Analysis of range-based key properties for sharded cluster of MongoDB

Pakorn Kookarinrat
Computer Science Department
Thammasat University
Pathumthani, Thailand
soulski@gmail.com

Yaowadee Temtanapat
Computer Science Department
Thammasat University
Pathumthani, Thailand
yao@cs.tu.ac.th

***Abstract*—MongoDB is one of the most popular NoSQL database nowadays. It is an open-source document-oriented database with flexible schema. To increase performances, it can easily scale both vertically and horizontally. For horizontal scaling, MongoDB uses auto-sharding technique to divide data and distribute it over multiple machines. However, in this technique, a DB administrator must choose a shard key for MongoDB to split its collection. Selecting a right key could improve the performance and capability of a database. Contrarily, a wrong key takes down performances and, in some serious case, could lead to a system halt. Therefore, it is important to choose a correct key. MongoDB has a general suggestion on the properties of its ideal shard key. For instance, a good shard key should have high degree randomness for write scaling and should contain high locality for range-query reading. In order to understand the impact of these properties on a shard key, this paper has analyzed and evaluated such suggested properties. We discussed how the variation of a shard key's choices could impact the DB performance and it gives the base to help a MongoDB admin to understand and could select a good shard key for his/her system.**

Keywords—shard key selection; sharding; horizontal scaling; MongoDB

I. INTRODUCTION

A NoSQL database is considered as a new breed of database systems focusing on scalability and performance. MongoDB is one of the most widely used NoSQL database systems today [11]. To support a large data set with high throughput operations, MongoDB uses sharding, an approach to horizontal scalability. Sharding splits a collection of large data and distribute them to multiple servers. To shard a collection, a DB administrator needs to pick a "shard key", a property chosen to evenly split partitioned data. Selecting a shard key is crucial to its performance. A good key could improve it while a bad key not only can degrade it but could also lead to a system halt like an incident with the FourSquare [3]. Since a shard key is used to break partitions, if data has a shard key with similar value, MongoDB will not be able to divide and distribute. Insertion of data would end up in a specific machine or a small set of machines. This machine would bear most of the loads of the system until it cannot further continue working.

There are two types of a shard key, a range-base and hash-base key. A range-base key divides data according to an interval of key values. A range-base key could gain a good read performance, if only a small group of shards are executed to retrieve data. In contrast, if a query must be independently read on many shards and then results have to be merged, the performance would be degraded. A hash-base key computes a hash value of a selected field in a collection to be its key. A hash-base key could gain a good write performance, if it could permit equality write queries to several machines. The choice of using which type of shard key depends mainly on applications. However, a hash-base key is supported solely on a single field and may be suitable only for some collections. For instance, in an application picking a sequential field to be its shard key, hashing may alleviate insertion hot spot. Since a range-base key could generally apply to a variety of systems, in this paper, we will focus only on a range-base key type.

From the general suggestion in the MongoDB document [3], a good shard key should have three main properties, *uniformity, randomness and locality*. Firstly, uniformity of shard key allows the system to distribute data evenly to multiple servers and make it easy to divide the content among shards. Secondly, randomness of key value could easily spread out write operations to multiple machines. Lastly, a good shard key should provide locality for reading query to avoid read operations to several machines.

In order to understand the effects of these three properties on shard keys to MongoDB's performance, we design experiments to realize their impacts, then analyze and evaluate the results of our experiments. Our contribution would give the base and concrete guideline to help MongoDB admins to understand and be able select a good shard key that would suitable to their systems.

The rest of the paper is organized as follows. Section II discusses background materials and related works. Section III presents our experimental design and how to set up the experiment system. Section IV reports experimental results. Section V concludes this paper.

978-1-4673-8611-1/15/$31.00 ©2015 IEEE

II. Background and Related Works

A. MongoDB

MongoDB, a NoSQL database, is a document-oriented database. It uses a flexible model to provide dynamic schema. Data is stored in BSON format, a binary-encoded serialization of JSON-like documents. The structure of MongoDB is different from traditional relational database systems. Since its main structure is a document base, a collection could be comparable to a table while a document could be equivalent to a record in a relational one.

MongoDB has no columns. Documents are comparable to records in a relational database. Documents consist of (field, value) pairs. Fields are strings, and values can be numbers, strings, arrays or objects. An object is itself a set of (field, value) pairs. Thus, its structure allows for unbounded nesting. A query could be requested by sending JSON-like format to compare with the collection in the system. MongoDB provides both shell and a variety of drivers to support most popular programming languages and development environments.

