CPSC 416 Distributed Systems

Winter 2023 Term 1 (October 17, 2023)

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

Logistics

Teaching Assistants

Andy Hsu (andy.hsu@alumni.ubc.ca)

Hamid Ramezanikebrya (hamid@ece.ubc.ca)

Jonas Tai (jonastai@student.ubc.ca)

Cathy Yang (kaiqiany@student.ubc.ca)

Office Hours

Remember: Use Piazza for **all** official course-related communications

- Not on Piazza? Not official.
- Canvas “comments/messages” **are not monitored**

Office Hours:

Who	When	Where
Tony	Monday 14:00-15:00 Wednesday 16:00-17:00	Discord
Andy	Thursday 19:00-20:30	Discord
Hamid	Friday 16:30-18:00	Kaiser 4075
Jonas	Thursday 13:00-14:00	X241
Cathy	Friday 09:00-10:30	X237

Self-Assessment

This week

- Design Project 3 Team Declaration Deadline (Today @ 23:59)
- Post-lecture self-assessment activity – Due Thursday (October 12 @ 17:00)
- Design Project 2 Feedback Due Thursday (October 19 @ 17:00)

Next week

- Design Project 3 Due Tuesday (October 24 @ 17:00)
- Design Project 2 Code Due Thursday (October 26 @ 17:00)
- Design Project 2 Implementation Report Due Thursday (October 26 @ 23:59)

Note:

- You are strongly encouraged to collaborate with others on this
- You should use tools at your disposal to answer these questions
- **Do not forget to submit it.**

Today’s Failure

Github.com Outage

Event: October 21, 2018 22:52 UTC

Planned outage: goal is to replace a failing 100Gb/s optical network device.

“Connectivity between these two locations was restored in 43 seconds, but this brief outage triggered a chain of events that led to 24 hours and 11 minutes of service degradation.”

Infrastructure: MySQL with Orchestrator to manage cluster topologies.

Note: Orchestrator uses **Raft** for consensus.

Github.com Outage

Network goes out: Raft starts “leadership deselection”

Note: optical link was between two Eastern US sites.

West coast data center and East coast Orchestrator form quorum

Fail over to clusters in West coast data center: write operations begin working.

Network fixed: traffic starts going to West coast site

Note: East coast had some updates that had not propagated to west coast yet.
This **blocked** primary returning to East coast.

Github.com

Things come unraveled due to increased latency, unexpected topologies.

Decision to degrade service rather than compromise consistency.

Start restoring databases from backup.

Restoration was started October 22, 2018 00:05 UTC

Restoration completed and service restored October 22, 2018 23:03 UTC

Twenty three hours to restore from a 43 second network disruption.

Takeaway: Recovery is the hard part.

Source

Kleppmann Chapter 7

Learning Goals (Kleppmann Chapter 7)

Understanding the role of transactions

Identify Challenges in Distributed Transactions

Distinguish Between Isolation Levels**

Recognize Common Anomalies

Appreciate Practical Implications

Transactions

A sequence of operations executed as a single unit of work ("atomic")

- Ideally it is a *minimal* set
- Transitions from one *consistent* state to another *consistent state*
- Enables recovery in case of handled failures – provide *data integrity*
- Permits concurrent operations
- Ensures isolation, so intermediate states are not visible

ACID – atomic, consistent, isolated, durable

Distributed Transactions

Transactions in distributed systems

- Not fate shared

Problem types: network issues (slow, out of order, dropped packets) and node failure

Concurrent access: how do we control this? How do we make it efficient?

Two-phase Commit Protocol:

- Leader (“coordinator”) proposes a set of changes
- Resources *prepare* their changes
- Leader *commits* - this is a *voting* phase
- Quorum = “unanimous consent”

Consistency Models

Strong Consistency

Eventual Consistency

Causal Consistency

Read-Your-Writes Consistency

Monotonic Read Consistency

Monotonic Write Consistency

Strong Consistency

Definition:

- All operations on the system appear in a single, agreed-upon order.
- Every read receives the most recent write

Example:

- Linearizable (read-after-write consistency)
- Serializable (isomorphic outcome for concurrent operations)

Strength: simplifies application logic

Weakness: Expensive (high latency/low throughput) in distributed systems

Eventual Consistency

Definition: Without further updates, all replicas will converge to the same value(s)

Example:

- CRDTs are data structures for implementing eventually consistent systems

Strength:

- High Availability, Good partition tolerance

Weakness:

- Applications may see inconsistent results

Causal Consistency

Definition:

- Causally related operations are visible in the same order
- Concurrent (non-causally related) operations may be seen in different orders

Example:

- Messaging systems

Strength: Stronger than eventual consistency, faster than strong consistency

Weakness: Requires tracking causal relationships between operations

Read-your-writes Consistency

Definition: Client will see its own writes

Example: Google Docs

Strength: Clients have a self-consistent view (but may see different data for other client writes)

Weakness: No guarantees about global ordering of operations across clients

Monotonic Read Consistency

Definition: client will always read the same or newer value

Example: Online shopping cart

Strength: clients do not see old values after new values

Weakness: no guaranteed global order across values

Monotonic Write Consistency

Definition: client writes are always ordered

Example: Online blogging platform

Strength: Provide write ordering guarantee for a client

Weakness: No ordering of writes for *other* clients

Snapshot Isolation

Use database snapshots to permit simultaneous transactions

Non-conflicting updates can proceed

Conflicting updates must be resolved (one transaction "aborts" and tries again)

Fast (except in conflict cases)

Anomalies and Issues

Dirty Reads: Reading data that is modified but not yet committed

Dirty Writes: Writing data that is modified in another transaction but not yet committed.

Read Skew: data read twice, returns different values (someone else is changing it)

Lost Updates: read/modify/write race to a single value

Write Skew: read/modify/write race to a multiple values

Phantom Reads: First transaction reads a set of objects (like a SELECT statement) and second transaction changes the set of objects, so first transaction sees different objects on a second read.

Combining Distributed Transactions and Consensus

Distributed Transactions: leader requiring 100% quorum

Distributed Consensus: *may* have a leader, *always* requires a quorum (but not necessarily 100%)

Can combine these two together:

- Sharded key-value store
- Replicated Databases with SQL transactions

Isolation Level Trade-offs

Spectrum (fast to safe)

- Read Uncommitted
- Read committed
- Repeatable Read
- Serializable
- Snapshot Isolation

Trade-offs:

- Higher isolation = lower performance
- Lower isolation = higher anomaly risk

Practical Implications

Real-world challenges:

- Operational complexity
- Latency Considerations
- System Failures
- Scalability
- Changing Requirements

Monitoring, logging, and alerting:

- Anomaly detection
- System behaviour insight
- Issue Alerting
- Performance Tuning
- Auditing & Compliance
- Learning & Adapting

Questions?


UBC
THE UNIVERSITY OF BRITISH COLUMBIA