Raft-based Replicated Block Store

Shichun Yu, Yuting Yan, Zhikang Hao

Outline

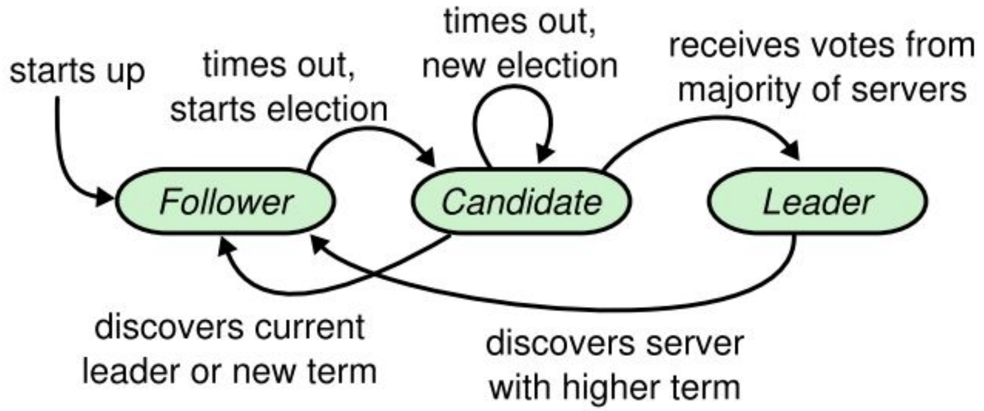
Implementation

- General architecture of Raft consensus algorithm
- Our special decisions and optimizations

Test

- Consistency Test
- Performance Test

State machine of Raft Server

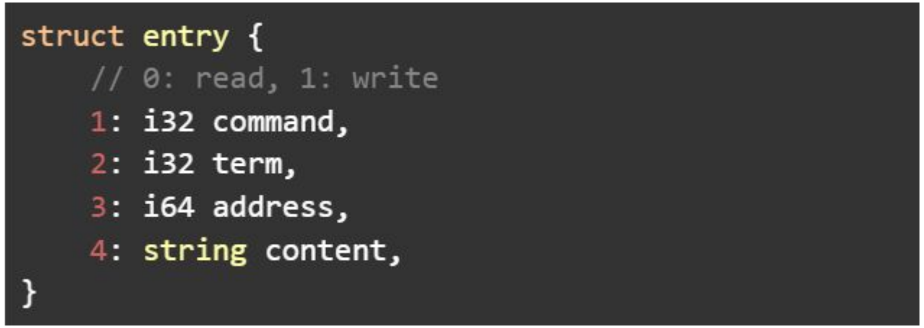

starts up
times out, starts election
times out, new election
receives votes from majority of servers
Follower
Candidate
Leader
discovers current leader or new term
discovers server with higher term

Persistent States: Term, Vote, Log[]

Volatile States: commitIndex, lastApplied, **leaderID**

Volatile State on Leader: nextIndex[], matchIndex[]

Log Entry



```
struct entry {
    // 0: read, 1: write
    1: i32 command,
    2: i32 term,
    3: i64 address,
    4: string content,
}
```

In this project, for the 4K block store requests,

the size of content will always be 4K.

