Comparative Performance Analysis of various NOSQL databases

Shubham Bothara
Illinois Institute of Technology
Big Data Technologies

1 INTRODUCTION

In response to explosion in the volume of data stored, the frequency in which this data is accessed, and performance and processing needs, a wide variety of different database technologies were developed grouped under the umbrella of NoSQL databases.

"A NoSQL database is a database that provides a mechanism to store and retrieve data other than the tabular relations used in relational databases." [1] These databases are schema-free. They support replication and are eventually consistent. They are also designed to handle can handle huge amounts of data.

There are four general types (most common categories) of NoSQL databases as follows:

- **Column-oriented**: e.g. Apache Cassandra,
- **Document store**: e.g. MongoDB,
- **Key-Value stores**: e.g. DynamoDB
- **Graph-based**: e.g. Neo4J

Each of these categories has its own specific attributes and limitations.

In this paper, we discuss various NoSQL database technologies. We then compare their performance on datasets of incremental sizes.

2 COLUMN-ORIENTED NOSQL DATABASES: APACHE CASSANDRA

"In column-oriented NoSQL database, data is stored in cells grouped in columns of data rather than as rows of data. Columns are logically grouped into column families. Column families can contain columns that can be created at runtime or while defining the schema." [2] In Column-Oriented NoSQL Databases, read and write is done using columns rather than rows. Values of a single column are stored contiguously and Every column is treated individually. This makes the search/access and data aggregation (e.g. COUNT, SUM, AVG, MIN, MAX) faster.

2.1 APACHE CASSANDRA: OVERVIEW

Apache Cassandra is a column-oriented, distributed open source NoSQL database. It is massively scalable, consistent with ability to manage large amounts of structured, semi-structured, and

unstructured data. It was developed at Facebook with its distribution design based on Amazon's Dynamo while its data model is based on Google's Bigtable. Other features of Cassandra include continuous availability, linear scalability, operational simplicity and fault tolerance.

Cassandra has a query language of its own known as Cassandra Query Language (CQL), that supports SQL-like commands. CQL has many features for querying data but is not as extensive as SQL. For example, CQL does not support joins and sub-queries. Basic queries on Cassandra include the GET, SET and DEL. The GET command is used to read data back from either a whole column family or a desired column from the column family. If we want to update data or create new ones, the SET command is used. With DEL command, we can delete either a column or the entire column family.

2.2 APACHE CASSANDRA: DATA MODEL

The main components of Apache Cassandra data model are: Cluster, Keyspace, column and column families.

- **Cluster:** In a distributed system, Cluster is the outermost container in Cassandra. Every node in a distributed system has a replica for fault tolerance, which takes charge in case of a failure. These nodes are arranged in Cassandra in a cluster in a ring format and Cassandra assigns data to them.
- **Keyspace:** The outermost container for data in Cassandra is keyspace. The basic attributes of Keyspace in Cassandra are:

    1. **Replication placement Strategy**: It defines the strategy to place replicas in the ring.
    2. **Replication factor**: It is the number of machines in the cluster to receive a replica of the data.
    3. **Column Families**: A keyspace is a container of one or more column families. Column Families represent the data structure.

- **Column:** A column is the basic data structure of Columns as a tuple represented as name: value: timestamp

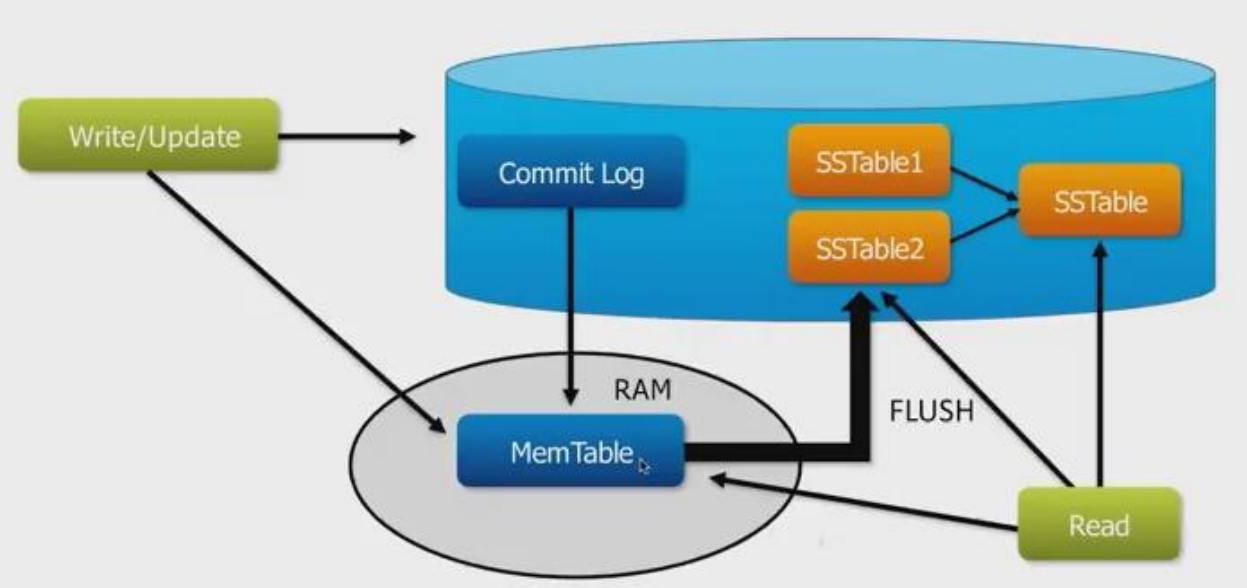
- **Column Families:** A column family is a collection for ordered rows containing a collection of columns. It is represented as couple, name: value where name is the column family name and value is an array of columns.

