Koustuv Sinha & Hitesh Agarwal

koustuvsinha@gmail.com,hitesh.iem@gmail.com

Abstract

Due to Data Explosion, PetaBytes of data are generated in the Internet today thanks to Social Media. RDBMS readily fails in handling this huge volume of data. So, in this Project we explore the possibilities presented by NoSQL databases like MongoDB in handling Big Data, which in our case, is Human Genome Data.

Data warehousing & mining using mongodb

Final Year Project under the guidance of Prof Ee Kian Wong

# Contents

1. The Problem - Big Data

2. Why Not RDBMS

3. What is MongoDB

4. How MongoDB solves the Problem

5. Understanding MongoDB - a sample Project

6. Problems faced - and their solutions

7. Warehousing Human Genome Data

8. The way ahead

The Problem

# Data Explosion resulting in Big Data

Thanks to Social Media and other related Internet applications, Data available on the Internet have literally exploded over the past decade. Some statistics shown below expresses the concerns:

1. **Facebook** - The most popular social networking site generates data in size of Peta Bytes every day. In addition to scanning 105 terabytes of data every 30 minutes -- a process Facebook's products team often use to gauge how products are doing -- the company manages millions of photos and logs billions of likes to make sure its site is tailored to its users.

Here's a breakdown of how much data flows through the Facebook machine each day:

* 2.7 billion likes made daily on and off of the Facebook site
* 300 million photos uploaded
* 70,000 queries executed by people and automated systems
* 500+ terabytes of new data "ingested"

2. **Amazon.com** - One of the best online megastore as well as a provider for Cloud services handles data of more than 10 PetaBytes as of today. It handles millions of back-end operations every day, as well as queries from more than half a million third-party sellers. The core technology that keeps Amazon running is Linux-based and as of 2005 they had the world’s three largest Linux databases, with capacities of 7.8 TB, 18.5 TB, and 24.7 TB.

3. **Walmart** - handles more than 1 million customer transactions every hour, which is imported into databases estimated to contain more than 2.5 petabytes (2560 terabytes) of data – the equivalent of 167 times the information contained in all the books in the US Library of Congress.

These were a mere sample of the volume of data that is being transferred over the internet today. Now beyond Internet domain, other fields which generate huge data are as follows:

1. **Large Hadron Collider** - LHC experiments represent about 150 million sensors delivering data 40 million times per second. There are nearly 600 million collisions per second. After filtering and refraining from recording more than 99.999% of these streams, there are 100 collisions of interest per second. As a result, only working with less than 0.001% of the sensor stream data, the data flow from all four LHC experiments represents 25 petabytes annual rate before replication (as of 2012). This becomes nearly 200 petabytes after replication.

2. **Human Genome** - Decoding the [human genome](http://en.wikipedia.org/wiki/Human_Genome_Project) originally took 10 years to process, now it can be achieved in less than a week: the DNA sequencers have divided the sequencing cost by 10,000 in the last ten years, which is 100 times faster than the reduction in cost predicted by [Moore's Law](http://en.wikipedia.org/wiki/Moore%27s_Law).

3. **Climate Data** - The [NASA](http://en.wikipedia.org/wiki/Nasa) Center for Climate Simulation (NCCS) stores 32 petabytes of climate observations and simulations on the Discover supercomputing cluster.

4. **Flights** - A Boeing 737 generates 240 terabytes of flight data during a single flight across the US.

And the list goes on.

These huge amount of data sets are collectively known as Big Data. Storing, querying and analyzing this huge data sets are a big computational challenge. Traditional database softwares such as Relational Database cannot handle such big data.

Why not RDBMS

# Why RDBMS is not suitable to handle Big Data

Relational Database Management System came in 1980’s and became perfected in the late 1990’s. But, the data computer scientists had to handle in 1990’s is nowhere compared to the scenario now. As we have seen, Big Data is constantly exploding, breaking every barrier of data. Ten years from now, the data size which increase exponentially in Zettabyte or Yottabyte scale. So, in short, the landscape has completely changed for RDBMS to exist. The reasons are as follows:

1. **Vertical Expansion**: RDBMS systems expand vertically, that is more you put data in, the larger the data grows within the server. Nowadays, server capability ranges in TeraBytes of data. But, even if server technology advances in the coming years, it cannot keep up with the Big Data explosion. Soon, we will face a storage crisis, where a single server cannot handle large data sets.

