Distributed Systems

CAP, and More Consistency Models

CUHK

CAP theorem

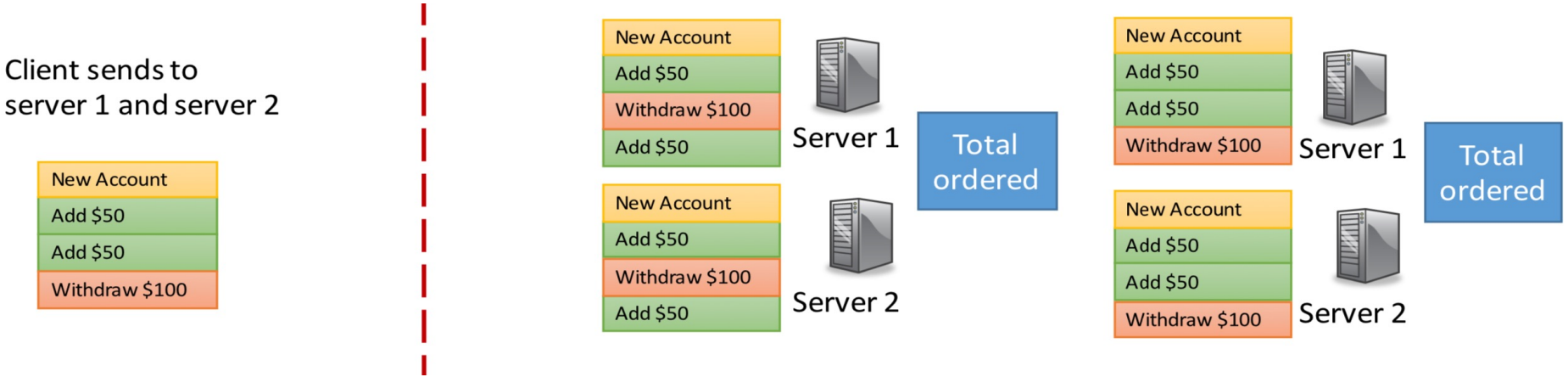
- Fault model now considers **network partitioning** (i.e., message lost, beyond fail-stop failure) happens
  - FLP considers fail-stop only
  - CAP adds a focus on availability
- Tradeoff on
  - Choosing C over A
    - Your request may timeout
  - Choosing A over C
    - Your request returns
      - But the results might not be the latest or consistent

Consistency Model for RSM

- Strongly related to ordering
- Because of CAP
  - Internet companies prefer to keep A, so sacrifice C
- From Strong to Weak
  - From no divergence across replicas to allowing divergence
    - Linearizable consistency (a.k.a. strong consistency)
    - Sequential consistency (=total ordering)
    - Causal consistency
    - FIFO consistency
    - Eventual consistency

Consistency models

- Linearizable consistency (a.k.a strong consistency)
  - All operations appear to have executed atomically in an order that is consistent with the global real-time ordering of operations. (Herlihy & Wing, 1991)
  - “Behave like a single machine”: (1) total order, (2) order match real-time (3) reads its own write
- Sequential consistency (total order)
  - All operations appear to have executed atomically in **some** order that is consistent with the order seen at individual nodes and that is equal at all nodes. (Lamport, 1979)

