Research on The Improvement of MongoDB Auto-Sharding in Cloud Environment

Yimeng Liu, Yizhi Wang,Yi Jin
School of Computer and Information Technology
Beijing Jiaotong University
Beijing , China

Abstract—**With the rapid development of the Internet Web 2.0 technology,the demands of large-scale distributed service and storage in cloud computing have brought great challenges to traditional relational database.NoSQL database which breaks the shackles of RDBMS is becoming the focus of attention.In this paper, the principles and implementation mechanisms of Auto-Sharding in MongoDB database are firstly presented, then an improved algorithm based on the frequency of data operation is proposed in order to solve the problem of uneven distribution of data in auto-sharding.The improved balancing strategy can effectively balance the data among shards,and improve the cluster's concurrent reading and writing performance.**

key words—**NoSQL, MongoDB, Auto-Sharding, balance strategy**

I. INTRODUCTION

In recent years,with the rapid growth of the amount of data and the development of Internet web 2.0 technology, how to efficiently store, process and extract large amounts of data becomes an urgent problem, Cloud computing emerged in this context. Cloud computing is the delivery of computing as a service rather than a product, whereby shared resources, software, and information are provided to computers and other devices as a metered service over a network (typically the Internet)[1]. Many universities, vendors and government organisations are investing in research around the topic of cloud computing, for example: Amazon launched Simple Storage Service (S3) and Elastic Compute Cloud(EC2), Google proposed GFS, BigTable and MapReduce, which have all been successfully used in production environment. Distributed File System can organize the huge amounts of data in cloud computing, and also be able to efficiently read the cloud data, but we still need specialized data management tools for better management of structured data.

Cloud Data Management is a new data management concept with the development of cloud computing, it must be able to efficiently manage of large data sets in the cloud, and quickly locate specific data in massive data sets, which makes the Cloud Data Management with the following common characteristics: (1) high concurrent read and write performance, (2) efficiently store and access huge amounts of data, and (3) high scalability and high availability requirements of the database. In the face of these demands, the traditional relational data management system(RDBMS) has encountered an insurmountable obstacle. Therefore, NoSQL database systems rose alongside major internet companies, such as Google, Amazon, Twitter, and Facebook which had significantly different challenges in dealing with data that the RDBMS solutions could not cope with. These companies realized that performance and real time nature was more important than consistency, which traditional relational databases were spending a high amount of processing time to achieve. As such, NoSQL databases are often highly optimized for retrieve and append operations and often offer little functionality beyond record storage. The reduced run time flexibility compared to RDBMS systems is compensated by significant gains in scalability and performance. Often, NoSQL databases are categorized according to the way they store the data and fall under categories such as key-value stores(e.g. Dynamo[2]), BigTable implementations[3] and document store databases(e.g. MongoDB[4]). However, due to the immature technology of cloud data mangement, there are still many issues need to be addressed in actual production environment.

This paper discusses the design principles and implementation mechanism of MongoDB database, and focuses on the principle of Auto-Sharding. The goal of Auto-Sharding is to split data up across machines and rebalance automatically making it possible to store more data and handle more load without requiring large or powerful machines. But the balancer algorithm isn't so intelligent that data isn't evenly distributed among servers. In order to solve this problem, an improved FODO (i.e. frequency of data operation)algorithm is proposed. The FODO algorithm is based on the frequency of data operation and taking the load of server into consideration. The data balancing strategy based on FODO algorithm can effectively balance the data among servers, and improve the cluster's concurrent reading and writing performance.

II. AUTO-SHARDING IN MONGODB

A. Brief introduction of MongoDB

MongoDB (from "humongous") is an open source document-oriented NoSQL database system written in the C++ programming language. It manages collections of BSON documents. Development of MongoDB began in October 2007 by 10gen. MongoDB features[5]: Document-oriented storage, JSON-style documents with dynamic schemas offer simplicity and power; full index support; replication and high availability; Auto-Sharding, scale horizontally without compromising functionality; Map/Reduce, flexible aggregation and data processing; GridFS, store files of any size without complicating your stack.

At the heart of MongoDB is the concept of a document which is the basic unit of data for MongoDB, roughly

equivalent to a row in a RDBMS. Similary, a collection can be thought of as the schema-free equivalent of a table. A single instance of MongoDB can host multiple independent databases, each of which can have its own collections and permissions.

