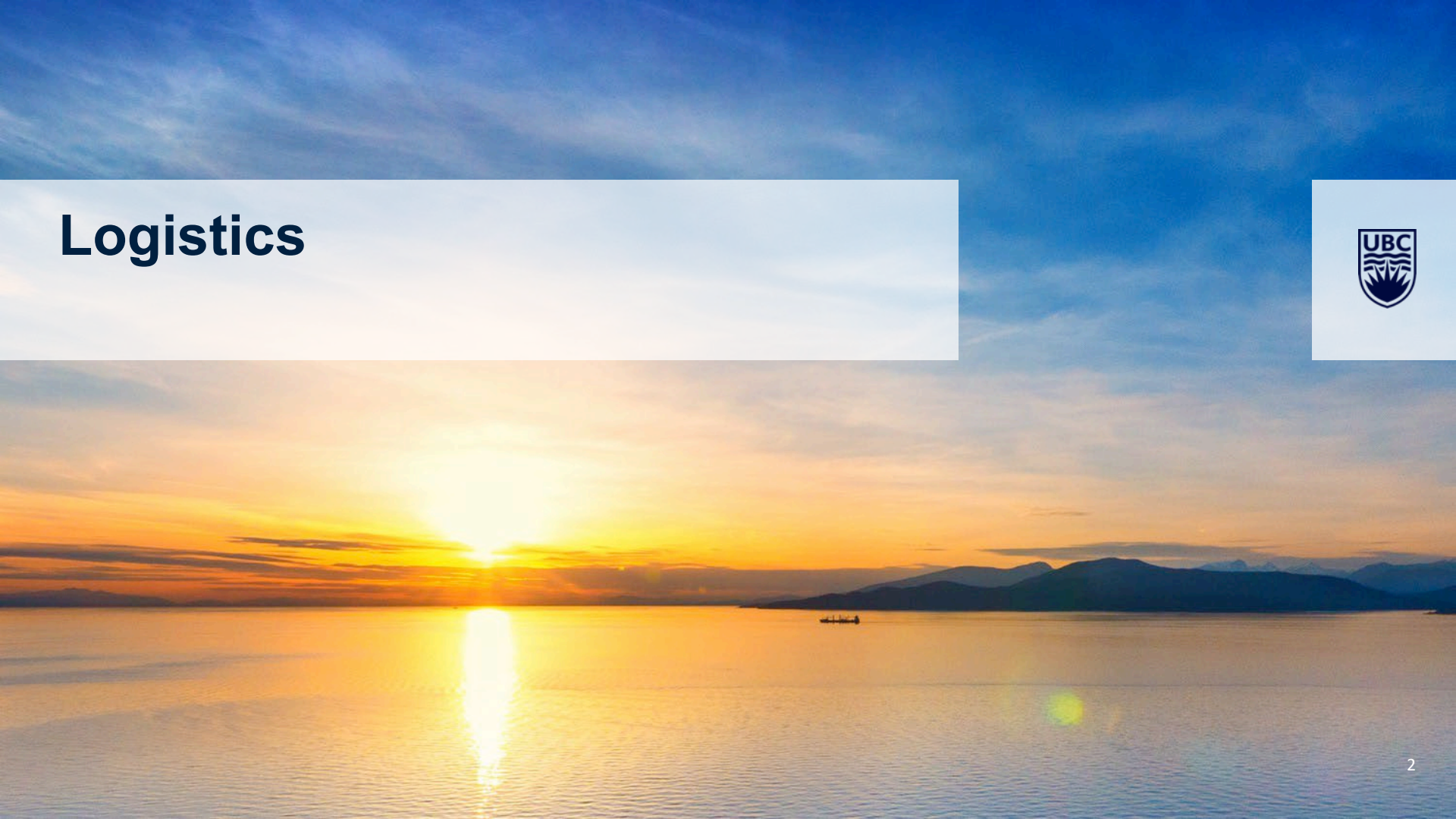
CPSC 416 Distributed Systems

Winter 2022 Term 2 (February 28, 2023)

Tony Mason (fsgeek@cs.ubc.ca), Lecturer

Logistics



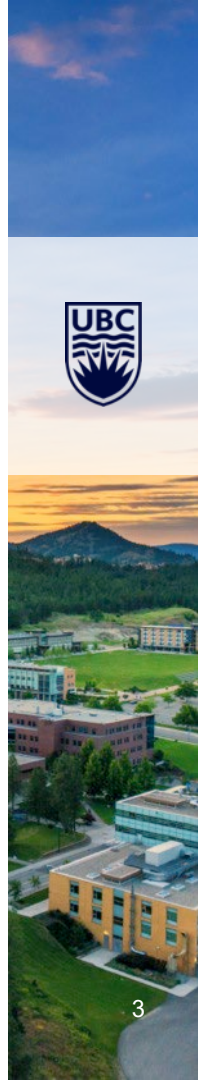
Deadlines

Project 3 Released. Late Deadline: April 13, 2023. Report Grades Pending.

