Replicated Database using Raft

Group: 14

- Madhushree Nijagal
- Priyavarshini Murugan
- Tanisha Hegde

Raft Server – Design and Architecture

•1 Client, 3 Servers

DB Server (3 nodes):

•LevelDB - File Based NoSQL Database

•Operation supported:

Get(key)

Put(key, value)

Raft Server (3 nodes):

- All servers start as Followers with term=0
- Followers convert to Candidates at randomized time and Leader is selected
- Client requests Read and Write operations only to the Leader
- Leader replicates the commands on majority of Servers

Raft Server – Logs & Communication

Communication

AppendEntries RPC

- Leader sends Heartbeats (empty AppendEntries RPC) to Follower to indicate liveness
- Leader sends AppendEntries RPC to Follower with new log entries for data replication

RequestVote RPC

- On election timeout, Followers become Candidates and send RequestVote RPC

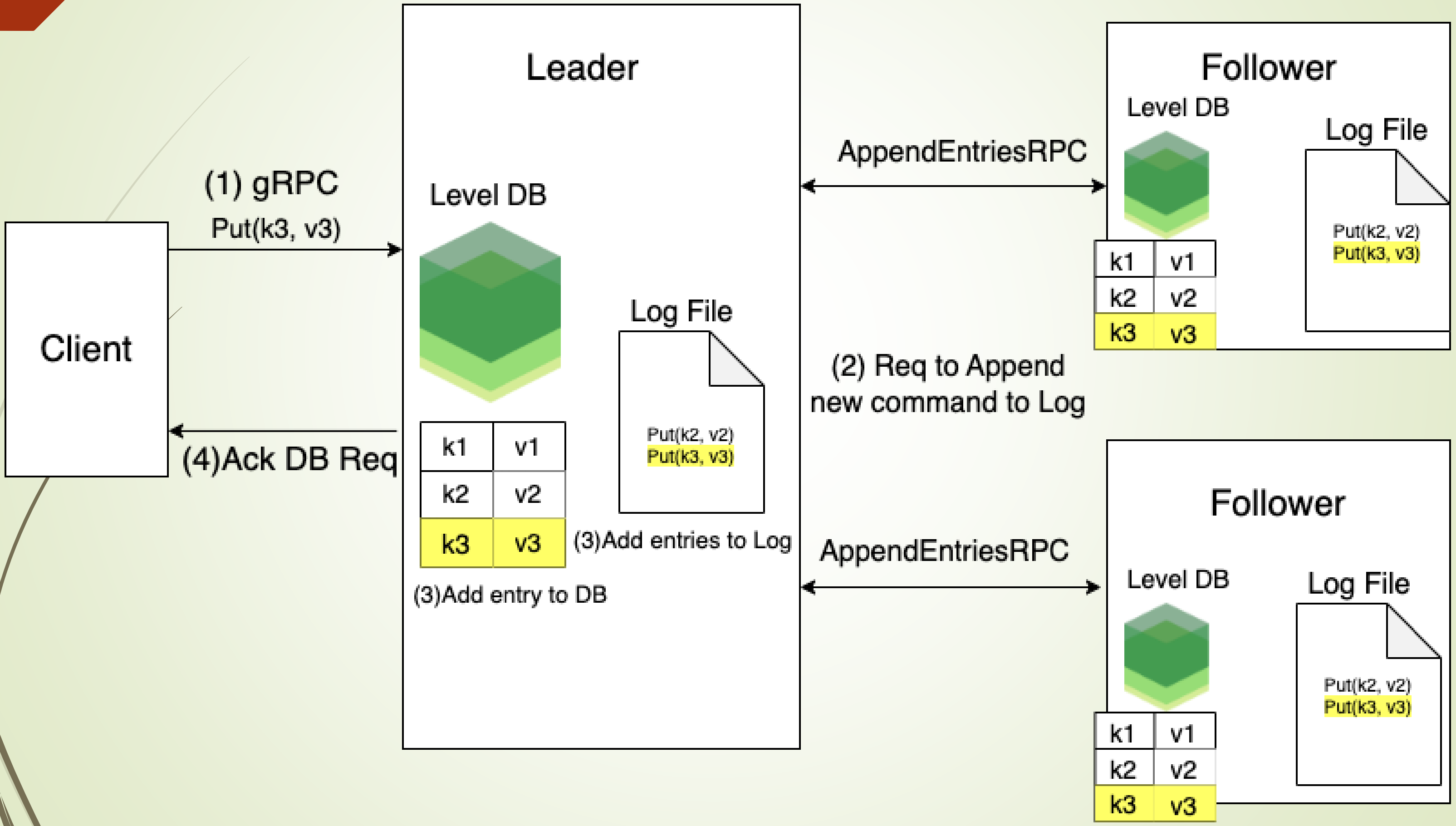
gRPC Request

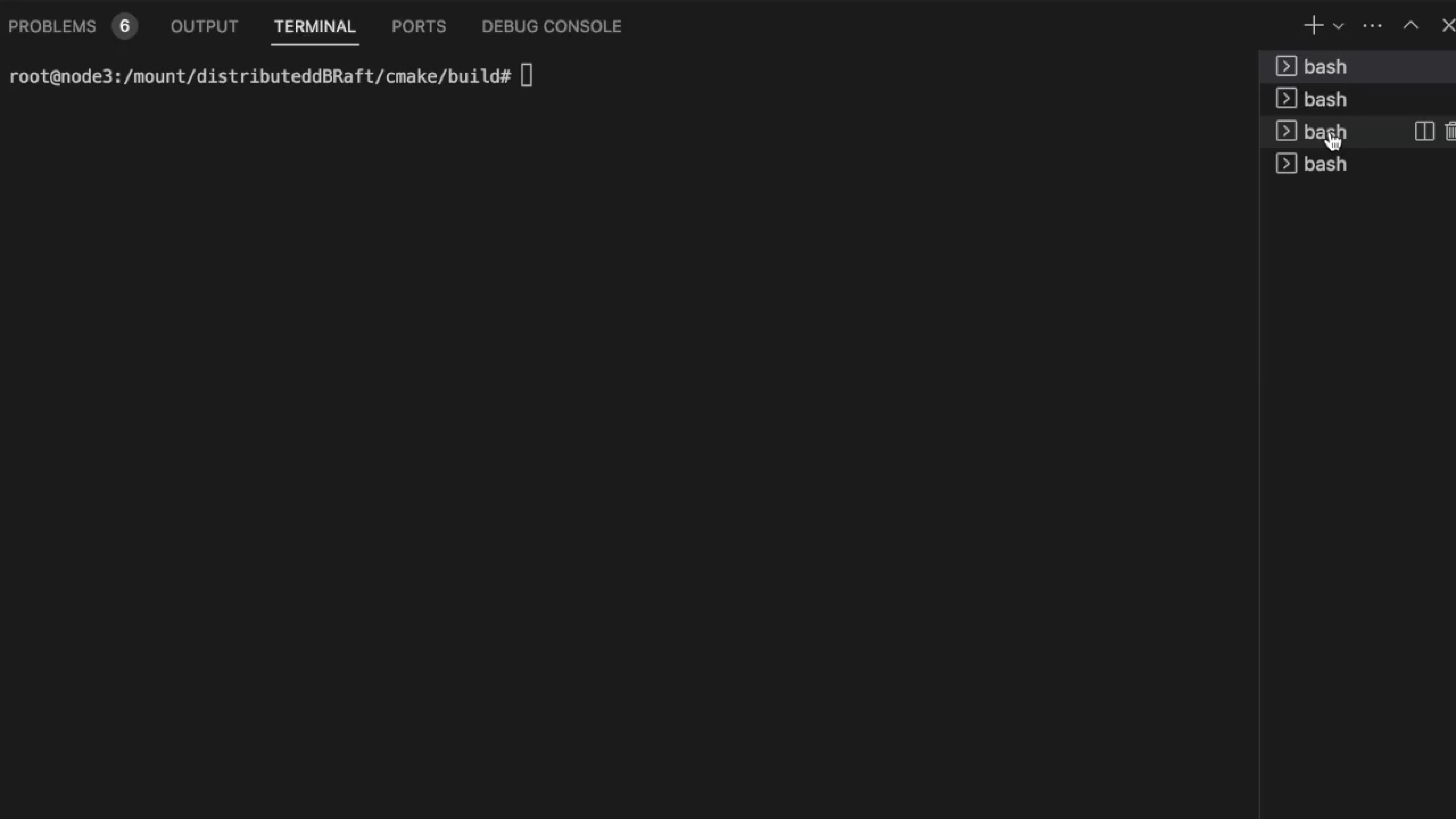
- Client send Read and Write Request to Leader

Logs

- Each raft server (Leader & Followers) have their own Log files that maintain all client Put requests.
- Followers truncates conflicting entries from own and appends Leader log entries to its own log
- Applies them to their own storage.

Design and Flow





```
root@node3:/mount/distributeddBRaft/cmake/build# 
```

Test Cases

	node1	node2	node3	Leader	Log results	Client
1	Follower (assume leader after election)	Follower	Follower	Election decides who becomes the leader	Leader(n1) logs replicated in followers (n2,n3)	Directed to the leader