B. *Auto-Sharding architecture*

Sharding refers to the process of splitting data up and storing different portions of the data on different machines. By splitting data up across machines, it becomes possible to store more data and handle more load without requiring large or powerful machines[6]. MongoDB supports Auto-Sharding, which eliminates some of the manual sharding, the cluster can split up data and rebalance automatically. MongoDB sharding provides: (1) automatic balancing for changes in load and data distribution, (2) easy addition of new machines without down time, (3) no single points of failure, and (4)Automatic failover.

The basic concept behind MongoDB's sharding is to break up collections into smaller chunks. These chunks can be distributed across shards so that each shard is responsible for a subset of the total data set. To partition a collection, MongoDB specify a shard key pattern which names one or more fileds to define the key upon which we distribute data. Because of shard key, chunks can be described as a triple of collection, minkey and maxkey. Chunks grow to a maximum size, uaually 200MB, once a chunk has reached that approximate size, the chunk splits into two new chunks. The architecture of Auto-Sharding is displayed in Fig. 1.

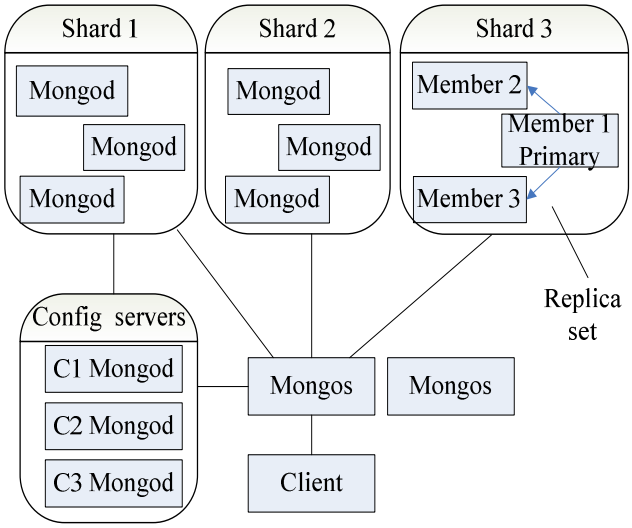

Figure 1. Architecture of Auto-Sharding .

A MongoDB shard cluster consists of two or more shards, one or more config servers, and any number of routing processes. Each of the components is described below.

1) Shard: Each shard consists of one or more servers and stores data using mongod processes. In production situation, each shard will consists of a replica set to ensure availability and automated failover.

2) Config server: It stores the cluster's metadata which includes basic information on each shard server and the chunks contained therein.

3) Mongos(Routing Processes): It can be thought of as a routing and coordination process. When receiving requests from client, the mongos routes the request to the appropriate server and merges any results to be sent back to the client.

C. *The balancing strategy of Auto-Sharding*

MongoDB uses balancer to keep chunks evenly across all servers of the cluster. The unit of transfer is a chunk, and balancer waits for a threshold of uneven chunks counts to occur. In the field, having a difference of 8 chunks between the least and most loaded shards showed to be a good heuristic. Once the threshold is reached, balancer will redistribute chunks, until that difference in chunks beween any two shards is down to 2 chunks.

In order to reduce the amount of data transferred, each shard contains multiple ranges. When a new shard added into the cluster or some of the shards contain too much data to reach the threshold, the balancer will skim chunks off of the top of the most-populous shard and move these chunks to the least-populous shard, allowing the data evenly distributed in the cluster by moving the bare minimum.

The goal of the balancer is not only to keep the data evenly distributed but also to minimize the amount of data transferred. The balancer's algorithm isn't terribly intelligent. It moves chunks based on the overall size of the shard[7]. The migration chunks are just the ones located on the top of each shard, but the operation of data is not taking into consideration. The data transferred between shards may not often be used, it will makes the system load can not achieve an effective balance. It is necessary to improve the balancing strategy of Auto-Sharding in MongoDB.

III. ALGORITHM BASED ON THE FREQUENCY OF DATA OPERATION

A. *Basic idea of FODO algorithm*

In order to solve the problem of data's uneven distribution in auto-sharding, an improved algorithm (FODO algorithm) based on the frequency of data operation is proposed.

