CAP Theorem and Eventual Consistency

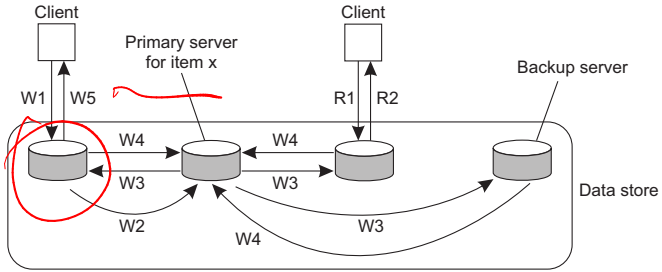
Distributed Systems Spring 2020

Lecture 15

Agenda

- Last time: How consistency models can be implemented
  - Primary-based methods
- CAP theorem and its implications
  - Most famous observation made about distributed systems in the last 2 decades
- Eventual Consistency
- Next time: CRDT's

Recap: Primary based protocol

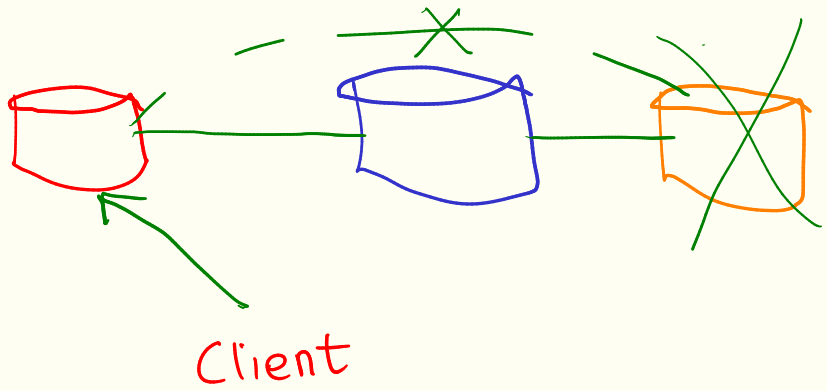


W1. Write request
W2. Forward request to primary
W3. Tell backups to update
W4. Acknowledge update
W5. Acknowledge write completed

R1. Read request
R2. Response to read

Tradeoffs In Consistency Models

- Sequential Consistency: Blocking writes via multicasting
- Eventual Consistency: Non-blocking writes lazily propagated
- What if we cannot write to a replica?
  - Perhaps due to Network or hardware failure
- Should the write operation continue?
  - For strong consistency, we cannot allow write to proceed
  - System becomes "unavailable" to the client



CAP Theorem

C, A, and P:

- Consistency: Strong Consistency (Linearizability)
- Availability: Clients can always perform operations
- Partition Tolerance: If some replicas are unreachable, system function should not be compromised

Naive **CAP theorem: Pick any two***

With a few important caveats!

- CAP theorem is widely misunderstood and misapplied

C, A

A, P

C, P

✓

C, A, P

X

CAP Theorem Details

- Partition tolerance usually cannot be sacrificed
- But, dont need to sacrifice either C or A when there are no partitions.
- Partitions usually detected via timeouts
- System can enter "C" or "A" mode if partition detected
  - Cancel operation and have reduced availability
  - Continue operation and risk inconsistency

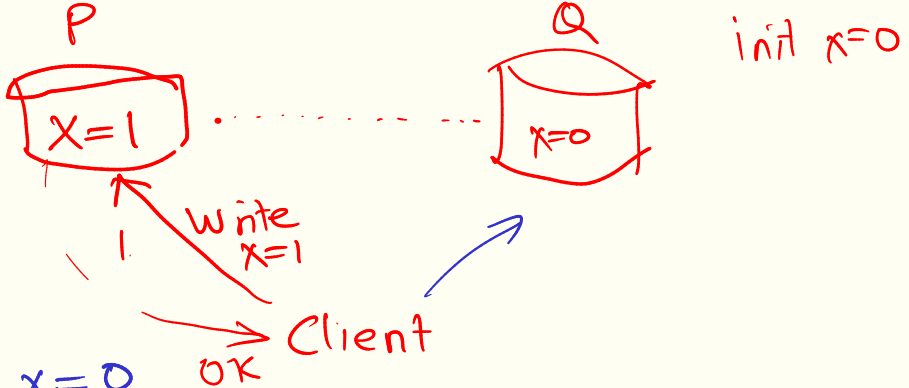
A more precise statement of the CAP Theorem:

If there is a network partition, you must choose either consistency or availability.