2	Leader	Follower	Follower	n1 continues as leader	Leader(n1) logs replicated in followers (n2,n3)	Directed to the leader
2.a	Leader(crash)	Follower(assume leader after election)	Follower	Election start between n2 and n3 and next leader is chosen	Leader (n2) logs replicated in followers (n3)	Retries until leader is elected
2.a.1	Leader(crash and restart as leader AFTER n2 is chosen as the current leader)	Follower (assume next leader)	Follower	[1] Election starts between n2 and n3 and timeout decides the next leader [2] n1 becomes follower even if restarted as leader.	[1] Leader(n2) logs replicated in followers (n3) [2] n1's logs made consistent with the leader(n2)	Requests are redirected from n1 to n2
2.a.2	Leader(crash and restart as leader BEFORE election)	Follower	Follower	n1 continues as leader	Leader(n1) logs replicated in followers (n2,n3)	Requests are still sent to n1
2.b	Leader	Follower	Follower(crash)	n1 continues as leader	Leader(n1) logs replicated in followers (n2)	Requests are still sent to n1

Statistics & Performance

- Election Timeout = 5s + random (0-100ms)
- Leader heartbeats = every 1s
- Wait for votes during election = 10ms
- Writing (put) 4kB data 1000 times followed by 2kB data 1000 times from client = ~15s for 4kB ~21s for 2kB => Log size increase
- Reading (get) 4kB data 1000 times followed by 2kB data 1000 times from client = ~17s for 4kB ~22s for 2kB
- Servers on local nodes => complete flow with 300 times 20 bytes = ~3ms
- Servers on different nodes => complete flow 300 times 20 bytes = ~7ms
- Leader Log size: 2Mb , Time taken for a restarted follower to make its logs consistent = ~5s