- **Super Column:** A Super Column is a collection of Column Families as an array represented by the couple, name: value, where name is the super column name, and value is an array of column families. The array is sorted by column family name

2.3 APACHE CASSANDRA: ARCHITECTURE

2.3.1 Given below are the key components of Cassandra:

- **Node** - Data is stored in Nodes.
- **Data center** - Collection of related nodes.
- **Commit log** - Every write operation is written to the commit log. It is a crash recovery mechanism.
- **Cluster** - A cluster contains multiple data centers.
- **Mem-table** - A mem-table is a memory-resident data structure. After commit log, the data will be written to the mem-table. Sometimes, for a single-column family, there will be multiple mem-tables.
- **SSTable** - It is a disk file. The data is flushed to, from mem-table, when its contents reach a threshold value.
- **Bloom filter** - It is a special kind of cache that determines whether an element is part of the set. Bloom filters are accessed after every query.



2.3.2 Write Operations

Every write activity is captured by the nodes in the **commit logs** written in the nodes. Then the data is captured and stored in the **mem-table.** Whenever the mem-table is full, data will be written into the **SStable** data file. Cassandra periodically consolidates the SSTables, discarding unnecessary data.

2.3.3 Read Operations

During read operations, Cassandra gets values from the mem-table and checks the bloom filter to find the appropriate SSTable that holds the required data.

2.3.4 Cassandra Query Language

Cassandra is accessed through its nodes using Cassandra Query Language (CQL). CQL treats the database **(Keyspace)** as a container of tables. CQLSH a prompt is used to work with CQL.

2.3.4.1 CQL Commands

- **CREATE KEYSPACE** - Creates a KeySpace in Cassandra.

```
Syntax:        CREATE KEYSPACE | SCHEMA IF NOT EXISTS keyspace_name
               WITH REPLICATION = map
               AND DURABLE_WRITES = true | false
```

- **USE** - Connects to a created KeySpace.

```
Syntax:        USE keyspace_name
```

- **ALTER KEYSPACE** - Changes the properties of a KeySpace.

```
Syntax:        ALTER KEYSPACE | SCHEMA keyspace_name
               WITH REPLICATION = map
               | WITH DURABLE_WRITES = true | false
               AND DURABLE_WRITES = true | false
```

- **DROP KEYSPACE** - Removes a KeySpace

```
Syntax:        DROP KEYSPACE | SCHEMA IF EXISTS keyspace_name
```

- **CREATE TABLE** - Creates a table in a KeySpace.

```
Syntax:        CREATE TABLE IF NOT EXISTS keyspace_name.table_name
               (column_definition, column_definition, ...)
               WITH property AND property ...
```

- **ALTER TABLE** - Modifies the column properties of a table.

```
Syntax:        ALTER TABLE keyspace_name. table_name instruction;
```

- **DROP TABLE** - Removes a table.

```
Syntax:       DROP TABLE IF EXISTS keyspace_name.table_name
```

- **TRUNCATE** - Removes all the data from a table.

```
Syntax:       TRUNCATE keyspace_name.table_name
```

- **CREATE INDEX** - Defines a new index on a single column of a table.

```
Syntax:       CREATE CUSTOM INDEX IF NOT EXISTS index_name
              ON keyspace_name.table_name ( KEYS ( column_name ) )
              USING class_name WITH OPTIONS = map
```

- **DROP INDEX** - Deletes a named index.

```
Syntax:       DROP INDEX IF EXISTS keyspace.index_name
```

- **INSERT -** Adds columns for a row in a table.

```
Syntax:       INSERT INTO keyspace_name.table_name
              ( identifier, column_name...)
              VALUES ( value, value ... ) IF NOT EXISTS
              USING option AND option
```

- **UPDATE** - Updates a column of a row.

```
Syntax:       UPDATE keyspace_name.table_name
              USING option AND option
              SET assignment, assignment, ...
              WHERE row_specification
              IF column_name = literal AND column_name = literal . . .
              IF EXISTS
```

- **DELETE** - Deletes data from a table.

```
Syntax:       DELETE [ column_name [ , column_name ] [ … ]
              FROM [ keyspace_name. ] table_name
              [ USING TIMESTAMP timestamp_value ]
              WHERE row_specification
              [ IF [ EXISTS | condition [ AND condition ] [ … ] ] ]
```

- **SELECT** - This clause reads data from a table

```
Syntax:       SELECT select_expression
              FROM keyspace_name.table_name
              WHERE relation AND relation ...
              ORDER BY ( clustering_column  ASC | DESC ...)
              LIMIT n
              ALLOW FILTERING
```

3 DOCUMENT-STORE NOSQL DATABASES: MONGODB

Document store databases are schema-less database, where the data is stored as key-value pairs, with the values as documents with some structure. Data and relationships are stored as a collection of independent documents. Documents can contain many different key-value pairs, or nested documents. Documents are generally encoded as XML, JSON or BSON. CouchDB and MongoDB are the most popular document based databases.