In MongoDB, the unit of transfer is a chunk. There are n chunks in a shard, the ith chunk is represented as Ci and its frequency of data operation value is F_DOi. In MongoDB database, the main operations of data are insert, find, update and delete. As the data in the cluster rarely be frequently deleted, the manipulations which affect system performance mainly concentrated in the first three operations. We use Ii, Fi and Ui to represent the number of these three kinds of operations to a chunk. So the value of F_DOi in normalized form is (1)

$$F\_DO_i = \frac{I_i}{\sum_{i=1}^{n} I_i} + \frac{F_i}{\sum_{i=1}^{n} F_i} + \frac{U_i}{\sum_{i=1}^{n} U_i} \quad (1)$$

However, the insert, find and update operations have different impacts on the cluster load and should not be treated equally. In Auto-Sharding environment, it's not necessary to make a physical connection when query record from database because cashes have already kept the fresh data. Moreover, each shard will consists of a replica set in production situation. The secondary nodes in a replica set can read the data written

from the primary node, so it can lighten the querying load on the primary node to a certain extent. Contrarily, the insert and update operations result directly to database every time when they commit a transaction, which occupy a bigger part of the whole cluster workload. In particular, insert data will cause the number of data in each shard is different. Once the threshold is reached, balancer will balance data automatically, which will greatly consume system resources. So FODO algorithm adds a parameter inc (always >1) before inserting part in the equation, called insert coefficient, to put more weight on insert operation. Here is the final FODO value definition in (2)

$$F\_DO_i = inc \times \frac{I_i}{\sum_{i=1}^{n} I_i} + \frac{F_i}{\sum_{i=1}^{n} F_i} + \frac{U_i}{\sum_{i=1}^{n} U_i} \qquad (2)$$

The F_DO value of each shard is the sum of its containing chunk's F_DOi value, that is (3)

$$s(F\_DO) = \sum_{i=1}^{n} c_i(F\_DO_i) \qquad (3)$$

We need to modify data structures recording the chunk's information, adding I, F, U and F_DOi four variables to store insert, find, update and F_DOi values. Each operation to the data must be recorded in the four variable of the corresponding chunk.

B. Balancing strategy of FODO algorithm

F_DOi value indicates the frequency of data operations in each chunk. If a chunk's F_DOi value is high, it means data in this chunk is frequently used. The threshold of data redistribution is still the difference in the number of chunks in each shard, but the migration chunks are selected based on its F_DOi value. There are trade-offs between overall size of shard and operating frequency of data in this balancing strategy. The process of data balance has three steps.

1) The threshold of data migration: Calculate the number of chunks in each shard. If the difference is greater than 8, then begin to balance the data until the difference is less than 2. Chunks will be migrated from the most-populous shard called from-shard to the least-populous shard called to-shard.

2) Choose the migration chunks: Calculate the difference of F_DO value between from-shard and to-shard. If f(F_DO)>t(F_DO), then choose the chunk with the maximum F_DOi value in from-shard. Otherwise, choose the chunk with the minimum F_DOi value in from-shard.

3) Migrate chunks: Migrate the selected chunk in step 2) to the to-shard, and recalculate the F_DOi value of each chunks both in from-shard and to-shard.Repeat step 1).

The flowchart of data balance process is displayed in Fig. 2.

Moreover, we should determine the value of inc. The role of inc is to put more weight on insert operation, so the value should be greater than 1. The specific value of inc relies on server-related parameters and system load, we can setthe initial value and then adjust it according to the actual needs of business.

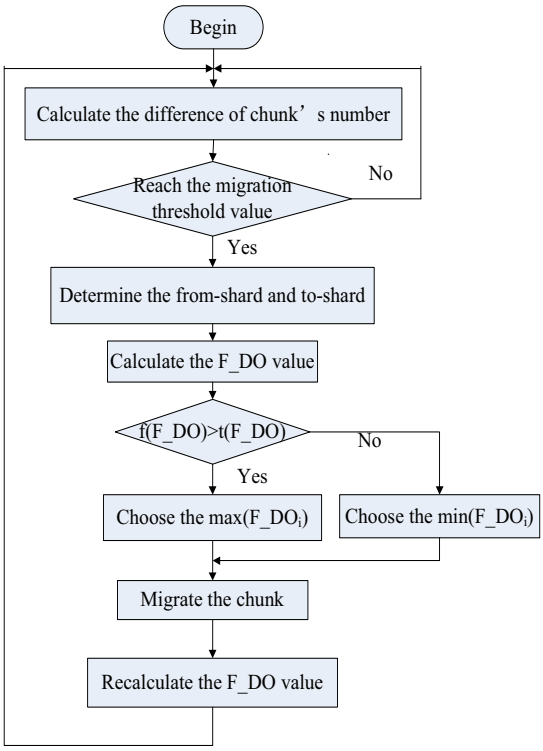

Figure 2. The flowchart of data balance process

IV. EVALUATION OF PERFORMANCE

The test environment based on the MongoDB Auto-Sharding cluster, it contains 10 virtual machines, each of which has the same hardware configuration: 8GB of memory and CPU speed is 2.4G HZ. Each virtual machine has a Linux RedHat operating system, and be connected by the 100Mbps LAN. The MongoDB version is 1.8.2, and the Auto-Sharding cluster has three shards, each of which consists of a replica set. Each replica set contains three nodes: one primary node and two secondary nodes. We should run the rs.slaveOk() command on the secondary nodes in order to query data from them.