2. **Complex Joins**: One can argue, however, that if the data is partitioned in several servers parallel, then RDBMS can handle big data. True, but it has to then perform complex joins to get the whole data. Joins in Relational database require huge overhead in terms of data and computation. So, complex joins will eventually slow down the system.

3. **Querying and extracting information**: RDBMS is not suitable for current Object Oriented Programming parlance, as data is handled more as an object now. So, to query and extract big data from the data sets, the problem lies in extracting each piece of information and storing it in separate fields of an object. Same goes for inserting an object in the database.

4. **Fixed Schema**: Relational databases require that schemas be defined before you can add data. For example, you might want to store data about your customers such as phone numbers, first and last name, address, city and state – a SQL database needs to know what you are storing in advance.

This fits poorly with agile development approaches, because each time you complete new features, the schema of your database often needs to change. So if you decide, a few iterations into development, that you'd like to store customers' favorite items in addition to their addresses and phone numbers, you'll need to add that column to the database, and then migrate the entire database to the new schema.

If the database is large, this is a very slow process that involves significant downtime. If you are frequently changing the data your application stores – because you are iterating rapidly – this downtime may also be frequent. There's also no way, using a relational database, to effectively address data that's completely unstructured or unknown in advance.

So, for these reasons, RDBMS is not suitable to handle the Big Data Problem.

MongoDB

# The leading NoSQL database

MongoDB is an open-source database used by companies of all sizes, across all industries and for a wide variety of applications. It is an agile database that allows schemas to change quickly as applications evolve, while still providing the functionality developers expect from traditional databases, such as secondary indexes, a full query language and strict consistency.

MongoDB is a document database that provides high performance, high availability, and easy scalability. It is a document oriented database, which uses JSON styled documents to store data. It has separate Query system which is rich of functionality and document based. Large Files can be easily stored by GridFS. Some salient features of MongoDB are :

Document Database

* Documents (objects) map nicely to programming language data types.
* Embedded documents and arrays reduce need for joins.
* Dynamic schema makes polymorphism easier.

High Performance

* Embedding makes reads and writes fast.
* Indexes can include keys from embedded documents and arrays.
* Optional streaming writes (no acknowledgments).

High Availability

* Replicated servers with automatic master failover.

Easy Scalability

* Automatic sharding distributes collection data across machines.
* Eventually-consistent reads can be distributed over replicated servers.

Fortune 500 companies and startups alike are using MongoDB to create new types of applications, improve customer experience, accelerate time to market and reduce costs.

MongoDB is a leading NoSQL database. NoSQL encompasses a wide variety of different database technologies and were developed in response to a rise in the volume of data stored about users, objects and products, the frequency in which this data is accessed, and performance and processing needs. Relational databases, on the other hand, were not designed to cope with the scale and agility challenges that face modern applications, nor were they built to take advantage of the cheap storage and processing power available today.

The Solution

# How MongoDB solves the problem

1. **Horizontal Expansion**: The MongoDB database grows horizontally, ie it can be easily partitioned or “sharded” into parallel servers without the overhead caused by Complex joins in case of RDBMS. As the size of the data increases, a single machine may not be sufficient to store the data nor provide an acceptable read and write throughput. Sharding solves the problem with horizontal scaling. With sharding, we can add more machines to support data growth and the demands of read and write operations.

So, if we have 1 TB of data, we can shard in 4 parts each having 256 MB of data each. In MongoDB, this sharding is handled automatically once the shards have been identified. There can be any number of shards depending on the need of the application as well as storage constraints.

2. **Dynamic Schema**: The schema of a collection is not fixed, it can be changed as and when required by the application. That makes it easy to make significant application changes in real-time, without worrying about service interruptions – which means development is faster, code integration is more reliable, and less database administrator time is needed.

3. **Suitable for Object Oriented Programming**: As MongoDB is a document based storage, the objects can be stored as it is in the database in JSON format. So, no need for extracting each piece of data and storing in collection. Also, same in case of retrieval, as Object Oriented data strcutures get automatically mapped to the data in the collection.

4. **Replication**: MongoDB also support automatic replication, meaning that we get high availability and disaster recovery without involving separate applications to manage these tasks. The storage environment is essentially virtualized from the developer's perspective.

