Introduction to Consensus

Ashvin Goel

Electrical and Computer Engineering
University of Toronto

Distributed Systems
ECE419

Overview

- Motivation for consensus
- What is consensus
- Intuition for consensus

Review

- We have looked at two replication schemes based on FIFO-total order broadcast
  - Primary-backup replication
  - State machine replication
- We have seen that FIFO-total order broadcast can be implemented by using a leader
  - One node is designated as leader (sequencer)
  - To broadcast message, node sends it to the leader
  - Leader broadcasts it via FIFO broadcast
    - Ensures FIFO-total order broadcast
  - Recall we assumed that the leader does not crash

How can total order broadcast be made fault tolerant?

- Leader is a single point of failure
  - If leader fails, broadcast stops! No more replication.
- Option 1: handle leader failure manually
  - An operator can designate another node as leader,
    and reconfigure other nodes to use the new leader
  - Works well for planned maintenance, e.g., software updates
  - Problem: when leader fails suddenly, manual failover takes time
- Option 2: use external view server
  - Used it for primary-backup replication to handle primary failure
  - Problem: what happens if view server fails, use Option 1?

Automating fault tolerance

- Leader is a single point of failure
  - If leader fails, broadcast stops! No more replication.
- Option 3: ideally, automate leader switching
  - Straw man solution:
    - Let’s say we have two servers S1 and S2
    - If both are up, then S1 is leader
    - if S2 sees S1 is down, S2 takes over as leader
    - What could go wrong?
    - Network partition ... split brain!
      - Hard to distinguish between “server down” and “network down”
      - This seemed hard to solve for a long time…
  - How can we solve this problem?
    - Using consensus algorithms

What is consensus?

- A set of nodes need to agree on a single data value, e.g., a single leader, in the presence of failures
  - Each node may propose a value
  - A consensus algorithm decides on one of those values
- Requirements
  - Agreement: No two correct nodes decide differently
  - Integrity: No node decides twice
  - Validity: Any value decided was proposed by some node
  - Termination: Each correct node eventually decides a value

correctness (safety)

progress (liveness)

Consensus vs. total order broadcast

- Consensus and total order broadcast are equivalent
- When nodes need to broadcast messages in total order:
  - Use consensus to decide on first message to deliver
    - All nodes will deliver this message first
  - Do the same thing for second, third, …, messages
    - All messages are delivered to nodes in the same order
- Common consensus algorithms:
  - Paxos: single-value consensus
  - Multi-Paxos: generalization to total order broadcast
  - Raft, Viewstamped Replication, Zab: FIFO-total order broadcast

Consensus system model

- Paxos, Raft, etc. assume best-effort links, partially synchronous, crash-recovery system model
- There are also consensus algorithms for insecure links, partially synchronous, byzantine failure system model
  - E.g., PBFT, blockchain, discussed later
- Why not asynchronous?
  - Cannot use timeouts to detect failures
  - FLP result: no deterministic consensus algorithm is guaranteed to terminate in an asynchronous crash-stop system model

Consensus challenges

- In a partially synchronous system, we can use timeouts to eventually detect failures, but no bounds on delays
  - Safety: Need to ensure correctness without depending on timing
  - Liveness: Need to ensure progress even when failure detection is potentially incorrect

Intuition for consensus

- Multi-Paxos, Raft, etc. are leader-based schemes
- Key safety requirement: only one leader
- Since a leader can fail, we will weaken this requirement
  - There should only be one leader at any time
- How to ensure only one leader at a time without depending on timing or correct failure detection?
  - Aren’t we back to the leader switching problem?
- Key idea: use majority voting to elect a leader
  - With majority voting, only one leader can be elected at a time
  - Avoids dependence on correct failure detection for correctness!

