CPSC 416 Distributed Systems

Winter 2022 Term 2 (February 14, 2023)

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

Logistics

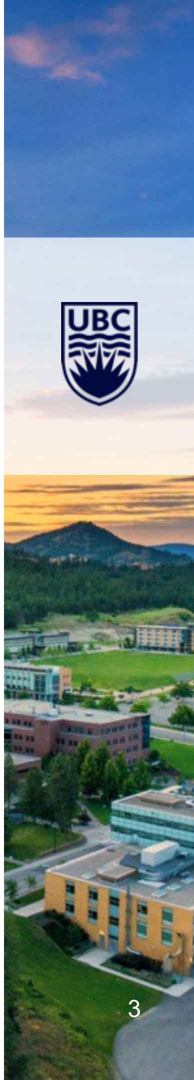
Deadlines

Project 3 Released. Initially Due: **February 13, 2023**. Next Monday

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.



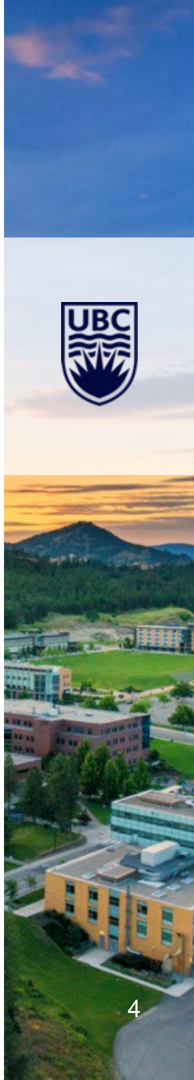
Deadlines

Alternate Path 1 & 2: Review in progress

- Piazza private threads created
    - **Teams have been given feedback on their proposal.**
    - **Several Path 1 teams have outstanding questions to answer (as of 2023/02/08)**
    - **Weekly updates due each Monday @ 23:59 PT**
- **Proceed according to your plan.**

Instructor Office Hours:

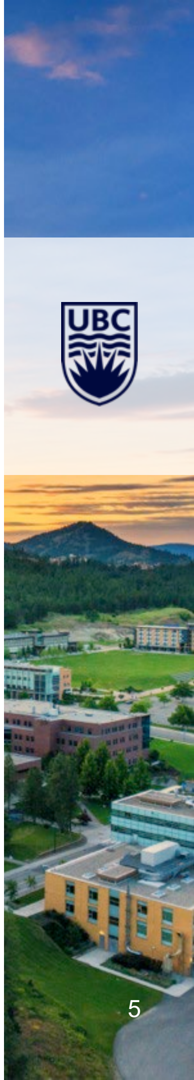
- Zoom Office Hours (Tuesday) @ 13:00-14:00
- Discord (Casual) Office Hours (Thursday) @ 14:00-15:00



Readings

Required:

Recommended:

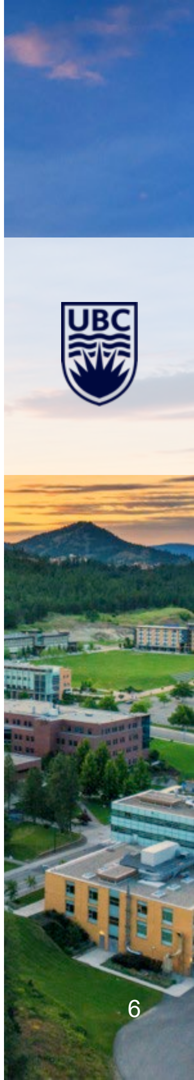


Questions?

Questions about the class?

Questions about the previous lecture?

Funny stories to share?



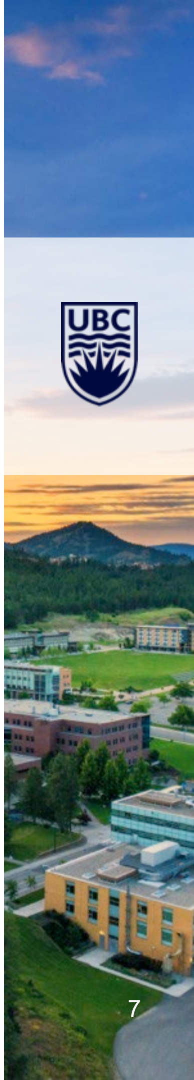
Deadlines

Project 3 Released. Initially Due: **February 13, 2023**. Next Monday

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.



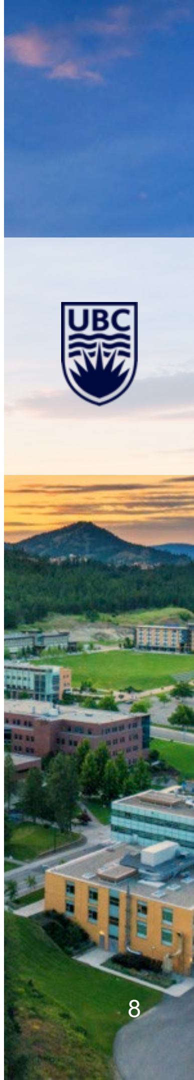
Deadlines

Alternate Path 1 & 2: Review in progress

- Piazza private threads created
    - **Teams have been given feedback on their proposal.**
    - **Several Path 1 teams have outstanding questions to answer (as of 2023/02/08)**
    - **Weekly updates due each Monday @ 23:59 PT**
- **Proceed according to your plan.**

Instructor Office Hours:

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



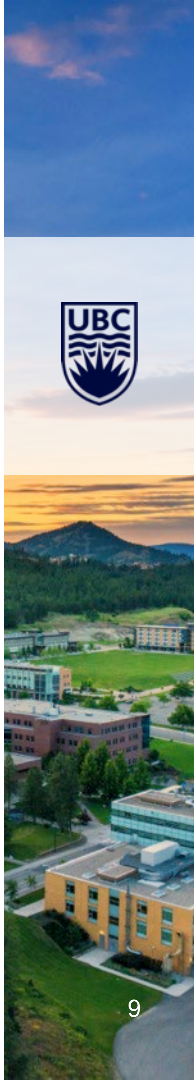
Readings

Required:

- Weighted Voting for Replicated Data (Gifford)

Recommended:

- Kleppman's notes on distributed systems (See Section 5.2)
- Crumbling Walls: A Class of Practical and Efficient Quorum Systems

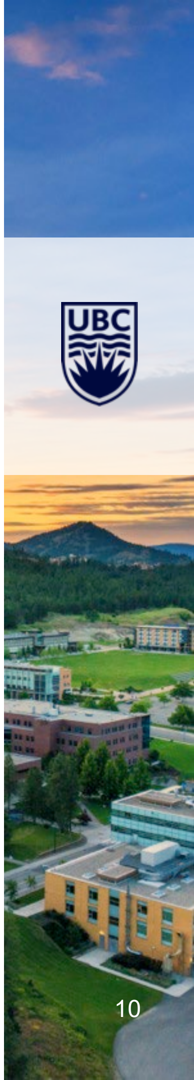


Questions?

Questions about the class?

Questions about the previous lecture?

Funny stories to share?



Today’s Failure

Amazon Outage

Event start: November 26, 2004

Event ends: December 6, 2004

TL;DR Version

- Amazon home page went offline
    - Amazon.com
    - Amazon.ca
    - Amazon.co.uk

Mitigation

In fact, November 2004 was *before* Amazon Web Services Existed!

- This was (partial) motivation for its creation

[Dynamo: Amazon's highly available key-value store]

Reliability at massive scale is one of the biggest challenges we face at Amazon.com, one of the largest e-commerce operations in the world; even the slightest outage has significant financial consequences and impacts customer trust. The Amazon.com platform, which provides services for many web sites worldwide, is implemented on top of an infrastructure of tens of thousands of servers and network components located in many datacenters around the world. At this scale, small and large components fail continuously and the way persistent state is managed in the face of these failures drives the reliability and scalability of the software systems.

Lesson Goals

Quorum Replication

Review:

- Primary/Backup
- Chain Replication
- CRAQ

Consistency (again)

Quorum Replication (Readers + Writers)

Crumbling Walls (Generalized Quorum Replication)

Primary/Backup Replication

Basic model

- Primary: responsible for all read/write activity
- Backup(s): maintain copies of primary database

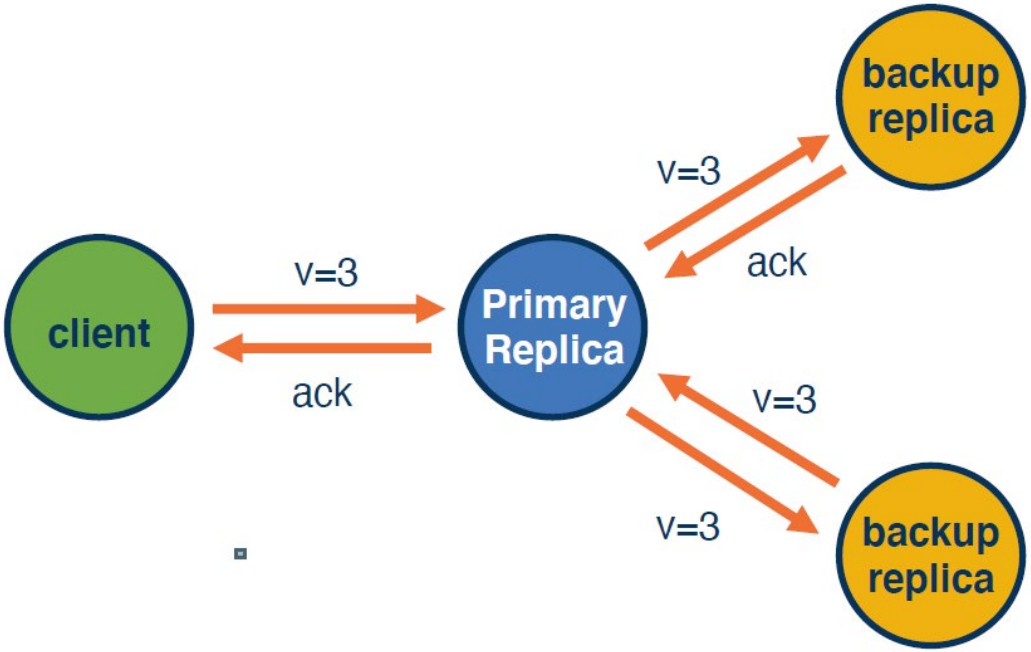
Chain Replication

- Each backup *chains* to the next
- Minimizes message overhead
- Write head, read tail