MongoDB uses sharding to improve a horizontal scale-up. To enable sharding, we need to tell MongoDB which data and collection that we would like to do sharding and which attribute in the document will be used as a shard key. An attribute(s) serving as a shard key must exist in all documents and will be indexed for later use. There are three components in sharded cluster, namely, shard, config and route (see Fig. 1).

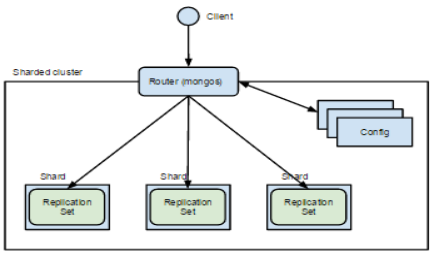

Fig. 1. sharded cluster

- **Shard**: A shard is a component storing split data. Usually, it is constructed from MongoDB database and it could use a replicate set to increase availability. A client, generally, can make a read or write request on a shard via a router. A client could also make a request directly to a shard, however, the client will receive only data held in that shard.
- **Config**: A config is a component that stores the configuration of a shard cluster. It is constructed from MongoDB. A config component will receive read and write a configuration's request from a router.
- **Route:** A router is a connection point of shard clusters. A router is constructed from a program named MongoS. A client will manage and request data of shard cluster through a router. When the router receive a request, it reads its configuration, an information about the shards' locations that hold the requested data. The route then make a request to those shards. After getting results back from shards, the router merges and sends them back to the client.

B. Related Works

Some of previous researches have tried to improve sharding performances of MongoDB by modified its balancer. One of them [1] use algorithm that could balance chunks from a shard having the highest insert, read and update to the one that has the lowest load. In this work, they slightly improve its read and write performance. Another research [2] tried to improve performance of sharding as follows. First, they switched a hot primary node on a replicate set to the cold secondary node. Second, they added more nodes to help alleviate the workloads. Last, they modifies a balancer that balances chunks in a sharding cluster to make all nodes to use resources such as CPUs, memories and network instead of balancing a number of chunks to make all nodes using resource evenly.

All related works try to improve performance of sharding by focused on how to distribute workload among nodes. However, in our work, we look into the source of the problem that an administrator has to make decision on which attribute should be used as a shard key before start working on sharding . Also, the knowledge gains here will help us to monitor the sharding problem before it is too late to fix.

III. Design of Experiments

This work investigates the effects of shard key choices to sharding performance on both read and writes operations. As mentioned earlier, the three main suggested properties of a shard key namely, uniformity, randomness, and locality could affect the sharded cluster on read and write operations. In this section, we explain how we generate our experimental data to evaluate such properties and the set of write-read queries on the data to measure the effects of write and read performances. The system configuration and additional tool are stated at the end of the section.

A. Organizing Test Data and Query Operations

1) Experiments for Writing Performance

Two main properties that could affect write performance are uniformity and randomness. Since horizontal scaling depending on how well the data could be distributed evenly. Uniformity allows even storing while randomness helps to temporally distribute operation to multiple shards. Thus, the shard keys for testing are divided into 3 groups: random, sorted, and combined shard key.

A random shard key aims to perform a uniformity test. A shard key in this category is divided into 4 ranges, 1 to 1,000,000, 1 to 666,666, 1 to 333,333 and 1 to 3. The first three ranges are used to test the effect of range boundary toward write performance. The last group is designed to see the effect of jumbo chunk to the performance and also what would happen if a shard exhausts its resources.

A sorted shard key is picked to test the effect on the sharding write performance when incoming data would arrive in order. Since a sorted shard key would be accumulated in a

single shard before the chunk is big enough to split out. Thus, it could load one machine at all time.

A combined key is a combination of random and sorted fields. We choose this type of key since it is an easy form that an admin trends to use in order to gain uniformity and randomness for a shard key. The key will be compared its performance with the above two groups.

2) Experiments for Reading Performance

MongoDB could gain good read performance if a small group of shards is run to retrieve the data. Read, generally, involves distributing sub-queries to a collection of related servers and then waiting for multiple results to be merged. Thus, the more involved machines, we would expect the poorer read performance. To test sharding read performance on data set up in the previous section, we provide the following criteria of reading, single key target, range key target and index target.

A single key target query is a query requesting for a single value of the shard key. Thus, it is expected that the query would be sent to a single shard.