Project 4 Released. Initially Due: March 13, 2023

Project 5 Released Due: April 13, 2023

All project work is due April 13, 2023. Late projects have a 75% score cap.



Deadlines

Alternate Path 1 & 2: Review in progress

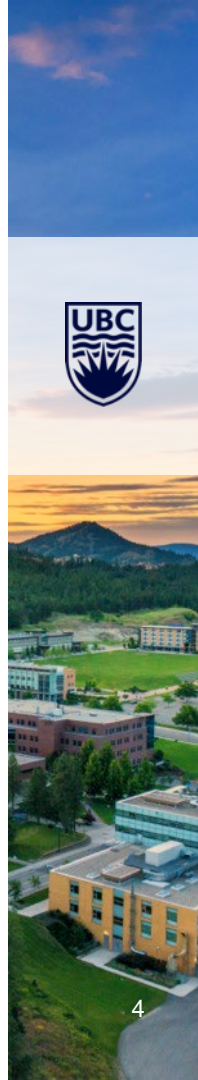
- Piazza private threads need TLC
  - **Weekly updates due each Monday @ 23:59 PT**

Instructor Office Hours:

- Zoom Office Hours (Tuesday) @ 13:00-14:00
- Discord (Casual) Office Hours (Thursday) @ 14:00-15:00

TA Office Hours:

- Eric: Friday 9-11 am (in-person and Zoom)
- Japraj: Wednesday 3-5 pm (Zoom)
- Yennis: Thursday 2-4 (Zoom), Friday 2-4 (in-person)

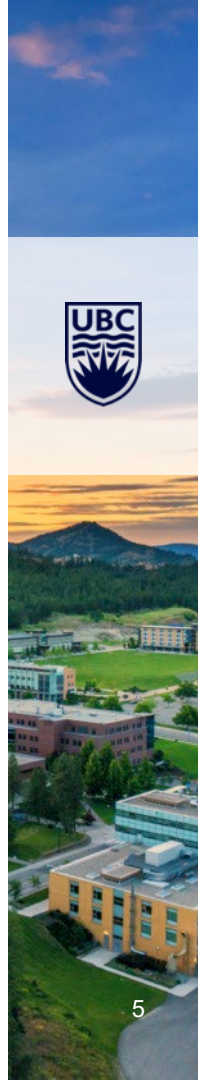


Readings

Required:

Recommended:

- In Search of an Understandable Consensus Algorithm
- Paxos vs Raft: Have we reached a consensus on distributed consensus (Video)

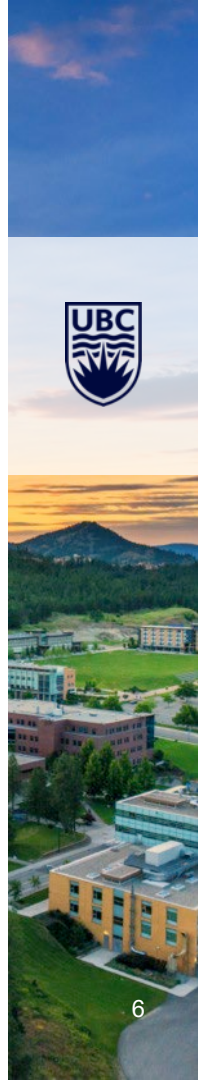


Questions?

Questions about the class?

Questions about the previous lecture?

Funny stories to share?



Today’s Failure

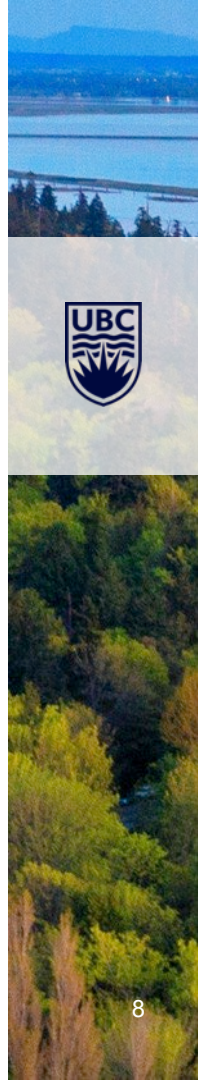
Taking Too Long is Bad

This is my *personal* experience.