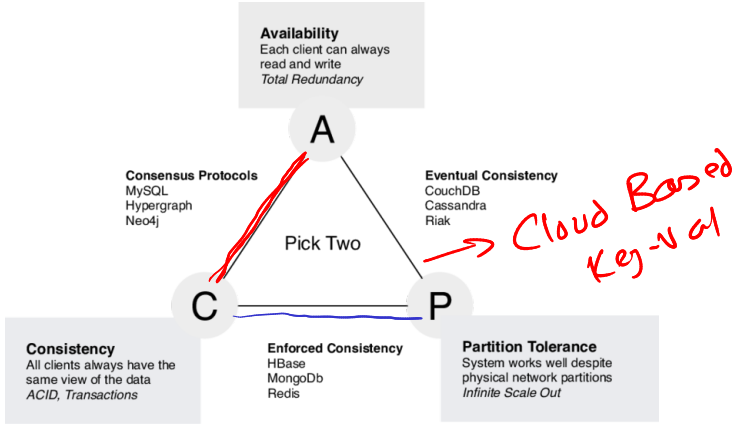
Intuitive "Proof"

- System with two replicas P, Q.
- Client writes to P
- Network partition, so P cannot update Q
- Client reads from Q, but gets stale result. Hence not consistent.
- If P cannot update Q, it can refuse to process update. Hence not available.

QED



Distributed Storage Systems Tradeoffs



Partitions

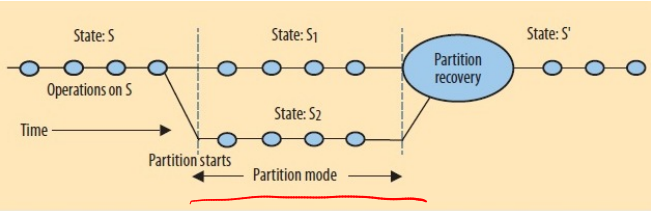
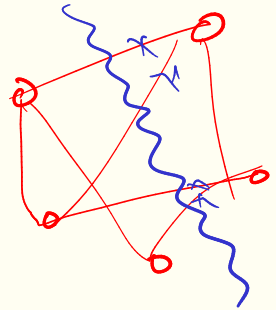


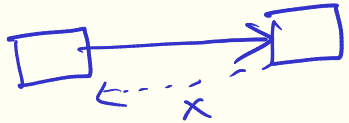
Figure 1. The state starts out consistent and remains so until a partition starts. To stay available, both sides enter partition mode and continue to execute operations, creating concurrent states S_1 and S_2, which are inconsistent. When the partition ends, the truth becomes clear and partition recovery starts. During recovery, the system merges S_1 and S_2 into a consistent state S' and also compensates for any mistakes made during the partition.

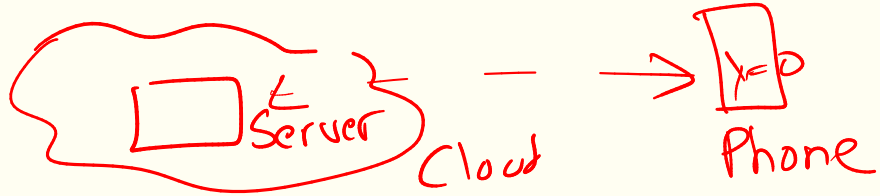
Ideal for A,P Systems

Dealing with Partitions

- No global view of partitions: some replicas may be unreachable from certain sources
  - Can have a "one-sided" partition
- Offline-mode: long periods of unavailability
- Mask unavailability: log operations and replay later
  - Credit-card machines
- "Merge" partitions using version vectors, or last-write-wins
  - Source control systems typically use this
- Prohibit "risky" operations (such as deletes)

