EECS 491
Introduction to Distributed Systems

Fall 2019

Harsha V. Madhyastha
Consistency vs. availability

- Replica can execute client requests only if in same partition as majority

- What consistency can we offer if we want any replica to always serve client requests?
Eventual Consistency

- If no new updates, all replicas eventually converge
- **Approach:** Apply updates in same order at all replicas in a manner that preserves causal ordering
  - Use Lamport clocks

- Principles to enable any client to always serve reqs:
  - Maintain log to apply updates immediately, **roll back** later
  - Exchange updates, not state, when syncing
  - Use **version vector** for efficient sync
  - Include **conflict resolution policy** in update
Eventual Consistency

- Never know whether some write from the past may yet reach your node
  - So all entries in log must be tentative forever
  - All nodes must store entire log forever

- To commit updates and garbage collect them
  - Rely on highly available primary to order updates
  - Order chosen by primary must preserve causality
Committing Updates

- At any replica:
  - Stable state
  - Log of tentatively ordered updates (order based on Lamport clock timestamps)

- Upon sync with primary
  - Receive updates in order
  - Apply updates to stable state and prune log
Replicated State Machines

- Logical clock based RSM
  - Cannot progress if any replica is unavailable
  - Eventually consistent even if intermittent connectivity

- Primary backup replication
  - Can replace primary/backup upon failure
  - Unavailable until failed replica is replaced

- Paxos based RSM
  - Available as long as majority in same partition
Consistency Spectrum

Eventual | Read-after-write | Causal | Sequential | Linearizability

Consistency

Ease of programming

Latency

Availability
Web Services in the Cloud

Storage Service

VM
VM
VM

Amazon Web Services
Windows Azure
Google Cloud

October 10, 2019
EECS 491 – Lecture 12
Example Scenario

- Photo sharing web service deployed in AWS
- Use Amazon S3 for storage
  - User name → (# of photos, <List of album names>)
  - Album name → <List of photo URLs>
  - Photo URL → Image data

- Problems application must cope with because GETs may not reflect all completed PUTs?
  - Think through how to serve requests to upload photos and add them albums
Problems with Eventual Consistency

- Lack of referential integrity
  - Total # of photos != sum of # of photos in each album
- Updates may get lost
  - Initialize album1 and add photo1
  - Read album1’s content and add photo2
  - Read album1’s content and add photo3
  - Album1 may eventually have only photo1 and photo3
- Why is Amazon S3 still useful at all?
  - Read-after-write consistency for initial PUT
  - Most data is write-once
Design Challenge

How to satisfy following goals?

- Minimize latency for reads and writes
- Ensure total ordering of writes
- Okay to respond to reads with stale data but minimize occurrences
How often does this system violate linearizability?
Linearizability checker

- Many clients continually issue reads and writes
- For each op, log time of request/response
- For each read, log response value

- Construct graph with one vertex per operation
- Directed edge from Op1 to Op2 if
  - Op2 was issued after response for Op1 received
  - Op2 read value written by Op1

- Merge reads into writes
- Cycle → Linearizability violation
Linearizability checker

(a) Execution.

(b) Graph before merge.

(c) Graph after merge.

(a) Execution

(b) Graph before merge.

(c) Graph after merge.
Consistency at Facebook

- 0.0004% of reads violate linearizability
  - Despite system not designed for linearizability!

- Implication:
  - Whether a system offers linearizable consistency cannot be ascertained via measurements
  - Must reason about system’s design
If partition,
- Choose availability vs. consistency

Else,
- Choose latency vs. consistency

Externally visible effects may not reveal if system chooses latency over consistency
Announcements

- **Project 3:**
  - Paxos-based key-value store (due Oct. 31st)
  - Overview of project at next Friday’s discussion

- **Tomorrow’s discussion:**
  - Implementing eventually consistent key-value store

- **Use Fall break to review material for mid-term**