Understanding majority voting

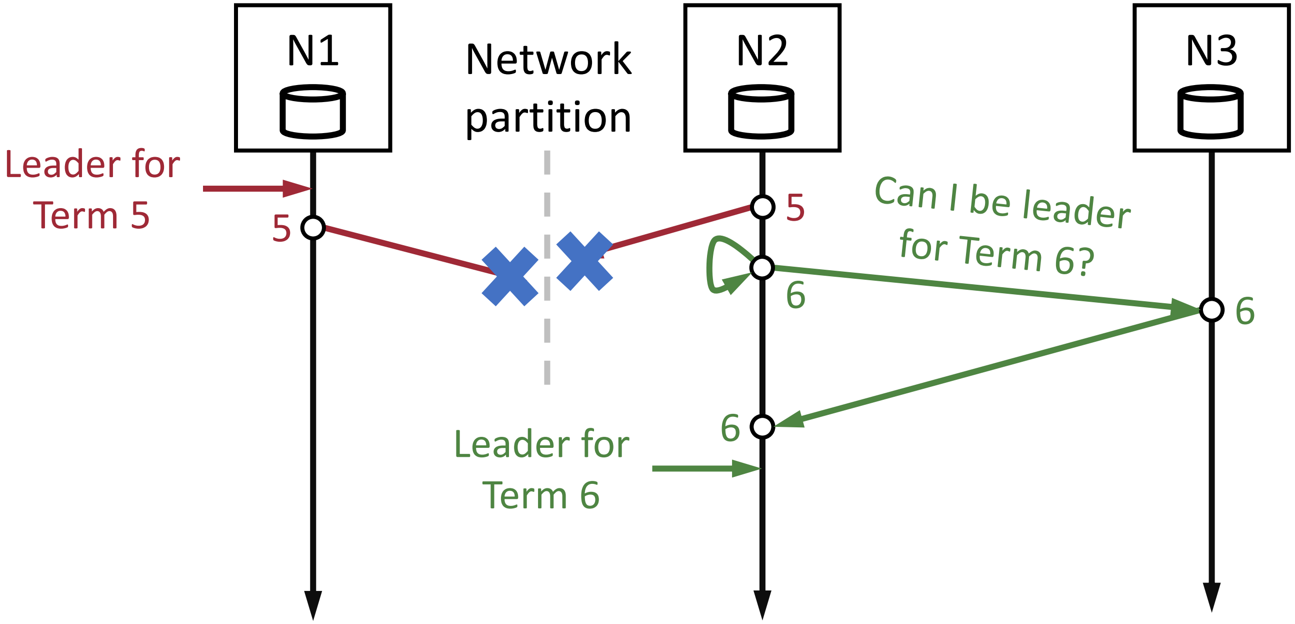
- Majority is based on all servers, not live servers
  - With 3 servers, majority is 2 servers
- If no majority, then wait
  - With 3 servers, if 2 have crashed, then wait for 1 to come back
- If majority available, then proceed
  - With 2f+1 nodes, allows dealing with f failures
- Key property of majority is that any two intersect
  - Allows conveying most recent information about voting process
- Ok, back to electing a leader

Leader election

- Nodes use failure detector to detect crashed leader
  - E.g., based on timeout (no message from leader for some time)
- When a node detects (potentially) crashed leader,
  it becomes candidate (for leadership)
- Candidate starts election by
  - Incrementing a counter (e.g., Raft term) to indicate new election
    - A term lasts until the next election
  - Asking other nodes to vote to accept it as new leader for the term
- If majority vote for candidate, it is elected for the term
  - Other nodes only vote at most once per term (or election)
  - Due to majority, two leaders cannot be elected for same term

Have we ensured one leader?

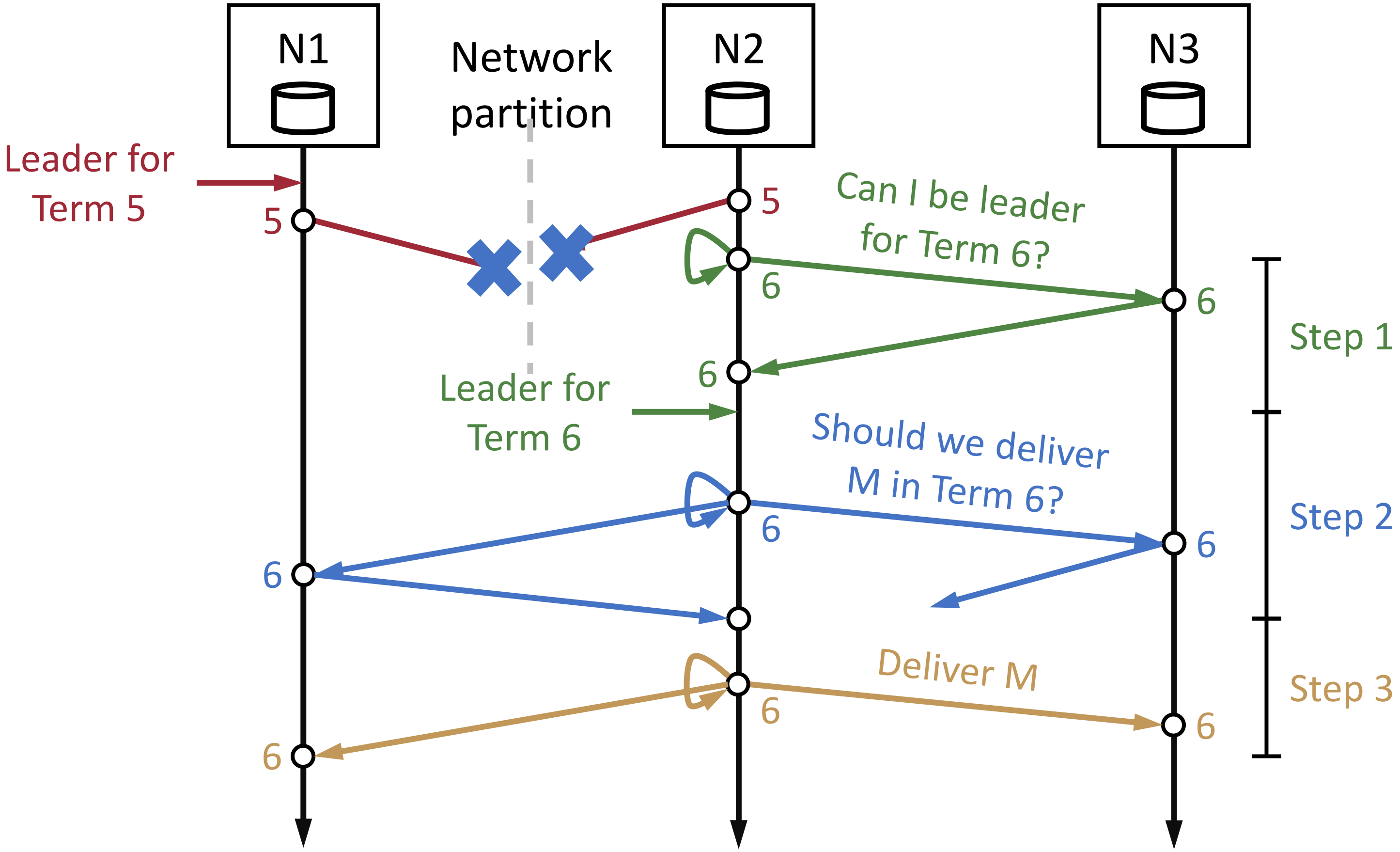
- Leader election guarantees (at most) one leader per term
- But failure detection is imprecise
    - Leader from another previous term may still be running
    - Cannot prevent multiple leaders from different terms



Am I a leader?

- Only leader of latest term must serve as leader when deciding a value, e.g, delivering next message
    - But a leader for a new term can be elected at any time!
- How can a replica check whether it is leader of latest term?
- Once again, ask a majority …
- Why does this work?
    - Since leader election requires a majority vote, there will be at least one node that will know if a new leader has been elected
    - Doesn’t depend on timing!

Leader asks majority before delivering message



Conclusions

- Consensus: set of nodes need to agree on single value
    - E.g., a single leader at a time
- Challenge: correct failure sensing is not possible
    - E.g., if we use a timeout to detect and remove a faulty leader, it may still believe it is a leader
    - Need to ensure correctness and progress without depending on correct failure sensing
- Solution: get permission from majority of participants
    - Avoids split brain issues, since two majorities not possible
        - Leader-based scheme: get majority when 1) electing leader, 2) delivering messages. Intersection property ensures one leader at a time delivers messages in order.
    - f failures possible with 2f+1 participants, good availability