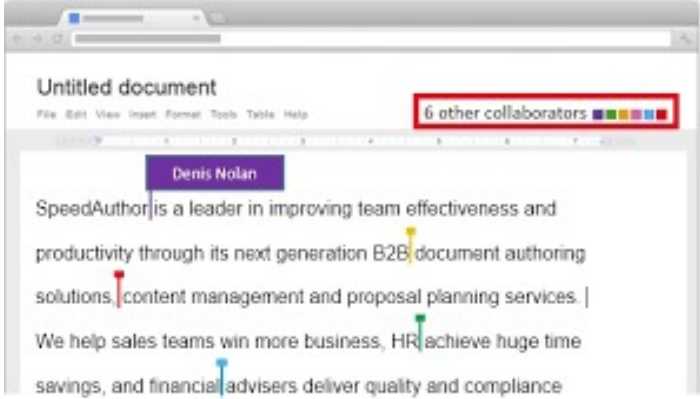


Eventual Consistency

- No lost updates
  - Eventually your updates will be propagated to all replicas
- Order doesn't matter
  - Add Iphone first vs add iphone case first to your shopping cart doesn't matter
- No safety:
  - Some reads will base their work w/o seeing your update
- Need application-level reconciliation that is suitable to individual applications
  - Shopping cart in Amazon (Dynamo)
    - Added item never lost
      - But deleted item can be resurfaced ^_^"
    - Add-to-cart(Cola)
    - Add-to-cart(iPhone13)
    - Delete(Cola)
    - Add-to-cart(iPhone13 case)
    - Network partition, client restart, whatoever
    - Get-cart() → only see iPhone 13
    - Add-to-cart(iMac) **//allow it to carry on**
    - Get-cart() -> see everything back: Cola, iPhone13, iPhone 13 case, iMac
      - The point is,
        - **the iMac is added on an obsolete shopping-cart but it is still there**

Eventual Consistency

- E.g., CRDT (Conflict-free Replicated Data Type)
  - An abstract data type whose all possible internal states form a lattice
  - Two instances of a CRDT can be merged
  - Eventually, the states of two CRDT replicas that may have seen different orders can always be merged into a new final state (the tip of the lattice) that incorporates all the inputs seen by both.
  - https://crdt.tech/



https://www.serverless.com/blog/crdt-explained-supercharge-serverless-at-edge

Causal Consistency

- Remember “I lost my son” and “Thanks God” example?
- Best consistency in the potential presence of P:

1. Availability. All operations issued to the data store complete successfully. No operation can block indefinitely or return an error signifying that data is unavailable.

2. Low Latency. Client operations complete “quickly.” Commercial service-level objectives suggest average performance of a few milliseconds and worse-case performance (i.e., 99.9th percentile) of 10s or 100s of milliseconds [16].

3. Partition Tolerance. The data store continues to operate under network partitions, e.g., one separating datacenters in Asia from the United States.

4. High Scalability. The data store scales out linearly. Adding N resources to the system increases aggregate throughput and storage capacity by $O(N)$.

To appear in *Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP'11)*

Don’t Settle for Eventual:
Scalable Causal Consistency for Wide-Area Storage with COPS

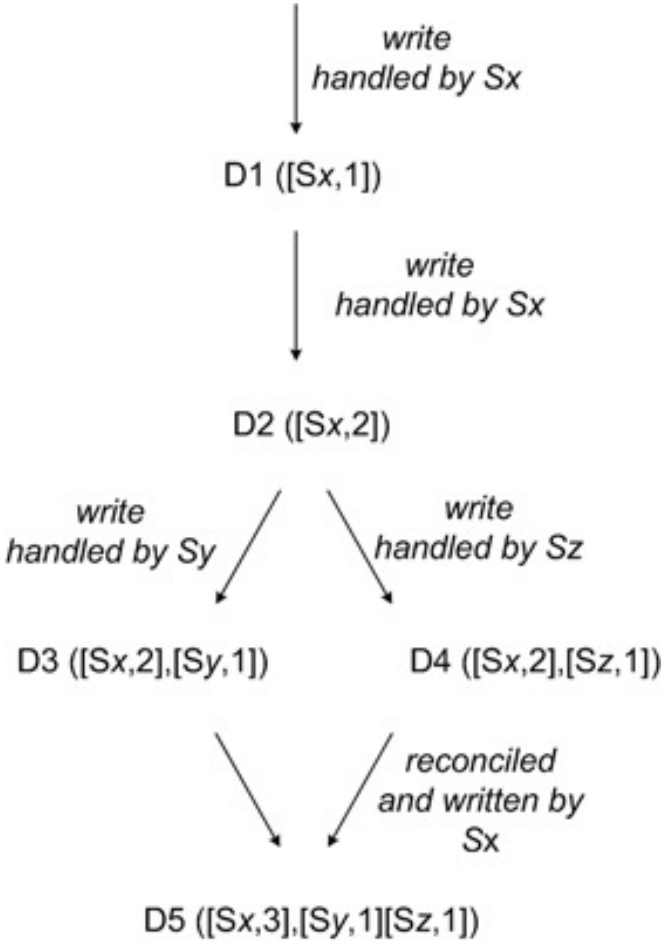
Wyatt Lloyd*, Michael J. Freedman*, Michael Kaminsky†, and David G. Andersen‡
*Princeton University, †Intel Labs, ‡Carnegie Mellon University