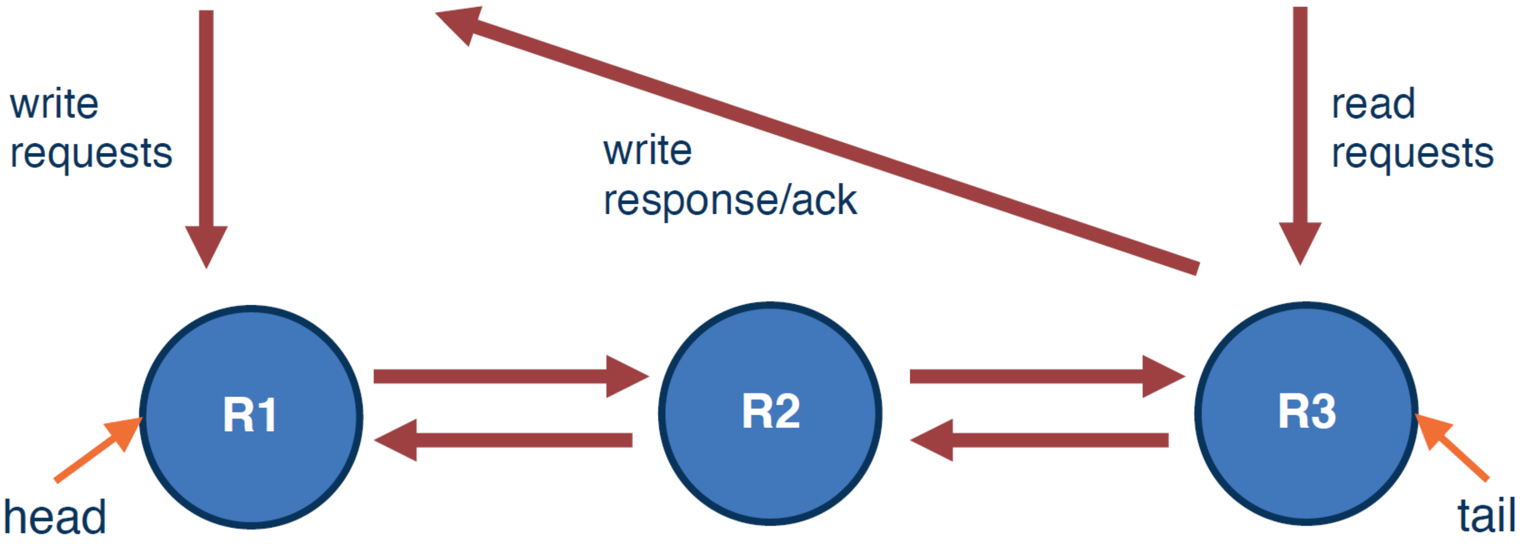
CRAQ

- Allows backups to service *read* operations
- Uses dual values for in-flight changes (must read final backup if dual)

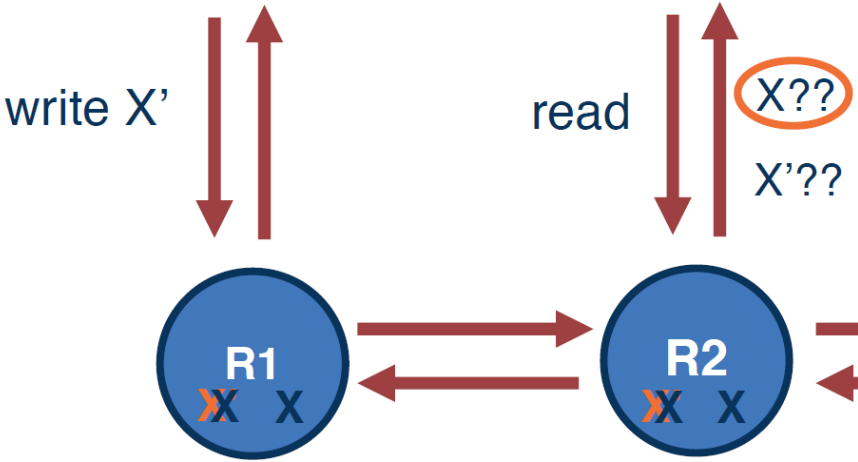
Replication



Chain Replication



CRAQ



More detail in paper…

Disadvantages of Replication

Availability

- Primary: single point of failure
- *Or* configuration database: single point of failure
    - Project 3 – ViewServer is a configuration database

Scalability

- Write-heavy workloads suffer (CRAQ)
- All workloads suffer (Chain Replication)

Partition Resistance

CAP Theorem (Again)

Consistency

- Strongest consistency model is **sequential** consistency
- Weaker models exist (e.g., linear consistency)

Availablity

- Can we continue servicing clients

Partition tolerance

- Networks are garbage
- Many failures are *transient*
- How to handle *without* sacrificing **C** or **A**?

CAP (More formal)

Consistency:

- *A data item behaves as if there is only a single copy*

Availability:

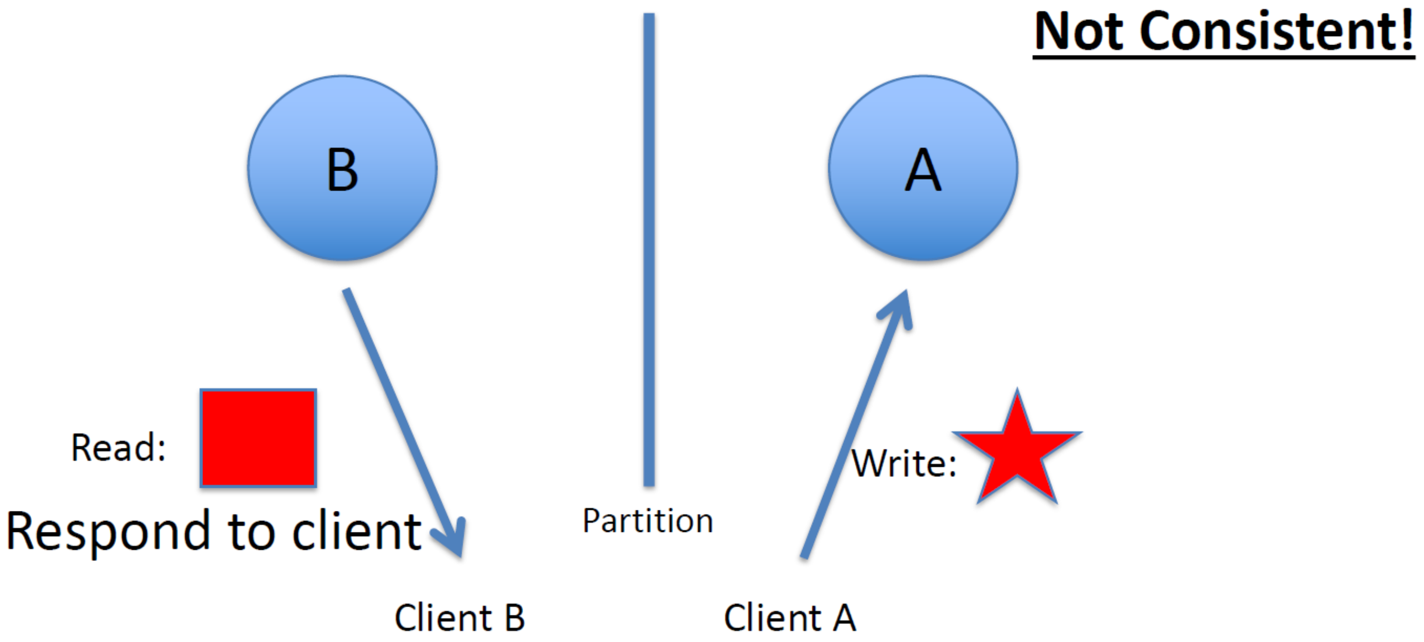
- Some level of nodes failing does not interfere with other nodes continuing to provide services.

Partition-tolerance:

- Services continue to be available in exactly *one* partition.

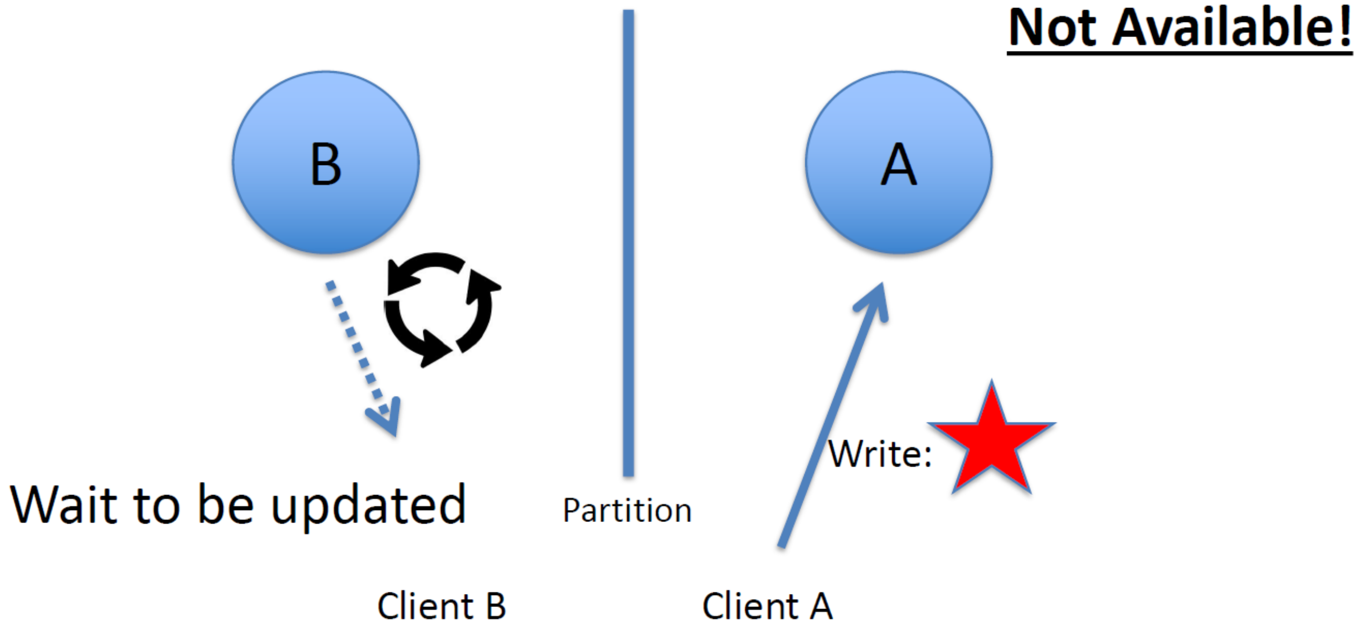
CAP: Proof by intuition (1)

- A simple proof using two nodes:



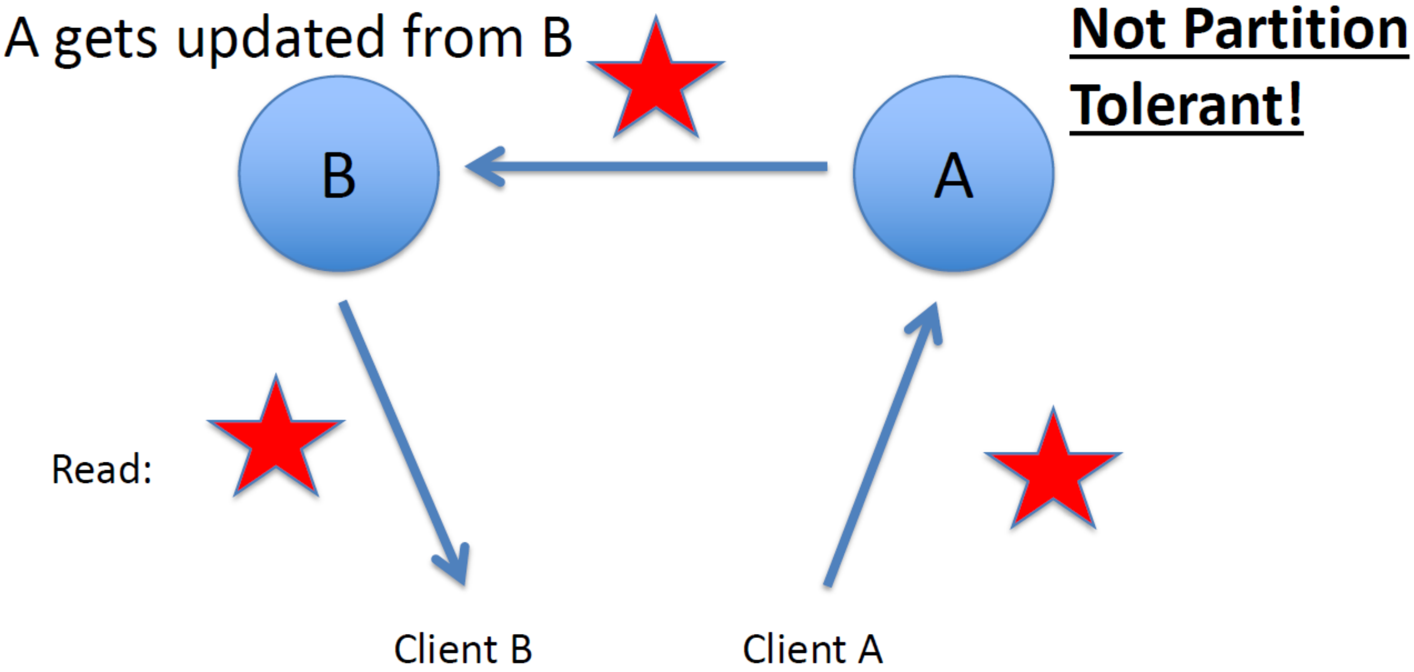
CAP: Proof by Intuition (2)

- A simple proof using two nodes:



CAP: Proof by Intuition (3)

- A simple proof using two nodes:



CAP: Summary

Partiton is a *property of the network*

- We don't really get to control this

Consistency + Availability are *goals of the system*

- Reality: *some* failure is better than *complete* failure

In other words: *failure happens*

Our job:

- Minimize the impact of failure
- Continue providing services whenever possible
- Manage tradeoffs between *consistency* and *availability*

Consistency Models

Strong Consistency = only one path through the (distributed) system

Single Consistency = pick one path through the (distributed) system

- Same outcome *regardless* of which picked (“equivalence”)

Causal Consistency

- Guarantee ordering of causally connected events
- **No ordering** for concurrent (not causally connected) events

Eventual Consistency

- Recovery will (or could) restore a stronger consistency “at some point”
- Favours *availability* over strong consistency

Quorum Replication

Basic idea:

- Each copy of a replicated copy has a *weight*
- Access is done via a *transaction*
    - Acquires r weighted votes (read)
    - Acquires w weighted votes (write)
    - $r + w > \frac{1}{2}(\Sigma_0^n r_\omega + \Sigma_0^m w_\omega)$

This ensures:

- There is a non-null intersection between every read quorum and every write quorum.

Use version numbers to determine which copies are current.