Project:

- A file server
- Multiple client computers
- A storage *fabric* (Fibre Channel)
  - Strongly interconnected
  - Storage is accessible from multiple machines

Goal: Use an existing file server (SMB on a Windows Server system) but allow direct storage usage on clients.



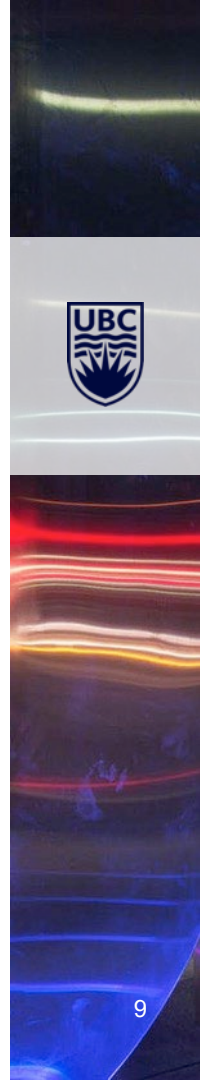
Taking too Long

Server:

- Standard Windows Server
- SMB file server
- NTFS file system

Client:

- SMB Client (initially Windows)
- File system filter:
  - Capture *read* and *write* operations
  - Satisfy via direct I/O to attached storage

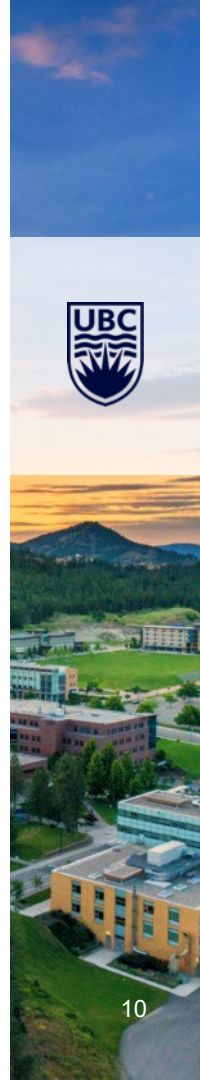


Taking too long

Basic idea:

- File server owns metadata
  - File extension
  - File truncation
  - Location information (tunneled IOCTL query)
- Client:
  - Forwards size changing operations
  - Queries file server for *location* (tunneled IOCTL)
  - Direct I/O (read/write)

Benefit: file server is not a performance bottleneck



What took too long?

Simple operation:

- Client application: extend the file by 10GB
- Client: request 10GB extension on file server
- Server: request NTFS extend file by 10GB
- NTFS: zero newly allocated storage region

Problem?

- It took > 600 seconds (10 minutes) to extend the file.
- **Physical disk is slow**
- Zero-filled disk region
- SMB protocol would time out after 10 minutes

Resolution: Don’t take so long

We fixed this by:

- Building a *disk filter*
- Detect when NTFS was extending
- Return immediately (do not zero disk)

Why was this OK?

- Zeroing the disk was a *security* features
- It was shared disk. The clients could already read it.
- Performance was more important than security

Result?

- System worked and was fast enough!

Lesson Goals

Raft

Review Paxos

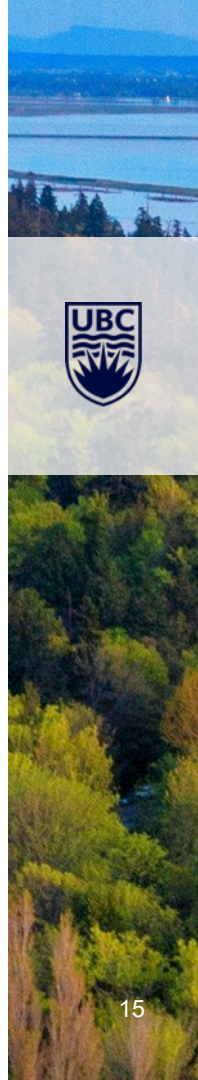
Explain Raft Protocol

Paxos Review

Challenges

- How does Paxos work?
- Why does Paxos work?
- How to build a real system with Paxos?

