Replicated Block Store using Raft

CS 739, Project - 4, Group 5

Aakarsh Agarwal
Muhammad Asad Khan
Bijan Tabatabai
Salman Munaf

(Dept. of Computer Sciences/CDIS)

Table of Contents

- Design decisions
  - Replication strategy - Raft - Leader election, log replication
  - Durability
  - Walkthrough
  - Crash recovery protocol
- Performance
  - Experimental setup
  - Read perf - 3 nodes vs 5 nodes
  - Write perf - 3 nodes vs 5 nodes
- Demos

Design decisions - Replication Strategy

- Raft (3/5 server nodes, can tolerate f=1/2 failures)
  - 3/5 servers - all are started as **Followers** with initial term = 0
  - **Followers** change to **Candidates** using randomized election timeouts, elects Leader
  - Client sends read and write requests only to **Leader**
  - Reads involve **Leader** reading from its disk and sending data back to the client
  - Writes involve **Leader** replicating on majority of servers -> committing to its state machine -> making Followers commit, piggybacked on successive heartbeats

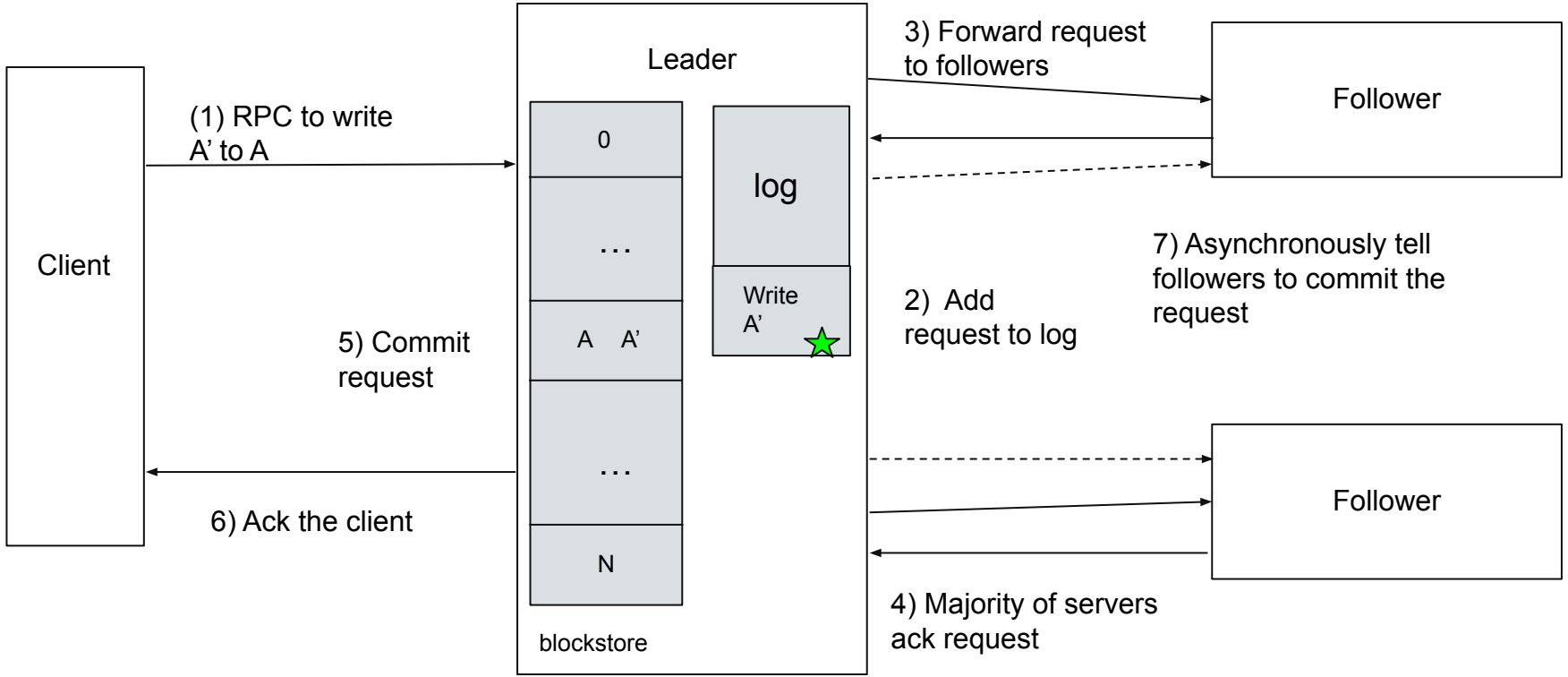
Design decisions - Communications

- Communication from Leaders to Followers/Candidates
  - AppendEntries
    - Heartbeats: Followers/Candidates use election timeout to identify Leader crash
    - Copy/Replication of data: Log/write data is copied to the other server and other server acknowledges it after copy has been made
- Communication from Candidates to Followers
  - Request Vote
- Communication between Client and Server
  - Read request
  - Write request

Design decisions - Durability

- Data stored in single file
- Undo logging
  - Before overwriting data, save old data into undo file
  - When write completes, delete undo file
  - If undo file exists on server startup, apply undo to data
- Reader/Writer locks serialize writes to 4K aligned blocks of data
  - If write is unaligned, must grab two locks
- Raft log persisted in single file
  - Simplifies undo logging - truncate to offset

Design decisions - Walkthrough of normal write



