B669 Project Report

Comparison of Cassandra and MongoDB on Functionality and Performance for a University Library Text Corpus

Group members
Harsh Savla (hsavla@indiana.edu)
Sandeep Taduri

Under the guidance of
Prof. Beth Plale

at

Indiana University Bloomington

Table of Contents

1. Introduction

In this project, we study the new breed of NoSQL data stores:-Cassandra and MongoDB. We model the provided data in each system, understand the required queries and carry out a performance evaluation of write, read and query times in each system. We present our ideas and findings in this report.

2. Schema

Cassandra

Cassandra is a structured key-value store. Keys map to multiple values which are grouped into column families. The column families are fixed when a Cassandra database is created, but columns can be added to a family at any time.

There are two super column families: PageVolumeSFamily and SizeVolumeSFamily. The query to create the PageVolumeSFamily is:

```
create column family PageVolumeSFamily with column_type = 'Super'

and key_validation_class = 'UTF8Type'

and default_validation_class = 'UTF8Type'

and comparator = 'LongType(reversed = true)'

and subcomparator = 'UTF8Type'
```

Note: The actual value of the column does not matter.

The schema for the SizeVolumeSFamily is exactly similar except that in this case the columns represent the size of the volume.

Pros:

We have adopted the wide row approach to schema design and have exploited the natural sorting provided by Cassandra. The super column is sorted in reverse order. Hence the volume with maximum number of pages will occur first. Similarly, for the volume with max size will occur first.

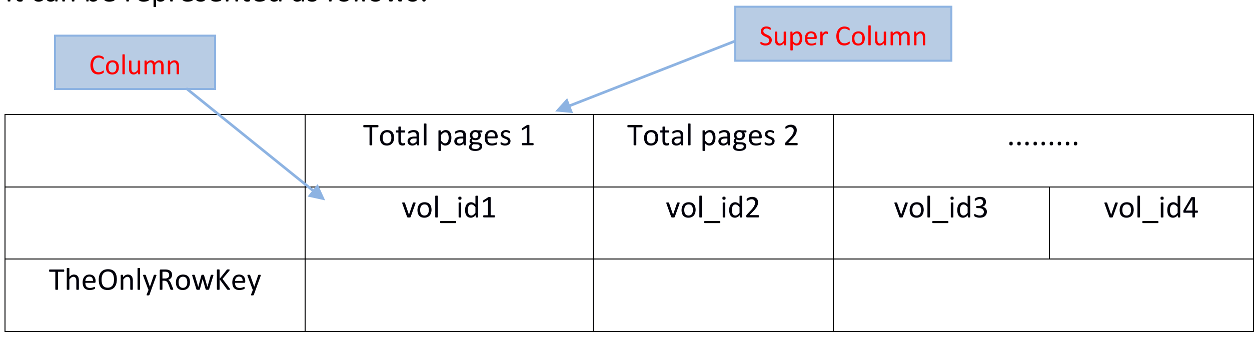
Cons:

There is only one row. In the case where there we are using distributed set of nodes, this scheme will fail as Cassandra uses key hashes to distribute the keys across machines.

<u>Description:</u>

There is only row key. The super columns are the total number of pages in a volume. The columns are all those volumes having the number of pages equal to the super column.

It can be represented as follows:



	Total pages 1	Total pages 2	....	
	vol_id1	vol_id2	vol_id3	vol_id4
TheOnlyRowKey				

For eg. If volume_1 has 100 pages, volume_2 and volume_3 have 200 pages then the structure would look something like follows:

	100	200	
	volumeid_1	volumeid_2	volumeid_3
TheOnlyRowKey	[No Use]	[No Use]	[No Use]

There is one Standard column family to store the actual volume records. The following query is used to create the column family:

create column family VolumeRecordsCFamily with column_type = 'UTF8Type'

and key_validation_class = 'UTF8Type'

and default_validation_class = 'UTF8Type'

and comparator = 'UTF8Type'

MongoDB

MongoDB is a high performance document-based NoSQL database. The documents in MongoDB are BSON objects which are conceptually similar to JSON objects.

There are 2 collections:- volumemetada stores the mete-data such as page count and volume size for each volume whereas the volumerecords collection stores the page content for each page in the volume.

