Distributed Algorithms 60009 - Raft consensus

Pengfei Xiao(px23), Qiancheng Fan(qf23)

February 24, 2024

1 Architecture

The architecture of Raft is illustrated in Figure 1 . It depicts the connections and message transmission among the various key modules.

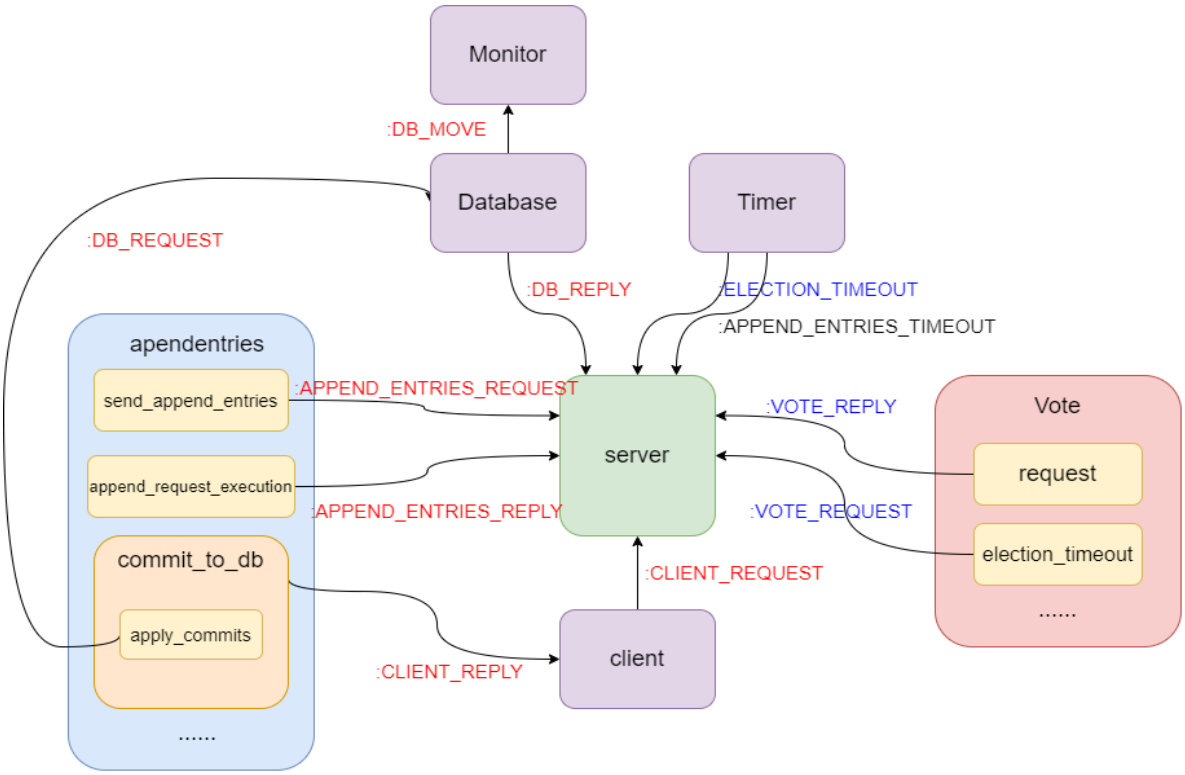

Figure 1: Architecture of raft

The typical communication process when a client sends a request to the server is illustrated in Figure 1 with the text in red font. In this description, we temporarily do not consider scenarios involving leader changes and voting.

1. The client sends `:CLIENT_REQUEST` to the server. If the current server is not the leader, it directly returns the current leader in the servers. Otherwise, proceed to Step 2.
2. The leader sends `:APPEND_ENTRIES_REQUEST`, including the entry, to all servers (except itself).
3. Other followers receiving `:APPEND_ENTRIES_REQUEST` fill the entry into their own logs (if valid) and send `:APPEND_ENTRIES_REPLY` to the leader.

4. The leader receives `:APPEND_ENTRIES_REPLY` from followers and updates the next_index and match_index for the corresponding follower located at the leader.

5. The leader attempts to update commit_index (once a majority of followers have completed log writing) and synchronizes the update with other followers. The leader writes the updated log to the server's database, sends `:DB_REQUEST`, and other followers subsequently send `:DB_REQUEST` for synchronization (DB will also send `:DB_MOVE` to the monitor.ex for monitoring purposes while not the main part of the C-S process).

6. After the request is successfully committed, the leader sends `:CLIENT_REPLY` to the client, including custom reply information.

7. The client receives `:CLIENT_REPLY`, completes subsequent recording tasks on the client side, and begins the next iteration of the loop.

2 Design and Implementation

2.1 Design

In general, the modules are divided based on logical functions. All interactions with the client are handled by the leader, while other followers synchronize logs and databases. The system includes independent debug and monitor modules, facilitating debugging and obtaining effective outputs.