Leader Election

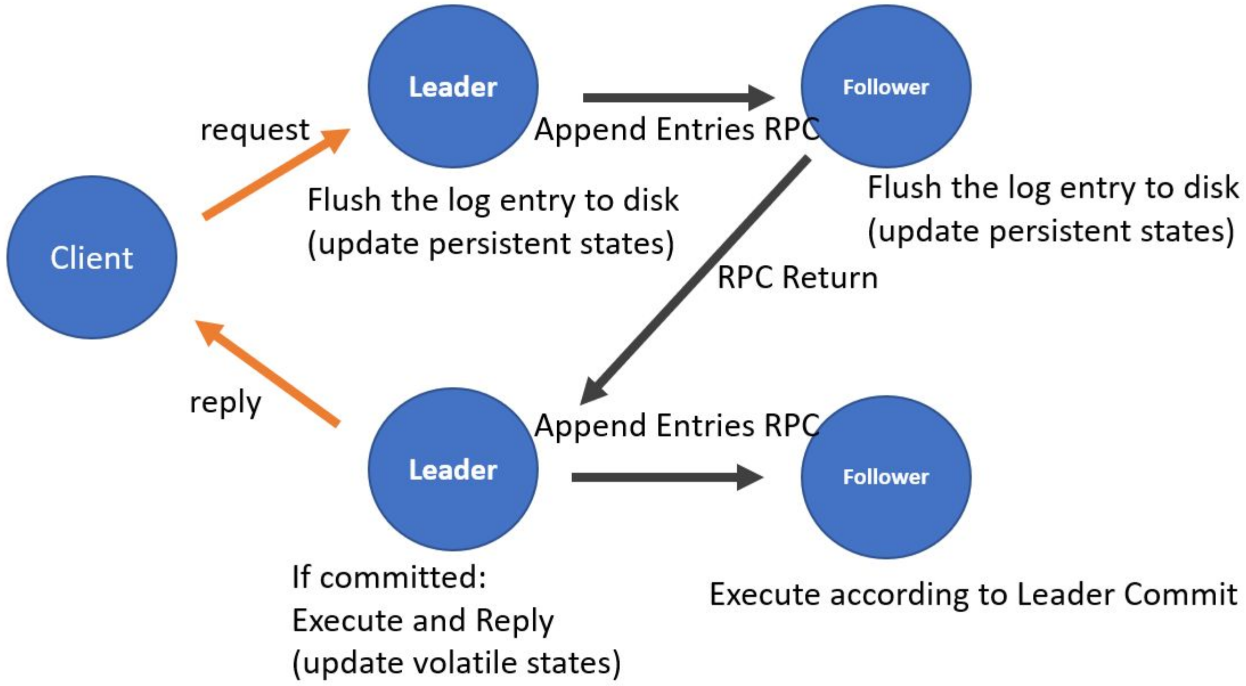
- Candidates send RequestVote RPC

To prevent the livelock, follow the guide of MIT raft project, quote

“only restart your election timer if a) you get an AppendEntries RPC from the current leader (i.e., if the term in the AppendEntries arguments is outdated, you should not reset your timer); b) you are starting an election; or c) you grant a vote to another peer.”

Ref: https://thesquareplanet.com/blog/students-guide-to-raft/

Normal Operation

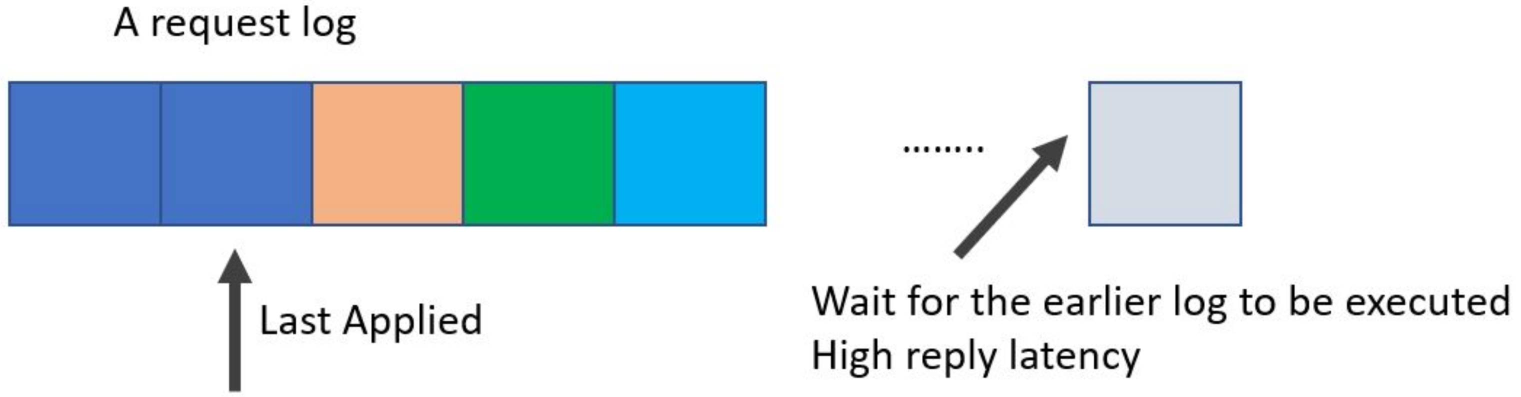


Exactly-once Semantic (not fully implemented)

- Read logs are also replicated through the servers.
- Every request has a sequence number (monotonically non-decreasing for each client).
- Servers keep the completion record (a map from <client id, seq num> to execution result), but the CRs are not persistent.
- The Servers keep track of the largest seq number for each client, and if a request with lower seq num comes, reply immediately without executing twice.   (**only implemented in the leader**)