1. volumemetada

```
json structure for this collection:

    [

                {

                        volumename: 'vol1',

                        pagecount : '20',

                        volumesize : '20000'

                },

                {

                        volumename: 'vol2',

                        pagecount : '30',

                        volumesize : '30000'

                }

                .....

                .....

    ]
```

2. **volumerecords**

```
    json structure for this collection:

    [

                {

                        volumename : 'vol1',

                        pagerecords: [

                                        {p1: 'p1_content'},

                                        {p2 : 'p2_content'}

                                        ....

                                        ....
```

```
                                                ]
                    },
                    {
                                volumename : 'vol2',
                                ........
                    },
                    ......
    ]
```

3. Workflow

- Before the parsing of files, we create the database schema with appropriate column families in Cassandra and collections in MongoDB.
- In our program, we have separate classes to handle the database CRUD operations for Cassandra and MongoDB. We start with Cassandra, insert all the required data into it, and then do the same for MongoDB. So we recurse twice on the root directory.
- We start by recursively exploring each directory from the root directory.
- If the file encountered is an XML file, the meta-data such as number of pages is extracted from the volume record populated by the METSParser.
- The volume record name along with its meta-data is inserted into the appropriate data store :- Column family for Cassandra and Collection for MongoDB.
- If the file encountered is a zip file, we extract each .txt entry (a page) from the file and put its content in a hashmap whose key is the page number or the name of the zip entry file.
- The content is encoded with UTF8 encoding before putting it in map.
- In Cassandra, we use Hector client's 'Mutator' object to perform the insertion operation.
- For MongoDB we use 'BasicDBObjectBuilder' object to put entire map (consisting of page id as key and page content as value) into the data collection at once.

4. Query Support

Cassandra

<u>Queries:</u>

Q1. Query 1 to get volume record with max number of pages.

Query > **get PageVolumeSFamily['TheLonlyRow'] limit 1;**

Ans > **super_column=1148 (page_count), column=nnc2.ark:/13960/t59c70b1c (vol_id)**

Q2. Query 2 to get volume with max size.

Query > **get SizeVolumeSFamily['TheLonlyRow'] limit 1;**

Ans > **super_column=3736540 (size) , column=nnc2.ark:/13960/t2r50b64n (vol_id)**

Q3. Query 3 to get average volume size.

Query > **Included in the code**

Ans > **670550.43 bytes**

MongoDB

Q1. Query 1 to get volume record with max number of pages.

Query > **db.volumemetada.find().sort({pagecount : -1}).limit(1)**

Ans > **vol_id: nnc2.ark:/13960/t59c70b1c, page_count: 1148**

Q2. Query 2 to get volume with max size.

Query > **db.volumemetada.find().sort({volumesize : -1}).limit(1)**

Ans > **vol_id: nnc2.ark:/13960/t2r50b64n, vol_size: 3736540 bytes (~3.73 MB)**

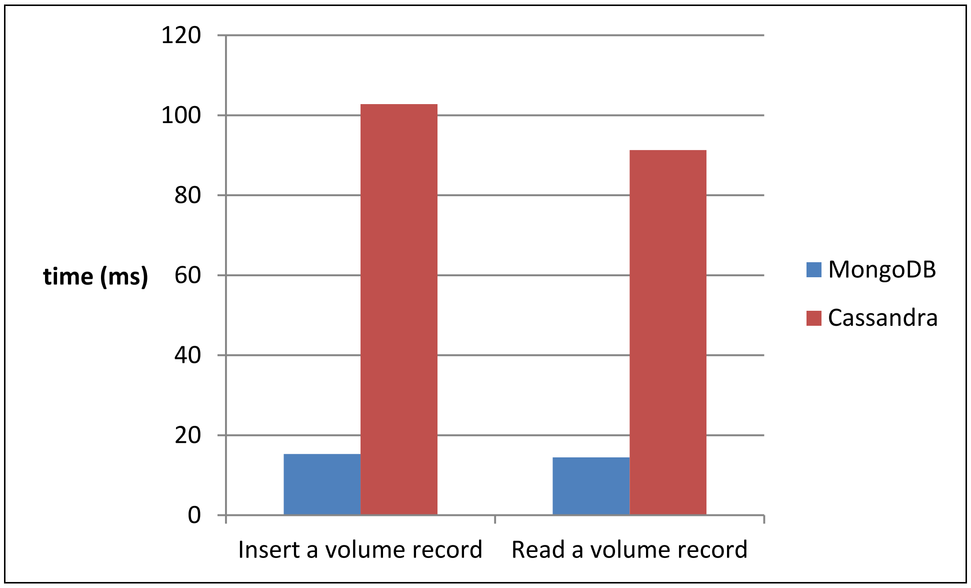
Q3. Query 3 to get average volume size.