A range key target query involves a range of documents. We divide this type of query into two subtypes, a small and big range of data. For a small range query, it is involved about 1% range in a chunk. Thus, it is likely to cover only a couple of shards. A big range query covers 3 chunks of data. It is possible that it will request 2-3 shards to answer such query.

Finally, an index target requests data that are indexed but not a shard key. It would look up on the index and possibly scan requests to all shards in order to get an answer.

B. Test Data Collection

To make a consistent testing, we generate data according to the conditions described in Section III.A.1. The schema of the testing data is shown in Table 1.

TABLE I. SCHEMA OF TEST COLLECTION

Name	Description
sort	Sort
m_sort	Sort range 1-333,333
random	Random range 1-1,000,000
s_random	Random range 1-3
m_random	Random range 1-333,333
b_random	Random range 1-666,666

C. System Configuration

We run MongoDB in a simulated environment. The experiments are run in the distributed setting on 6 virtual machines using virtualization tool running a single Intel i7 CPU at 3.6 GHz. It has 16GB main memory and a 225 GB SSD hard disk. The experiment components comprise with 3 shard instances, 1 config instance and 1 router instance.

We setup up 3 level of resource allocation on sharded cluster that hold 30%, 40% and 60% of resources, in order to see the effects of sharding on how it works under resource pressure.

We run a bash script to set up and connect all components to make it easy to construct a new sharding environment.

D. Docker Monitoring Tool

To monitor a docker instance's resource utilization, we use a monitoring docker tool named Docker-stats. However, this tool supports only a console monitor. In order to collect testing results, we write an additional program to read the docker resources in every second. The plugin is written in golang programming language. The results are fed to R for further evaluation. The results and their evaluation are explained in Section IV.

E. Testing Process

After designing and arranging necessary tools for testing, this section explains the process of our testing.

Step of testing:

1. Install a shard key to sharded clusters
2. For each experiment,
   a) Run the bash program to insert each group of testing data
   b) Collect the running time and the resource's usage of each test from the monitoring tool
   c) Run the bash program for retrieving query as plan
   d) Collect the time and the resource's usage of each reading test from monitoring tool, i.e. number of involved shard per query, number of reading index

IV. EXPERIMENTAL RESULTS

We tested all experiments as described in Section III. The results are as follows:

A. Write Performance Results

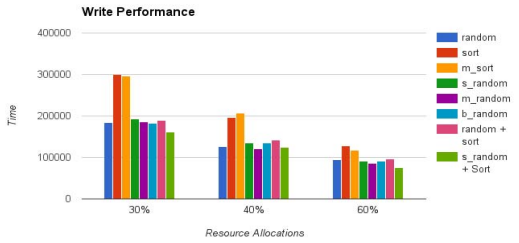

Fig. 2. Write performance of shards on sharded cluster

From fig. 2, we could easily see that a random shard key gives the best performance in writing even in the case that may cause a jumbo chunk in the sharded cluster. Range boundary is not significantly affected the write performance. As shown in Fig 3, all random keys have no significant difference on t-test. For a sorted shard key, the write performance could be degraded on any range of data because data was sent to only one shard instead of any available shards to process it parallelly. The combination key gives a similar result as in the

random key's using in its combination. Finally, resource limiting may degrade sharding performance. However, it decreases evenly on any pattern of shard keys, Therefore we can make assume that on any level of resource, sharding key affect performance pattern will not be change.

	random	sort	m_sort	s_random	m_random	b_random	random + sort	s_random + sort
random		0.002145	0.004333	0.821082	0.592717	0.851094	0.495558	0.335027
sort			0.995251	0.003953	0.000940	0.003543	0.008595	0.000469
m_sort				0.007282	0.002029	0.006636	0.014606	0.001065
s_random					0.319964	0.974756	0.547337	0.229523
m_random						0.324896	0.121420	0.705558
b_random							0.581489	0.233645
random + sort								0.093707
s_random + sort								

Fig. 3. T-test for independent write performance

B. Read Performance Results

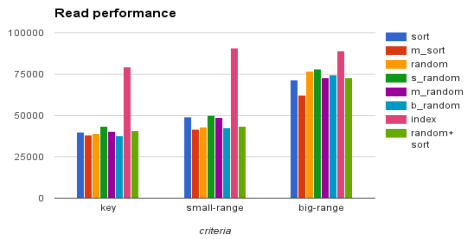

Fig. 4. Read Performance of shard keys on sharded cluster

	sort	m_sort	random	s_random	m_random	b_random	index	random+sort
sort		0.63126	0.96140	0.81937	0.98056	0.89349	0.03122	0.92931
m_sort			0.72018	0.49857	0.62035	0.77940	0.00977	0.71887
random				0.80478	0.94501	0.94007	0.05622	0.97495
s_random					0.83978	0.74027	0.05992	0.76337
m_random						0.87785	0.03547	0.91204
b_random							0.04506	0.96097
index								0.03564
random+sort								