3.1 MONGODB: OVERVIEW

MongoDB is an open-source document store working on concept of collection and document. Collection is a group of MongoDB documents. It is the equivalent of an RDBMS table. A collection exists within a single database. Collections do not enforce a schema. A record in MongoDB is a document, which is a data structure composed of field and value pairs. Documents within a collection can have different fields. Typically, all documents in a collection are of similar or related purpose. They have dynamic schema. Dynamic schema means that documents in the same collection do not need to have the same set of fields or structure, and common fields in a collection's documents may hold different types of data. The values of fields may include arrays, other documents and arrays of documents. It provides high performance, high availability, and automatic scaling.

The key features of MongoDB are:

- It is Schema less, i.e. Number of fields, content and size of the document can differ from one document to another.
- It has a clear structure
- No complex joins.
- Fast in-place updates

The following table shows the relationship of RDBMS terminology with MongoDB.

MongoDB	RDBMS
Database	Database
Collection	Table
Document	Tuple/Row
Field	column
Embedded Documents	Table Join
Primary Key (Default key _id provided by mongodb itself)	Primary Key

3.2 MONGODB: DATA MODEL

Data in MongoDB has a *flexible schema*. MongoDB's collections do not enforce document structure. It is schema-less. Each document can match the data fields of the represented entity, even if the data has substantial variation. As a result of the document model, data in MongoDB is more localized, which dramatically reduces the need to JOIN separate tables. The result is dramatically higher performance and scalability across commodity hardware as a single read to the database can retrieve the entire document. In practice, however, the documents in a collection share a similar structure.

Write operations are atomic at the document level in MongoDB. No single write operation can affect more than one document or more than one collection. A data model with embedded data allows all related data for an entity to be present in a single document. This facilitates write operations for an entity.

3.2.1 MongoDB Commands

1. **use** - Used to create database.

```
Syntax:          use DATABASE_NAME
```

2. **dropDatabase** - Used to drop an existing database.

```
Syntax:          db.dropDatabase()
```

3. **createCollection** - Used to create collection.

```
Syntax:          db.createCollection(name, options)
```

4. **drop** - Used to drop a collection from the database.

```
Syntax:          db.COLLECTION_NAME.drop()
```

5. **ensureIndex** - To create an index

```
Syntax:          db.COLLECTION_NAME.ensureIndex({KEY:1})
```

6. **insert** – Add a document to a collection

If we want to insert one record:

```
Syntax:          db.collection.insertOne()
```

If we want to insert multiple records:

```
Syntax:          db.collection.insertMany()
```

7. **find** - Retrieves documents from a collection

```
Syntax:          db.collection.find()
```

If we just want to see one document from a collection, we can use findOne() method

```
Syntax:          db.collection.findOne()
```

8. **update** – To modify existing documents in a collection

```
Syntax:          db.collection.updateOne()
```

If we want to modify multiple documents

```
Syntax:          db.collection.updateMany()
```

If we want to replace a record

```
Syntax:          db.collection.replaceOne()
```

9. **delete** - Deletes documents permanently from the database.

If we want to remove one documents

```
db.collection.deleteOne() New in version 3.2
```

If we want to remove multiple documents

```
db.collection.deleteMany() New in version 3.2
```

4 EXPERIMENTATION

4.1 DATASET

The dataset chosen consists 1,000,000 most frequent 3-grams from the Corpus of Contemporary American English (COCA). The instances comprise of following 4 attributes:
Freq: # of times the 3 words occur in group
Word1: The first word.
Word2: The second word.
Word3: The third word.

4.2 EXPERIMENTATION PLAN

To evaluate the performance of the databases, 6 queries listed below were run three times each on incremental no. of records of 100,000 to 1,000,000 and the time required for the query execution was calculated.

The data is stored in the databases in the following format:
Cassandra: Each Sample is stored as a new table in Cassandra. Each table has the following structure:

```
cqlsh> use project;
cqlsh:project> select * from sample0;

 word1          | srno   | freq | word2        | word3
----------------+--------+------+--------------+-------------
         await | 136503 |   36 |          the |     arrival
```

MongoDB: Each Sample is stored as a new collection in MongoDB and each document has following format:

```
> use mydb
switched to db mydb
> db.sample0.find().pretty()
{
        "_id" : ObjectId("5904e12ec5a215142ac624c5"),
        "word1" : "distorted",
        "srno" : 235785,
        "word2" : "by",
        "freq" : "111",
        "word3" : "the"
}
```

Queries performed on Databases:

1. Create a table and insert n records in data, where n is the sample size.
2. Read all the records inserted ordered by frequency
3. Count distinct words in word1.
4. What is the maximum frequency for word1 = “the”
5. Delete records with word1 = 'san'
6. Update record srno = 466405, freq= 30, word1 = 'law' with value of word2 = ‘firm’

4.3 EXPERIMENT

Following is a snapshot of Table created constituting time required in seconds for each query.