Why Quorum Replication

It works correctly even if some copies are inaccessible

It can be implemented on top of a transactional storage layer

- No communications needed

It provides serial (strong) consistency

Weights can be adjusted to balance performance against reliability

Quorum Replication Examples

Read one, write all

Majority consensus

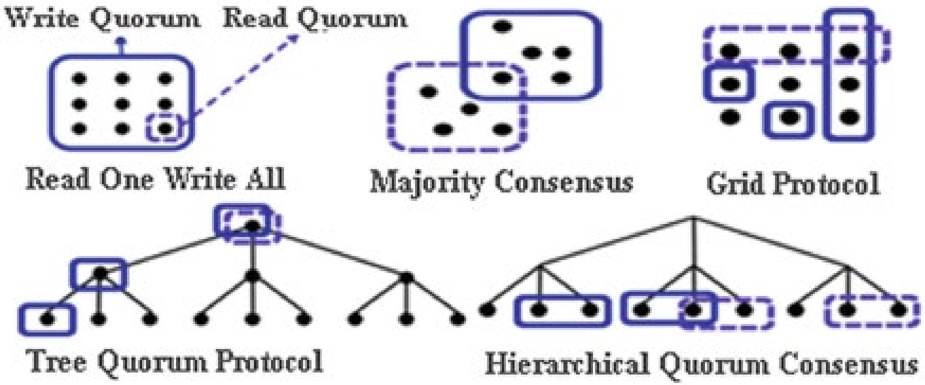
Tree Quorum Protocol

Weighted Voting Strategy

Hierarchical Quorum Consensus

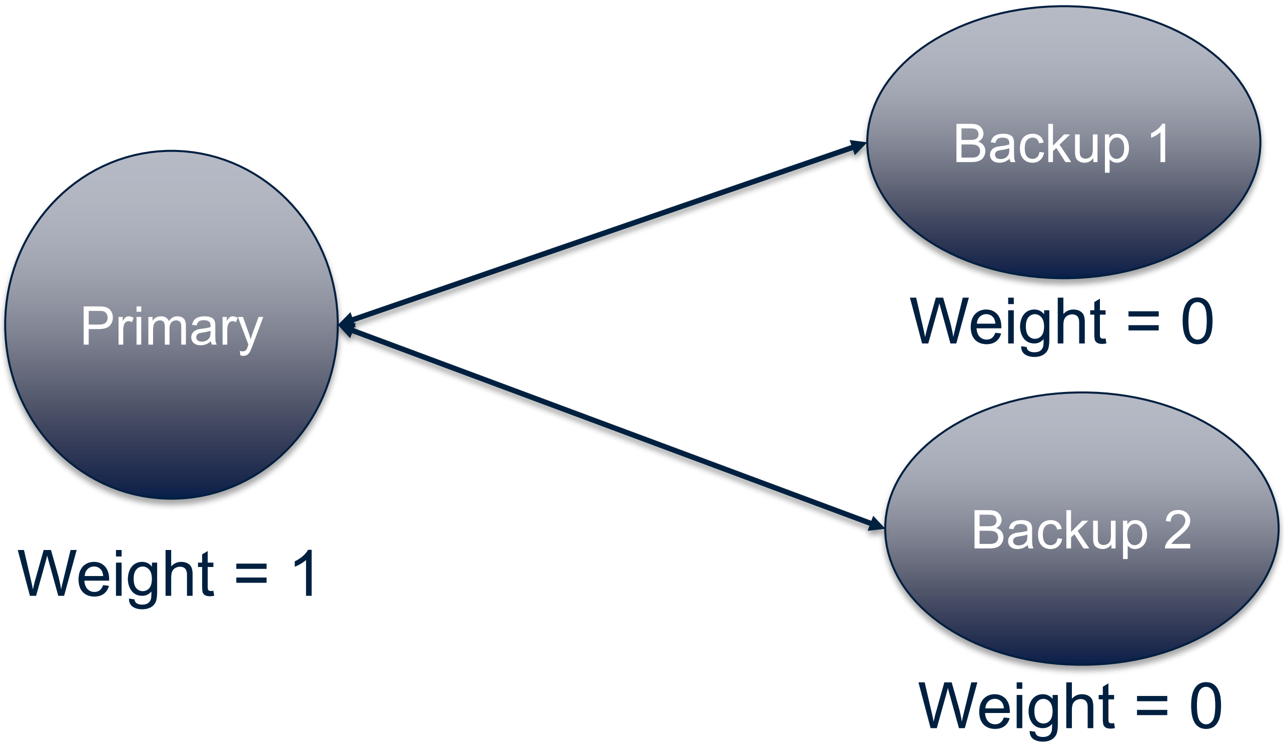
Grid Protocol

Triangular Lattice Protocol

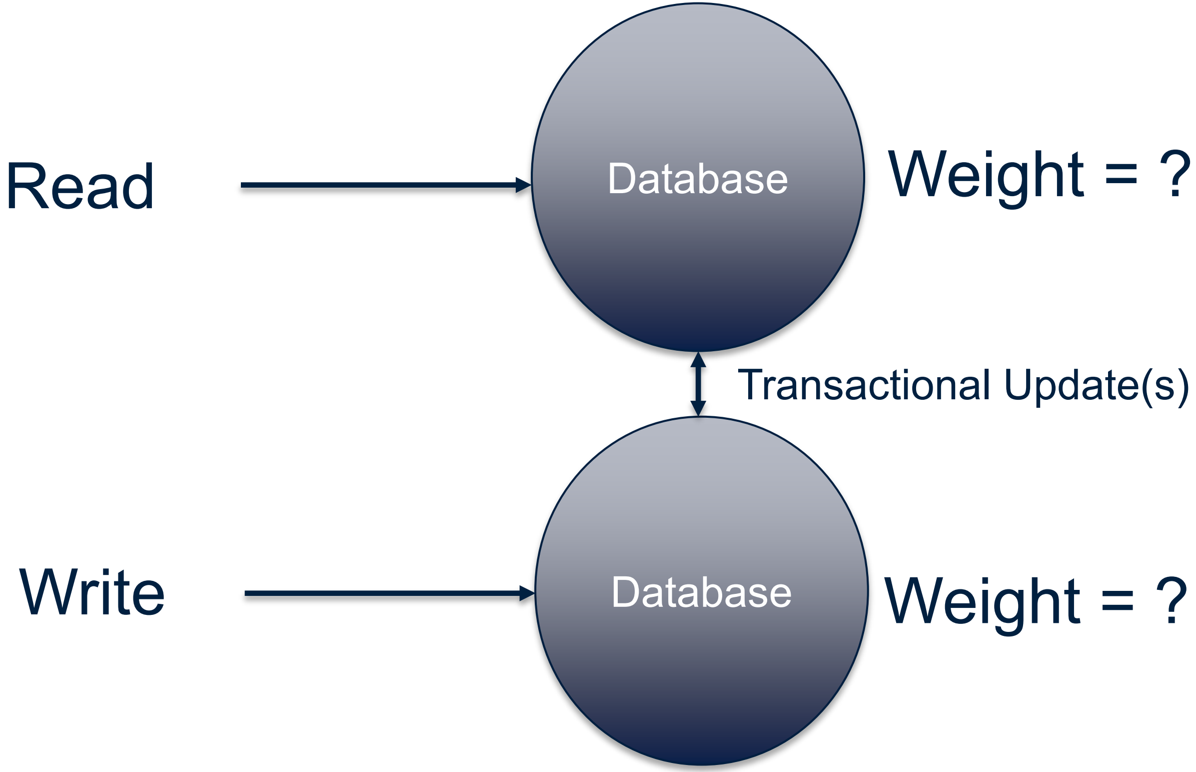


Primary-Backup is Quorum Consensus

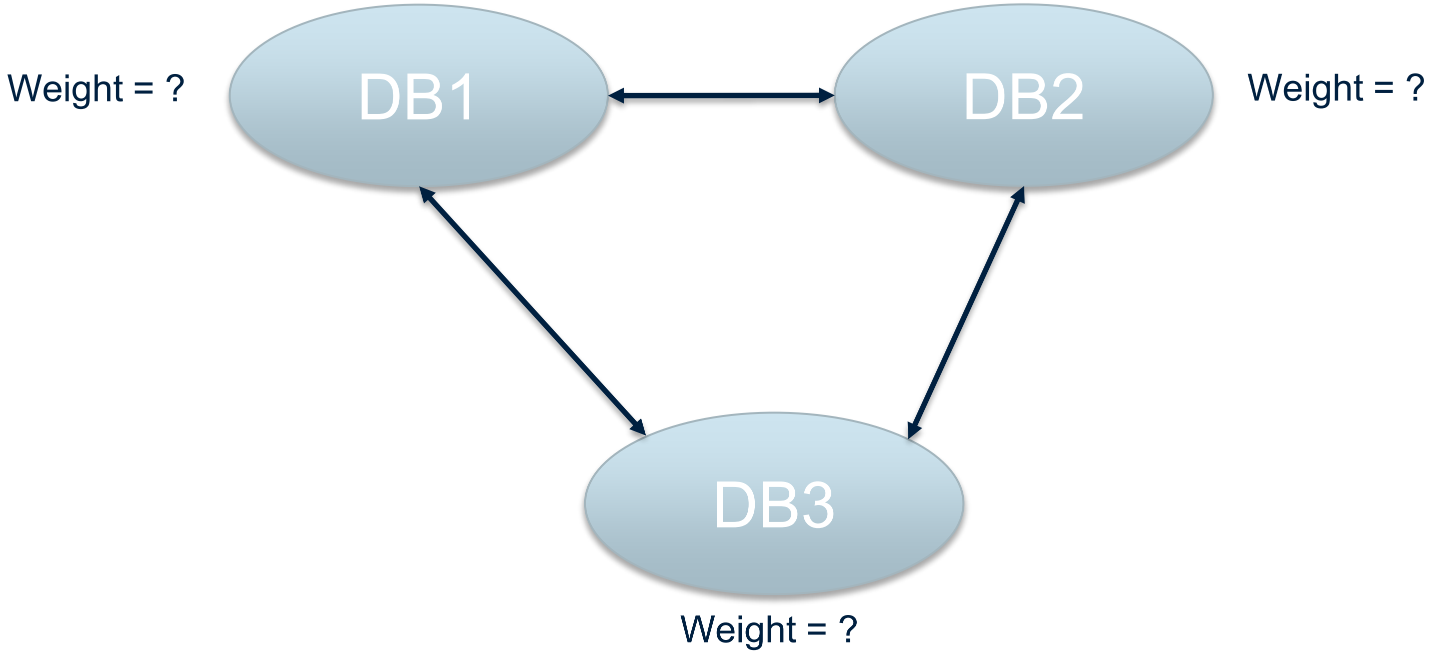
Extending Primary Backup (Trivial Quorum)