Fig. 5. T-test for independent read performance

	Key			Small range			Big range		
	Number of scan	Number of shards	Execution time	Number of scan	Number of shards	Execution time	Number of scan	Number of shards	Execution time
sort	1	1	0	1	1	0.5	2	2	0
medium sort	1	1	0	1	1	0	1.25	1.25	0
random	1	1	0	1	1	0	1.5	1.5	0.75
small random	1	1	0.5	1	1	0	1.75	1.75	0
medium random	1	1	0	1	1	0.5	2.25	2.25	0.75
big random	1	1	0	1	1	0	2.25	2.25	0.25
index	3	3	0	3	3	0	3	3	0.75
combine random+sort	0.5	1	1.75	1	1	0	2.25	2.25	0.25

Fig. 6. Detail resource usages of read queries

From Fig. 4, a wider range query request gets worse read performance than a smaller range one. However, the index key performs poorly in all types of queries since it needs to use all shards to process the result as shown in Fig 6. This decreased performance is significantly different from others on t-test as shown in Fig 5. The sorted key does better than the random key on big-range query because it uses a lower number of shards as shown in Fig. 6.

In conclusion, searching from an index or wide range search would degrade the system performance due to the number of involved machines delivering the results. Of course, the exact match or narrow range of search has better performance than index or wide range query. The number of involved shards has significant effect on read performance. The more shards involve in read, the poorer performance would be. Thus, locality has an impacts on performance.

V. Conclusion

MongoDB becomes a popular choice of NoSQL database usage today because it is an open-source software that could support big data without a restrict schema like a traditional relational database system. However, to gain a benefit of its horizontal scaling, one has to decide a right shard key which is not an easy task, especially for a small company that has no or minimal DBA skill. To understand the effects of a shard key to the MongoDB scaling performance, this paper evaluated the effect of selecting range-base shard key. We simulated an environment and measured the read/write performance of variety of keys. We found that a shard key with randomness and good locality could give a decent performance on write and read. The range of key might create hot spot on some shards. However, if it is widen enough to split the content to sharded cluster, it would not harm much on its performance. Also, in case that an intended shard key has nearly sorted values, combining a small range of random values to it might give acceptable performance for both read and write.

From the results of this study, one direction that we could further our work is to provide a method to semi-automatically pick a good shard key based on history of system usages. Another direction that we plan to explore is to create a good monitoring tool to help a MongoDB admin to decide when to shard or change a wrong key, if already sharded, before it is too late. There are quite a lot of monitoring data such as queue lengths, flush time, lock percentages, faults and other history of usages could be helped to create such tool.

References

[1] Xiaolin Wang, Hapeng Chen and Zhenhua Wang, “Implement of Dynamic Load Balancing in MongoDB” DASC'13 Proceeding, pp. 124-130.

[2] Yimeng Liu, Yizhi Wang and Yi Jin, “The improvement of MongoDB Auto-Sharding in Cloud Environment”, ICCSE 2012, pp.851-854.

[3] MongoDB document available at: http://info.mongodb.com/rs/mongodb/images/MongoDB_Architecture_Guide.pdf.

[4] MongoDB architecture guide available at: http://info.mongodb.com/rs/mongodb/images/MongoDB_Architecture_Guide.pdf.

[5] Foursquare Outage available at: http://highscalability.com/blog/2010/10/15/troubles-with-sharding-what-can-we-learn-from-the-foursquare.html.

[6] NoSQL available at: https://en.wikipedia.org/wiki/NoSQL.

[7] CAP theorm available at: https://en.wikipedia.org/wiki/CAP_theorem.

[8] MongoDB Lesson at pantip available at: http://macroart.net/2013/10/mongodb-lessons-learned-on-pantip/.

[9] How to choose shard key game available at: http://www.kchodorow.com/blog/2011/01/04/how-to-choose-a-shard-key-the-card-game/.

[10] Cardinality available at: https://en.wikipedia.org/wiki/Cardinality

[11] NoSQL ranking available at: http://db-engines.com/en/rank