ISSN 2395-1621

MongoDB Performance Tuning using Sharded Cluster and Redis Cache

Doppa Srinivas, Prof. Kale Navnath, Prof. Sarika V.Bodake

doppa.srinivas35@gmail.com

Computer Science
Padmabhushan Vasantdada Patil Institute Of Technology
(Savitribai Phule Pune University)
Pune,India

ABSTRACT

With the rapid development of the Database technology, the demands of large-scale distributed service and storage 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 Redis Cache is used to handle 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.

Keywords—Sharding, NoSQL, Redis, Database

ARTICLE INFO

Article History

Received:22nd March 2021

Received in revised form :

22nd March 2021

Accepted: 24th March 2021

Published online :

24th March 2021

I. INTRODUCTION

The term "NoSQL" was first introduced in 1988 for the relational databases not having SQL interfaces [1]. However, the term was re-introduced in 2009 for the kind of modern web-scale databases which trade transactional consistency over large-scale data distributions and incremental scalability. NoSQL databases were not originally meant to replace traditional databases, rather they are more suitable to adopt when relational databases does not seem appropriate. The main reasons behind adopting NoSQL databases are their simple yet flexible architecture and the capability of handling large amount of multimedia, word processing, social media, emails and other unstructured data files [6]. While, the conventional relational databases are hard to scale-out mainly due to their pre-defined schemas and I/O performance bottlenecks [2, 4]. These issues have made relational databases difficult to fit in the new computing paradigms such as Grid and Cloud applications, data warehousing, web2.0, social networking etc [7]. Contrary to it, NoSQL databases are becoming a primary choice for cloud applications because of their highly available, reliable and scalable nature [8].

The BASE (Basically Available, Soft State and Eventually Consistent) properties of NoSQL databases allow them to be scalable and thus, NoSQL systems inherently support auto-sharding phenomenon [2]. Auto-sharding is the automatic and native horizontal distribution of data among different severs in NoSQL databases which, in turn, increase the performance and throughput of database operations [9]. Another significant advantage of auto-sharding is load balancing across the cluster such that a single server does not get overloaded with all the queries. This makes NoSQL databases a good candidate for high transaction and write-intensive database applications [4, 5].Redis cache is to further improve the performance of the database.

Redis cache

Caching is a technique used to accelerate application response times and help applications scale by placing frequently needed data very close to the application. Redis [11], an open source, in-memory, data structure server is frequently used as a distributed shared cache (in addition to being used as a message broker or database) because it enables true statelessness for an applications' processes, while reducing duplication of data or requests to external data sources. In the redis cache we store the mongodb data so that for the next request we will not hit the database again.

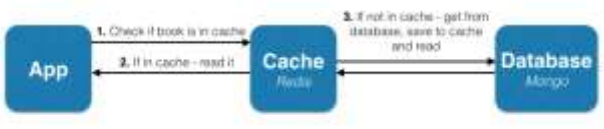
Why cache

The cache's main purpose is to reduce the time needed to access data stored outside of the application's main memory space. Additionally, caching is also an extremely powerful tool for scaling up external data sources and mitigating the effects that usage spikes have on them. An application-side cache effectively reduces all resource demands needed to serve data from external sources, thus freeing these

resources for other uses. Without the use of a cache, the application interacts with the data source for every request, whereas when a cache is employed only a single request to the external data source is needed, with subsequent access served from the cache. Additionally, a cache also contributes to the application's availability. External data sources may experience failures that result in degraded or terminated service. During such outages the cache can still serve data to the application and thus retain its availability

What is an application cache

Once an application process uses an external data source, its performance can be bottlenecked by the throughput and latency of said data source. When a slower external data source is used, frequently accessed data is often moved temporarily to a faster storage that is closer to the application to boost the application's performance. This faster intermediate data storage is called the application's cache, a term originating from the French verb "cacher" which means "to hide." The following diagram shows the high-level architecture of an application cache:



Sharding

Sharding is a method for distributing data across multiple machines. MongoDB uses sharding to support deployments with very large data sets and high throughput operations.

Database systems with large data sets or high throughput applications can challenge the capacity of a single server. For example, high query rates can exhaust the CPU capacity of the server. Working set sizes larger than the system's RAM stress the I/O capacity of disk drives.

There are two methods for addressing system growth: vertical and horizontal scaling.

Vertical Scaling involves increasing the capacity of a single server, such as using a more powerful CPU, adding more RAM, or increasing the amount of storage space. Limitations in available technology may restrict a single machine from being sufficiently powerful for a given workload. Additionally, Cloud-based providers have hard ceilings based on available hardware configurations. As a result, there is a practical maximum for vertical scaling.

Horizontal Scaling involves dividing the system dataset and load over multiple servers, adding additional servers to increase capacity as required. While the overall speed or capacity of a single machine may not be high, each machine handles a subset of the overall workload, potentially providing better efficiency than a single high-speed high-capacity server. Expanding the capacity of the deployment only requires adding additional servers as needed, which can be a lower overall cost than high-end hardware for a single machine. The tradeoff is increased complexity in infrastructure and maintenance for the deployment.

