Storing of Unstructured data into MongoDB using Consistent Hashing Algorithm

Saranraj Sankarapandi
PG Scholar, IIIT-Srirangam, Tiruchirapalli, Tamilnadu, India.

Dr.M. Sai Baba
Associate Director, RMG, Indira Gandhi Centre for Atomic Research, Kalpakkam, Tamilnadu, India.

S.Jayanthi
Assistant Professor,Department of Computer Science & Engineering,Anna University, BIT Campus Tiruchirapalli, Tamilnadu, India.

E.Soundararajan
Scientific Officer/E,SIRD,Indira Gandhi Centre for Atomic Research,Kalpakkam, Tamilnadu, India.

Abstract — In the modern world network users or internet users are increasing incredibly day by day because of that more and more unstructured data's are producing and consuming over the network. And how to maintain those data and improve the availability and scalability of the storage system becomes a considerable challenge. Nowadays some of the NoSQL databases are supported the unstructured data management and provide different advantages for the unstructured data management e.g. CassandraDB, CoughDB, MongoDB, DynamoDB etc. MongoDB that is providing the most flexible query functions for the unstructured data management compared to the other databases like Dynamo Db, Cassandra DB. The main objective of this paper is to store a large amount of unstructured data into the MongoDB with using the Consistent hashing algorithm. The consistent hashing algorithm is one of the algorithm for the storing the documents into the database using the consistent hash ring.

Index Terms — MongoDB, Unstructured data, NoSql, Storage, Consistent hashing algorithm

1. INTRODUCTION

A relational database is broadly used the database application to storing and retrieving data. Managing a large amount of data like the internet was incompetent in RDBMS. To conquer this problem, NOSQL comes into reality. The term NOSQL is short for Not Only SQL and was introduced in 2009. The name attempt to tag the appearance of a mounting number of non-relational, distributed data storage that frequently did not attempt to give ACID. NOSQL is not an instrument, but a methodology collected of numerous corresponding and challenging tool. The main advantage of the NOSQL database is that dissimilar to the relational database they hold unstructured data such as documents, e-mail, multimedia and social media proficiently. Most of the familiar features of NOSQL database can be summarized a schema is not predetermined, does not support join operations, high scalability and reliability, very simple data model, very simple query language and high availability. There are many benefits of NoSQL as compared to RDBMs, but also there are many obstacles to overcome before they can appeal to conventional enterprises. Few of the challenge and improvement which means RDBMs systems are stable and abundantly functional whereas NoSQL is in pre-production versions with many key features are yet to be implemented, Support, Administration and NoSQL database is still in learning mode. There is three category of NoSQL data model, a) Key- Value stores; in this a value corresponds to a key and data are stored as a key-value pairs. E.g.Redis b) Column-Oriented stores in this database contain one extendable column of directly related data and use a table as the data model but do not support table relationship. E.g. Cassandra, HBase c) Document oriented stores in this data are stored and organized as a collection of document, but the value of a document database is stored in JSON or XML format. E.g. MongoDB, CouchDB. MongoDB is a document-oriented database developed by 10gen. It manages a collection of JSON documents. With the rapid development of the social web as well as cloud computing, the traditional database cannot manage with the basic demands of availability, scalability, storage of enormous data and fast data backup and recovery. Currently, NoSQL database is used by the developer for storing a large amount of database. The most NoSQL systems occupy a distributed architecture, with the data detained in a redundant manner on several servers, and partitioning scheme believes on consistent hashing to distribute the load across multiple storage hosts.

Unstructured data is a general label for recitation data that is not enclosed in a database or some other type of data structure. Unstructured data can be textual or non-textual. Textual unstructured data is generated in media like e-mail messages,

power point presentations, word documents, collaboration software and instant messages. Non-textual unstructured data generated in media like JPEG images, MP3 audio files and Flash video files. If left unmanaged, the absolute volume of unstructured data that's generated each year within a venture can be expensive in terms of storage. Unmanaged data can also cause a responsibility if information cannot be located in the event of an observance or proceedings. The information enclosed in unstructured data is not constantly easy to find. It requires that data in both electronic and hard copy documents and other media be scanned so a search application can parse out a concept based on words used in unambiguous context this is called semantic search.

2. OVERVIEW OF MONGODB

2.1 MONGODB

MongoDB is the schemaless document-oriented database. The name Mongo DB comes from the name "humongous". The database is proposed to be scalable and is written in C++. The main reason for moving away from a relational model is to make scaling easier. MongoDB is also schema - free a document key are not predefined or fixed.

2.1.1 Features:

MongoDB support BSON(Binary JSON) data structures to store complex data types and also supports complex query language. It gives the high-speed admittance to store the mass data. It should stores and distribute large binary files like images and videos and instead of stored procedures, developers can store and use JavaScript functions and values on the server side. Then it supports an easy-to-use protocol for storing large files and it gives a fast serial performance for single clients. And the MongoDB uses memory mapped files system for the faster performance.