Note: difficulty increases in that list above



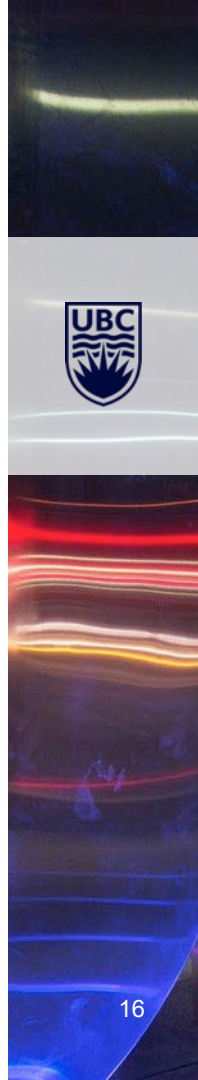
Why is Replication Difficult?

Partition: split brain challenge

- Primary and backup cannot communicate
- Clients can communicate with primary and/or backup

How to handle the failure:

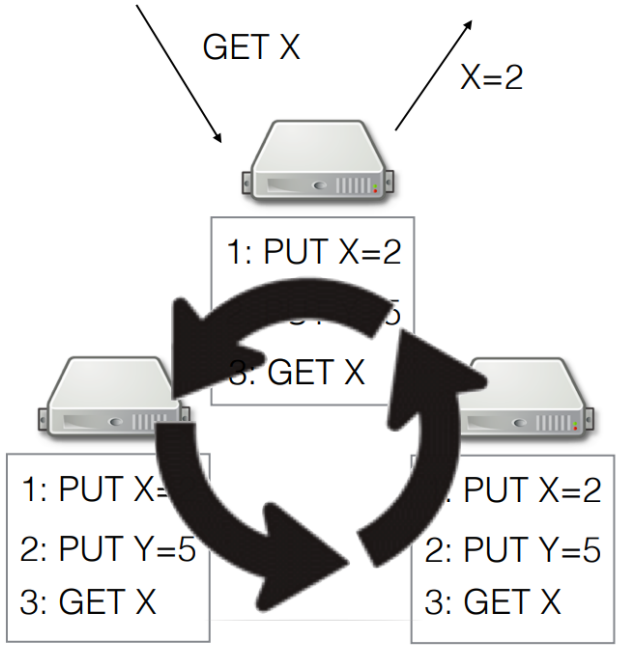
- Backup self-promotes
- Risk: Primary isn’t dead
- Consistency lost



State Machine Replication (Example)

Scenario:

- Three replicas, **no** primary, **no** view server
- Replicas maintain operations log
- Clients send requests to subset of replicas
- Replica **proposes** client request for consens
- Consensus:
  - Commit operation in log
  - Return result to client



Leaderless versus Leader

Original Paxos

- Proposers, accepters, learners
- No leader
- Parallelism by running Paxos instance per log entry

Multi-Paxos

- Leader election (via original Paxos)
- Failure protocol: elect a new leader
- Leader handles commits (similar to 2PC)

Multi-Paxos is more popular for implementation. Project 4 is Multi-Paxos (because we need that for Project 5)

Original Paxos

Each replica maintains an operations log

Client sends request to *any* replica

Replica initiates Paxos proposal

- Uses its latest sequence number
- *Propose* does not mean *accepted*

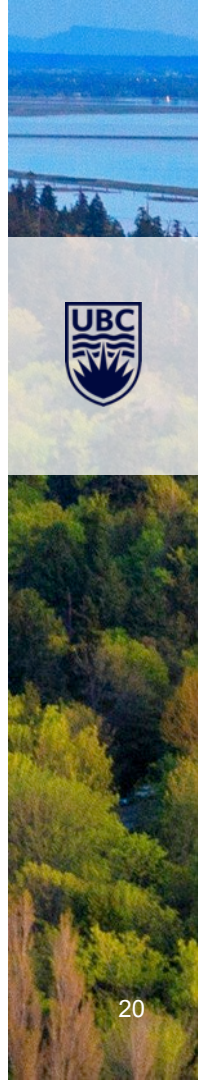
Proposers collect votes

- Quorum reached
  - Record in log
  - Return result to client

Practical Implementation Notes

Replica normally consists of *Proposer, Accepter,* and *Learner*

This is true for Project 4.



Paxos: Reaching Agreement

Client 42:

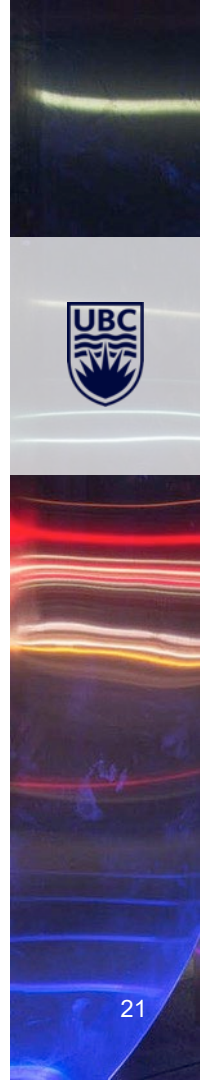
- Sends Put(x)=42 to Server 1

Client 13

- Sends Put(x)=13 to Server 3

Cannot *accept* both values

We protect against *partition* by insisting on a quorum



Why Quorum Matters

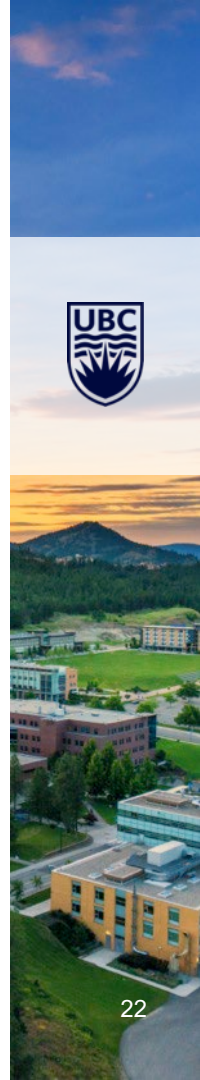
Recall: Paxos tolerates *f* failures.

This means we need **2f+1** replicas

If *n* is the number of replicas, and *f* is the number of failures:

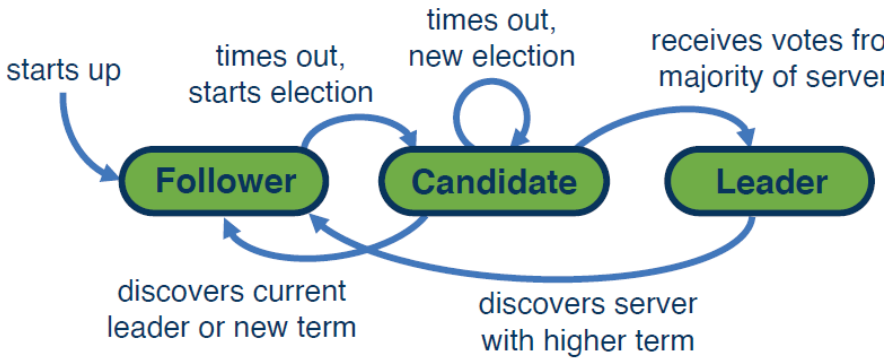
- Have to consider when we have *f* failures.
- Requires we have at least *one* non-failed case left

$$(n - f) - f \geq 1 \Rightarrow n \geq 2f + 1$$

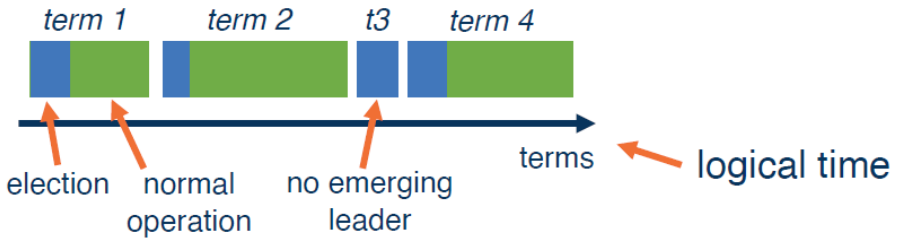


Raft

Leader election phase:



Followed by normal operation…
Log Replication phase:



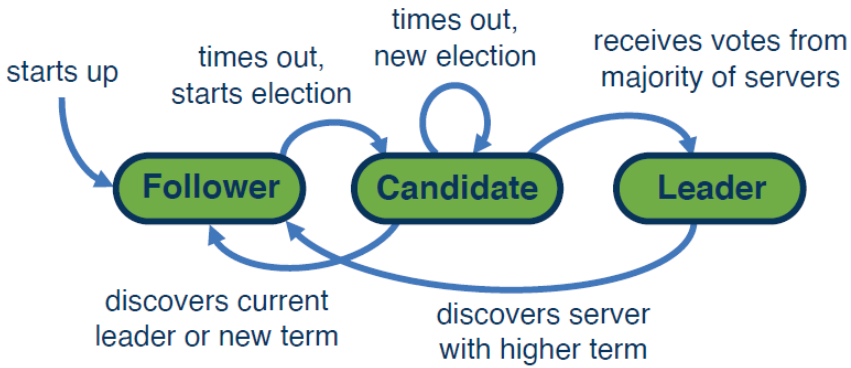
Raft: Leader Election Protocol

Follower time-out: Call Election

Leader candidates “declare”

- Term Number
- Log Index
- (Isn’t this just a view proposal?)