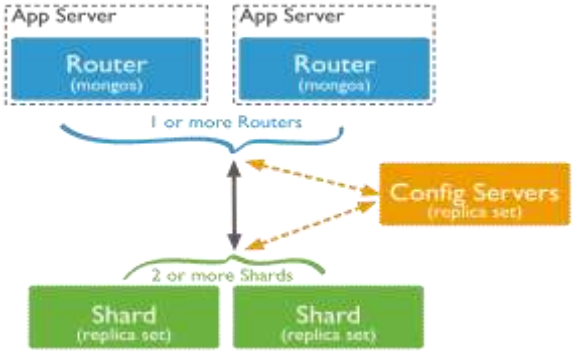
MongoDB supports horizontal scaling through sharding.

II. SHARDED CLUSTER

A MongoDB sharded cluster consists of the following components:

- **Shard**: Each shard contains a subset of the sharded data. Each shard can be deployed as a replica set.
- **Mongos**: The mongos acts as a query router, providing an interface between client applications and the sharded cluster.
- **Config Servers**: Config servers store metadata and configuration settings for the cluster. As of MongoDB 3.4, config servers must be deployed as a replica set (CSRS).

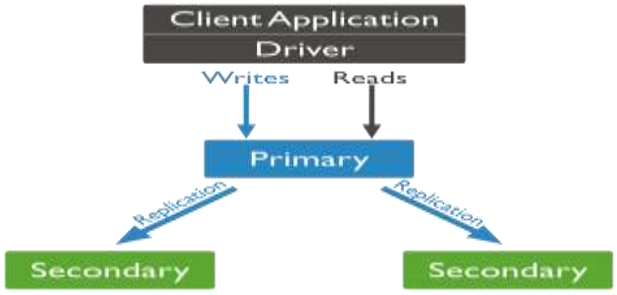
The following graphic describes the interaction of components within a sharded cluster:



Replica Set

Replication provides redundancy and increases data availability. With multiple copies of data on different database servers, replication provides a level of fault tolerance against the loss of a single database server.

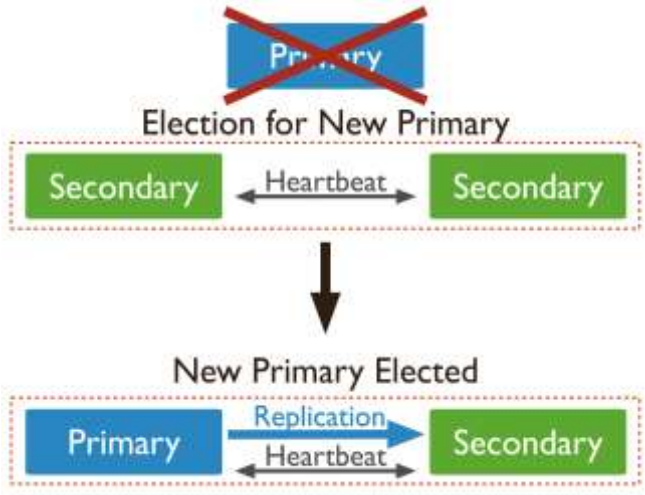
A replica set is a group of mongod instances that maintain the same data set. A replica set contains several data bearing nodes and optionally one arbiter node. Of the data bearing nodes, one and only one member is deemed the primary node, while the other nodes are deemed secondary nodes.



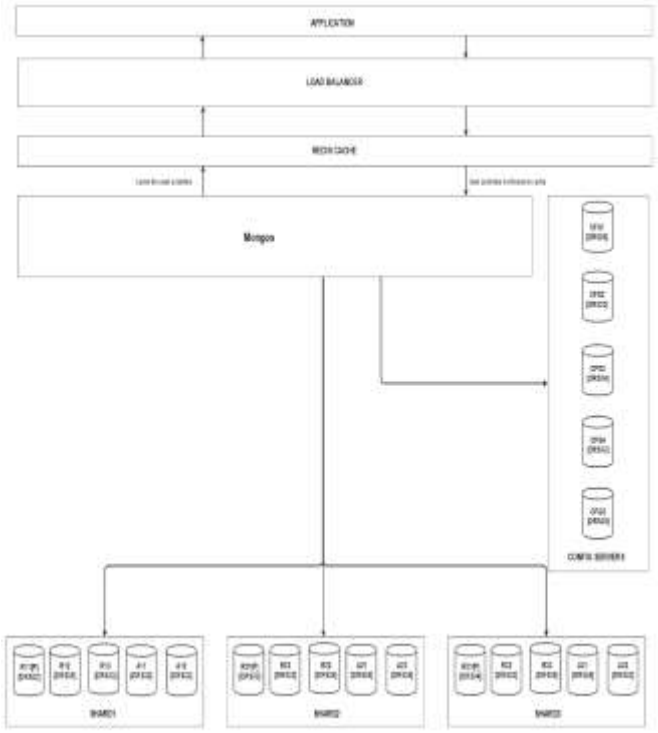
The primary node receives all write operations. A replica set can have only one primary capable of confirming writes with { w: "majority" } write concern; although in some circumstances, another mongod instance may transiently believe itself to also be primary. [1] The primary records all changes to its data sets in its operation log, i.e. oplog.