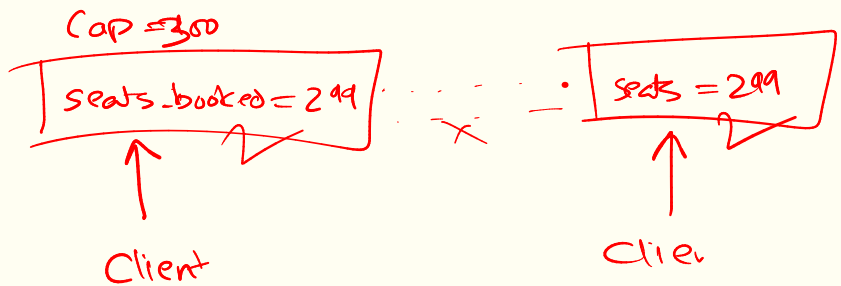






Compensating for Mistakes

- Mistakes made during partitions can be fixed in many ways
- Airline overbooking: compensation is literal $
- Amazon shopping carts: Union of two carts. Deleted items may reappear, and customer manually corrects final cart, or escalates to customer service
- ATMs: Withdrawals can proceed even when partitioned. But banks place a hard-limit on withdrawals to limit the risk.

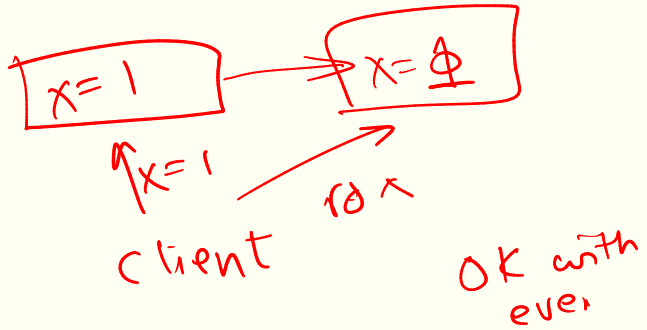


Eventual Consistency

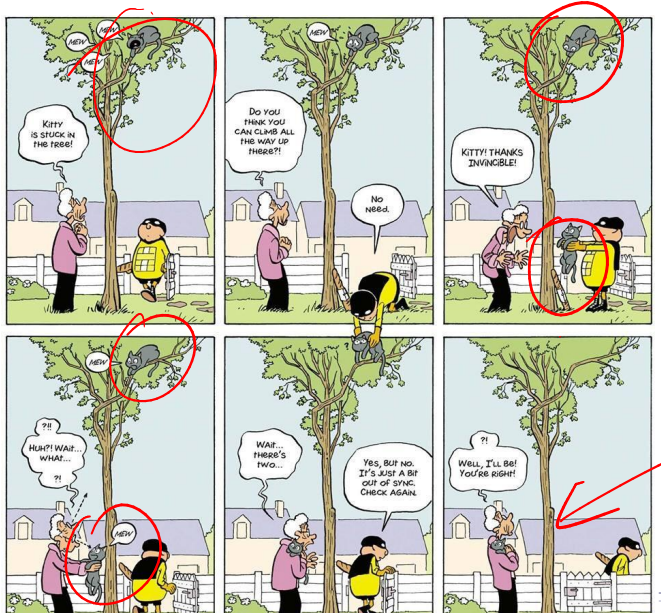
- If no additional updates are made to a given data item, all reads to that item will eventually return the same value.
- No guarantees about when the replicas will converge
- Usually implemented through **async** writebacks
- Conflicts merged/resolved asynchronously in the background
- On conflict: Arbitrate or roll-back

What are the safety and liveness properties?

Eventually consistent
~~esp~~ with infrequent writes



Eventual Consistency In a Nutshell



Eventually, System is consistent

PACELC

- If there is a Partition, how does the system tradeoff A and C
- Else, how does it tradeoff latency and C
- Generalization of CAP
- Latency differences between synchronous and async operations, location of primary, etc.

Consistency	Reads	Writes	Comments
Linearizability	Slow	Slow	ToB for all ops
Sequential	Fast	Slow	Local-read algo
Causal	Fast-ish	Fast	Wait for causally preceeding ops
Eventual	Fast	Fast	Easy to implement

Causal possible even with partitions

CAP Thm misapplication

⇒ Eventual Consistency

More Eventual Consistency

- Can probabilistically bound the staleness
- New approaches can provide stricter consistency guarantees on top of eventually consistent stores
  - COPS, Eiger, Bolt-on causal Consistency, ...

Causal Consistency