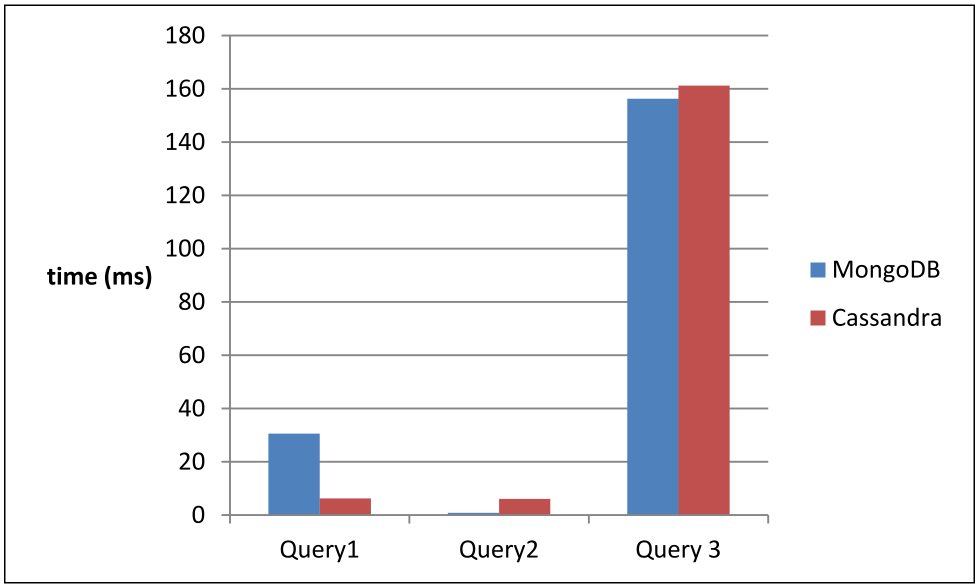
Query > **Included in the code**

Ans > **670550.43 bytes**

5. Performance Analysis

We time each of the queries as well as time the operations of inserting and reading a volume. We present our results below.





- We observed that the first time when MongoDB runs, it takes a lot of time to perform the query evidenced in Query 1. Next time, MongoDB stores the results in RAM and hence subsequent queries including the first run faster.
- We also observe that the time for inserting a volume is more than the time to read a volume for Cassandra and MongoDB.

6. NoSQL Technology

- NoSQL seems to be an incredibly powerful technology in the hands of developers who wish to scale their applications. During the course of this project we realized that the learning curve for NoSQL databases is not very steep and the tutorials available online more or less covers the common use cases. It seems the advantages of such databases can be truly seen only when the data is modeled properly. This requires thorough understanding of the data stores and may even require one to go underneath the surface, for example, studying the freely available source code of the implementations.

- Traditional SQL solutions require developers to normalize their data and hence there are a lot of tables created even for a small web enabled application. The NoSQL solution requires us to de-normalize data for best results. It enables us to access a document and get all the information related to it without ever us worrying about time and computation expensive query joins.

- For example, consider a scenario where we need to store a person's account information. In SQL we would basically create 3 tables:- Customer, Account and the relationship table between them. So if the query was to get all customer names for a particular account number we would need to join 2 tables. In a distributed environment it may be possible for the two tables to exist on different nodes. This query would then become very expensive. Instead, NoSQL allows us to put all the information related to a person in the data structure related to the person. This is not only more intuitive but also has massive performance benefits. Additionally it allows us scale horizontally (add more machines) instead of traditional SQL where we can scale vertically (by buying more powerful servers).

- Today, the amount of user-generated data such as comments or posts is enormous. This requires that data stores have the ability to write data very fast. It also requires them to have a flexible, dynamic schema which can grow horizontally (read columns), and vertically (read rows). NoSQL solutions have both these advantages and are hence ideal for large web applications such as Twitter and Facebook.

7. References

- http://www.mongodb.org
- http://cassandra.apache.org/
- http://stackoverflow.com
- http://maven.apache.org/
- http://hector-client.github.com/hector/build/html/index.html

8. Acknowledgement and Feedback

We thank Professor Beth Plale and instructors Miao Chena and Yuduo Zhou for their continued support throughout the course of the project.
We would also like to mention that the Piazaa platform proved to be very helpful for getting our doubts clarified.