No. of records -->		102000		204001	
Query No.	Run #	Time required in Cassandra	Time required in MongoDB	Time required in Cassandra	Time required in MongoDB
1	1	114.037354	23.65061474	210.9455848	47.27450514
	2	103.9547513	23.52874351	227.2488482	47.01571274
	3	105.8618305	23.57024002	221.2183807	47.80712223
	Avg	107.9513119	23.58319942	219.8042712	47.36578004
2	1	0.000290155	0.000209093	0.000338554	6.53E-05
	2	0.000366449	0.000202417	0.000286579	6.79E-05
	3	0.000455379	0.000198364	0.000263929	6.29E-05
	Avg	0.000370661	0.000203292	0.000296354	6.54062E-05
3	1	0.040290356	10.20999837	0.05957675	19.99713516
	2	0.067286491	6.872420073	0.053408861	19.94910192
	3	0.039939165	10.31710267	0.054956436	20.05431819
	Avg	0.049172004	9.133173704	0.055980682	20.00018509
4	1	0.00069809	8.32E-05	0.001087904	9.94E-05
	2	0.000754356	8.06E-05	0.001027107	9.30E-05
	3	0.000689507	0.000114202	0.000669241	9.23E-05
	Avg	0.000713984	9.26654E-05	0.000928084	9.48906E-05
5	1	0.001029253	0.03919816	0.001043081	0.076003313
	2	0.001114368	0.038089991	0.001328468	0.072303772
	3	0.001409769	0.037473917	0.001342773	0.073604107
	Avg	0.001184464	0.038254023	0.001238108	0.073970397
6	1	0.001138449	0.041039944	0.001007318	0.081515789
	2	0.000967979	0.039074898	0.000920773	0.07207346
	3	0.000962019	0.041248322	0.000855207	0.082269907
	Avg	0.001022816	0.040454388	0.000927766	0.078619719

From the experiment data generated, for each query the plot of no. of records against time required is as follows:

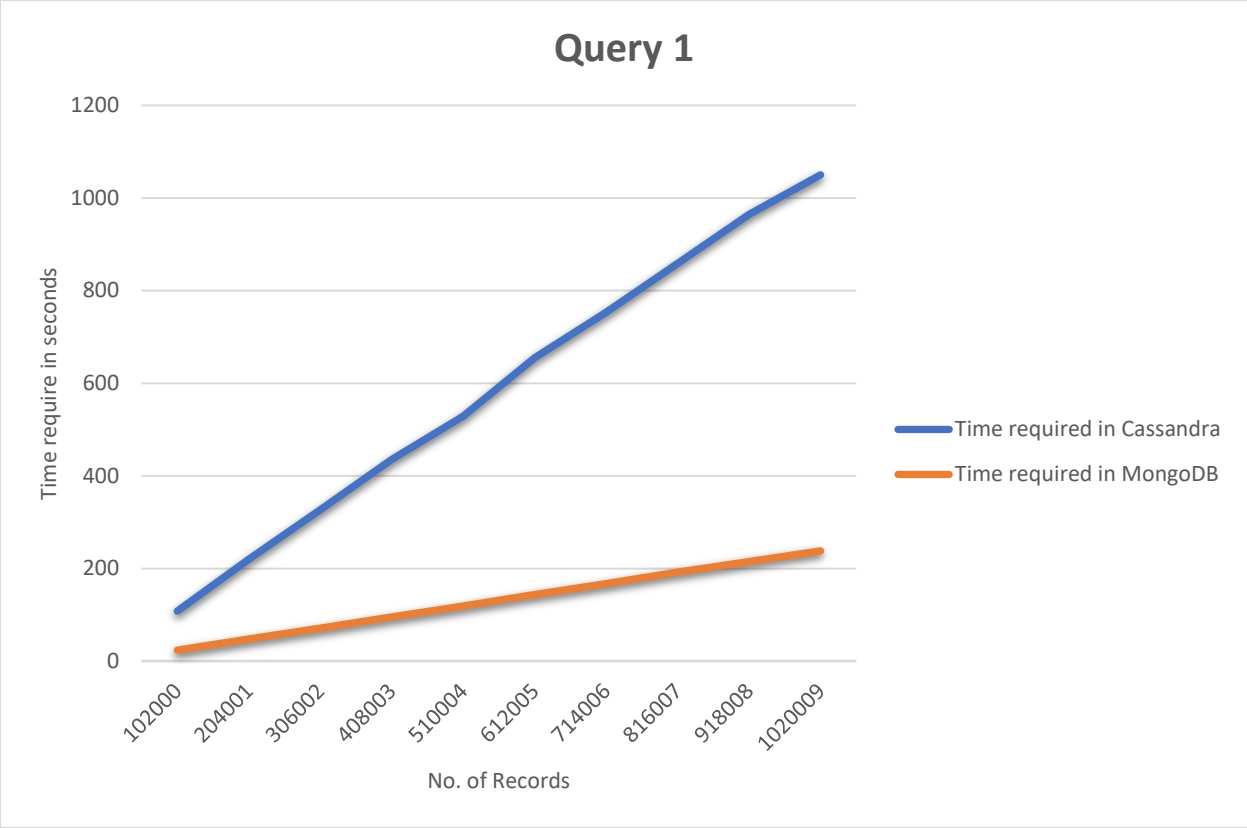


Figure 4-1: Query1

The Figure 4-1: Query1 shows that the time required to insert records is linear in both MongoDB and Cassandra. Also, Cassandra requires significantly more time than MongoDB for records insertion. This is due to the columnar data storage of Cassandra.