5. **Map/Reduce**: Map-reduce is a data processing paradigm for condensing large volumes of data into useful aggregated results. For map-reduce operations, MongoDB provides the “[mapReduce](http://docs.mongodb.org/manual/reference/command/mapReduce/" \l "dbcmd.mapReduce" \o "mapReduce)” database command. Through this we can map similar data sets and combine them to get a reduced set. It is particularly important in case of Data Mining.

Using MongoDB

# A sample Library Management System

To get familiar with the working of MongoDB, we chose to do first a sample Project which is a conventional Library Management System.

The choice of Programming Language is JAVA. The database drivers for JAVA is readily available in MongoDB website.

This is a simple Library Management System, having the following entities:

Book, Author, Publisher – core classes

*Book*

id integer

name String

pub Publisher

isbn String

authors ArrayList<Integer>

category String

price Double

status String

*Author*

id integer

name String

address String

email String

phone String

books ArrayList<Integer>

*Publisher*

id integer

name String

address String

email String

phone String

books ArrayList<Integer>

This Library System has Many to Many relationship among Book and Author class and Many to One relationship between Book and Publisher class.

The source code of this sample project is available on <https://github.com/koustuvsinha/mongodb> .

# Problems faced - and their solution

**Many to Many Relationship Management:** Book and Author class has many to many relationship. A Book class points to multiple Authors, as well as an Author class points to multiple Books. The problem lies that, what is the optimal way of saving both the classes, so that this relationship is maintained.

**Answer:** We have made an integer array in both class, containing the id’s of the other classes. Book has an Integer array containing Author Id’s and vice versa.

**Transaction ACID Property handling:** MongoDB doesn’t support transaction controls like “Rollback” or “Commit” as in Relational Database. So, to ensure atomicity and concurrency, we have to devise a unique way to handle database transactions.

**Answer:** We have created commit & rollback equivalent in MongoDB as follows:

Two tables: *backup\_insert* and *backup\_update*.

Whenever any data is inserted in the database, it is also inserted in the *backup\_insert* table. When “rollback” is called, the data in the *backup\_insert* is extracted and deleted from the main table, else when “commit” is called, the data stored in *backup\_insert* or *backup\_update* is deleted.

Similar operations for *backup\_insert* table.

**Consistency Problem**: Solved by using MVCC or Multi version Concurrency Control emulation driver known as “mongomvcc”.

**Encapsulation**: library\_manager class handles all the operations with the MongoDB database, thus encapsulates the details from the library\_clerk class, who calls the methods available. The library\_manager class supports Generics, so can be invoked with reference to core classes.

Human Genome Data

# The main project

To analyze the potential of MongoDB, we have selected to do Data Warehousing on Human Genome Data, which is publicly available at National Center for Biotechnology Information (NCBI) website.

Objectives:

* Store the Human Genome data in MongoDB
  + Human Genome Data is now publicly available in NCBI website. But it is of 12 TB + size, ideal for our Big Data emulation.
  + First, we downloaded a single chromosome gene data of a test case human being - the size amounted to 950 MB
  + To insert it in MongoDB, we have to use GridFS
  + If the size of data to be stored in MongoDB is greater than 16 MB, then GridFS is used, which divides the file in chunks limiting the chunk size to 256 KB.
  + When we query the file, GridFS automatically reassembles the chunk as needed.

The Genome Data is in a binary format .BAM. So, we can either store it in Binary Format as it is, or we can convert it into .SAM using samtools.

64 Bit version of mongodb is preferred since 32 bit versions has a size limitation of 2GB data.

Practical Scenario: Insertion of 950 MB data took 7 seconds.

The way ahead

# The project roadmap

Until now we have understood the environment of MongoDB and how to operate it. We have started to accumulate and insert the Human Genome Project data in MongoDB. The next steps to do in the coming months are:

* Gain domain knowledge on Genomics
* Get 12 TB volume of Genetic Data
* Store it using Sharding
* Make clusters using commodity hardware
* Explore data mining
* Search for patterns

# Conclusion

This project would not have been possible without the able guidance of our Mentor Prof Ee Kian Wong. Also, we would like to thank the following entities for providing top-notch solutions whenever needed.

1. MongoDB documentation: <http://docs.mongodb.org/manual/>

2. Help regarding MongoDB programming practices: Stack Overflow

3. Java - MongoDB tutorials: MKyong.com

4. GitHub - for hosting our project

Thank you,

Koustuv Sinha & Hitesh Agarwal