All nodes vote (“accepters”)



RAFT Leader Election Rules

Property 1: At most one leader per term

Rule 1: Leader has Quorum (“majority”)

- New term, Candidate becomes leader

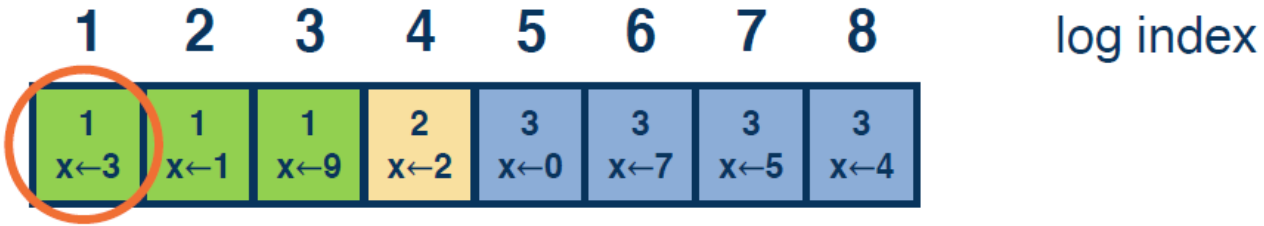
Rule 2: Block old leaders

- Node: only vote for a candidate if their log is newer
- Could be *same* term number: log just longer
- Higher term number: new log
- Losing candidate: just another follower

Rule 3:

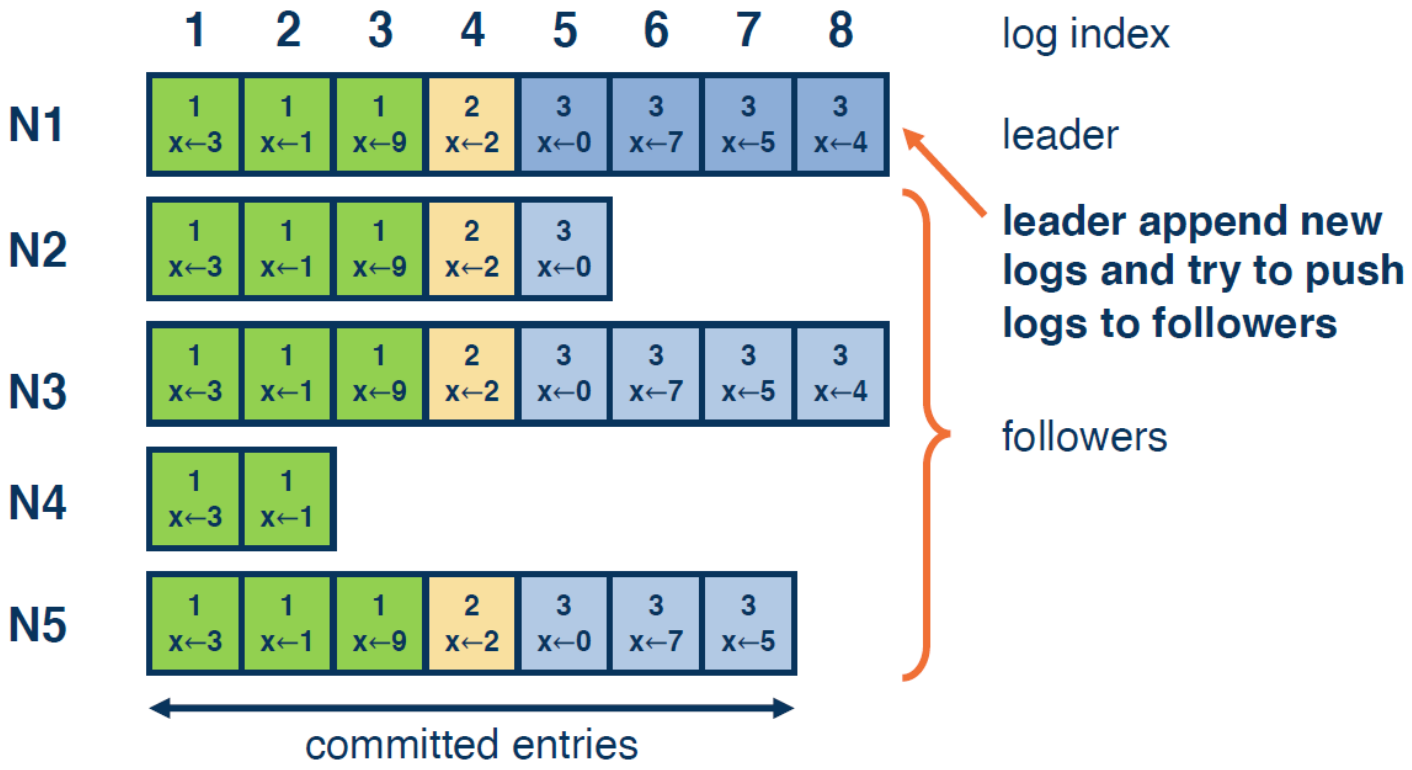
- No winner/network partition? Try again
- Random timeouts to minimize split vote risk

