Research and design of distributed key-value storage system based on Raft consensus algorithm

Lin Wang*, Xiaobing Chen, Weifeng Li

1 Huaiyin Institute of Technology, Huaian, JiangSu, 211600, China
*Corresponding author’s e-mail: 1181301328@hyit.edu.cn

Abstract. With the rapid development of the Internet, data storage technology has gradually become a major research field in the industry. But stand-alone storage has poor performance in terms of consistency or fault tolerance. Therefore, a new type of key-value storage system for unstructured data storage is widely used. This article mainly researches and designs a distributed key-value storage system based on the Raft consensus algorithm.

1. Introduction

With the development of the mobile Internet, how to store and apply the large-scale data generated by it has become a hot issue. Most of the data is unstructured, and in today’s Internet applications, storage systems have high concurrent access requirements, but traditional relational databases have poor scalability and low concurrency performance, and are not suitable for unstructured data. Storage, therefore distributed NoSQL storage systems have gradually become mainstream. But the three indicators of consistency, availability, and partition fault tolerance directly affect the distributed storage system. While through distributed horizontal expansion, more machines mean a higher risk of failure, and distributed systems need to be fault-tolerant [1]. We can guarantee consistency through some distributed consensus algorithms. These include Paxos, ZAB, Raft, Gossip. The consensus algorithm selected in this article is Raft, which is designed based on the Multi-Paxos idea. This paper is mainly based on the Raft consensus algorithm to research and design a distributed key-value storage system.

2. Materials and methods

2.1. Functions and application scenarios

Due to the poor performance of single-machine relational storage of unstructured data[2], distributed key-value storage systems have gradually become mainstream. Compared with stand-alone relational storage, its idea is to discard transaction operations and relational data structures in relational databases, exchange simple storage models for higher storage performance, and reduce the consistency requirements of data copies to ensure eventual consistency of data copies. Compared with the traditional relational data storage system, the key-value storage model adopted by this system is simpler and more efficient.

2.2. Introduction to Raft consensus algorithm

In essence, the Raft consensus algorithm is to achieve consensus on a series of values and the consistency of each node in the cluster through all methods based on the leader node. Its main modules are: leader election, log replication, and member changes. Raft is a strong leader model, which ensures consistency...
by implementing log replication on other nodes by the leader node. Due to the existence of a leader node, if the leader node fails and the node is unavailable, the cluster needs to elect a leader to generate a new leader. In Raft, time is divided into tenure, and each elected leader will manage the cluster until the end of the tenure. The cluster can update some expired information through tenure. For example, if the old leader node finds that the term number is smaller than other nodes, it will update the term information to the latest term. Each server in Raft must be in one of the three states of leader, follower, and candidate. During normal operation, a Raft cluster can only have one leader. Figure 1 is the Raft state transition diagram.

Figure 1 Raft state transition diagram

The communication method between distributed clusters is mainly through remote procedure call (RPC). RPC provides a higher level of abstraction than Socket network programming interface. In the Raft consensus algorithm, there are mainly two types of RPC: the candidate is the leader RequestVote RPC initiated during the election process, and AppendEntries RPC initiated by the leader to replicate log entries and heartbeat mechanisms.

If a node does not receive the heartbeat information of the leader node within a specified time, it will become a candidate node, and then send RPC information requesting voting to other nodes, and other nodes that receive the voting request can only vote within a term number. Give a vote, and secondly, vote only for candidates whose log integrity is higher than their own. The leader node is selected through a round of voting mechanism. If the election process is not handled properly, it is likely to cause livelock. Therefore, the leader election process can be optimized by random timeout.

The fundamental reason for Raft to achieve consistency lies in its log replication module. In Raft, the leader's log is used as the standard to achieve consistency in the log of each node. When the client requests, the leader creates a log entry and updates it to its local log. Then the leader replicates the log entry to other follower nodes through AppendEntries RPC, and submits the log entry to its state machine. In this process, the leader will check the log inconsistencies of the follower nodes, and will force an update to cover the inconsistent log entries.

2.3 Overall Design

According to the theoretical basis of the Raft protocol, we divide the content that needs to be implemented into the following parts: leader election, log module, node communication, and state machine. The specific cluster architecture and functional module structure are shown in Figure 2:
The system architecture adopts the traditional C/S architecture. The system is divided into two roles, namely the leader and the follower. They communicate data through RPC. If the client request is not the leader node, then a redirected message will be returned to the client, and the IP address of the leader node will be returned to the client.

2.4. Implementation

According to the above functional module structure diagram, we need to consider the technical selection and specific implementation of the four major modules.

First, discuss the technical selection of the RPC communication mechanism. The most popular ones in the industry are the HTTP-based RESTFUL API or the direct use of the transport layer protocol TCP or UDP. Considering that the cluster nodes communicate frequently and are usually in the same local area network, we choose the TCP protocol for reliable transmission. In the specific framework selection, we chose Netty\textsuperscript{3} which is a network programming framework that provides asynchronous and event-driven, and is the first choice for Java network programming. The log storage module selected RocksDB\textsuperscript{4}, which is a database engine based on the log structure. It provides high-performance storage.

In the interface design, we finally designed the following most important interfaces, as shown in Table 1:
Table 1. Project Interface

| Interface | Description                          |
|-----------|--------------------------------------|
| Consensus | ensure the data consistency of the nodes |
| LogModule | operate the log                      |
| RpcServer | RPC server                            |
| RpcClient | RPC client                            |

According to the above technical analysis, the technical selection can be summarized as shown in Table 2:

Table 2. Technical Selection

| Module            | Description |
|-------------------|-------------|
| Programmer Language | Java 1.8    |
| Network framework  | Netty 4.1.42.Final |
| Storage           | RocksDB     |
| Tools             | Guava       |

3. Results

Distributed clusters are deployed on different ports of a single machine to simulate clusters.

The test results of leader election are shown in Figure 3:

As can be seen from the figure, in this round of election, the node of port 8776 won the election and became the leader node.

The client sends a PUT command to create a key-value pair. Figure 4 shows the result of the server RPC response received by the client.

It can be seen from the figure that the client receives the message that the node on port 8779 is added successfully.

Figure 5 shows the result after the server receives and executes the RPC message requesting to create a key-value pair.
Figure 5 result of execution after the server receives request

It can be seen from the figure that the log of this operation was added successfully.

4. CONCLUSIONS

Since Raft is a strong leader model, if the system is partitioned due to a network failure, the entire cluster cannot provide external services because there is no leader node. From the CAP theory, the Raft protocol can meet consistency and partition fault tolerance. But it cannot meet availability. At the same time, due to the existence of the strong leader model, all client requests will pass through the leader. This will lead to lower performance of the distributed cluster using the Raft protocol, so the performance optimization of the system is a major problem. Therefore, the future optimization goal of this system is mainly to optimize network communication and underlying mechanism.

REFERENCES

[1] Zhike Wang. Research on the consistency of distributed systems based on Raft consensus algorithm [D]. Xidian University, 2019.
[2] Lin Feng. Design and implementation of distributed key-value pair storage system [D]. South China University of Technology, 2017.
[3] Netty.Netty Project [EB/OL]. https://netty.io/
[4] RocksDB.RocksDBOverview [EB/OL]. https://rocksdb.org/.