High-availability MongoDB Cluster Configuration Solutions

Alibaba Cloud Follow
Mar 2, 2018 · 7 min read

Introduction

High Availability (HA) refers to the improvement of system and app availability by minimizing the downtime caused by routine maintenance operations (planned) and sudden system crashes (unplanned).
This post extensively talks about High-availability Cluster Solutions along with several MongoDB High-availability Cluster Configurations.

High-availability Cluster Solutions

The high availability of computer systems has different manifestations at different levels:

(1) Network high availability
Thanks to the rapid development of network storage, the network redundancy technology undergoes constant improvements. The key application for improving the high availability of IT systems lies in high network availability. The high availability and high reliability aspects of a network are different. We achieve high availability of a network through network equipment redundancy by matching redundant network equipment, such as redundant switches, routers and so on.

(2) Server high availability
Primarily, we achieve the high availability of servers using server cluster software or high availability software.

(3) Storage high availability
Storage high availability refers to achieving high availability of storage using software or hardware technologies. The main technical indicators are the storage switching, data replication, and data snapshot features. When a storage device fails, another spare storage device can quickly take over to continue the storage service without interruptions.

MongoDB High-availability Cluster Configuration

A shortened form of high-availability cluster is HA cluster.
A cluster is a set of computers that provide users with a set of network resources as a whole.
These individual computer systems are the nodes of the cluster.
Building a high-availability cluster requires a reasonable allotment of roles of multiple computers, as well as data recovery and consistency mechanisms. The primary methods are as follows:

(1) Master-slave Approach (Asymmetric)

When the master node is running, the slave node is in monitoring and preparation status; when the master node fails, the slave node takes over all the tasks of the master node. After the master node resumes normal services, the services switch back to the master node as per the user's settings-automatic or manual. Also, it ensures data consistency through the shared storage system.

(2) Dual-machine Running Approach (Dual Active Mode)

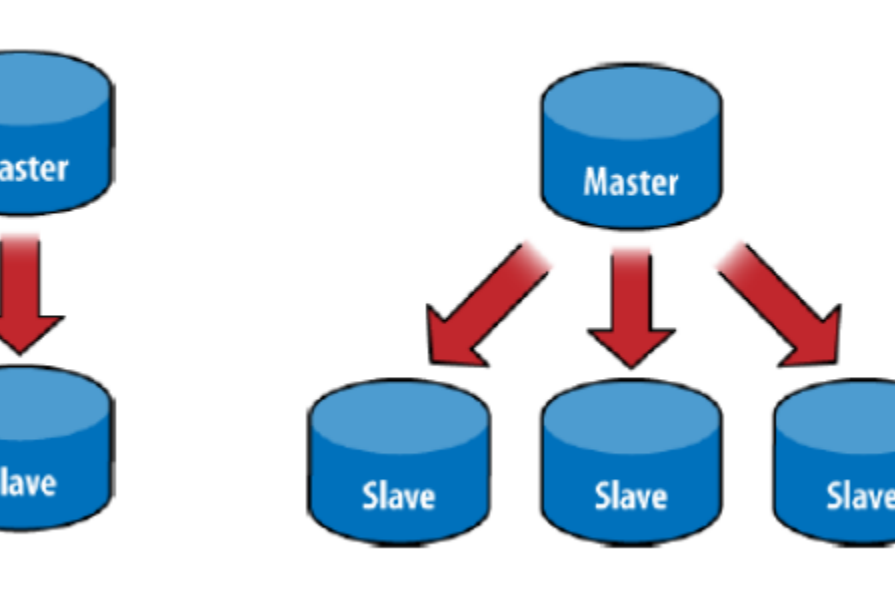
Two master nodes run their services at the same time and mutually monitor each other's status. When either master node fails, the other master node immediately takes over all tasks to ensure everything runs in real time. It stores key data of the application service system in the shared storage system.