Raft Log Replication



each log entry contains the operation and term number

Raft Log Replication



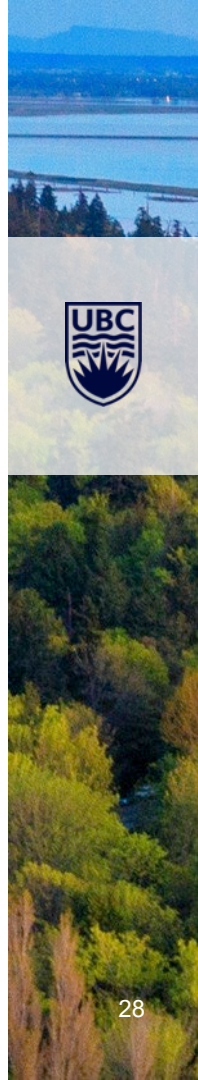
Raft Log Replication

Step 1: Leader pushes new log entry + previous entry to followers (heartbeat)

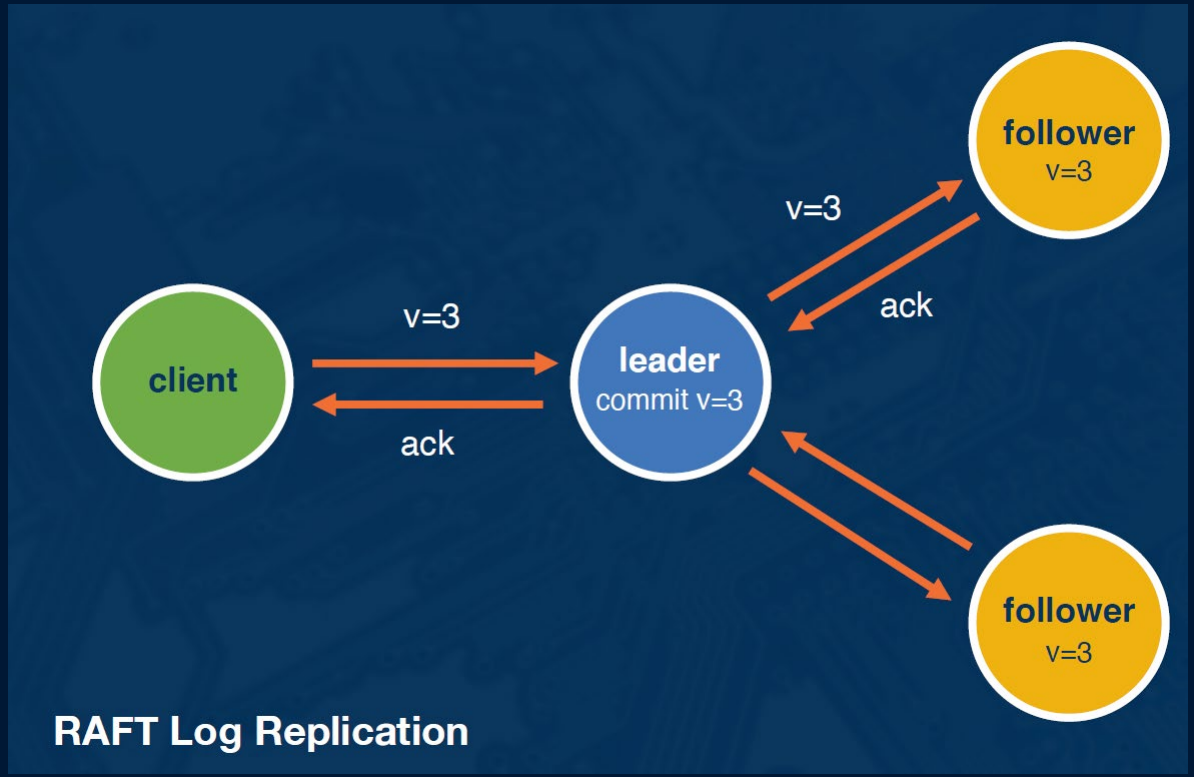
Step 2: If follower has previous log entry: sends ack to leader

Step 3: Leader commits log entry once has quorum; send ack to client

Note: outdated followers catch up via heartbeat.