Performance optimization

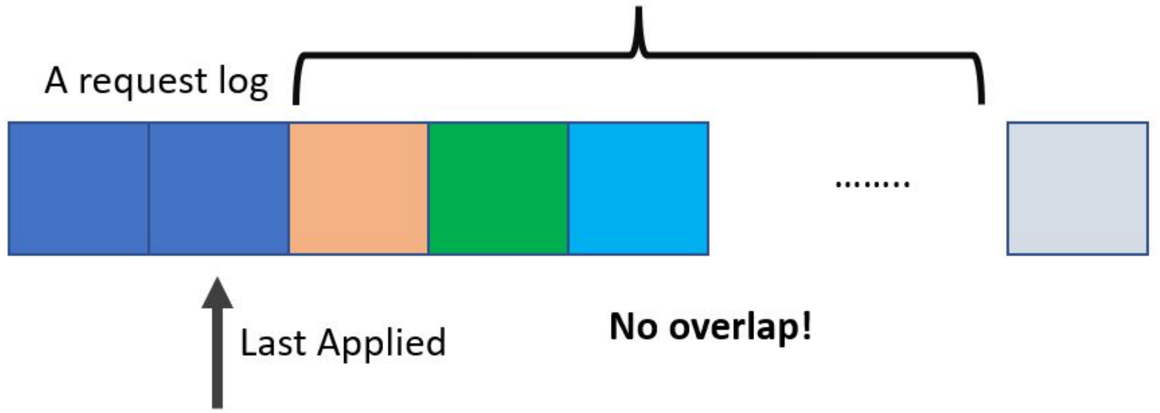
How to improve the latency when having large amount of requests?



Simple latency optimization

Sometimes there is no need to wait!

Execute and reply immediately if there is no overlap to the requests come earlier.



Insert the executed request log index to a set.

Update “lastApplied” accroding to this set.

Test RPCs support

Implement extra RPCs to make debug and test easier.

We implemented Two RPCs

to make comparison of states between leader and followers.

- CompareStates (Log, term, vote)
- CompareBlockStore (compare the persistent Block Store)

Correctness Tests

Case 1: follower crash

Proof: resist one crash

& block any modification with 2

crashes (Leader completeness)

Recovered follower will be

Consistent after several seconds

Method: manually crash servers

Correctness Tests

Case 2: General Leader crash case

Proof: new leader will be elected

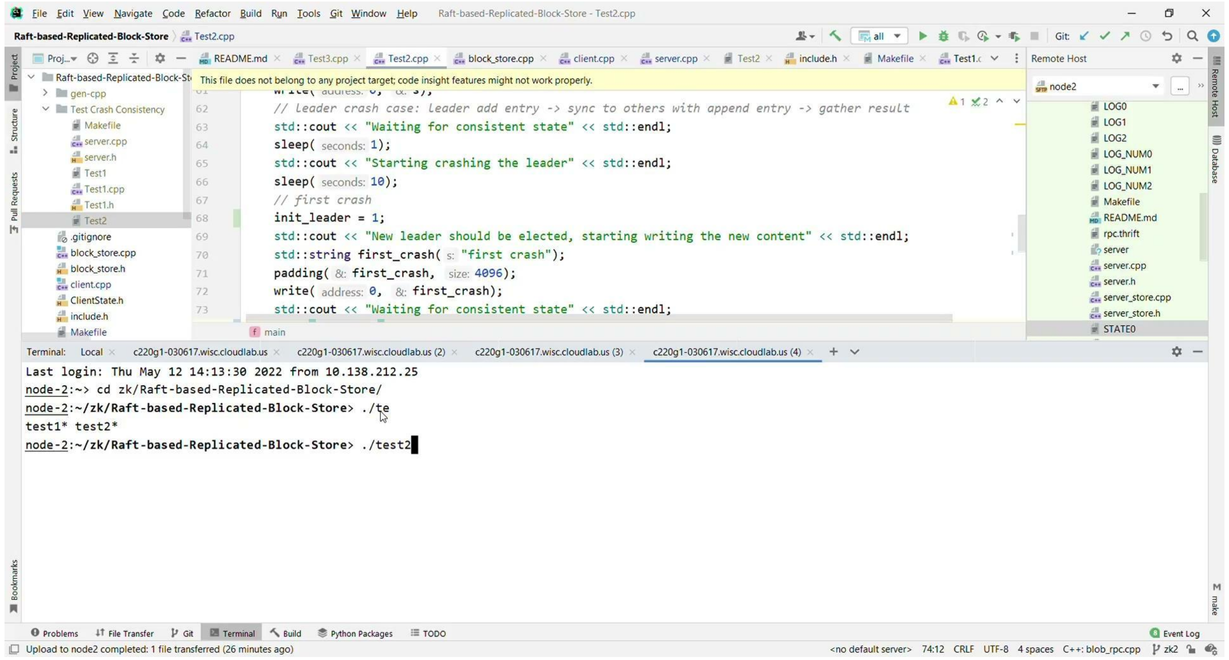
Logs in other servers keeps consistent

Recovered server will become follower (we update terms)