2.2.2 Data Design:

MongoDB database holds a set of collections and a collection has no pre-defined schema like tables, and stores data as documents. BSON (binary JSON) are used to store documents. A document is a set of fields and can be thought of as a row in a collection. It can hold complex structures such as lists and a document. Each document have the ID field, that is used as a primary key and each collection can hold any kind of document, but queries and indexes can only be made against one collection. MongoDB has supported the indexing over embedded objects and arrays thus have a special feature for arrays called "multi-keys". That feature is allows using the array as an index, that can be used to search documents by their linked tags. Figure1 shows the structure of MongoDB and completely explaining the MongoDB how it is working and gives the full architecture of the document – oriented data store.

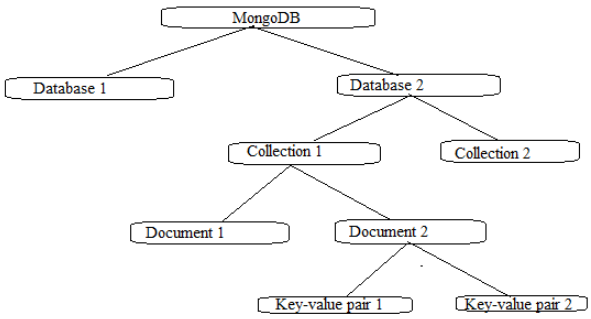

Figure 1: Structure of MongoDB

2.2.3 Query Language:

MongoDB has its own query language named Mongo Query Language and to retrieve certain documents from a database collection, thus the query document is created containing the fields that the preferred documents should match. For example,

2.2.4 Application:

MongoDB is using the RESTful (Representational State Transfer) Application protocol interface (API). It is the architecture style for designing networked applications. It is contains the stateless, client-server, cacheable communications protocol (e.g., the HTTP protocol). A rESTful application using the HTTP requests to post read data and deletes the data.

```
### Insert Command ###
db.users.insert ({ user id:"abc123", age: 55, status:"A"})

### Drop Command ###
db.users.drop ()

### Select Command ###
db.users.find ({ status:"A", age: 55})

### Delete Command ###
db.users.remove ({ status:"A"})
```

2.2.5 Architecture:

MongoDB cluster is built up using three main components namely Shard nodes, Configuration servers and Routing services or mongos as shown in Figure 2. Shard nodes: A MongoDB cluster it should containing one or more shards, and the each shard node is dependable for storing the real data into the database. Each of the shard is consists of either one node or a replicated node which just hold data for that shard. Read and write queries are a retreat to the appropriate shards. A replicated node contains one or more servers, where one of the server is acting as a primary server and others servers secondary servers. If the primary server will fail one of the secondary servers automatically takes over as primary. All

write and reliable reads go to the primary server and all eventually consistent reads are distributed among all the secondary servers.

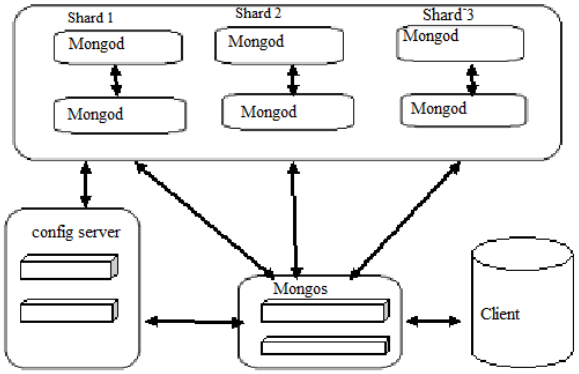

Figure 2: MongoDB Architecture

3. RELATED WORK AND ARCHITECTURE OF THE UNSTRUCTURED DATA STORAGE

3.1 Related Work

MongoDB is providing the more and more confidence to do the work because that database it is containing the lots of methods to store the data's into the database eg. GridFS technology, NodeJS and Replica set technology etc. In the previous work they are trying to store a large amount of data into the MongoDB database but they had met some problem and they did not successfully store the data into the MongoDB database. The previous work they had chosen the clustering technology to store the data into the database but they successfully did the clustering but the clustered node storage is not supported by the MongoDB and its supporting clustering after the data storage inside into the database. In this paper I have created the one new temporary data storage that is called as the store data storage that is the temporary storage only. Consisting hashing algorithm is using this project for the storage of the unstructured data into the MongoDB database.

Thus the consisting hashing algorithm is very useful to store the large amount of unstructured data into the MongoDB database. The Consistent hashing aim is to provide a uniform data sharding around the cluster with the least data lose. And the pseudo code is given in the fig.4.

