CPSC 416 Distributed Systems

Winter 2023 Term 1 (October 26, 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 2 Peer Feedback: Implementation Report Due Thursday (October 26 @ 23:59)

Next week

- Self-Assessment Due Tuesday (October 31 @ 17:00)
- Design Project 3 Peer Feedback: Design Due Tuesday (October 31 @ 17:00)
- Self-Assessment Due Thursday (November 2 @ 17:00)

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

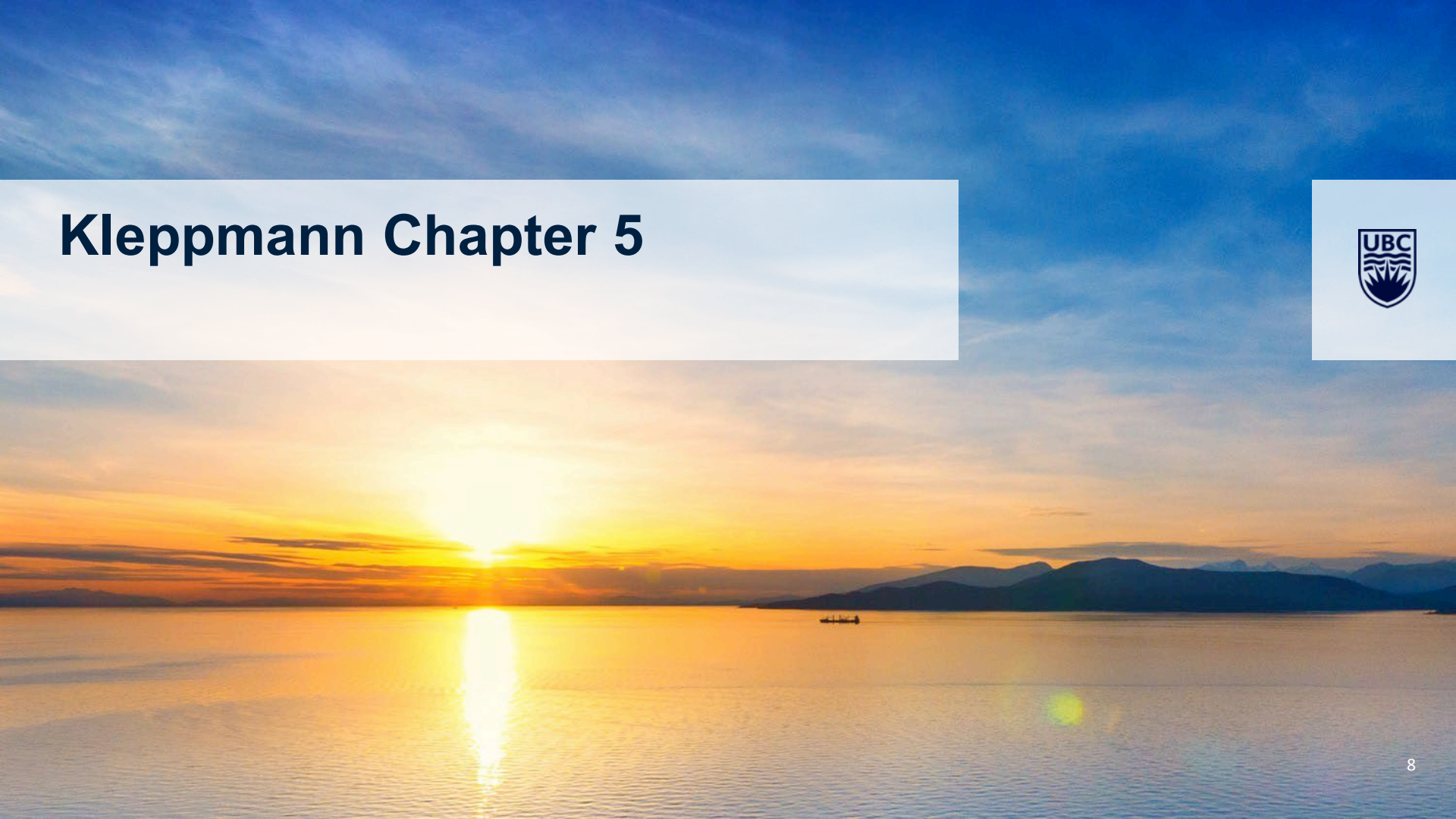
Failure Avoided

Using TLA+

“TLA$^+$ made it tractable for an ordinary software engineer to reason about a tricky distributed systems problem, and it found a bug introduced by an “optimization” I tried to add (classic). The bug required 12 sequential steps to occur and would not have been uncovered by ordinary testing.”

Andrew Helwer (Blog)

Kleppmann Chapter 5



The major difference between a thing that might go wrong and a thing that cannot possibly go wrong is that when a thing that cannot possibly go wrong goes wrong it usually turns out to be impossible to get at or repair.

—Douglas Adams, *Mostly Harmless* (1992)

Learning Goals (Kleppmann Chapter 5)

What is replication

Leaders & Followers

Replication Lag

Multi-leader replication

Leaderless replication

Replication

Maintaining *identical* copies of data on different machines

“Degenerate” case for 2PC/3PC transactions

- 2PC/3PC is about *consistency across databases*
- Replication is about *consistency across **identical** databases*

Read-only data is easy to replicate

Read-write data:

- How we replicate
- How we deal with “out of sync” due to transient failure (failure-resume model)

Leaders & Followers

Replica = a copy of the database

Database write must propagate to every replica

Leader/follower model

- Writes go to leader
- Writes propagate to followers *from* the leader
  - Ordering details are specific to the protocol.
- Reads usually come from any of the replicas

Synchronous versus Asynchronous update (consistency) – Chain Replication, CRAQ.

Configuration changes (add remove/followers, replace leader)

Catchup

- Follower: incremental restore from log(s)
- Leader: much harder to implement correctly

Replication Lag

Eventual consistency – higher availability, weaker consistency (CA trade-off)

Consistency models (again):

- Read your own writes
- Monotonic reads
- Consistent prefix reads (causality related operations, preserve write order)
- Read-after-write
- Strong consistency – transactions/quorum consensus

Multi-leader Replication

Leader-based replication

- Simple to understand/implement
- Creates a bottleneck
- Complex failure recovery semantics

Multi-leader:

- When performance benefit outweighs added complexity
  - Write conflict handling (sync/async, avoidance, merging)
- Disconnected operation
- Collaborative editing

Topologies

Leaderless Replication

When every node is a leader, no node is a leader

- No need to handle leader failure
- Client directly proposes changes
- Reads require querying multiple nodes
  - Highest version wins
  - Can you see how this might work with quorum consensus?
- Catch-up
  - Reader submits latest version to stale nodes
  - Special background process “consistency checks” replicas

Monitoring system (“detect problems before they become serious problems”)

“Sloppy” quorum

Multi-data center

Concurrent writes (back to conflict resolution)

Conflict Resolution (Redux)

Last writer wins

- We have to define what this means (e.g., no *a priori* definite order)

Happens-before (causal relationship)

Write-merging

- Push the problem back to the client

Versioning

- Use *vectors* – version number *per replica*

Questions?


THE UNIVERSITY OF BRITISH COLUMBIA
THE UNIVERSITY OF BRITISH