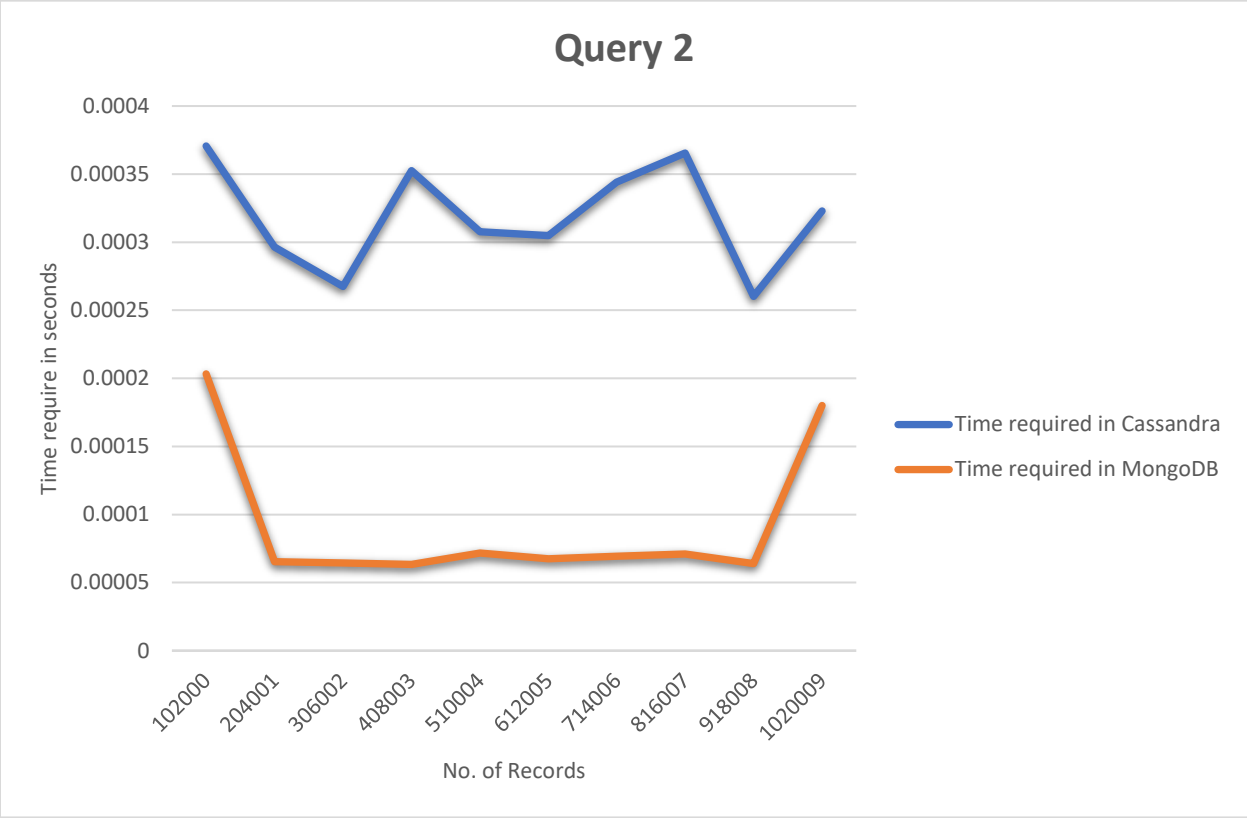


Figure 4-2: Query2

The Figure 4-2: Query2 shows that the time required to retrieve the records is more for Cassandra than MongoDB.