(3) Cluster Running Approach (Multi-active mode)

Multiple master nodes work together, each running one or several services, and each defining one or more backup nodes for the services. When any master node fails, the master nodes takes over the services that were running on it.

The practices of MongoDB cluster configuration also follow these schemes, mainly involving the master-slave structure, replica set and sharding approaches.

Master-slave Structure



Generally, we use the master-slave architecture for backup or read-write splitting. The structure exists in two structure divisions, one-master-one-slave structure and one-master-multiple-slave structure.

There are primarily two roles:

(1) Master

The master node can both read and write data. When handling modified data, the op-log will synchronize the updates to all the connected slave nodes.

(2) Slave

The slave node can only read data, but not write data. It automatically synchronizes data from the master node.

We do not recommend MongoDB to use the master-slave architecture because it is impossible to recover the master node automatically after a failure. The replica set solution intended for introduction later would be ideal. The master-slave architecture remains unused unless the number of replica nodes exceeds 50. Normally, one need not use so many nodes.

In addition, the master-slave architecture does not support the chained structure. It is only possible to connect the slave node directly to the master node. Redis master-slave architecture supports the chained structure and allows us to connect the slave node to another slave node, hence the slave node of a slave node.

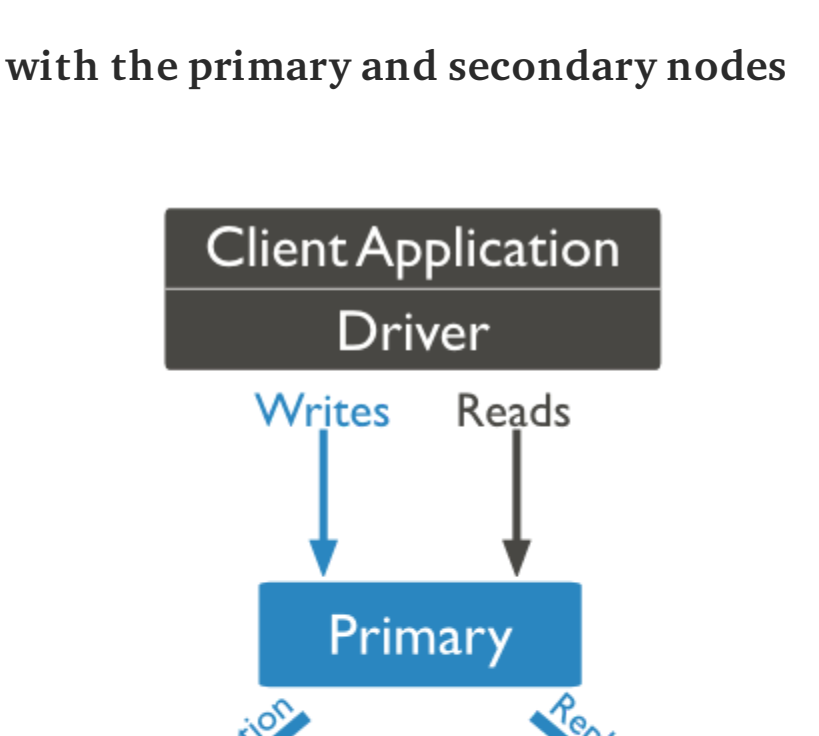
Replica Set

The replica set mechanism of MongoDB has two main purposes:

• One is for data redundancy for failure recovery. When the hardware fails, or the node is down for other reasons, you can use a replica for recovery.

• The other purpose is for read-write splitting. It routes the reading requests to the replica to reduce the reading pressure on the primary node.

1.Replica set with the primary and secondary nodes



The replica set is a set of MongoDB instances which share the same data content. It primarily involves three roles:

(a) Primary node

The primary node receives all the write requests, and then synchronizes the changes to all the secondary nodes. A replica set can only have one primary node. When the primary fails, the other secondary nodes or the arbiter node will re-elect a primary node. The primary node receives read requests for processing by default. If you want to forward the read requests to a secondary node, you need to modify the connection configuration on the client.