Logs are consistent after and state machine is up-to-date

Correctness Tests

Video



This file does not belong to any project target; code insight features might not work properly.

```cpp
    // leader crash case: leader add entry -> sync to others with append entry -> gather result
    std::cout << "Waiting for consistent state" << std::endl;
    sleep( seconds: 1);
    std::cout << "Starting crashing the leader" << std::endl;
    sleep( seconds: 10);
    // first crash
    init_leader = 1;
    std::cout << "New leader should be elected, starting writing the new content" << std::endl;
    std::string first_crash( s: "first crash");
    padding( &: first_crash,  size: 4096);
    write( address: 0,  &: first_crash);
    std::cout << "Waiting for consistent state" << std::endl;
```

```
Last login: Thu May 12 14:13:30 2022 from 10.138.212.25
node-2:~> cd zk/Raft-based-Replicated-Block-Store/
node-2:~/zk/Raft-based-Replicated-Block-Store> ./te
test1* test2*
node-2:~/zk/Raft-based-Replicated-Block-Store> ./test2
```

Upload to node2 completed: 1 file transferred (26 minutes ago)

Correctness Tests

Case 3: Election behavior

Proof: the server with highest vote priority will be the leader

Performance Tests

Environment:

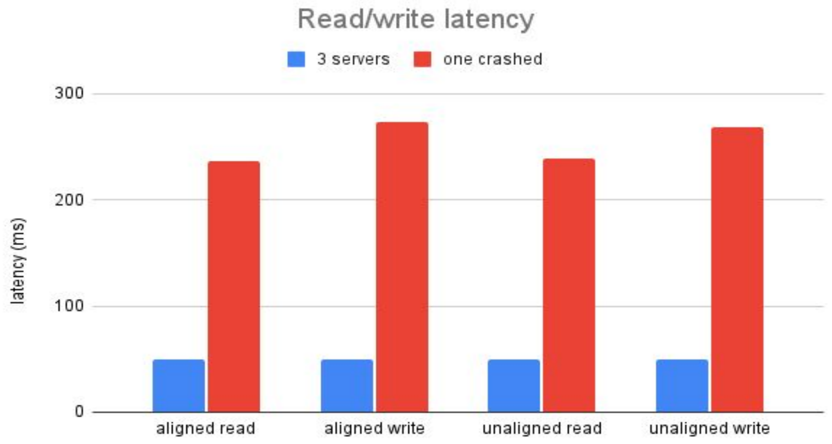
4 cloudlab c220g1 linux servers, 1Gb cable connection

