Analysis of Data Replication Mechanism in NoSQL Database MongoDB

Yunhua Gu, Xing Wang, Shu Shen, Sai Ji, Jin Wang
Jiangsu Engineering Center of Network Monitoring, School of Computer & Software, Nanjing University of Information Science & Technology, Nanjing, 210044, China

***Abstract*--The replication mechanism of NoSQL database MongoDB includes Master/Slave structure and Replica Set. Write operation implement on Master, Slaves will send the synchronize data command asynchronously to Master to update its data. Read operation just implement on Master to provide the strong consistency, while read operation implement on Slave to provide the eventual consistency. Replica set is a group of servers which run Mongod and store the copy of the same data with automatic failover and automatic recovery of member nodes.**

I. INTRODUCTION

In the Web 2.0 era, the Pan-data becomes the main content of data storage in Internet applications. Pan-data includes documents, e-mail, video, pictures and other types of data, which unlike the strict structured data of relational data, is usually not based on row and column format, and has huge data amount which requires a lot of storage space[1]. Many Internet applications such as search engines, their requirements of consistency and integrity can be reduced, but need more availability, scalability and performance[2]. The traditional relational databases are confronted with great difficulties. Even using the hierarchical caching, Web services, load balance and other optimization methods, it is hard to meet the requirements.

II. DISTIBUTED STORAGE

A. Sharding

Sharding occurs on a per-collection basis. One or more fields in collection are selected as shard key, in the order of which the data is sorted and divided into multiple chunks in default maximum size of 200M. When the size of chunk is over the maximum by inserting the data, the system automatically divides it into two chunks and stores in different shard. The data stores to shard in chunks, when the load needs balancing, data moving in chunks[3]. The config-servers store the cluster's metadata, which includes basic information on each shard server and the chunks contained therein. The chunk information involves the collection name which chunk belonged to, minimum of shard key, maximum of shard key and the shard chunk stored in. A two-phase commit is used to ensure the consistency of the configuration data among the config servers.

The Mongos process can be thought of a manager as a routing and coordination process that makes the various components of the cluster look like a single system. It connects each shard in the cluster with ping command every 30 seconds to check them alive or not and read or update the config-information on config-server. All commands about show, add, delete shard and load balancing run on it, and it is Mongos that modifies the corresponding metadata on config-servers.

B. Config server

When the client application issues the commands, Mongos connects to the config-server, decides the target shard by comparing the shard key and submits the data to the management program Mongod on it. The Mongod will implement the actual operation on shard and return the results to Mongos, which merges all the results and return to client and updates the metadata on config-server if necessary.

The config-servers store the cluster's metadata, which includes basic information on each shard server and the chunks contained therein. The chunk information involves the collection name which chunk belonged to, minimum of shard key, maximum of shard key and the shard chunk stored in. A two-phase commit is used to ensure the consistency of the configuration data among the config -servers.

The Mongos process can be thought of a manager as a routing and coordination process that makes the various components of the cluster look like a single system.. It connects each shard in the cluster with ping command every 30 seconds to check them alive or not and read or update the config-information on config-server. All commands about show, add, delete shard and load balancing run on it, and it is Mongos that modifies the corresponding metadata on config servers[4].

III. DATA REPLICATION MECHANISM

A. Master/Slave Model

MongoDB uses Master/Slave and Replica Set as its replication mechanism. Servers in a Mater/Slave replication have different roles, at the same time there is only one Master server, the others are Slave, as shown in Fig.1. Write operation implement on Master, Slaves will send the synchronize data command asynchronously to Master to update its data. Read operation just implement on Master to provide the strong consistency, while read operation implement on Slave to provide the eventual consistency. The Master/Slave replication don't provide the automatic failover, if the Master is down, Slave must shut down and restart to change its role to Master.

B. Replica set

Replica set is a group of servers which run Mongod and

978-1-4799-8745-0/15/$31.00©2015 IEEE

store the copy of the same data with automatic failover and automatic recovery of member nodes. Fig.2. displays the model of replica set.