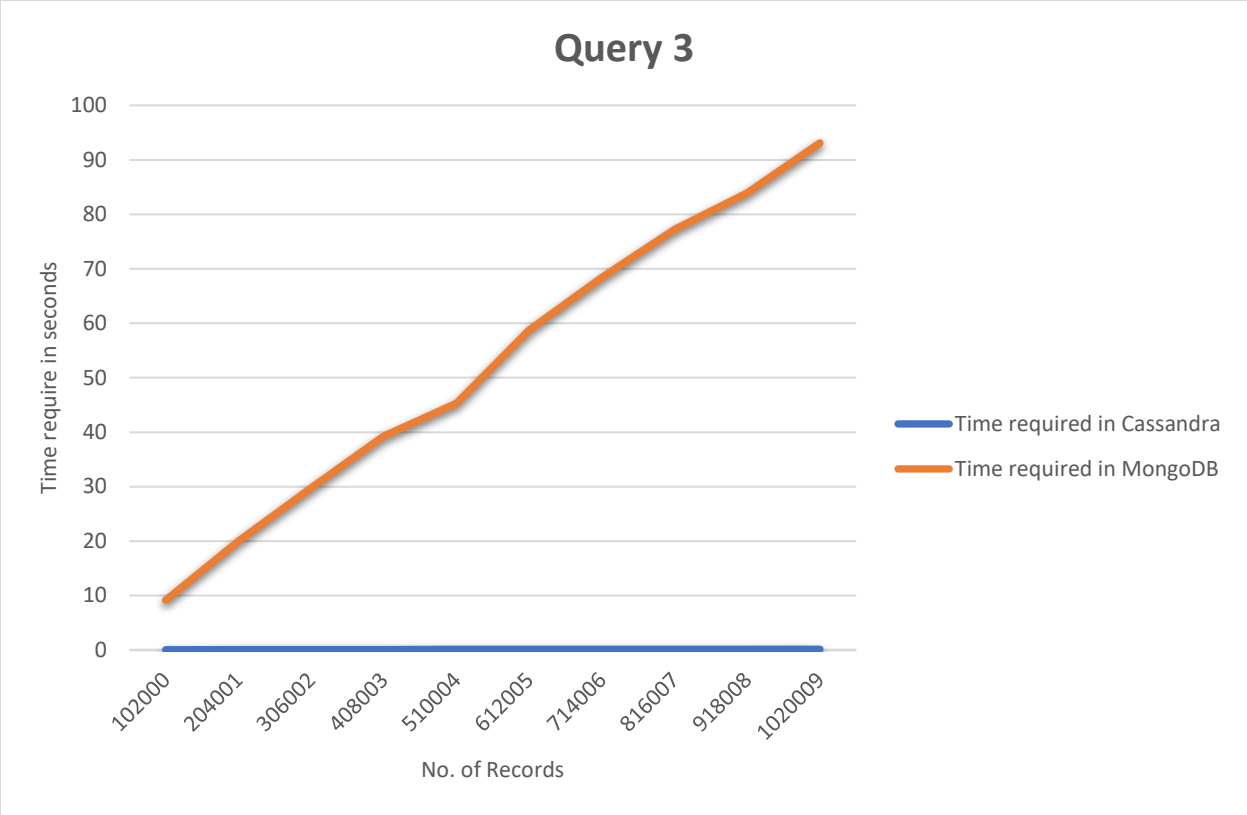


Figure 4-3:Query3

Figure 4-3 shows that MongoDB requires significantly large amount of time to count the number of distinct records than Cassandra. This is due to the columnar storage of data in Cassandra, which makes querying on a particular column faster. While MongoDB has to traverse through all the documents, Cassandra just scans through the particular column.