(b) Secondary nodes

The secondary node maintains the same data set with the primary node. When the primary node fails, the secondary nodes participate in the election of a new primary node.

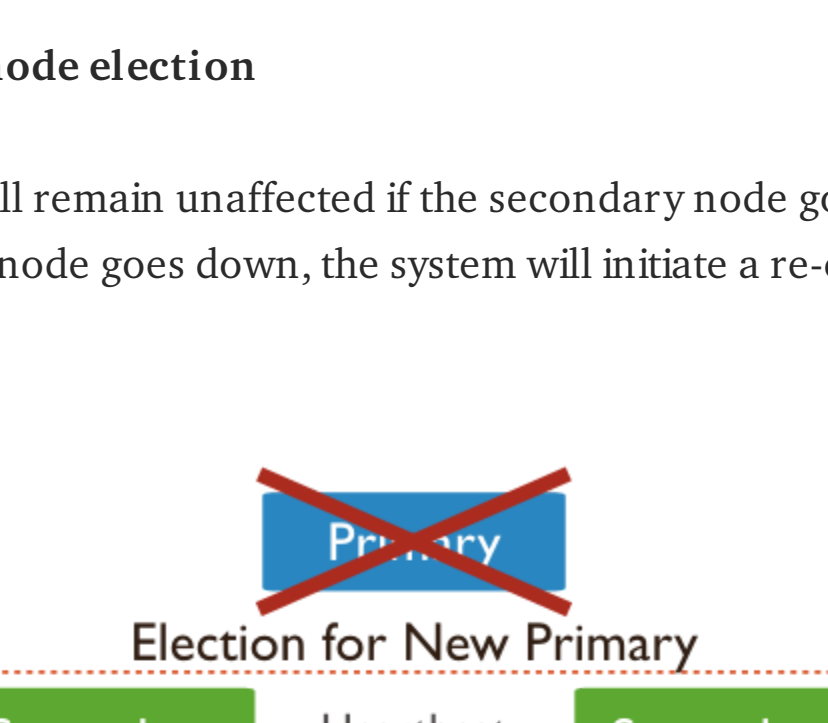
© Arbiter nodes

An arbiter node does not store data or participate in the primary node election, but it performs election voting. The arbiter node can reduce the hardware requirements for data storage, as the arbiter runs with minimal demand on hardware resources. However, it is important that one should not deploy the arbiter node on the same server with other data nodes in the production environment.

Note: The number of nodes capable of automatic failover in a replica set must be odd to avoid a tie when it comes to voting at primary node election.