We realize the FODO algorithm in MongoDB, and compare the two algorithm by testing the concurrent read and write performance of the cluster. The data set used in test is a simple line data, including four fields of int, long, string and double. In addition, the value of inc parameter in FODO algorithm is 1.5.

In order to see the effect of the FODO algorithm, we only add two shards, insert 1000000 records and use randomly generated id to do find and update operations. Then we add the third shard to the cluster. The purpose of this action is to ensure the cluster has data at the beginning of the test and the value of F_DO is not zero.

Firstly, we test the concurrent writing performance of the cluster. We insert 10000000 records into the cluster and remain the overall amount of records be same under different number of concurrent. The concurrent writing performance of this two algorithm is shown in the Fig. 3.

Remain the concurrent number and records be unchanged, and test the concurrent reading performance of the cluster. The concurrent reading performance of this two algorithms is shown in the Fig. 4.

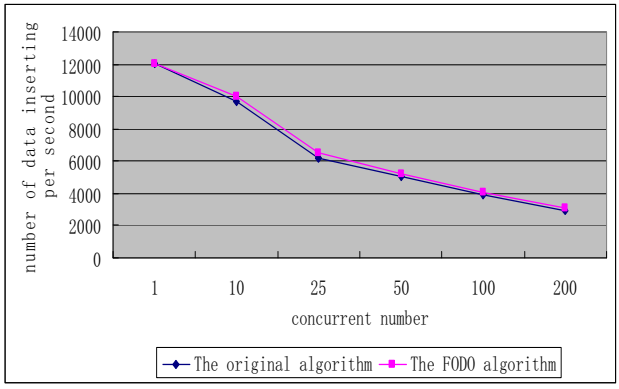

Figure 3. Concurrent writing performance

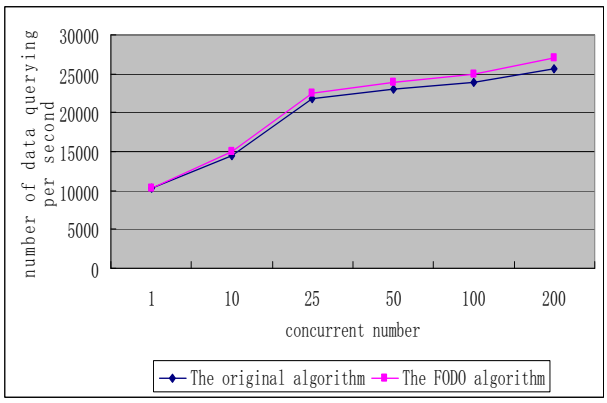

Figure 4. Concurrent writing performance

V. CONCLUSION

This paper analysises the principle of the MongoDB Auto-Sharding. For the problem of uneven distribution of data among shards, we introduce an improved balancing algorithm-algorithm based on the frequency of data operation(FODM). A data balancing strategy based on FODO algorithm is proposed and its effectiveness is verified by experiments. The concurrent writing and reading performance of the Auto-Sharding cluster is significantly improved. And more aspects of the FODO algorithm could be explored, such as determination of inc value.

REFERENCES

[1] Peter Mell, Tim Grance. "The NIST Definition of Cloud Computing". National Institute of Science and Technology. Retrieved 24 July 2011.

[2] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, et al. Dynamo:Amaon's Highly Available Key-value Store. SOSP'07. Stevenson, washington, USA:2007.

[3] Fay Chang, Jeffery Dean, Sanjay Ghemawat, et al. Bigtable: A Distributed Storage System for Structured Data. 7th Symposium on Operating System Design and Implementation. Seattle, WA, USA: 2006.

[4] 10gen. MongoDB. http://www.mongodb.org, 2011-07-15.

[5] MongoDB. features.http://www.mongodb.org/.

[6] Kristina Chodorow, Michael Dirolf. "MongoDB: The Definitive Guide". O'Reilly Media,September 2010. p135

[7] Kristina Chodorow. "scaling MongoDB". O'Reilly Media,January 2011. p13.