Cloud and Cluster Data Management

SCALABLE CONSISTENCY AND TRANSACTION MODELS

PART 3

Outline

- Sharding & Replication –
  - What are they?
  - What issues do they introduce?
- CAP Theorem (Consistency, Availability, Partition-Tolerance – can't have it all)
  - Also look at consistency vs. latency
- Eventual Consistency
  - What is it?
  - What are different models of eventual consistency?
  - Also look at configurations of readers and writers for replication and consistency
- Vector clocks: causal consistency

Configurations

- Definitions
  - **N**: number of nodes that store a replica
  - **W**: number of replicas that need to acknowledge a write operation
  - **R**: number of replicas that are accessed for a read operation
- W+R > N
  - e.g. **synchronous replication** (N=3, W=2, and R=2)
  - write set and read set always overlap
  - strong consistency can be guaranteed through **quorum protocols**
  - risk of reduced availability: in basic quorum protocols, operations fail if fewer than the required number of nodes respond, due to node failure
- W+R = N
  - e.g. **asynchronous replication** (N=2, W=1, and R=1)
  - strong consistency cannot be guaranteed

Based on: “Eventually Consistent” by W. Vogels, 2008

Configurations

- R=1, W=N
  - optimized for **read access**: single read will return a value
  - write operation involves all nodes and risks not succeeding
- R=N, W=1
  - optimized for **write access**: write operation involves only one node and relies on asynchronous updates to other replicas
  - read operation involves all nodes and returns “latest” value
  - durability is not guaranteed in presence of failures
- W < (N+1)/2
  - risk of conflicting writes
- W+R <= N
  - **weak/eventual consistency**

Based on: “Eventually Consistent” by W. Vogels, 2008

Participation Q1

Your team is designing a **global e-commerce database** with:

- **High read volume** (customers browsing products).
- **Critical financial transactions** (orders/payments must be consistent).
- **Multiple data centers** across different regions.

You must choose a **replication model**:

1. **Strong Consistency ($W+R > N$)** – Guarantees latest data, but increases latency.
2. **Eventual Consistency ($W+R \leq N$)** – Faster but may return stale data.

Team Roles:

- **Database Engineer:** Focus on consistency & replication.
- **App Developer:** Focus on read/write performance.
- **Business Manager:** Focus on availability & cost.

Discussion Questions (5 min):

1. Which model does your team choose and why?
2. What is the biggest risk of your choice?
3. How would you adjust W and R to balance performance & consistency?

BASE

- **B**asically **A**vailable, **S**oft state, **E**ventually Consistent
- As consistency is achieved eventually, conflicts have to be resolved at some point
  - read repair
  - write repair
  - asynchronous repair
- Conflict resolution is typically based on a global (partial) ordering of operations that (deterministically) guarantees that all replicas resolve conflicts in the same way
  - client-specified timestamps
  - vector clocks