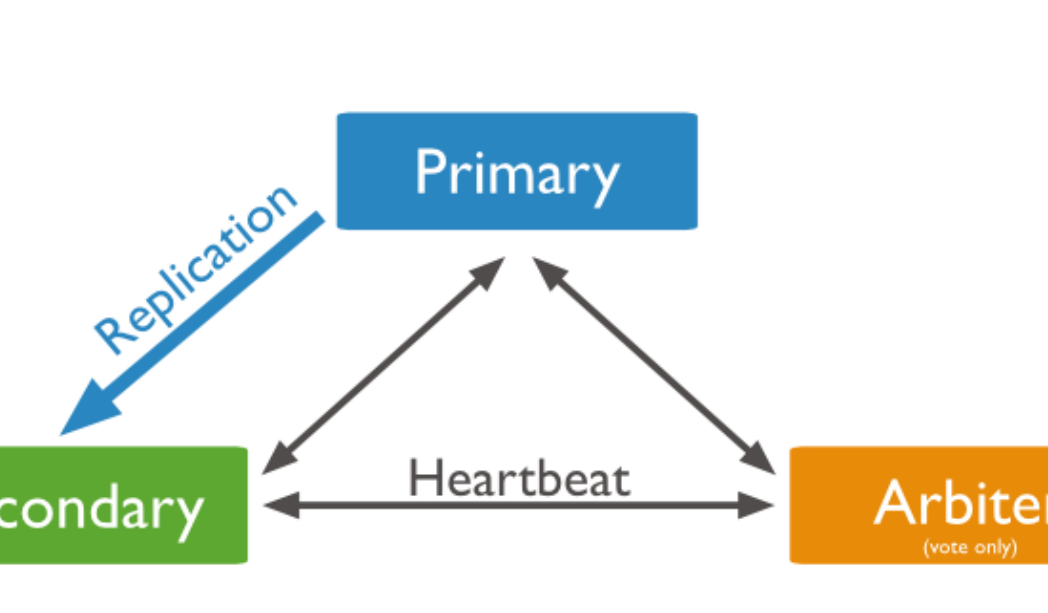
(d) Primary node election

The service will remain unaffected if the secondary node goes down, but if the primary node goes down, the system will initiate a re-election of the primary node:



2.Establish a replica set using the arbiter node

The replica set comprises an even number of data nodes, plus an arbiter node:

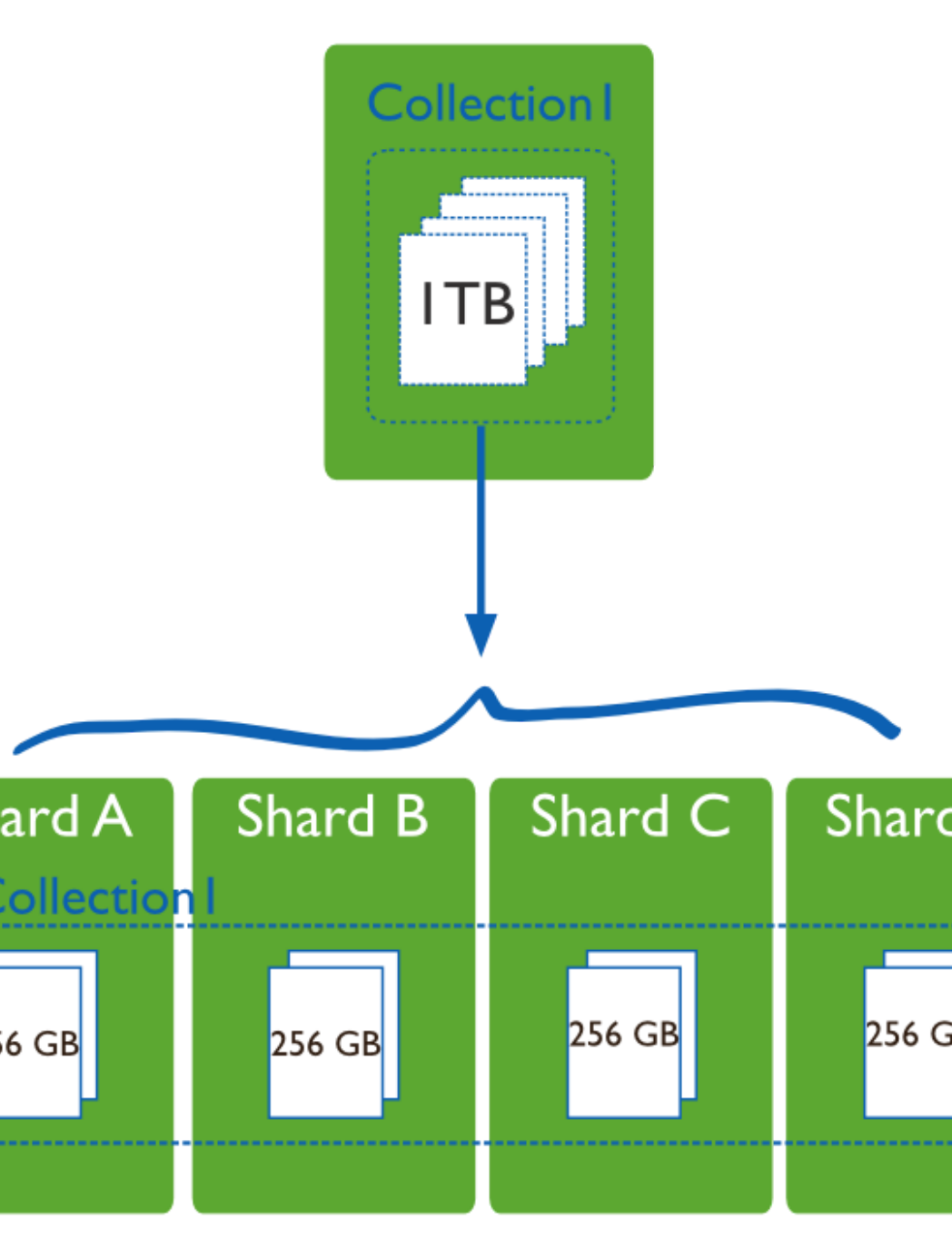


Sharding Technology

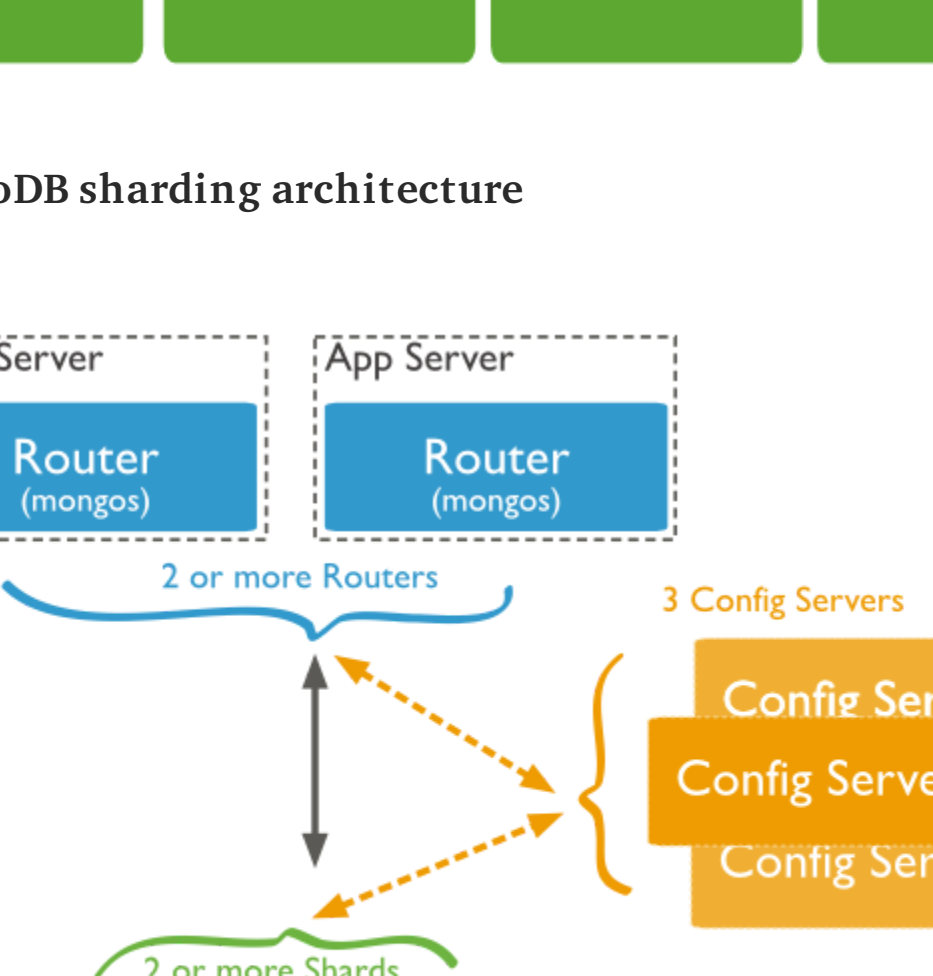
When the data amount is relatively large, we need to shard and run the data on different machines to reduce the CPU, memory and IO pressure. Sharding here refers to the "Database sharding technology".