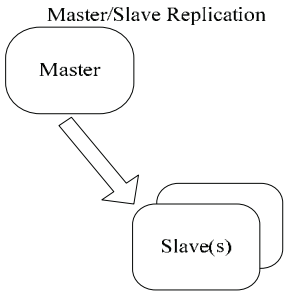

Fig. 1. Master/Slave Model.

There are three server states in replica set: primary, secondary and recovering. Only one server is primary in replica set at a given point in time. Primary server provides write and strong consistency read, while secondary server provides eventual consistency read. Recovering server is a server getting back in sync before entering secondary mode. A write is only truly committed once it has replicated to a majority of members of the set. Writes which are committed at the primary of the set may be visible before the true cluster-wide commit has occurred. Thus we have "READ UNCOMMITTED" read semantics.

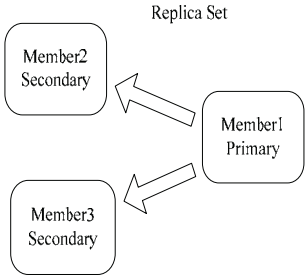

Fig. 2. Replica Set Model.

An election of primary will be held when a replica set initializes. First check the priority of each node with the highest is the primary, otherwise the consensus protocol will compare the MaxAppliedOpTime on each node, the latest is the primary. After the primary is elected, suppose it has the latest data in replica set, all secondary will synchronize data with it.

Any secondary node which finds the primary is down can initiate the election to elect a new primary, that implements the automatic failover. This process takes about 20-30s. During this time there is no primary node in replica set, so write operation and strong consistency read can not be implemented, while eventual consistency can always be implemented. If the secondary node in replica set is down, wait until it recovers and adds to replica set, the following same as add a new node to replica set: inform the primary then take the full resync operation to copy the data, and change the state to secondary, in this process, the state and performance of replica set is not affected.

In single point storage the driver will typically take a comma separated list of host[:port] names. This is a seed host list, which need not be every member of the set. These seed host will inform the driver about the other members in replica set. The driver will look for the primary from the seeds and submit the client application's issues to it. With sharding, the client connects to Mongos process, which will automatically find the right member of the set.

MongoDB implements complicated aggregate operations by some utility functions and map/reduce.

It includes utility functions which provide server-side count, distinct and group by operations. Count() returns the number of objects in a collection or matching a query. The distinct command returns a list of distinct values for given key across a collection. Group() returns an array of grouped items, similar to SQL's group by.

Map/reduce in MongoDB is useful for batch processing of data and aggregation operations. It is similar in spirit to using something like Hadoop with all input coming from a collection and output going to a collection. Often in a situation where you would have used GROUP BY in SQL, map/reduce is the right tool in MongoDB. Map/reduce implements aggregation operations about statistics, classification, consolidation and so on of data set, such as find the average value of a certain group.

IV. CONCLUSION

NoSQL is a revolution in the field of database, which changes people's inherent concept of relational data. It is a new thought to database field. MongoDB is one of the greatest NoSQL database, which is document-oriented and schema-free, and supports complicated data structure and large amount of data storage, with rich query and high scalability and performance. However it has no transaction consistency and is lack of constraint of paradigm. These features make MongoDB can provide great service in some certain application scenarios and in others it can not meet their needs. So we have to deep analyze the requirements of applications and the features of MongoDB to select the database reasonably. The features of NoSQL database and traditional relational database are complementary. Depending on the applications characteristics to select an appropriate database can achieve better results.

REFERENCES

[1] Tudorica B, Bucur C. A comparison between several NoSQL databases with comments and notes[C]. Roedunet 2011 10th International Conference on:1-5.

[2] Coulter T. Costing: non-traditional data stores versus traditional DBMS technologies[C]. Technology Management in the Energy Smart World (PICMET), 2011 IEEE Proceeding of PICMET'11: July 31-Aug 4. 2011:1-15.

[3] MongoDB Introduction. http://www.mongodb.org/ display/DOCS/Introduction, 2014.11.

[4] Kristina C, Michael D. MongoDB：the definitive guide[M]. O' Reilly Media. 2010.9.