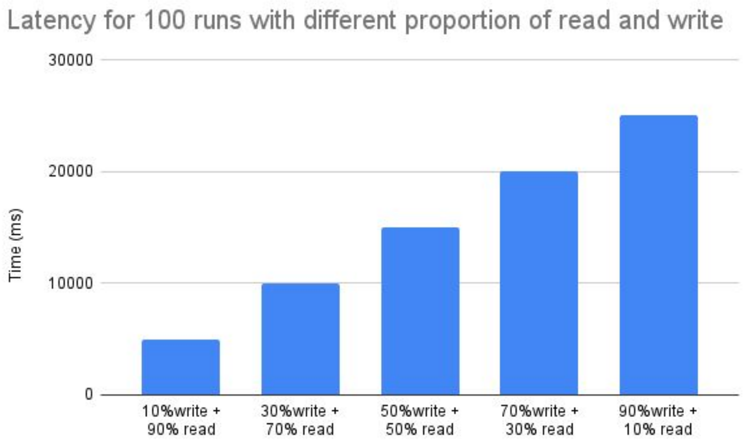
3 distinct raft nodes

1 distinct client node

Performance Tests

- Read/Write latency, appendEntry interval with 50ms

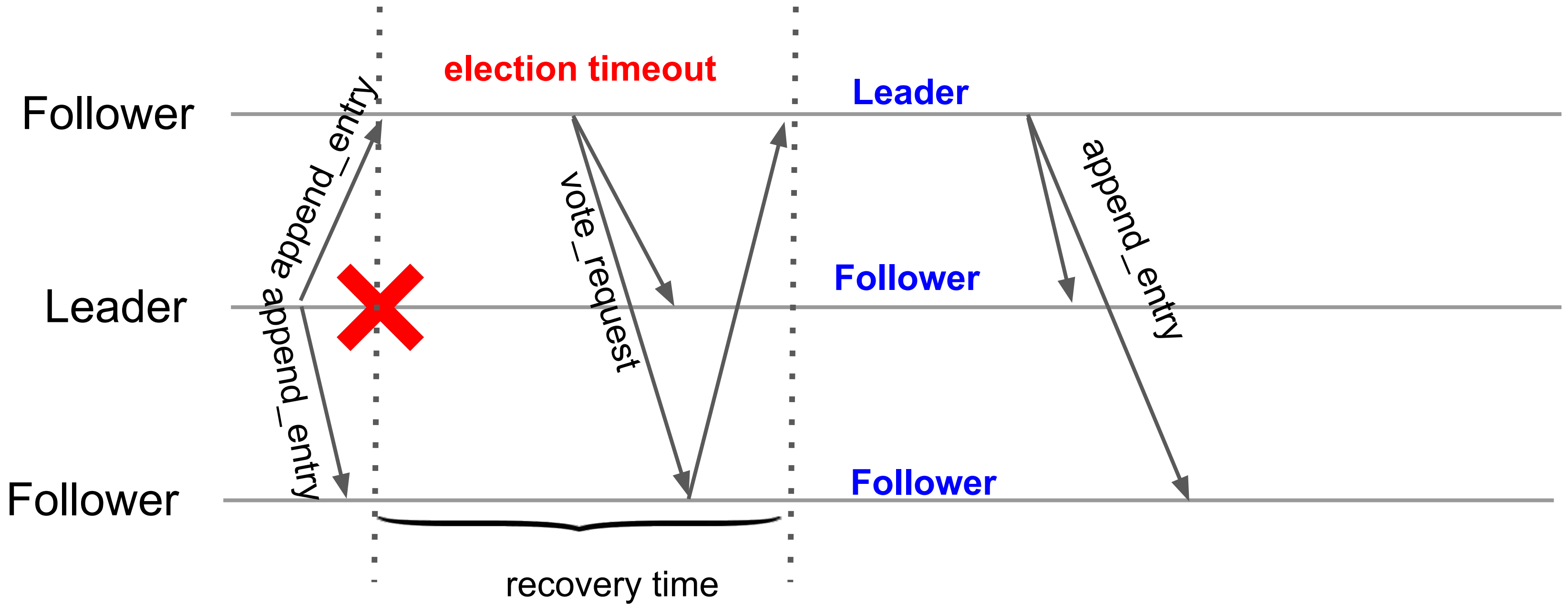




- Aligned operation or unaligned operation make no much difference in latency;
- Crashed server in a cluster would increase latency;
- Write latency is larger than read.

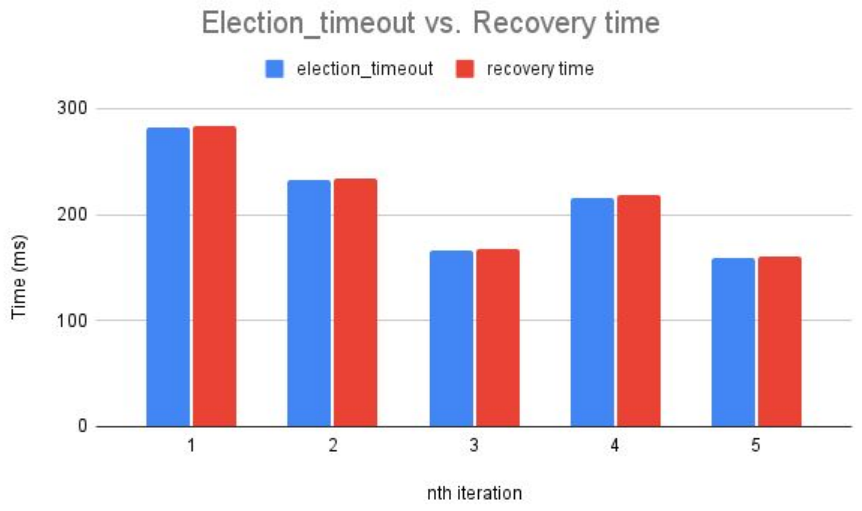
Performance Tests

- Crash Recovery process



Performance Tests

- Recovery Time



- Recovery time is highly related to election timeout.

Conclusion

- Write latency is larger than read;
- A failed server will impact the performance of system.
- The recovery time is highly related to server’s election timeout.

Thank you!

Github Repo:
https://github.com/SmileIsThinking/Raft-based-Replicated-Block-Store