MongoDB sharding technology is similar to the horizontal split and vertical split of MySQL. The database mainly adopts two sharding approaches: vertical expansion and horizontal sharding.
Vertical expansion refers to cluster expansion, namely adding more CPU or memory resources, or disk space.

Horizontal sharding means to shard the data to provide services in a uniform way through the cluster shown below:



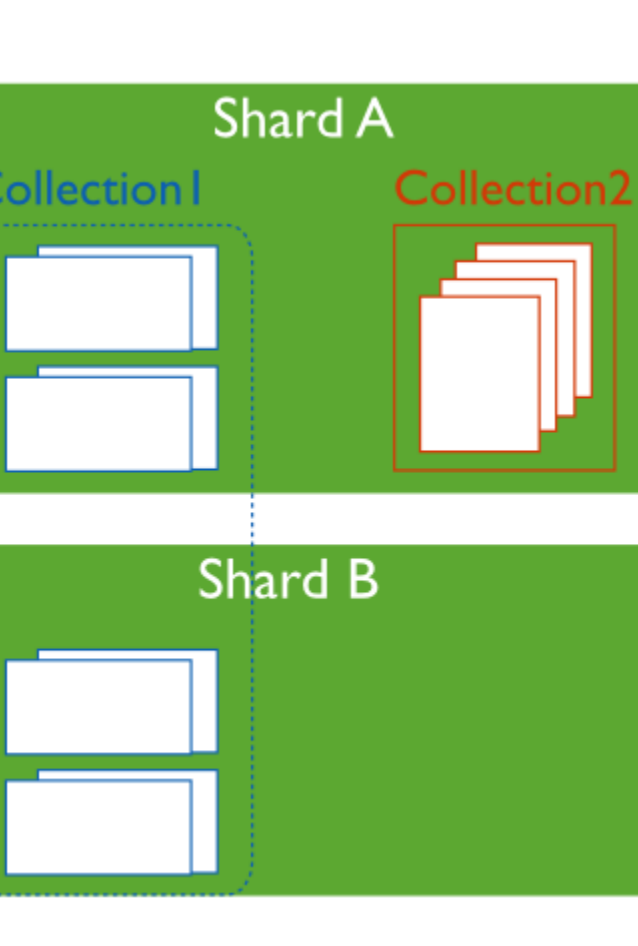
(1) MongoDB sharding architecture



(2) Roles in MongoDB sharding architecture
A. Data shards

A data shard serves to store data to ensure high availability and consistency of data. It can be a separate MongoDB instance, or a replica set.

Shard is generally a replica set in the production environment designed to prevent single points of failures in the data shard. A primary shard exists among all the shards that contain the unsharded data collection:



B. Query routers

A router is a mongos instance. The client has a direct connection to mongos, which routes read and write requests to the designated shard.

A sharding cluster can have a mongos instance or multiple mongos instances to ease the pressure from the client's requests.

C. Configuration servers

A configuration server saves the metadata of the cluster, including the routing rules for each shard.

Conclusion:

To make sure that a production cluster has no single point of failure, it important to provide a sharded cluster for high availability. In addition to thoroughly introducing the availability concerns involving MongoDB deployments with highlights of potential failure scenarios and their available resolutions, this post reveals the distinguishing features of the high availability cluster solutions.

Furthermore, it explores the sharding technology and adequately describes the roles of MongoDB sharding architecture as applicable to data shards, query routers and configuration servers.

Reference:

https://www.alibabacloud.com/blog/High-availability-MongoDB-Cluster-Configuration-Solutions_p490866?spm=a2c41.11248211.0.0

Cloud Computing · Mongodb · Cluster · Sharding

WRITTEN BY
Alibaba Cloud
Follow
Follow me to keep abreast with the latest technology news, industry insights, and developer trends.