3.2 Architecture Diagram

The architecture diagram is clearly explaining about the unstructured data storage into the MongoDB database. In the first step we will extract the unstructured data from the data source. After collecting the unstructured data we are going to do the categorization process in the second step. In the categorization process we are categorizing the unstructured data, in this step what I am going to do means splitting the unstructured data with the category of files means if .pdf, .mp3, .mp4 etc files are splited by the category using the parsers. Then the third step after the categorization is store the large amount of the unstructured data into the temporary data storage that the temporary data storage is called as MyStore data store. And then finally we will store the unstructured data from the MyStore data storage. Figure 3 explains the clear process of the unstructured data into the MongoDB database.

The main idea is to hash both data ids and cache-machines to a numeric range using the same hash-function. E.g. in Java a primitive type int has a number range of values between -231 to 231-1. Assume the interval is [0, 231-1] for simplicity (java primitives cannot be unsigned). Now let's join starting and ending points together of this interval to create a ring so the values wrap around. We do not have 231 -1 available servers, the large size of the ring being merely intended to avoid collisions. As a hash function a good choice is either be e.g. MD5 or SHA-1 algorithm. As a machine's number we can take its IP-address and apply that hash function to it. By taking from the result the first 8 bytes we can map it to our ring [0,231-1]. Both the nodes and the keys are mapped to the same range on the ring. Ok, now we need to understand how to identify on this ring which data ids belong to which server's IP. It's really simple, we just move clockwise starting from zero (starting point on the ring) following the main rule of consistent hashing: If IP-n1 and IP-n2 are 2 adjacent nodes on the ring all data ids on the ring between them belong to IP-n1.

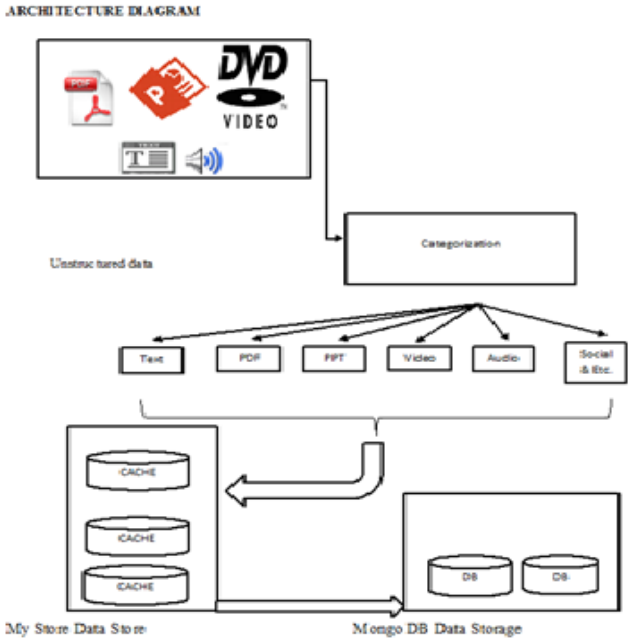

Figure: 3 Architecture Diagram

That's it. Using consistent hashing we do not need to rehash the whole data set. Instead, the new server takes place at a position determined by the hash value on the ring, and part of the objects stored on its successor must be moved. The reorganization is local, as all the other nodes remain unaffected. if you add a machine to the cluster, only the data that needs to live on that machine is moved there; all the other data stays where it is. Because the hash function remains unaffected, the scheme maintains its consistency over the successive evolutions of the network configuration. Like naive hashing, consistent hashing spreads the distributed dictionary almost evenly across the cluster. One point to mention is what happens when a node goes down due to some disaster. In this case consistent hashing alone doesn't meet our requirements of reliability due to loss of data. Therefore there should definitely be replication and high availability which is feasible and out of scope of this introduction. You may want to find good references at the end of this article to find out more.

```
class HashRing { public: typedef std::map<size_t, Node>NodeMap;
HashRing(unsigned int replicas) : replicas_(replicas),
hash_(HASH_NAMESPACE::hash<const char*>()) { 41}
HashRing(unsigned int replicas, const Hash& hash) : replicas_(replicas),
hash_(hash) { } size_t AddNode(const Node& node);
 void RemoveNode(const Node& node);
const Node&GetNode(const Data& data)
const; private: NodeMap ring_;
const unsigned int replicas_; Hash hash_; };
template <class Node, class Data,
class Hash> size_t HashRing<Node, Data,
Hash>::AddNode(const Node& node) {
size_t hash; std::string nodestr = Stringify(node);
for (unsigned int r = 0; r < replicas_; r++) {
 hash = hash_((nodestr + Stringify(r)).c_str()); ring_[hash] = node; }
 return hash; } template <class Node, class Data, class Hash> void HashRing<Node, Data,
Hash>::RemoveNode(const Node& node) { std::string nodestr = Stringify(node);
for (unsigned int r = 0; r < replicas_; r++) { size_t hash = hash_((nodestr + Stringify(r)).c_str());
ring_.erase(hash); 42

} } template <class Node, class Data,
class Hash> const Node&HashRing<Node, Data, Hash>::GetNode(const Data& data)
const { if (ring_.empty()) {
throw EmptyRingException(); }
size_t hash = hash_(Stringify(data).c_str());
typename NodeMap::const_iterator it;
// Look for the first node >= hash it = ring_.lower_bound(hash);
if (it == ring_.end()) {
// Wrapped around; get the first node it = ring_.begin(); } return it->second;
}
```