Consistency vs Scalability Tradeoff

- Recently, from C vs A to C vs S
- "*The first principle of successful scalability is to batter the consistency mechanisms down to a minimum, move them off the critical path, hide them in a rare visited corner of the system, and then make it as hard as possible for application developers to get permission to use them*"
  - By James Hamilton (AWS VP)
- That is, the coordination used for upkeeping consistency (e.g., Paxos, Raft, 2PC) is the bottleneck of both throughput and scalability
  - 90% of time can spend on waiting for coordination
  - How can we avoid coordination as Hamilton recommend?

Eventual Consistency in Dynamo

- Use of
  - Vector-clock
  - Multi-versioning
  - Reconciliation
- D is the object (shopping cart)
  - D5 is version 5 of the shopping-cart
- S is the server



Causal+ (a.k.a. convergent causal consistency)

- Original definition of Causal Consistency:
  - Refer to causal ordering
  - Said nothing about ”convergent” (replica consistency)
- Causal+ //but many people regard this as the casual consistency
  - Causal consistent + “Convergent”: Convergent causal consistency

The Potential Dangers of Causal Consistency and an Explicit Solution

Peter Bailis[†], Alan Fekete[◇], Ali Ghodsi[†,‡], Joseph M. Hellerstein[†], Ion Stoica[†]

[†] University of California, Berkeley [◇] University of Sydney [‡] KTH/Royal Institute of Technology

{pbailis, alig, hellerstein, stoica}@cs.berkeley.edu, alan.fekete@sydney.edu.au

Confluence consistency

- A new consistency model that requires no coordination
- Confluence looks at agreement of **application outcomes**
  - It rules out **application-level** inconsistency due to races and non-deterministic ordering of messages, while
    - **permitting non-deterministic ordering and timings of lower-level operations** that may be costly (or sometimes impossible) to present in practice
- Whereas any previously seen consistency is defined based on the input operation order (broadcast) or the RSM state
  - E.g., inconsistent RSM (DB) states can still lead to consistent application outcomes

Confluence consistency

- What type of applications (problems) can be coordination-free?
- Answer: Monotonic problems
  - Knowing more would (or would not) change the outcome of the program P?
  - Existence problems vs Non-Existence problems
  - E.g., Given a graph,
    - “is there a cycle?” once a positive example is found, the answer is affirmative
      - Distributed deadlock detection
    - “is there no cycle?” = for-all questions
      - Distributed garbage collection

https://blog.acolyer.org/2019/03/06/keeping-calm-when-distributed-consistency-is-easy/

CALM

- Consistency And Logical Monotonicity (CIDR'11)
  - "*A program has a consistent, coordination-free distributed implementation if and only if it is monotonic*"
  - "*If you keep things CALM then you are always in a position to 'carry on' without needing to coordinate*"
- Then, what problems are *monotonic*, what are not?
  - Analogy: what problem is P, or NP?
  - Def: Confluent operation
    - An operation is confluent if it produces the same sets of outputs for any non-deterministic ordering from the same set of inputs
    - Like commutative (e.g., addition, multiplication) operation,
      - **the order of receiving and applying does not matter;**
      - **may diverge in the middle but eventually converge**

Confluence operations

- Confluence can be applied to
  - individual operations, and composed to
    - components in a dataflow, or
    - even entire distributed programs.
- If we restrict ourselves to building a program by composing confluent operations, the whole program is confluent by construction and thus is **coordination-free!**

Confluence operations in hindsight

- E.g., Amazon Dynamo KV store (shopping cart)
  - Order of “add iPhone” and “add Mario Kart” doesn’t matter as long as the replicas receive the set same of “shopping cart”
  - How to handle “delete” then?
    - Keep a deleted-set
      - Delete items are appended-only to “Delete-set”
      - Then can merge as usual
  - What about “checkout” then?
    - Things would change if “checkout” message arrives earlier than insert/delete (all insert/delete shall be discarded after checkout)
      - Only use coordination on checkout
- That is, identify which part of the program needs (no) coordination and program separately.