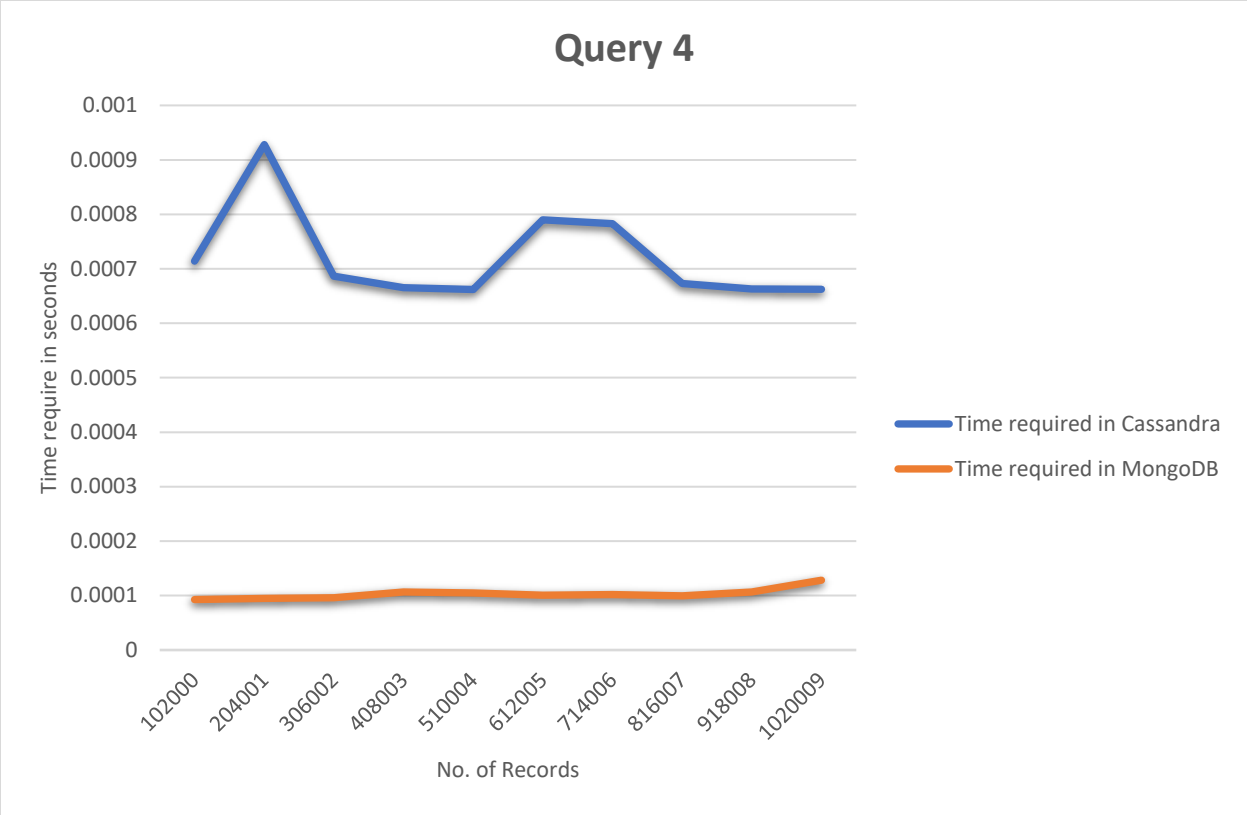


Figure 4-4: Query4

Figure 4-4 shows that MongoDB requires significantly less amount of time to retrieve the max frequency of the word 'the' in word1 than Cassandra. Cassandra searches for the word 'the' in the column word1 and then maps to frequency to find all frequency and retrieve the max frequency. This is significantly larger than the time required by MongoDB to scan each document sequentially and retrieve the max frequency.

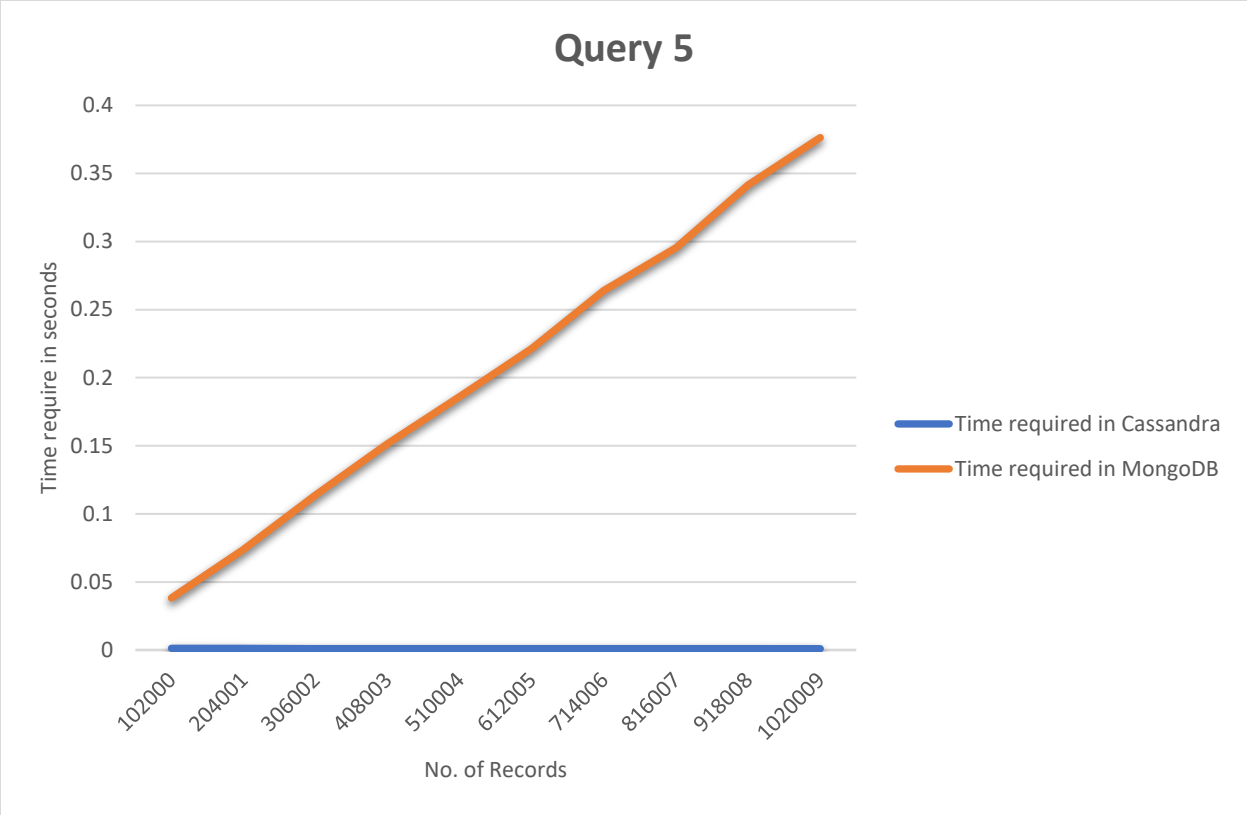


Figure 4-5:Query5

Figure 4-5 shows that MongoDB requires significantly large amount of time to delete records than Cassandra. This is due to the columnar storage of data in Cassandra, which makes querying on a column faster, and on match it deleted the record. While MongoDB has to traverse through all the documents to search for the document, Cassandra just scans through the particular column and removes the record.

