MongoDB Sharded Cluster Design and Deployment Postprint

Authors: Hongzhang Yang

Date: 2023-10-08T00:00:00+00:00

Abstract

MongoDB is a relatively mature NoSQL database product that is widely adopted. For the construction of MongoDB sharded clusters, MongoDB Enterprise Edition provides auxiliary tools such as OpsManager, which implement comprehensive automated solutions for deployment, upgrade, monitoring, backup, and recovery. By contrast, building sharded clusters with MongoDB Community Edition requires certain accumulated experience. This paper systematically organizes the rules and characteristics of MongoDB replica sets, validated in experimental environments, striving to be more practical, readable, and operable. It also discusses the construction of MongoDB sharded clusters, designs corresponding solutions for the problems to be solved and factors to be considered, and provides recommendations for deployment practices.

Full Text

Preamble

Design and Deployment of MongoDB Sharded Cluster Solutions for ChinaXiv Partner Journals

(Data Technology Department, Communication Technology Bureau, Xinhua News Agency, Beijing 100803)

Abstract: MongoDB has become a mature NoSQL database product with widespread adoption. While MongoDB Enterprise Edition provides auxiliary tools such as OpsManager that enable complete automated deployment, upgrade, monitoring, backup, and recovery solutions, building a sharded cluster using MongoDB Community Edition requires considerable expertise. This paper systematically organizes the rules and characteristics of MongoDB replica sets, validated through experimental environments to ensure practicality, readability, and operability. We discuss the design of MongoDB sharded clusters,

addressing key problems and considerations, propose corresponding solutions, and offer recommendations for deployment practices.

Keywords: MongoDB; Community Edition; Sharded Cluster; Replica Set; Sharding; Voting Node; Fault Tolerance Mechanism
Classification: TP393
Document Code: A
Article ID: 1671-0134(2021)03-111-03
DOI: 10.19483/j.cnki.11-4653/n.2021.03.031
Citation Format: Yang Hongzhang. Design and Deployment of MongoDB Sharded Cluster Solutions [J]. China Media Technology, 2021(03): 111-113.

1. Software Versions

- 64-bit operating system: CentOS 6.5+
- MongoDB 3.6 Community Edition

2. Replica Set Rules and Characteristics

2.1 Replica Set Member Roles and Quantity

A replica set consists of one primary node (P), several secondary nodes (S), and optionally an arbiter node (A) based on actual requirements. The total number of nodes in a replica set cannot exceed 50, with a minimum of 3 nodes required (excluding single-node or master-slave dual-node configurations in sharded clusters). Figure 1 [Figure 1: see original paper] illustrates two common three-node configurations: one primary with two secondaries, and one primary with one secondary and one arbiter. Single arrows indicate primary-secondary replication, while double arrows represent heartbeat signals.

If the total number of replica set nodes does not exceed 7, each node has voting rights by default. If the total exceeds 7, the excess nodes beyond 7 must be configured as non-voting. For example, in a 9-node replica set, 2 nodes must be designated as non-voting. In other words, if the number of voting nodes is less than the total number of replica set nodes, the difference must be configured without voting rights.

The total number of replica set nodes and the total number of voting nodes are generally configured as odd numbers, though this is not mandatory. Regarding the choice between odd and even numbers, one viewpoint suggests that even numbers may result in tied votes during elections, preventing the selection of a primary. However, this assumption leads to a paradox that contradicts MongoDB' s high-availability architecture characteristics. Experimental verification demonstrates that with an even number of nodes, a primary can still be elected as long as the election receives support from more than half of the members. The "majority" required for electing a primary refers to more than half of the voting nodes, not necessarily the majority of the total replica set nodes.

As shown in Figure 2 [Figure 2: see original paper], the fault tolerance number indicates the maximum number of voting nodes that can fail while still enabling normal primary election. As long as the number of failed voting nodes does not exceed the fault tolerance number, the replica set can successfully elect a primary. The figure also reveals that increasing from an odd to an even number of voting nodes does not improve fault tolerance; instead, it reduces replica set stability.