Design decisions - Crash Recovery Protocol - Crash identification

- Leader sends heartbeat to all nodes periodically
- If Follower does not receive heartbeat from Leader for a set election timeout (different and random per node), Leader crash is identified and election starts for a new term
- If Leader does not get back response from a node, Follower node crash is identified. However, no change in the working. Leader keeps on trying periodically

Design decisions - Crash Recovery Protocol (Leader crashes)

- Followers wait for heartbeats till election timeout (randomized timers across nodes), move to Candidates and start election
- Candidates request vote for a particular term, all nodes vote for only server per term number
- Becomes leader if gets majority of **Yes** votes
- **Yes** vote - If receiving node current term is not more than requesting node, has not voted for current term, log is as up-to-date as receiving node
- New Leader keeps on sending heartbeats to the crashed node
- Once crashed node is live, new Leader sends the new log entries at once, with commit information

Design decisions - Crash Recovery Protocol (Follower/Candidate crashes)

- No change in reads handled by Leader
- Leader keeps on sending heartbeats to identify the node reboot
- Once node is live, Leader sends the new log entries at once, with commit information

Design decisions - Crash Recovery Protocol (Client library)

- Client needs to hide crash from user
- Client makes the initial call to any one node (ideally Server 1)
- Client received Leader id and sends request to the Leader node
- Possible responses
  - Blockstore_Success - Client called Leader and request was executed successfully
  - Blockstore_Not_Primary - Client called Follower. Client will now call the other Leader
  - Failure/Crash Case
    - Client call to Leader fails
    - Client makes call to other node
      - Blockstore_Success - If node is Leader and request was executed successfully
      - Blockstore_Not_Primary - Client called Follower. Client will now call the other Leader