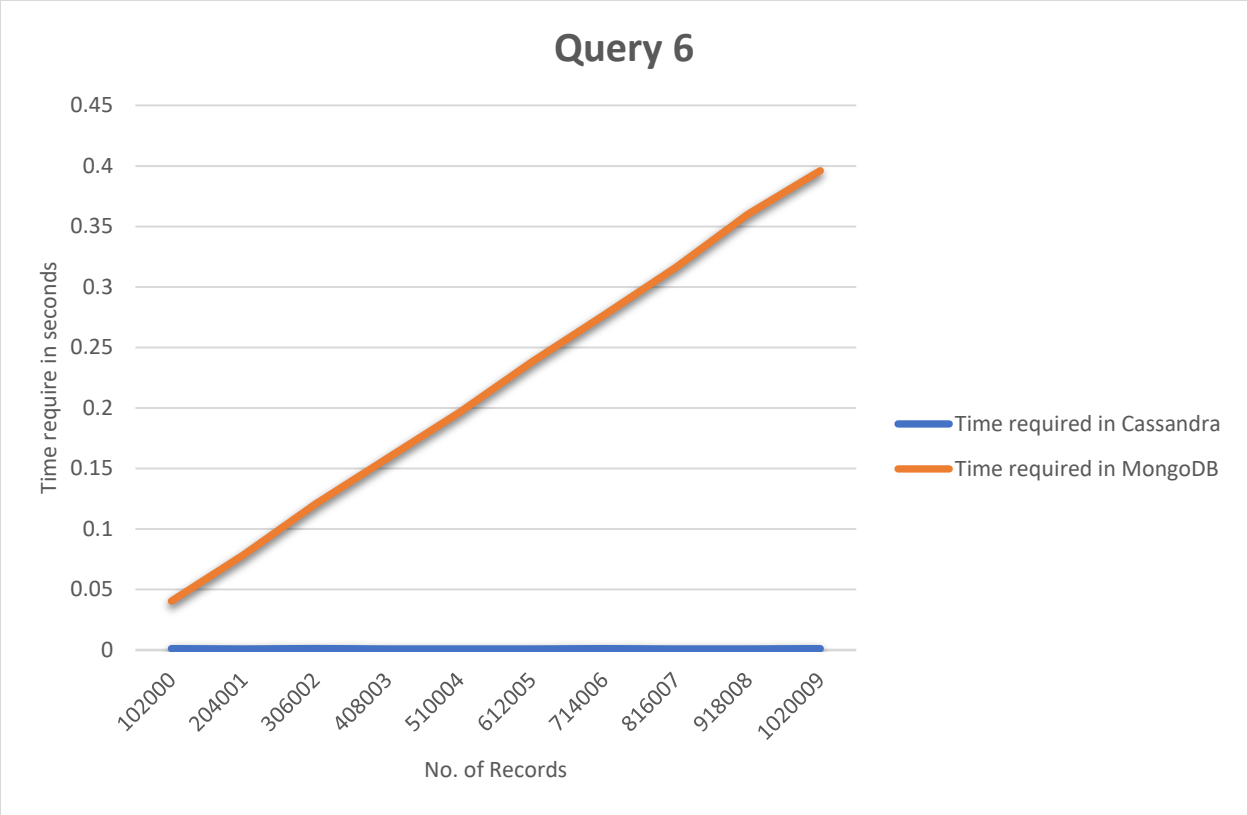


Figure 4-6:Query6

Figure 4-6 shows that MongoDB requires significantly large amount of time to update a record than Cassandra.

4.4 CONCLUSION

From the above experiment, we can conclude that Cassandra requires greater time to store records, due to it columnar storage. But, this makes it faster for querying. Both MongoDB and Cassandra are scalable. This is shown by the fact that they show linear increase in time for insertion or reading with increase in no. of records.

5 REFERENCES

[1] [Online]. Available: www.tutorialspoint.com/articles/what-is-nosql-and-is-it-the-next-big-trend-in-databases.

[2] [Online]. Available: (https://www.3pillarglobal.com/insights/exploring-the-different-types-of-nosql-databases).

[3] "NoSQL explained," [Online]. Available: https://www.mongodb.com/nosql-explained. [Accessed 04 2017].

[4] "Ngrams," [Online]. Available: http://www.ngrams.info/download_coca.asp. [Accessed 15 04 2017].

[5] [Online]. Available: http://api.mongodb.com/python/current/tutorial.html.

[6] [Online]. Available: https://www.tutorialspoint.com/mongodb/mongodb_quick_guide.htm.

[7] [Online]. Available: http://docs.datastax.com/en/cassandra/3.0/cassandra/architecture/archIntro.html.

[8] [Online]. Available: http://www.w3resource.com/mongodb/nosql.php.

[9] [Online]. Available: https://www.codeproject.com/Articles/630103/A-Beginners-Guide-to-NoSQL.

[10] [Online]. Available: https://www.3pillarglobal.com/insights/exploring-the-different-types-of-nosql-databases.

[11] [Online]. Available: http://cqlengine.readthedocs.io/en/latest/index.html.

[12] [Online]. Available: https://datastax.github.io/python-driver/cqlengine/queryset.html#retrieving-objects-with-filters.

[13] [Online]. Available: https://github.com/cqlengine/cqlengine.

[14] [Online]. Available: https://media.readthedocs.org/pdf/cqlengine/latest/cqlengine.pdf.