The secondaries replicate the primary's oplog and apply the operations to their data sets such that the secondaries' data sets reflect the primary's data set. If the primary is unavailable, an eligible secondary will hold an election to elect itself the new primary.

You may add an extra mongod instance to a replica set as an arbiter. Arbiters do not maintain a data set. The purpose of an arbiter is to maintain a quorum in a replica set by responding to heartbeat and election requests by other replica set members. Because they do not store a data set, arbiters can be a good way to provide replica set quorum functionality with a cheaper resource cost than a fully functional replica set member with a data set. If your replica set has an even number of members, add an arbiter to obtain a majority of votes in an election for primary. Arbiters do not require dedicated hardware.



III. ARCHITECTURE



The above diagram shows the architecture of mongodb sharded cluster with redis cache. I have created three shards with 5 replica set each (3 data nodes and 2 arbitrary nodes) on three virtual machines and five config servers are created in other VM.

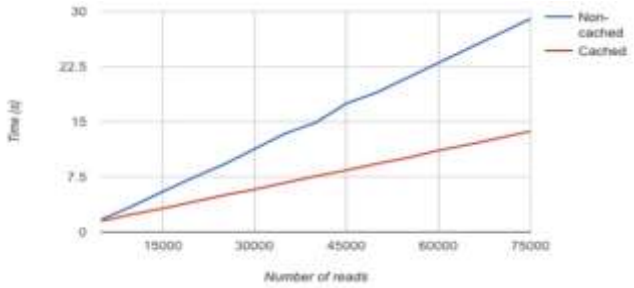
TABLE I.

S.No	Hardware configuration details		
	Virtual Machine Name	*RAM(in GB)*	*Number of Cores*
1	DRS32	16	8
2	DRS33	16	8
3	DRS34	16	8
4	DRS35	16	8

Rij means ith shard jth Replica set and same arbitrary nodes Aij.R12 means 1st shard 2nd replica set.P indicates the primary node.

IV. CONCLUSION

Under the above mentioned configuration the database is able accepts the 30000/sec requests with help of redis cache.



The above graph shows the improvement of the performance of database.

V. ACKNOWLEDGMENT

I thank Suman Bahera, Senior Software Engineer at TCS for his guidance in different core concepts of the mongodb.
I thank Sarika V.Bodake, Assistant Professor for assistance and for comments that greatly improved the manuscript.
I would like to show my gratitude to the Kale Navnath, Assistant Professor and Prof. Dr. B. K. Sarkar, Head, Department of Computer Engineering for sharing their pearls of wisdom with me during the course of this research, their comments on an earlier version of the manuscript, although any errors are my own and should not tarnish the reputations of these esteemed person.

REFERENCES

[1] R. Masood. "Fine-Grained Access Control for Database Management Systems." MS (CCS) thesis, National University of Science and Technology, Pakistan, 2013.

[2] 10Gen Corporation. "NoSQL Explained." Internet: http://www.mongodb.com/nosql-explained.

[3] IBM Corporation. "Data Security and Privacy – A holistic Approach." Internet: www.ibm.com/software/data/optim/protect-data-privacy.

[4] CodeFutures Corporation. "Cost-effective Database Scalability using database Sharding." Internet: www.codefutures.com/database-sharding.

[5] A. Viswanathan and C.J. Kothari. "Hibernate Framework-based database sharding for SaaS Applications." Internet: http://www.ibm.com/developerworks/library/os-hibernatesaas.

[6] C. Roe. "The Growth of Unstructured Data: What To Do with All Those Zettabytes?" Internet: http://www.dataversity.net/the-growth-of-unstructured-data-what-are-we-going-to-do-with-all-those-zettabytes, Mar.

[7] N. Hardiman. "Cloud computing and the rise of big data." Internet: http://www.techrepublic.com/blog/the-enterprise-cloud/cloud-computing-and-the-rise-of-big-data .

[8] A. Schram and K.M. Anderson. "MySQL to NoSQL: data modeling challenges in supporting scalability," in Proc. of the 3rd annual conference on Systems, programming, and applications: software for humanity, 2012, pp. 191-202.

[9] MongoDB Inc. "MongoDB sharding guide - Sharding and MongoDB Release 2.4.6." Internet: http://docs.mongodb.org/manual/sharding/, Mar. 2014.

[10] J.D. Meier, A. Mackman, M. Dunner, S. Vasireddy, R. Escamilla and A. Murukan. "Chapter 18: Securing Your Database Server." Internet: http://msdn.microsoft.com/en-us/library/ff648664.aspx, Jun. 2006

[11] Redis is an open source (BSD licensed), in-memory data structure store, used as a database, cache and message broker : https://redis.io/