2.3 Total Number of Voting Nodes

For example, with 3 nodes, 2 available nodes (67% availability) constitute a majority; whereas with 4 nodes, 3 available nodes (75% availability) are required to reach a majority.

2.4 Node Voting Rights

Voting rights are configured through the `votes` parameter, which defaults to 1 (representing one vote). The value cannot exceed 1, as a single node casting multiple votes is not permitted. A `votes` value of 0 indicates no voting rights (in such cases, the `priority` value must also be 0).

2.5 Node Priority

Priority is configured via the `priority` parameter, which accepts floating-point values ranging from 0 to 1000 [1], with a default value of 1. Higher values indicate higher priority, making a node more likely to be elected as primary. To favor a specific member for primary election, increase its priority value.

A replica set can have no more than 7 voting nodes.

2.7 Hidden Nodes

Hidden nodes are invisible to clients, which will not send requests to them [2]. Configuring a node as hidden prevents client requests. Although hidden nodes cannot become primary, they can vote normally during elections. Parameter settings: `priority: 0`, `hidden: true`.

2.8 Delayed Nodes

Delayed nodes must be hidden nodes and can vote normally during elections, but they do not receive client requests. A delayed node' s data lags behind the primary by a specified duration, preventing it from becoming primary. Example parameter settings: `priority: 0`, `hidden: true`, `slaveDelay: 259200` (in seconds, representing a 3-day delay).

The delay duration is determined based on data misoperation prevention requirements. For instance, a 3-day delay allows recovery of misoperations within the past 3 days, while a 7-day delay enables recovery within a week.

Additionally, based on business volume and data scale, configure the `oplogSizeMB` parameter in the node configuration file to ensure the oplog retention period exceeds the node delay time. The default size is 5% of available disk space.

2.9 When Voting Nodes Do Not Reach Majority

If voting nodes in a replica set do not constitute a majority (more than half of the total voting nodes), a new primary cannot be elected, and the current primary will automatically demote itself to a secondary.

3. Replica Set Design

3.1 Fault Tolerance Mechanism

To handle various emergencies such as node data corruption, physical machine crashes, or network disconnections, replica sets must maintain service availability when some nodes become unavailable, which imposes specific requirements on fault tolerance numbers.

A fault tolerance number of n means that the replica set can normally elect a primary even when n voting nodes fail. The replica set must contain a minimum of $2n+1$ nodes. For example, to tolerate 2 node failures, the replica set requires at least 5 nodes total.

3.2 Single Data Node Failure Recovery

Data node failure recovery employs different strategies based on the extent of damage. If only data is corrupted while hardware and other conditions remain normal, simply restore the data. For hardware failures, prepare a new node before restoring data.

Figure 3 [Figure 3: see original paper] illustrates data recovery to node S1. In a 1-primary/1-secondary/1-arbiter structure with only two data nodes, the data source must be the primary node (P). Copying large data volumes significantly impacts server and application performance. In contrast, a 1-primary/2-secondary structure with more than two data nodes can avoid using the primary as the data source by selecting a secondary node (S2) instead.

3.3 Use of Arbiter Nodes

Arbiter nodes participate only in elections and do not store data (they cannot be elected primary). Once configured as an arbiter, a node cannot be converted to a secondary; it must be removed from the replica set and re-initialized to join as a different role. Arbiter nodes can be deployed as needed, but additional arbiters neither improve data security nor enhance election efficiency.

Arbiter nodes serve a special purpose in cross-data-center and remote deployments. Their use ensures an odd number of voting nodes, improving replica set

stability while reducing data redundancy since arbiters do not store data.

3.4 Preventing Data Misoperation: Use of Delayed Nodes

Recovering from data misoperations requires delayed backup nodes (D nodes). Because primary-secondary replication is delayed by a specified period, operations performed on the primary during this interval have not yet been executed on the delayed node.

Based on hardware resource differences, it is recommended to designate the best-performing server as the primary, average-performing servers as secondaries, and the least powerful as arbiters. Designating the primary through priority settings also prevents an extreme scenario: when multiple shards experience node restarts and elections, using default priorities may result in all primary nodes being elected on the same host, leading to unreasonable resource distribution. By specifying primary nodes, this problem is completely resolved, enabling distributed business processing with primaries distributed across different hosts.

3.6 Use of votes:0, priority:0 Nodes

Nodes configured with `votes: 0, priority: 0` do not participate in elections, will not receive client requests, and serve solely as data backup nodes.

4. Cluster Deployment

4.1 Replica Set Nodes Across Data Centers and Remote Locations

To meet disaster recovery requirements across remote locations or different data centers, a multi-site architecture is necessary, as shown in Figure 4 [Figure 4: see original paper].

Data Center 1 serves as the primary site, with the primary node deployed there through priority settings. In scenario (a), the primary can be switched to Data Center 2 through replica set parameter adjustments. When communication between the two data centers is interrupted, the primary site can still conduct normal elections. However, if Data Center 1 experiences a complete failure, replica set availability cannot be guaranteed because Data Center 2 cannot achieve a majority to elect a new primary.

To solve this problem, scenario (b) incorporates Data Center 3 for deploying arbiter nodes. With equal numbers of nodes in Data Centers 1 and 2, regardless of which data center fails completely, the arbiter nodes can combine with the surviving data center's nodes to form a majority, ensuring replica set availability. Both hidden nodes (D) and arbiter nodes (A) possess voting rights despite never becoming primary.

4.2 Distributed Deployment of Replica Set Nodes

Deploying multiple nodes from the same replica set on a single host can render the entire replica set unavailable if that host crashes. To avoid this situation, replica set nodes must be distributed across different hosts.

4.3 Single Machine Single Node / Single Machine Multiple Nodes Deployment

Single-machine single-node deployment means one host runs only one node. Single-machine multi-node deployment means one host runs multiple nodes (from different replica sets). Since sharded clusters typically contain multiple shards and thus multiple replica sets, nodes belonging to different replica sets can be deployed on the same host. The number of nodes per host depends on hardware resources, primarily CPU and memory.

As shown in Figure 5 [Figure 5: see original paper], Server1 deploys one config node, one mongos node, the primary of shard1, and secondaries of shard2 and shard3.

4.4 Number of Shards and Cluster Scaling

MongoDB sharded clusters are used for large-scale data; otherwise, a replica set solution would suffice. One of MongoDB' s most significant advantages is its easy scalability. On one hand, this means the initial number of shards can be small (e.g., 2 shards), with additional shards added as needed. On the other hand, the data balancer migrates historical data relatively evenly across shards, and adding new shards triggers data migration to the new shards, which is time-consuming and resource-intensive. Therefore, shard addition should not be performed frequently. Only through thorough research and estimation of business volume and data scale can a forward-looking design be created that won' t require shard addition for an extended period.

4.5 Deploying Multiple Clusters on the Same Group of Servers

This deployment pattern is not recommended for production systems. However, if hardware resources are truly sufficient, it can be attempted for experimental purposes. Each node' s configuration file contains a `security.keyFile` parameter specifying the authentication file, with all nodes in the same MongoDB cluster using identical key files.

4.6 Config Server Deployment

MongoDB version 3.2 supports two config server deployment modes (choose one): - **SCCC Config Server**: Multiple config nodes in a mirrored configuration - **CSRS Config Server**: Multiple config nodes forming a replica set

Versions before 3.2 supported only SCCC; after 3.2, only CSRS is supported. When upgrading from version 3.2 to 3.4, config servers must first be configured as replica sets.

References

[1] MongoDB Manual[OL] https://docs.mongodb.com/
[2] Kristina Chodorow, translated by Deng Qiang and Wang Minghui. MongoDB: The Definitive Guide (Second Edition) [M]. Beijing: People' s Posts and Telecommunications Press, 2014(1).

Author Bio: Yang Hongzhang (1982-), male, from Huanggang, Hubei, Engineer at Xinhua News Agency Communication Technology Bureau.

Note: Figure translations are in progress. See original paper for figures.