Figure 4: Consistent hashing

Consistent hashing allows you to scale up and down easier, and makes ensuring availability easier. Easier ways to replicate data allows for better availability and fault-tolerance. Easier ways to reshuffle data when nodes come and go means simpler ways to scale up and down.It's an ingenious invention, one that has had a great impact. Look at the likes of Memcached, Amazon's Dynamo, Cassandra, or Riak. They all adopted consistent hashing in one way or the other to ensure scalability and availability. Want to know more about distributed databases in general and Riak in particular? You'll like the Riak Handbook, a hands-on guide full of practical examples and advice on how to use Riak to ensure scalability and availability for your data. In the next installment we're looking at the consequences and implications of losing key ordering in a Riak cluster. Virtual nodes minimize changes to a node's assigned range by a number of smaller ranges to a single node. In other words, amount of data to be moved from one physical node to others is minimized. Let's split a real node into a number of virtual nodes. The idea is to build equally-sized subintervals (partitions) for each real server on the ring by dividing the hash-space into P evenly sized partitions, and assign P/N partitions per host. When a node joins/leaves all data from partitions of all real servers are uniformly get assigned to a new server and given back to remaining ones respectively. The number of virtual nodes is picked once during building of the ring and never changes over the lifetime of the cluster. This ensures that each node picks equal size of data from the full data set, that is P/N and thus our data now are distributed more uniformly. This enforces that the number of virtual nodes must be much higher than the number of real ones.

4. CONCLUSION

In this work, I successfully stored the large amount of unstructured data into the MongoDB. MongoDB improve the high availability, and high scalability to the data storage. Consistent hashing algorithm is mainly used to store the with low data loss. And the consistent hashing algorithm is used to do the clustering inside into the MongoDB database. Using the GirdFS technology the data's should be separated into the shard that could be splitted into the many shards. Using the consistent hashing algorithm the data's are clustered so the retrieving will be very fast.. My Store dramatically improves the ability of handling failures and the availability, scalability of the system. I tried to store the data into the cloud using the MongoDB storage but due to some of the problem I could not store the unstructured data into the cloud environment.

5. FUTURE WORK

Main work to do here after creates the cloud environment using the MongoDB ops-manager I will store the large amount of the unstructured data into the cloud environment.

REFERENCES

[1] Gartner Inc. http://www.gartner.com/
[2] Abadi J (2009) Data management in the Cloud: limitations andopportunities. IEEE Data Eng Bull32(1):3–12
[3] Lakshman A, Malik P (2009) Cassandra—a decentralized structured storage system. In: Proceedings of the 3rd ACM SIGOPS international

workshop on large scale distributed systems and middleware(LADIS'09). ACM, New York, pp 35–40
[4] Clarence J. M. Tauro . A Comparative Analysis of Different NoSQL Databases on Data Model,Query Model and Replication Model. ACM SIGOPS Operating Systems Review archive Volume 44 Issue 2, April 2010 Pages 35-40
[5] Cassandra - A Decentralized Structured Storage System In: Proceedings of twenty-first ACM SIGOPS symposium on operating systems principles. ACM, NewYork, pp 205–220
[6] Abhinav Tiwari (2009) Data management in the Cloud: limitations andopportunities. IEEE Data Eng Bull32(1):3–12. ACMTransComputSyst 26(2):1–26
[7] Lakshman A, Malik P (2009) Cassandra: structured storage system on a p2p network. In: Proceedings of the 28th ACM symposium on Principles of distributed computing. ACM, New York, pp 5–5
[8] Ford D, Labelle F, Popovici F (2010) Availability in globally distributed storage systems. In: Proceedings of USENIX conference on operating system design and imlementation (OSDI'10). USENIX,Berkeley, pp 1–7
[9] Banker K (2011) MongoDB in action. Manning Press, USA
[10] Pritchett D (2008) BASE: an acid alternative. ACM Queue 6(3):48–55
[11] Jim G (1981) The transaction concept: virtues and limitations. In: Proceedings of the 7th international conference on very large databases (VLDB). IEEE, New York, pp 144–15412. Vogels W (2009) Eventually consistent. Commun ACM 52(1):40–44