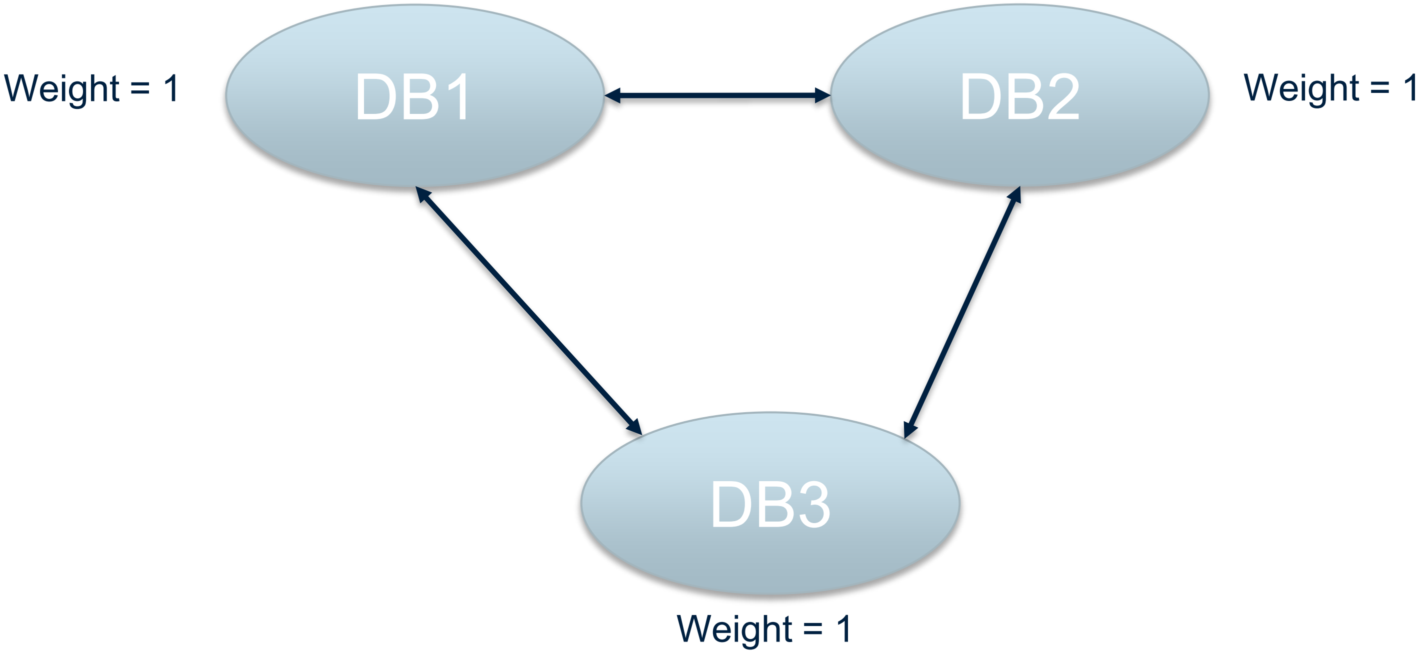
Multiple Writers



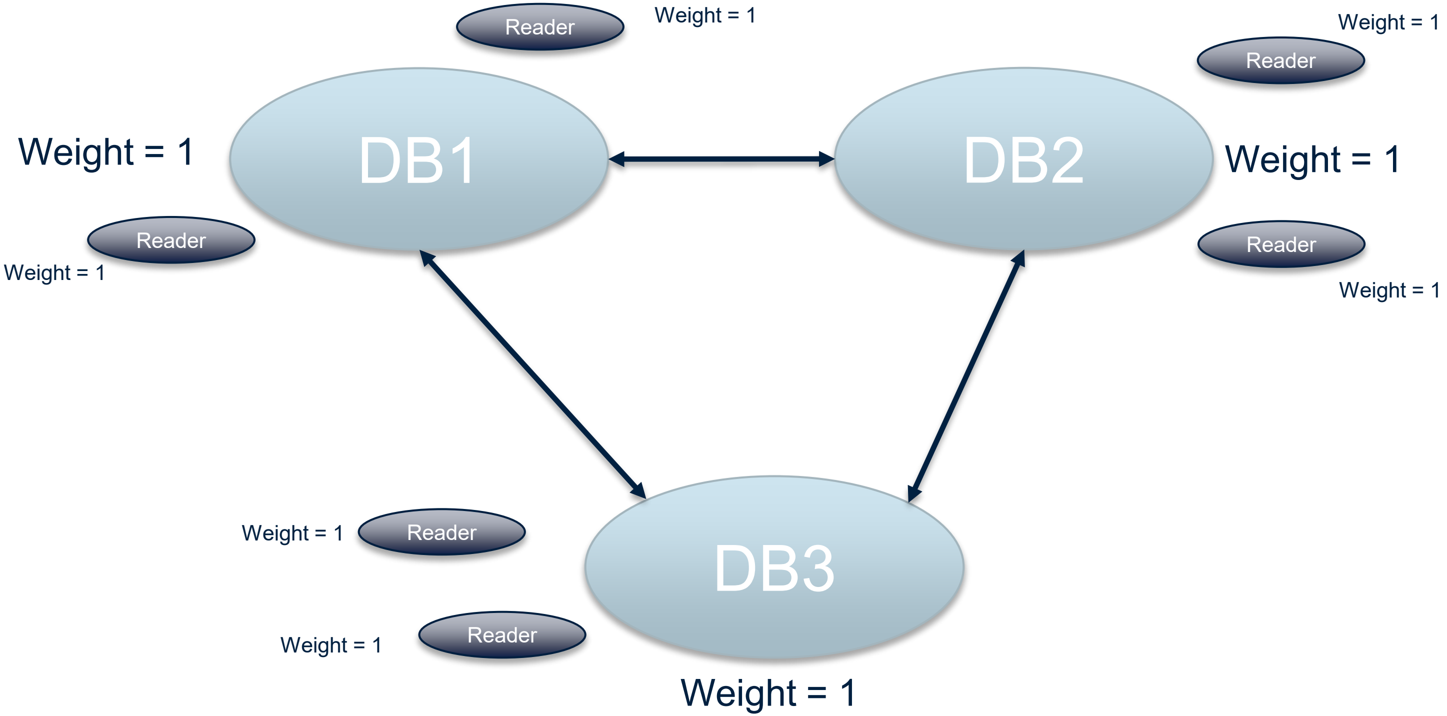
Multiple Writers: Problem



Multiple Writers: Problem



Readers + Writers



Choosing Weights

"By manipulating *r*, *w*, and the voting structure of a replicated file, a system administrator can alter the file's performance and reliability characteristics."

There *must be* some set of readers and writers (the quorum) that have the same value.

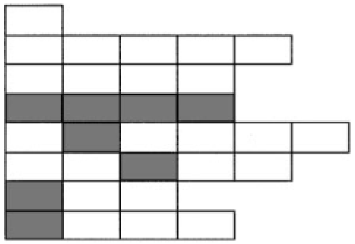
System must guarantee this behaviour.

Crumbling Walls

Elements are arranged in *rows*

Quorum is defined as the union of one full row plus one element from each column (picture is from the paper).

Note: Crumbling walls is a generalization of earlier quorum based solutions.



CWlog: Optimal Load, Small Quorum, High Availability

The load is:

$$\mathcal{L}(\text{CWlog}) = O\left(\frac{1}{\log n}\right)$$

Availability:

$$F_p(\text{CWlog}) = O(n^{-\epsilon})$$

For some constant $\epsilon(p) > 0$. (Note that p is the probability that each element fails)

Takeaway: we *can* construct systems with less than simple majority

Limitation: p need to be quite small ($p < 0.432$).

Challenges Remain

Quorum systems assume there is a Coordinator

- Project 3: ViewServer *is* a simple Coordinator

Coordinator needs to be high-availability as well

How do we construct such a system?

Lesson Summary

Quorum Replication

Review previous replication mechanisms

Revisit Consistency

Dive into CAP Theorem

Consider Voting (Quorum) protocols

Questions?

UBC

THE UNIVERSITY OF BRITISH COLUMBIA