Weekly Report – Raft

Tasks

- Continue to learn the raft algorithm;
- Consider a new algorithm to improve the performance of the raft algorithm;

Main Activities

1 Raft in detail

After some material of raft algorithm， the process of the raft algorithm becomes clear from vagueness. Now I will explain raft algorithm by my own comprehension.

To my point of view, raft algorithm could split up into four main function:

- Leader election
- Log replication
- Membership changes
- Log compaction

The first two(leader election and log replication) are core of the algorithm, and the latter two(membership changes and log compaction) are optional which solve some problem of a practical project. Next I will focus on the first two functions and illustrate some most important key elements in raft.

1.1 Leader election

Raft consensus algorithm is characteristic of strong leader, and use heartbeat to control the communication between nodes. Raft guarantee there are and only leader to issue log entries and use random timeout to avoid conflict.

Normally, there is one leader issue AppendEntris RPC to the other nodes which are follower. If a follower has not receive a RequestVote RPC within a **electionTimeout**, then the follower will **add its current term** and become a candidate.

electionTimeout have to satisfy the follow formula:

- broadcastTime << electionTimeout << MTBF

After become a candidate, the node initiates RequestVote RPC which contents information of its new term. The other nodes decide whether to accept the RequestVote RPC according to the term number. If the new term number is large then current term, the nodes will accept the RequestVote RPC otherwise not.

The term number is monotone increasing. It is the key factor that the raft algorithm can guarantee consistency.

There is another very important restriction when a node accept the RequestVote RPC:

- **If the candidate do not content the latest log which has been committed to state log machine, the RequestVote RPC will be deny.**

It is the restriction make raft more simpler to achieve consistency since the restriction limit data flow only from leader to followers.

If majority accept the RequestVote RPC, the candidate will become a new leader and the raft system will enter a new term of the new leader rude. Otherwise a new leader election process will be launched in the next term.

1.2 Log replication

In leader election process, key elements are electionTimeout, RequestVote RPC and term. As for the log replication process we will focus on AppendEntris RPC, log ID, and also term.

Because of the leader election mechanism, We can assume that the premise is one and only leader issue AppendEntris RPC to other followers.

Clients-leader communication and AppendEntris RPC are two relatively independent processes **(that is the point, it will be discuss more detailedly in part 2)**.

- Clients-leader communication process

    1. The only leader receive request from client;
    2. The leader save the log in its storage;
    3. If a log has been accept by majority, the leader will commit the log to state machine
    4. After the log has been committed, then the leader will response to the client.

- AppendEntris RPC process

    1. The leader maintain a nextIndex array which decide which log should be sent to the different follower;
    2. The leader constantly issue AppendEntris RPC to followers respectively, which contents prevLogIndex and term;
    3. After a follower received a AppendEntris RPC, it will match log according to prevLogIndex and compare term in AppendEntris RPC between its currentTerm to decide whether or not accept the AppendEntris RPC.
    4. If majority accept a log, the leader will commit the log to the state machine.

Clients-leader communication process is very simple, in this part I major concern about AppendEntris RPC process. Here are some of the most critical issues in the Appen process：

- When a leader commit a log, it would not just commit only one log but also the all log before the commit log which has not been commit.
- If a log which is not currentTerm is accepted by majority, the leader would not launch a new commit process.
- The key element in determining whether a follower accepts the AppendEntris RPC is the term. If the term not less the follower’s currentTerm it will accept otherwise not.

- To achieve consistency, followers have to modify its log entries and nextIndex in leader according to the matching of prevLogIndex.

2 A method to improve performance of raft

2.1 Add-Secretary Method

As mentioned in the 1.2 above, after receiving a request from the client, the leader simply keeps the log in the local store and wait the log to be accepted by majority. This means client interaction and AppendEntris RPC are two separate parts.

RPC is a very time-consuming process since the leader have to constantly read log from memory to issue AppendEntris RPC heartbeats to all followers.

Therefore, to optimize raft algorithm and speed up operation efficiency, we could assume that we have some log in the leader and accelerate AppendEntris RPC process by add some servers which we called secretary to make AppendEntris RPC execute concurrently.

We could assume that the method is reasonable by analyze the structure of the raft system.

- Most write operations are distributed evenly across the nodes
  (clients write log on leader, leader write almost the same log on each follower).
- However, most read operations are uneven
  (all followers read from just one leader)

The distribution of the read operation of the raft algorithm is uneven. Almost all the readings are from the leader. So we can add a mechanism to separate the read operation from the leader.

In Add-Secretary method, each log is read only once from the leader and don’t need read from leader in every heartbeat.

2.2 Problems We Are Facing

To achieve the Add-Secretary method, from my personal understanding there are several problems we have to solve.

1. Leader Election

   The leader may crash, then a new leader election will be launched and a new leader have been chosen, but the secretaries are attached to the leader, how to make all secretaries choose only one leader is a problem.

   If secretaries only get information from leader, it is impossible to keep consensus since some time the leader even don’t know it is not leader any more.

2. Log Replication

   In traditional raft algorithm, leader maintained many important information such as term,nextIndex and so on.

   These information is critical to issue AppendEntris RPC and have to been modify by the result of AppendEntris RPC, If AppendEntris RPC process distribute in several secretaries, it hardly maintain these information in all secretaries since these information is modified by AppendEntris RPC and different secretary launch different AppendEntris RPC.

If we could elect a leader in secretaries ?

Period: 2017.5.3-2017.5.8
Submitted by: jiacheng huang 2017.5.9