Twitter Dataset Analysis and Modeling

[-rcmenon@indiana.edu](mailto:-rcmenon@indiana.edu)

1. Overview:

This project implementation is to learn through hands on experience, handling of big data in real world applications and understanding big data processing needs and steps which are mainly data storing, querying and performance evaluation. Also analyzing storing of big data efficiently i.e data models.

1. Objective:

The objective of this project is to analyze and compare performances for accessing a large twitter dataset. The dataset consists of 2 collections 1) profile 2) tweets. The goal is to define two different data models of MongoDB in which the twitter data would be ingested. Then performance evaluation would be conducted for storing and querying of both these data models.

1. Description:

MongoDB:

NoSQL products (and among them MongoDB) should be used to meet challenges. If we have one of the following challenges, we should consider using MongoDB:

*High Write Load*

MongoDB by default prefers high insert rate over transaction safety. If we need to load tons of data lines with a low business value for each one, MongoDB should fit.

*High Availability in an Unreliable Environment (Cloud and Real Life)*

Setting replica set (set of servers that act as Master-Slaves) is easy and fast. Moreover, recovery from a node (or a data center) failure is instant, safe and automatic.

When it comes to the application of twitter high availability and high write load is of prime importance. In the project, I used a twitter dataset to perform query operations against it, using different data models, each one having several advantages over the other. Data in MongoDB has a *flexible schema*. [Collections](http://docs.mongodb.org/manual/reference/glossary/#term-collection) do not enforce [document](http://docs.mongodb.org/manual/reference/glossary/#term-document) structure. This flexibility gives you data-modeling choices to match your application and its performance requirements.

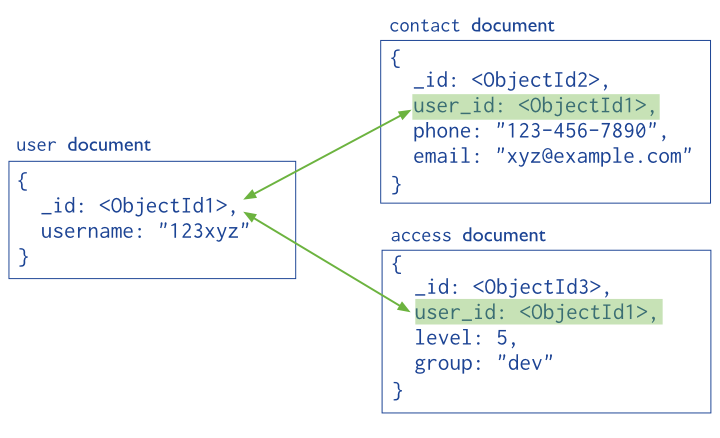
The two data models which are used are: 1) Referenced and 2) Embedded.

A data model organizes [data](http://en.wikipedia.org/wiki/Data) elements and standardizes how the data elements relate to one another. So in an application where there is a high demand of data, modeling is of outmost importance and a good data model always achieves better results with respect to performance and scalability.

As it is obvious mongoDB, which is a NoSQL system, has several advantages over SQL systems especially the lack of schema, horizontal scaling and sharding, MongoDB is the apt data storage mechanism for twitter.

Data Models:

1. Referenced(normalized) data model:



In general, we use normalized data models:

When embedding would result in duplication of data but would not provide sufficient read performance advantages to outweigh the implications of the duplication.

To represent more complex many-to-many relationships.

To model large hierarchical data sets.

How the twitter dataset maps to the referenced data model is explained in the code documentation.

1. Embedded data model:



In general, use embedded data models when:

When we have “contains” relationships between entities.

We have one-to-many relationships between entities. In these relationships the “many” or child documents always appear with or are viewed in the context of the “one” or parent documents.

How the twitter dataset maps to the Embedded data model is explained in the code documentation.

MapReduce using MongoDB:

The twitter dataset has a lot of documents in one collection and even a lot more documents in another collection. So the need is to merge those documents together but we don’t want to write a lot of code to achieve that. It also might take a lot of time to do it.

MongoDB comes with [map/reduce](http://docs.mongodb.org/manual/core/map-reduce/) which is an essential feature which we can use to achieve something like this. Map-reduce is a data processing paradigm condensing large volumes of data into useful aggregated results.

The power of mapreduce is truly realized in this project where two large collection of data is merged for better query performance. MapReduce was used for aggregating two collections into a single collection thus making use of the embedded data model, which gave faster query results.

1. Set-up for the project:

The project was implemented on a virtual machine (VM) which ran on Ubuntu operating system. The reason for choosing Ubuntu are its advantages over windows deploying the code for MongoDB. The code for running, building and deploying mongoDB is written in Java, it has a mongo driver which is added as an external library to the code. Later the entire program is bundled into a jar which can be run easily on the VM. The project is uploaded into a GitHub repo link provided below.

1. Conclusion:

So the project was completed as a part of the course I590: Projects in Big Data software. Today, huge repositories of structured, semi-structured and unstructured data collected across various digital platforms, social media and blogs or generated through simulation and modeling are at our disposal. These mass repositories are beyond the abilities of traditional database methods to analyze and understand effectively. This class played a major role in making us understand how big data analytics is an important concept to look forward in the future and the projects performed gave a hands on touch to the world’s biggest problem Big Data.

1. References:

[1] <https://wiki.cites.illinois.edu/wiki/display/forward/Dataset-UDI-TwitterCrawl-Aug2012>

[2] MongoDB Reference <http://docs.MongoDB.org/manual/reference>

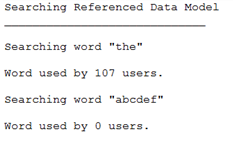
[3] Rui Li, Shengjie Wang, Hongbo Deng, Rui Wang, Kevin Chen-Chuan Chang: Towards social user profiling: unified and discriminative influence model for inferring home locations. KDD 2012:1023-1031

[4] GitHub: <https://github.com/cloud-class-projects/twitter-analysis.git>

**Project Results**

The project results are displayed in the VM terminal itself as I print out all the output in Java code. The purpose of this documents is to interpret the results of the performance of the data models.

1. Searching a high frequency word:



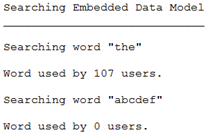
The program searches for a high frequency word “The” which is expected to be most common among tweets and the search result returns the word being used by 107 users. As it is expected the Embedded data model will also return the same results but let us take a look at the time.





So as we see here, there is a major difference in the time, as explained in the code documentation this a result of the referenced data model searching 2 collections to get the results, while on the other hand in the embedded data models aggregated output collection only once search is sufficient.

1. Searching a low frequency word:



Low frequency word abcdef is too rare to be used in tweets so as expected the users using word in their tweet is 0. Let us compare the timings.





In this scenario the referenced data model wins. In the code documentation as it mentions in the referenced data model the tweets collection is searched and when the word is not found, the list of output which should contain the UserIDs is empty so a simple if condition check exits the method of Embeddded data model. While on the other hand in Embedded data model the program searches in all tweets sub-document in the same collection ”reduced”, it even searches the empty tweets sub-document which is not mapped and hence the time difference. But it is noticed that the timings for embedded improves after consecutive runs.

Sample run after reducing data files:

