Raft consensus mechanism and the applications

Junjie Hu*, Ke Liu

1 School of Information Science and Engineering, Chongqing Jiaotong University, Nan'an, Chongqing, 400047, China
2 School of Materials Science and Engineering, Chongqing Jiaotong University, Nan'an, Chongqing, 400047, China
*Junjie Hu’s e-mail: hu@mails.cqjtu.edu.cn

Abstract. Raft consensus algorithm is one of the commonly used consensus algorithms in distributed systems. It is mainly used to manage the consistency of log replication. It has the same function as Paxos, but compared to Paxos, Raft algorithm is easier to understand and easier to apply to actual systems. The Raft algorithm is a consensus algorithm adopted by the alliance chain. This article describes the details of Raft consensus algorithm and its application in detail.

1. Raft Overview

1.1. Three states of Raft (role)

| Role     | Description                                                                 |
|----------|-----------------------------------------------------------------------------|
| Follower | Passively accepts requests from Leader. The initial state of all nodes is in the Follower state. |
| Candidate| Intermediate state transition from Follower to Leader.                       |
| Leader   | Responsible for interacting with the client and log replication (log replication is one-way, that is, the direction that the leader sends to Followers). Only one leader exists in the entire system at the same time, that is, two or more leaders will not appear at the same time. |

1.2. The transition relationship between the three states

The transition relationship between the three states is shown in Fig. 1. The Raft working mechanism in the next section will be specifically analyzed.
2. Key concepts

2.1. Copy the state machine
In a distributed system database, if the state of each node is consistent and each node executes the same command sequence, the entire distributed system will eventually get a consistent state[1]. That is, in order to ensure the consistency of the entire distributed system, we need to ensure that each node executes the same command sequence so that the logs of each node remain consistent[2]. Therefore, ensuring the consistency of log replication is the job of consensus algorithms such as Raft[3]. The replication state machine architecture is shown in Fig. 2.
correctly, the State Machine of each node will execute these commands in the same sequence, and finally get a consistent state[6]. The result of the consensus is then returned to the client, as shown in Fig. 3 below.

Fig. 3. Consensus process.

2.2. Term

In distributed systems, "time synchronization" is a big problem[7]. Because each machine may be always inconsistent due to its geographical location, machine environment and other factors. But in order to identify "outdated information", time information is essential.

The Raft consensus algorithm uses the concept of Term. The time is divided into Term (at the same time each node will maintain the Current Term locally), it can be considered as a logical time[8], as shown in Fig. 4 below.

Fig. 4. Term.

Each Term begins with a leader election and one or more Candidates run for Leader. If a Candidate wins the election, he will be the leader for the rest of the term. In some cases, multiple Candidate votes may be the same. At this time, Leader may not be selected (such as Fig. 3), then another Term will be started, and the next election will begin immediately. The Raft algorithm guarantees that there must be at least one leader in a given Term, and there can be no situation without a leader.

2.3. Heartbeats and Timeout

In the Raft consensus algorithm, there are two Timeout mechanisms to control Leader election:

- Election Timeout: the waiting time for the Follower to wait to become the Candidate state. This time is randomly set between 150ms and 300ms.
- Heartbeat Timeout: After a node becomes a leader, the leader will send an Append Entries message to other nodes. This information is transmitted through Heartbeat Timeout. When the follower receives Leader's heartbeat packet, it also resets the election timer.

3. Working Mechanism of Raft

3.1. Leader election

In the initial state, all nodes are started in the role of Follower, and Election Timeout is started (random time, reducing collision probability[9].
If a node finds that it has not received the heartbeat request sent by the Leader after the Election Timeout time, the node will become Candidate and stay in this state until one of the following three situations occurs:

- The Candidate wins the election
- Other Candidate wins election
- After a period of time, no server wins the election (enters the next round of Term elections and randomly sets the Election Timeout time)

The Candidate will then send Request Vote to other nodes. If more than half of the nodes agree, it becomes a leader. If the election has timed out and no Leader has been elected, it will enter the next term and re-elect.

After completing the Leader election, the Leader will periodically send Heartbeat to other nodes. Tell other nodes that the leader is still running, and reset the Election Timeout of these nodes. The re-election process is shown in Fig. 5.

3.2. Log replication

Client submits instructions to Leader (such as DET 5). After Leader receives the command, it appends the command to the local log. At this point, the command is in the Uncommitted state, and the replication state machine will not execute the command.

Leader copies the command (SET 5) to other nodes concurrently, and waits for other nodes to write the command to the log. If some nodes fail at this time or the command write time is too long, the Leader node will try again until all nodes have saved the command to the log. Then the Leader node submits the command (that is, the command is executed by the state machine, here: SET 5), and returns the result to the Client node.

After the Leader node submits the command, the next heartbeat packet will carry a message to notify other nodes to submit the command. After other nodes receive the Leader's message, they apply the command to the State Machine, and eventually the logs of each node remain consistent.

The Leader node records the maximum log index that has been submitted. Subsequent Heartbeat and Append Entries will carry this value. In this way, other nodes know which commands have been submitted, and can let State Machine execute the commands in the log, so that the state machine data of all nodes is consistent. The Leader election process is shown in Fig. 6.
In the case of inconsistent log content, the specific processing of the Raft consensus algorithm is as follows: as shown in the following figure, if in a distributed network, the log status of each node is as follows. When the Leader node sends a log copy request, it will bring the Index and Term of the last log record. At this time, the Leader node sends a log copy request <next Index: 8, command: x ← 4, the submitted log index is 7, and the term is 3>. At this point, after receiving the Leader's request, node A compares the Index and Term of the previous log record recorded by the leader node and finds that:

\[
\begin{align*}
\text{Index (Leader)} & > \text{Index (A)} \\
\text{Term (Leader)} & > \text{Current Term (A)}
\end{align*}
\]

The command was found to be absent from the node's log, and the request was rejected. At this point, the Leader node knows that an inconsistency has occurred and decrements the next Index. And send a log replication request to node A again until a node with consistent logs is found. Finally, the Follower node's log is overwritten with the Leader node's log content[10].

In other words, for requests with inconsistent log content, the Raft algorithm will overwrite the log content of the Follower node with the content of the Follower node. First find the location where the two log records are inconsistent for the first time, and then overwrite to the location of the most recently submitted command. The specific process is shown in Fig. 7.
3.3. Safety

The previous content discussed how the Raft algorithm conducts leadership elections and replicates logs. However, so far this mechanism cannot guarantee that every state machine can execute the same instructions in the same order. For example, when Leader submitted several log entries, a Follower might go down. This Follower was then elected as a leader and overwritten the original log entries with new ones. As a result, different state machines may execute different command sequences[1].

The Raft algorithm improves the Raft algorithm by adding a restriction during the Leader election phase. This restriction guarantees that for a fixed term, any leader has all the log commands submitted by the previous term. The Raft algorithm uses voting to prevent nodes that do not contain all log commands from winning elections.

If a Candidate node wants to win an election, it needs to communicate with most nodes in the distributed network. This means that each submitted log entry appears on at least one of the servers. If Candidate's log is as new as the log on the majority server, then it must contain all the submitted log entries. Request Vote RPC implements this limitation: RPC includes Candidate log information. If its log is newer than Candidate's, then it will reject Candidate's voting request[1].

Method for judging the new and old log content of two nodes: Raft algorithm determines which log content is newer by comparing the Index and Term of the last command in the log.

- If the terms of the two logs are the same, the content of the log with the larger term is updated.
- If the term number is the same, the content of the log is longer and the log is updated.

4. Case Study

How Raft maintains consistency when the network is partitioned. As shown below, we divide the distributed network into two subnets. The two subnets are the subnet AB and the subnet CDE respectively. At this time, the node B is a leader node.

However, there is no Leader node in subnet 1 due to network partitioning. At this time, the C, D, and E nodes did not receive the heartbeat of the leader node, which caused the Election Timeout to time out and entered the Candidate state. The entire distributed system began to conduct Leader elections.

At this point we assume that node C wins the election and becomes the leader node of subnet 1. Leader election process as shown in Fig. 8.
At this time, if there are Client nodes in the two subnets, they submit data to the Leader nodes in each subnet (such as: $X \leftarrow 3$). Since Leader node B in subnet 2 cannot be replicated to most nodes, its $X \leftarrow 3$ command will always be in the Uncommitted state. Since subnet 1 was successfully replicated to most nodes, $X \leftarrow 3$ finally reached a consensus on subnet 1. As shown in Fig. 9 below:

We assume that subnet 1 has undergone multiple elections and data interactions. The final log status of subnet 1 is shown in the following fig. 10:
At this time, the partition isolation status disappears. Leader C and Leader B send heartbeat requests respectively. In the end, Leader B found that Leader C had more votes than himself, so he switched to Follower status. With Log Replication, all node logs are finally agreed. As shown in Fig. 11 below:

![Diagram of network partition]

5. Raft Algorithm Overview
The core idea of the Raft algorithm is the same as that of other consensus algorithms, that is, it cannot require all individuals in the system to run without errors. As long as the majority nodes in the entire distributed system are running normally, the system can run well. In order to ensure the consistent behavior of all nodes in the system, Raft puts forward two extremely important core requirements:

- **Safety**: the safety of the system must be guaranteed no matter what.
- **Liveness**: The system must run and serve the client. It is not just that there are no problems, but also that the entire system can operate in a timely and good manner.

Raft consensus algorithm strengthens the status of Leader, divides the entire algorithm into two parts clearly, and uses the continuity of logs to make a good response strategy for different states of Leader. In order to ensure the correctness of the Leader, the Raft consensus algorithm emphasizes the legitimacy and uniqueness of the Leader. Only one legal leader can exist at the same time. At the same time, the newly elected Leader in the Leader election algorithm of the Raft consensus algorithm already has all the logs that can be submitted[13]. Therefore, the Raft protocol log can simply send and add log information only from the Leader to the Follower. Raft uses the continuity of the log to make a lot of simplifications to Paxos, so that the entire algorithm can be truly applied to various distributed problems[14].

Fig. 10. Leadership election with network partition.

Fig. 11. Leadership election with network partition.
Acknowledgments
Thanks to Professor Bo Mi for his patient guidance and the selfless help of the graduate team. The theoretical analysis guidance provided by Professor Bo Mi and the good suggestions provided by the graduate team make the writing and research process of the article highly efficient. Thanks for the support from the School of Information Science and Engineering of Chongqing Jiaotong University. Without their support, I would not be able to complete the full and efficient dissertation.

References
[1] BOLOSKY, W. J., BRADSHAW, D., HAAGENS, R. B., KUSTERS, N. P., AND LI, P. Paxos replicated state machines as the basis of a high-performance data store. In Proc. NSDI’11, USENIX Conference on Networked Systems Design and Implementation (2011), USENIX, pp. 141-154.
[2] GHEMAWAT, S., GOBIOFF, H., AND LEUNG, S.T. The Google file system. In Proc. SOSP’03, ACM Symposium on Operating Systems Principles (2003), ACM, pp. 29-43.
[3] O’NEIL, P., CHENG, E., GAWLICK, D., AND ONEIL, E. The log-structured merge-tree (LSM-tree). Acta Informatica 33, 4 (1996), 351-385.
[4] SCHNEIDER, F. B. Implementing fault-tolerant services using the state machine approach: a tutorial. ACM Computing Surveys 22, 4 (Dec. 1990), 299-319.
[5] ROSENBLUM, M., AND OUSTERHOUT, J. K. The design and implementation of a log-structured file system. ACM Trans. Comput. Syst. 10 (February 1992), 26-52.
[6] OKI, B. M., AND LISKOV, B. H. Viewstamped replication: A new primary copy method to support highly-available distributed systems. In Proc. PODC’88, ACM Symposium on Principles of Distributed Computing (1988), ACM, pp. 8-17.
[7] HUNT, P., KONAR, M., JUNQUEIRA, F. P., AND REED, B. ZooKeeper: wait-free coordination for internet-scale systems. In Proc ATC’10, USENIX Annual Technical Conference (2010), USENIX, pp. 145-158.
[8] Hunt P D, Konar M, Junqueira F, et al. ZooKeeper: wait-free coordination for internet-scale systems[C]. usenix annual technical conference, 2010: 11-11.
[9] LAMPORT, L. Time, clocks, and the ordering of events in a distributed system. Communications of the ACM 21, 7 (July 1978), 558-565.
[10] LAMPORT, L. The part-time parliament. ACM Transactions on Computer Systems 16, 2 (May 1998), 133-169.
[11] HERLIHY, M. P., AND WING, J. M. Linearizability: a correctness condition for concurrent objects. ACM Transactions on Programming Languages and Systems 12 (July 1990), 463-492.
[12] MORARU, I., ANDERSEN, D. G., AND KAMINSKY, M. There is more consensus in egalitarian parliaments. In Proc. SOSP’13, ACM Symposium on Operating System Principles (2013), ACM.
[13] VAN RENESSE, R. Paxos made moderately complex. Tech. rep., Cornell University, 2012.
[14] LAMPORT, L. Generalized consensus and Paxos. Tech. Rep. MSR-TR-2005-33, Microsoft Research, 2005.