinha, G. W. Romney, P. P. Dey, and M. N. Amin, "Estimation of database complexity from modeling schemas," *J. Comput. Sci. Coll.*, vol. 30, no. 2, p. 95–104, 2014.

[28] "What Are the Major Advantages of Using a Graph Database?," DZone Database, [Online]. Available: https://dzone.com/articles/what-are-the-pros-and-cons-of-using-a-graph-databa. [Accessed 15 1 2021].

[29] "Crimes - 2001 to present - Dashboard," Chicago Data Portal, [Online]. Available: https://data.cityofchicago.org/Public-Safety/Crimes-2001-to-Present/ijzp-q8t2. [Accessed 27 12 2020].

[30] "CQL for Apache Cassandra 2.0 & 2.1," Datastax, [Online]. Available: https://docs.datastax.com/en/archived/cql/3.1/cql/cql_reference/select_r.html. [Accessed 12 1 2021].

[31] "Start Developing with MongoDB," MongoDB, [Online]. Available: https://docs.mongodb.com/drivers/. [Accessed 14 1 2021].

## VIII. Appendixes

A) Steps taken for running queries

1. Cassandra

Step 1) Prompt 'cqlsh' in the VM terminal.

Step 2) Create keyspace

i) create keyspace project with replication={'class':'SimpleStrategy', 'replication_factor':3};

ii) use project;

Step 3) Create table

i) COPY crimes FROM '/home/Dataset2.csv' WITH DELIMITER='|' AND HEADER=TRUE;

Step 4) Run queries

i) TRACING ON;

ii) use project;

iii) SELECT * FROM crimes LIMIT 1000;

iv) SELECT * FROM crimes WHERE arrest = true LIMIT 1000 ALLOW FILTERING;

v) SELECT* FROM crimes WHERE primarytype = 'THEFT' LIMIT 1000 ALLOW FILTERING;

vi) SELECT * FROM crimes WHERE locationdescription = 'STREET' LIMIT 1000 ALLOW FILTERING;

vii) SELECT * FROM crimes WHERE primarytype = 'THEFT' AND locationdescription = 'STREET' AND year = 2016 LIMIT 1000 ALLOW FILTERING;

viii) SELECT * FROM crimes WHERE year > 2010 LIMIT 1000 ALLOW FILTERING;

2. MongoDB

Step 1) Load data from csv file via VM terminal after creating a collection called 'project;

i) mongoimport --type csv -d exercises -c crimes --type csv --file /home/Dataset2.csv --headerline

Step 2) Prompt 'mongo' in VM terminal

Step 3) Run queries

i) use project

ii) db.crimes.find().limit(1000).explain("executionStats")

iii) db.crimes.find({'Arrest':'true'}).limit(1000).explain("executionStats")

iv) db.crimes.find({"Primary Type":"THEFT"}).limit(1000).explain("executionStats")

v) db.crimes.find({"Location Description":"STREET"}).limit(1000).explain("executionStats")

vi) db.crimes.find({"Year":2016,"Primary Type":"THEFT","Location Description":"STREET"}).limit(1000).explain("executionStats")

vii) db.crimes.find({Year:{$gt:2010}}).limit(1000).explain("executionStats")

viii) db.crimes.find({$or:[{Year:2012},{Year:2013}]}).explain("executionStats")

B) Command Line Program to Calculate Execution Time in Node JS

```
const args = process.argv;


if (args[2]) {
  if (args[3]) {
    console.log(`START: ${args[2]} END: ${args[3]}`)
    var start = new Date(args[2])
    var end = new Date(args[3])
    var result = (end - start)/1000

    console.log(`\nTime Difference:${result} seconds`)
} else {
  console.log("please insert 2nd dates")}
 } else {
  console.log("please insert dates")
}
```