Raft: leader election, normal operation, leader changes, neutralizing old leaders.

Basic Submission

Name of student 1	Junda Feng
NetID of student 1	jundaf2
Repo URL	https://github.com/JundaFeng/mp2-raft.git

1. Name of student 1
   Junda Feng

2. NetID of student 1

3. Name of student 2 (optional)

4. NetID of student 2 (optional)

5. Repo URL (You are encouraged to create a private repo and share it with Github username: "dayuebai")

6. Commit to be graded.

Introduction

Raft is called log consensus, which means a set of events ordered in an agreed-upon sequence maintained by a cluster machine. Raft uses a consensus module to agree on a strict ordering of update (applied deterministically). In terms of which is next event, total ordering of events is quite similar to what consensus is trying to achieve.

Build application on top of Raft which tell how to order the updates. As long as the application is deterministic, Raft gives nice fault-tolerance and consistency.

Each process in Raft can only vote for one process in one term. Commitment for vote for somebody only last for that term. When start a new term and a new election, use a randomized timeout picked between some pretty big ranges. The change that the processes repeatedly get into deadlock states are low. The timeout determines when the new term starts. The timeout of how long since you received your last heartbeat to wait for until you declare somebody is failed is randomized. So the time that different processes detect the failure would be randomized between different processes. As a result, only one of the processes would detect this failure first, and the process that detects the failure first is the one most likely to be the elected leader. A new election call means a new term for an individual process.

Like Paxos, a negative vote is not necessary to send and the process could keep silent for a negative vote and do nothing. When a process receives positive votes from a majority of all the processes including itself, it sends applied-entries request (heartbeat). Once these heartbeats are sent, all processes including the one that disagreed with its leadership will acknowledge it as the new leader. All the servers switch their state to be ‘follower’ and give up their own election if there is any once they get the heartbeat. The responses to the heartbeat are not necessary either but they may help. Every time a follower get a heartbeat, it reset this timeout. The only thing a leader does is send its heartbeat.

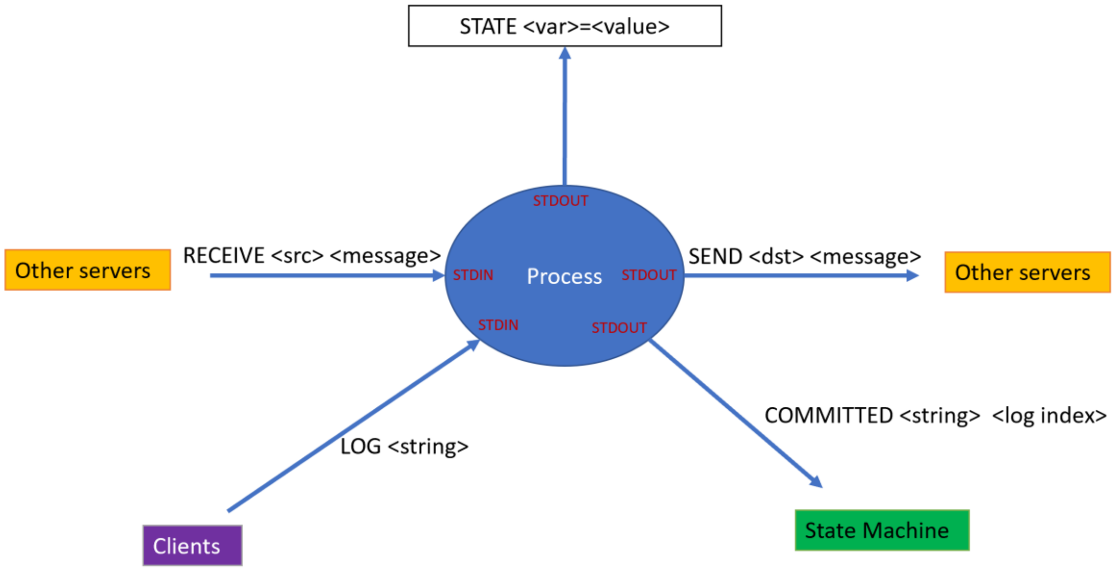
Safety concern is alleviated by the fact that every term has at most one leader. Thus even if a leader of a previous term that crashed for some reason resumes in a later term, it will move on to the next phase once it gets the heartbeat of the new term. In each term, at most one leader can be elected. The parameters are to minimize the chance that no leader is elected.

Associate events to a term can resolve conflicts. Raft is designed for crash-recovery failure. When a process recover from a crash, it find its log from stable storage. Leader acts like a global sequencer by adding commands to its log in order. This means there is only one process that determines the order of events in every term.

Log consistency achieved by raft is also a key component of Bitcoin. Partition is a common issue in networking. Typically data center are geographically distributed. Partition is when you have regions of your network that are separated such that you can talk within a region but you cannot talk between regions.

Framework

The interaction between the raft process and the framework can be described as the following schematic diagram.



- Unicast send: to send messages to the other nodes, the node should write a message to STDOUT:

  "SEND <DST> <message>\n"

- Unicast deliver: read from STDIN indicating the process has received from some other processes

  "RECEIVE <SRC> <message>\n"

- Log from clients: the framework will give entries to log. The string should be added to the log entries and committed.

  "LOG <string>"

- Acknowledge the message commit

  "COMMITTED <string> <log index>"

- Print out the State updates of the consensus group from the leader:

  "STATE <var>=<value>"

Executable Interface

Your executable will take 2 arguments:

.\raft <the identity of the current raft node> < the number of nodes >

Each event consists atomically of three steps:

- Receipt of a message by a process
- Processing of message
- Sending out all necessary messages into the global message buffer

Check Point 1: Leader Election

- Select one of the servers to act as leader
- Detect crashes, choose new leader

Server states:

- Leader
    - Discover higher term in RPC
        - Reject RPC with lower term
        - Revert to follower and update the term when seeing higher term
    - Send heartbeat
        - Send AppendEntries{} RPCs to followers
    - Receive command from clients
        - Append command to log
    - Commit new entries
        - Current term and early terms
        - Pass commands to state machine
        - Return result to clients
        - Send AppendEntries{<Committed entries>} RPCs to followers
            - Retries until success
            - Contains index # and the term of the last preceding entry
    - Keeps entry index for each Follower
        - Index of next log entry <next index> to send to that Follower
        - Initializing to (1+<leader's last index>)
    -
- Follower (initial state)
    - Receive AppendEntries RPCs from Leader
        - Check coherency based on the index and term
            - If not consistent, decrement <next index> 1 and try again
            - Overwrite inconsistent entry and delete all subsequent entries
        - Pass <Committed entries> to state machine
    - Timeout
        - Increment current term
        - $\rightarrow$ Candidate
    - Discover higher term
    - Discover current server
- Candidate

    - Start election
        - Vote for self
        - Send RequestVote RPCs to all other servers
            - Receive votes from majority
            - → Leader
    - Receive heartbeat RPC from valid leader
        - → Follower
    - Vote
        - Deny vote
            - According to the log info <Committed entries> in RequestVote RPCs, only vote for Candidate with more complete log.
    - Timeout
        - Increment term
        - Start new election

Log:

- Component
    - Log index
    - Term #
    - Command
- Storage
    - Stable storage for surviving crashes
- Commit
    - If known to be stored on a majority of servers
    - At least one new entry from leader’s term must also be stored on a majority of servers
    - Eventually be executed by state machine
- Consistency (Coherency)
    - The same index and term identify the same command on different servers
    - Logs are identical in all preceding entries
    - A committed log indicates its predeceasing logs are also committed.

Timeout are set to 100 ms.