Raft Log Replication



Raft Leader Properties

Append only

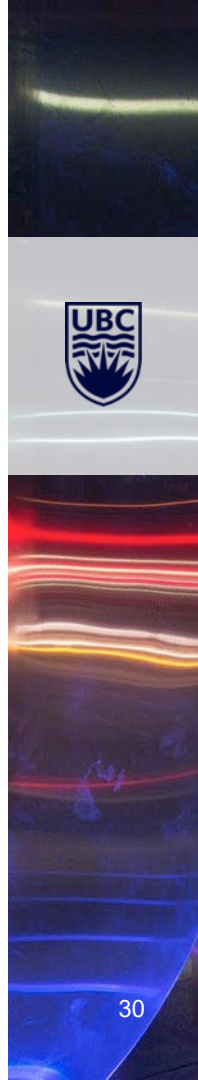
Log matching:

- If two entries in different logs have same index and term, all entries up to this point are the same.

Inconsistency:

- New leader does not know uncommitted log entries
- All followers must use *new leader's log*
- Leader election algorithm ensures new leader knows all *committed* logs

New leader commits uncommitted logs from previous terms *after* committing at least one log from current term.



Raft: Garbage Collection

Log length grows

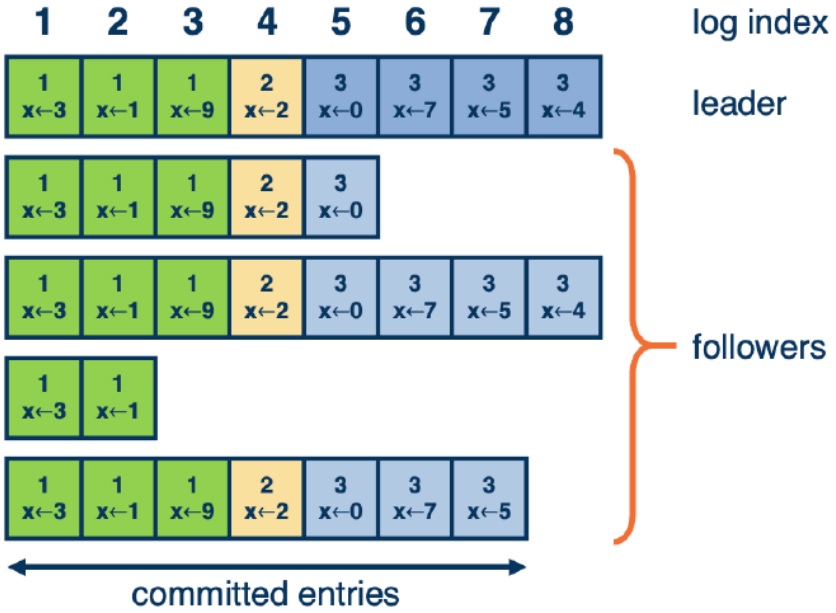
- Execution duration
- Nodes falling behind

Garbage collection:

- Snapshot
- Log truncation

Recovery Optimization:

- Leader sends snapshot via heartbeat



Raft Safety

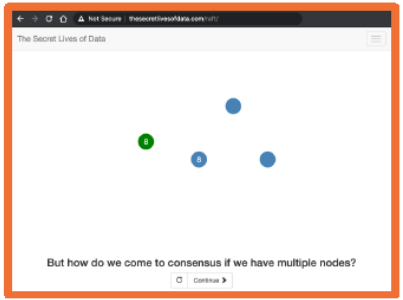
Property: Leader complete

- Log entires not overwritten after commit

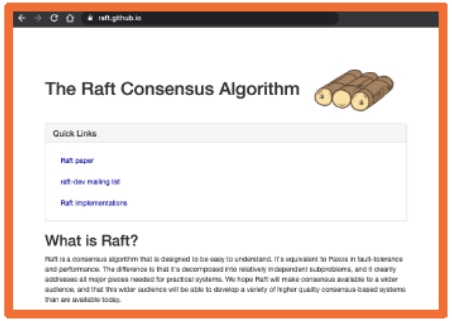
Property: State machine safety

- Log entry never overwritten in a node (see paper for formal proof)

Visualization(s)



http://thesecretlivesofdata.com/raft/



https://raft.github.io/

Lesson Review

RAFT

Paxos Review

Observe: Raft is Multi-Paxos (and Viewstamped Replication)

Questions?


UBC
THE UNIVERSITY OF BRITISH COLUMBIA