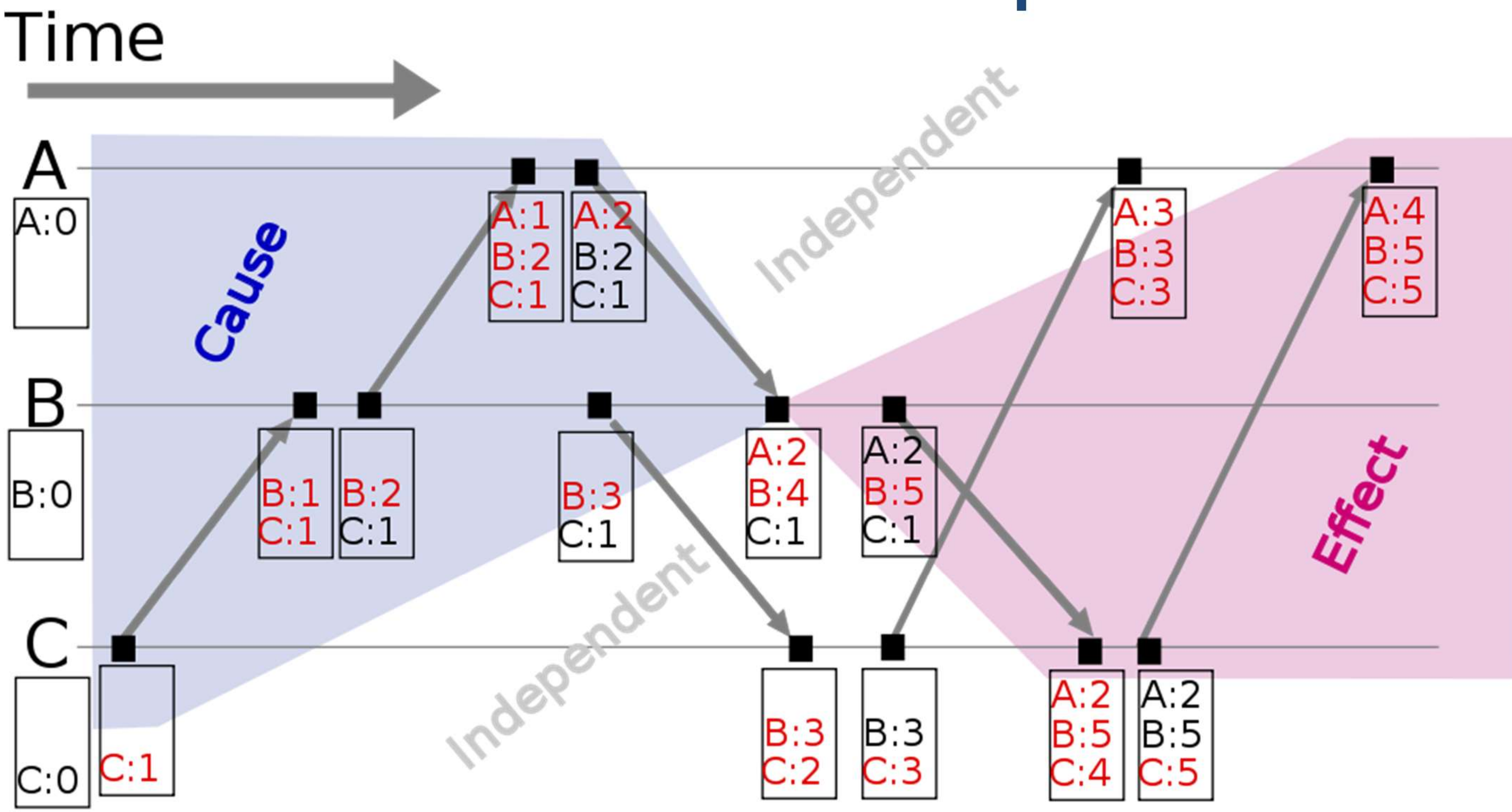
Vector Clocks

- Generate a partial ordering of events in a distributed system and detecting causality violations
- A **vector clock** of a system of *n* processes is an vector of *n* logical clocks (one clock per process)
  - messages contain the state of the sending process's logical clock
  - a local “smallest possible values” copy of the global vector clock is kept in each process
- Vector clocks algorithm was independently developed by Colin Fidge and Friedemann Mattern in 1988

Update Rules for Vector Clocks

- All clocks are initialized to zero
- A process increments its own logical clock in the vector by one
  - each time it experiences an internal event
  - each time a process prepares to send a message
  - each time a process receives a message
- Each time a process sends a message, it transmits the entire vector clock along with the message being sent
- Each time a process receives a message, it updates each element in its vector by taking the pair-wise maximum of the value in its own vector clock and the value in the vector in the received message

Vector Clock Example



Source: Wikipedia (http://www.wikipedia.org)

Participation Question 2

Detecting Causality Violations

- Your team is managing a distributed chat application where messages are replicated across multiple servers using vector clocks. Consider the following event sequence:

1. Alice sends a message: **"Let's meet at 3 PM"** (Vector clock: **[1,0,0]**).
2. Bob replies: **"Okay, see you then"** (Vector clock: **[1,1,0]**).
3. Alice, unaware of Bob's response, updates the time: **"Actually, let's meet at 4 PM"** (Vector clock: **[2,0,0]**).
4. Carol receives both messages at the same time.

Discussion Questions

1. **Using vector clocks, how can Carol determine if these messages are causally related or concurrent?**
2. **Which message, if any, happened before the other, and how do you know?**
3. **If Alice had seen Bob's response before updating the meeting time, how would that have changed her vector clock?**

References

- S. Gilbert and N. Lynch: **Brewer's Conjecture and the Feasibility of Consistent, Available and Partition-Tolerant Web Services.** *SIGACT News 33(2), pp. 51-59, 2002.*
- D. J. Abadi: **Consistency Tradeoffs in Modern Distributed Database System Design: CAP is Only Part of the Story.** *Computer, vol. 45, no. 2, pp. 37-42, Feb. 2012*.
- A. Thomson and D. J. Abadi**. The Case for Determinism in Database Systems.** *Proc. VLDB Endow. 3, 1-2 (September 2010).*
- D. Terry. **Replicated Data Consistency Explained Through Baseball.** *Commun. ACM 56, 12, Dec. 2013.*
- W. Vogels: **Eventually Consistent.** *ACM Queue 6(6), pp. 14-19, 2008.*