Experimental Setup

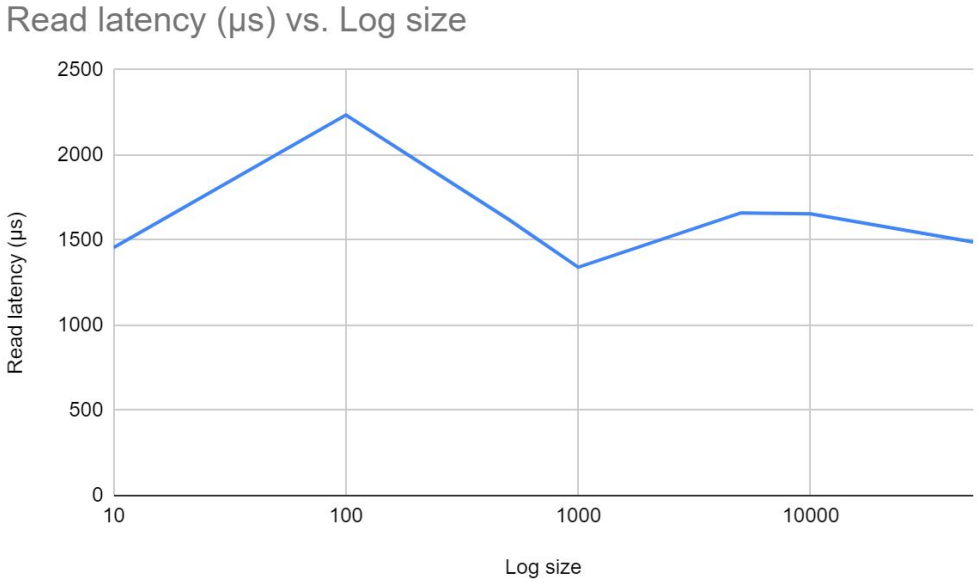
- For latency measurements, 6 Cloudlab C220g5 nodes connected on a 1Gbps network link
  - 5 servers ran on different nodes, 1 node dedicated to client

- For crash demonstrations, servers and client were run on one machine
  - Not ideal, but we had a limited number of machines and did not want to interfere with performance results

Read latency

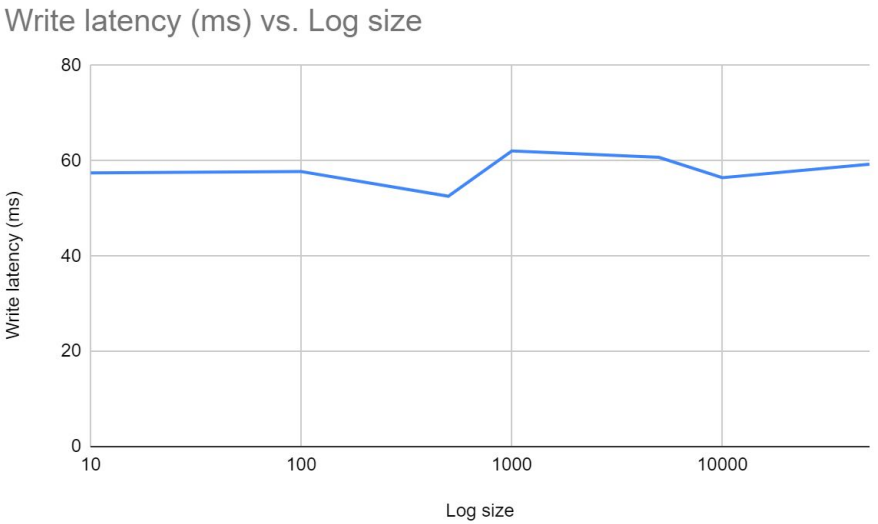
- The average READ latency on 5 servers was 1826.222 microseconds

- The average READ latency on 3 servers was 1806.112 microseconds



Write latency

- Average write latency on 5 servers of 10000 runs: 54.987ms
- Average write latency on 3 servers of 10000 runs: 52.811ms



Read latency with crashes

- The average READ latency in case of follower crashes remain the same
- The average READ latency in case of leader crashes increases as election timeout happens and a new leader is elected
- The failover time is on average 5.71s in case of 3 servers.

Write latency with crashes

- The average WRITE latency in case of follower crashes remain the same
- The average WRITE latency in case of leader crashes increases as election timeout happens and a new leader is elected
- The failover time is similar to previous slides which is on average 5.72s.

On to the fun part..
Demo!

Thank You!

Q/A?


WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON