THE UNIVERSITY *of* EDINBURGH

Distributed Systems Fall 2025

Yuvraj Patel

Today's Agenda

Exam Format

Exam Preparation

Revision

Kafka

Exam Format

Exam Preparation

Consensus with Paxos

Two phases – Prepare & Accept

Prepare Phase

- Find out any chosen values so far
- Block older and uncompleted proposals

Accept Phase

- Inform acceptors to accept a specific value

Algorithm – Prepare Phase

Proposer

- Choose proposal number n, send <prepare, n> to acceptors

Acceptor

- Only receiving a prepare message
  - If $n > n_h$, where n_h is the highest proposal seen so far by the acceptor

    $n_h = n$. (Promise to not accept older proposals)

    If no prior proposal accepted,

    reply <promise, n, NULL>

    Else

    reply <promise, n, (n_a, v_a)>
  - Else

    Reply <prepare-failed>

Algorithm – Accept Phase

Proposer

- If receive promise from majority of the acceptors,
  Determine any earlier chosen value v_a for n_a and choose latest value or any value v selected by the proposer
  send <accept, n, v> to acceptors

Acceptors

- If n >= n_h
  $n_a = n_h = n$
  $v_a = v$
- reply <accept, n_h>

Proposer

- When responses received from the majority
  If any $n_h > n$
  Start from prepare phase again
  Else
  Value is chosen

Paxos flow

Raft Basics

A node can be either follower, candidate, or leader

Raft divides time into terms of arbitrary length; terms are numbered consecutive integers

Each term begins with an election, where one or more candidates attempts to become a leader

- Two possible outcomes of an election – leader elected or split vote

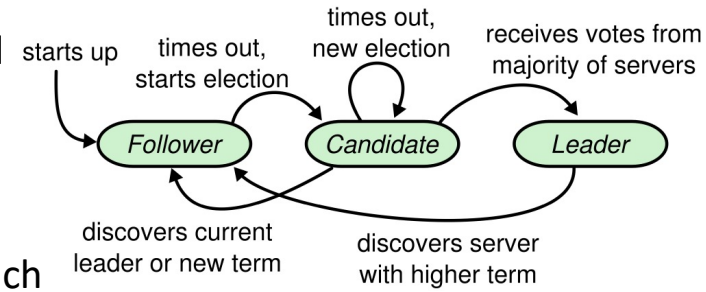
Term acts as a logical clock and helps detect obsolete information such as stale leaders

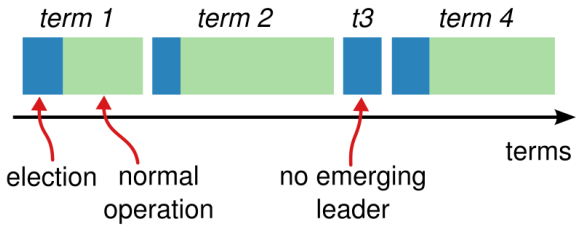
Each node stores a current term number, increases monotonically

Current terms exchanged while normal communication

- One node's current term smaller than others, it updates it term to larger value
- If leader/candidate discovers its term is out of date; revert to follower role

If node receives a request with a stale term number, reject the request





High-Level Understanding

Log entries over time

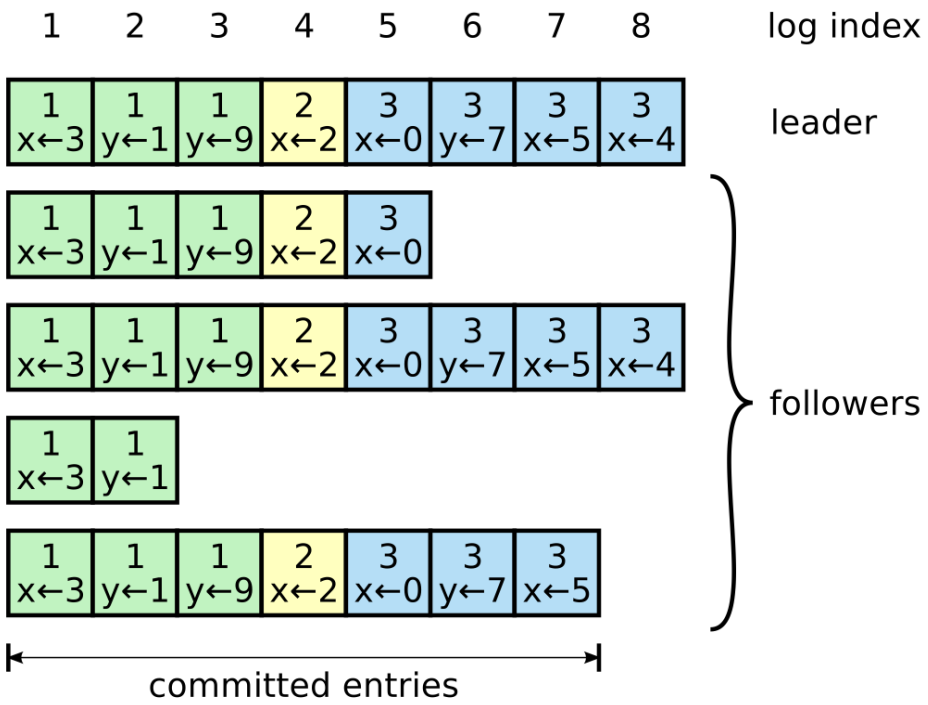
A leader's log is the ultimate truth

While election, ensure that the leader has all committed entries

Leader keeps track of each follower's log

Leader ensures all followers are up to date

- Either remove uncommitted log entries or append to log entries



Mutual Exclusion & Concurrency

Concurrency leads to non-deterministic behavior

- Different results even with same inputs

Race conditions: Specific type of bug

- Sometimes program works fine, sometimes it doesn't; depends on timing

Want to execute instructions as an uninterruptable group

- Want them to be atomic; appears that all execute at once, or none execute

Uninterruptable group of code is called critical section

Mutual exclusion for critical sections

- If thread A is in critical section C, thread B isn't
- It is fine if other threads do unrelated work

Distributed Locks

Cannot share local lock variables

Mutual exclusion in a distributed system

Central Solution

- Elect a central leader using election algorithm
- Leader keeps a queue of waiting requests from nodes who wish to access

Decentralized approach

- All nodes involved in the decision making of who should access the resource
- Ricart-Agrawala Algorithm – Use the notion of causality – rely on logical timestamps
- Token Ring Algorithm -- All nodes arranged in a ring fashion; Use token as a means of ownership

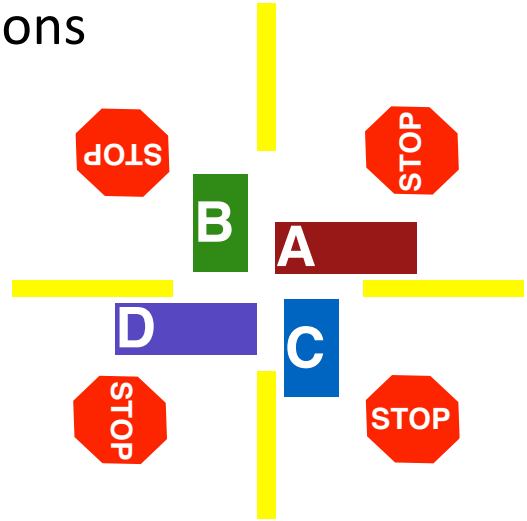
Deadlocks

No progress can be made because two or more nodes are each waiting for another to take some action and thus each never does

Deadlocks can only happen with these four conditions

1. Mutual Exclusion
2. Hold-and-wait
3. No preemption
4. Circular Wait

Can eliminate deadlock by eliminating any one condition



Kafka

END OF LECTURES
Good luck with the exam