Typically, for requests sent from the client, checks are performed upon receiving client requests and committing to the log. The corresponding cid is checked in the Log to prevent cases where the client submits a duplicate cid that has not yet entered the Log.

For monitoring timeouts, the send_after() function in timer.ex sends `:ELECTION_TIMEOUT` and `:APPEND_ENTRIES_TIMEOUT`. In the server, two different timeout functions are used for handling. `:ELECTION_TIMEOUT` is processed by the timeout function in vote.ex, where the sender can only be a follower or candidate. `:APPEND_ENTRIES_TIMEOUT` is processed by the timeout function in appendentries.ex, where the sender can only be a leader.

2.2 Implementation

Here is the part of the system that has been significantly modified based on the provided code framework.

server.ex Centrally receives all requests to the server and forwards them to the corresponding functions. Using a unified style to accept requests from different sources and recording them effectively serves the purpose of listening to events and dispatching them.

vote.ex Handles the logic related to elections in this file. Elections are initiated upon reaching the election_timeout in timer.ex. The transitions in the identity state machine, from Follower to Candidate and Candidate to Leader, are completed in this section. Additionally, if the leader discovers an out-of-date term, it will also transition to Follower.

appendentries.ex Contains logic related to log synchronization and commit requests. The log recorded in the leader is synchronized to followers by the leader's send_append_entries, which sends all entries to a specific follower. A request can only be committed and applied to the database after obtaining a majority match. When the client does not send any information, the

leader sends heartbeat packages to followers to maintain its leader role. Moreover, the leader and candidate automatically transition to followers when the server term is out-of-date.

clientrequest.ex Handles requests corresponding to `:CLIENT_REQUEST`. It involves checking whether the received client request is from the leader, and whether the client request's cid has already been recorded in the log.

3 Evaluation

3.1 Machine Description

We use Lab machine to run the experiments, its specifications are:

- OS version: Ubuntu 22.04.3 LTS
- Memory: 15GiB in total
- CPU: Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz, with 6 cores and 12 threads

3.2 Experiments evaluation

3.2.1 Normal situation: 5 clients and 5 servers

The normal case just works as expected. From the log file `01_5_servers_5_clients_normal` we can see the election timeout triggered when servers started and starts election stage, each server vote for themselves and send vote request to all servers and reply to requests by `vote_reply`. We can see the leader get selected at row 126.

The client request handling also looks fine, when the server received a request and it is not a leader, it replies with `:NOT_LEADER`, and trigger the leader query. If it is the leader, it replies `:LEADER`, and trigger the append entry process. The log shows all servers successfully received `APPEND_ENTRIES_REQUEST`, and the `APPEND_ENTRIES_TIMEOUT` also get correctly handled when it happened.

The log's output of db updates done also looks well, here I set the timelimit to be 40s, in the last db update at line 33527, we can see all server has 11 commit number except server2, this is because when the timelimit reached, some commits haven't been appended to servers yet, this is as expected.

3.2.2 High load: 10/20 clients and 5 servers

To test high load, we increase the number of clients to twice to increase the workload and stored the log at file: `03_5_servers_10_clients_highload`. The whole execution process does not have a big difference from the normal case, but we found the db's final commit number get a big increase from 11 12 to 49 51, which is also as expected since the total number of requests get increased.

This experiment shows our implementation is able to correctly handle requests when the workload is higher.

3.2.3 Failures: 5 clients and 5 servers with crashes

To test leader crash and make it viewable and controllable on log, I keep other part unchanged but let the ClientRequest function do one extra thing, if the receiver is a leader, it will generate a random number from 1 to 100. If the number is 100, the process will exit and generate log record. In our experiment, this setting works well, from the `02_5_servers_5_clients_crash.log` file, the crash

happened at line 10453 on server3. We can see when the crash happens, server3 is the leader, and after its crash, we can see there is a `:ELECTRON_TIMEOUT` called by server1, and another leader get generated at line 10482 that makes the processing works as usual. All following execution just the same as the normal case.

The log's data of "db updates done" also looks as expected, the other 4 servers all get the same commit number in the end, and the crashed server's state stays the same after the crash happened. Which shows our implementation can work as usual after a leader crashed.

3.2.4 Run for 60 seconds in normal case

Here what we do is just change the timelimit in normal case from 40s to 60s, and stored the log in `04_5_servers_5_clients_60s.log`. Here our observation is the db update's commit number on each server has an increase. We think this is because when the timelimit is 40s, there are some requested haven't been handled. When the timelimit increase from 40s to 60s, some extra requests have time to be committed, which makes the total number of db updates on each server increase. And this 60s version still not finish all request processing since the last db update's servers still do not have a consistent commit number, which means there are further processing needed.

3.2.5 Run with debug level 0

Here what we do is set the debug level to 0 and record the log in file: `05_5_servers_5_clients_debuglevel0.log`. Our observation on this is the processing speed becomes much faster with out the debug IO, the last db updates done's commit number is why higher than all previous experiments (330 331). This shows that Debug's IO operation takes a big part of processing power in this execution process.