Eventual Consistency Today: Limitations, Extensions and Beyond

Peter Bailis and Ali Ghodsi, UC Berkeley

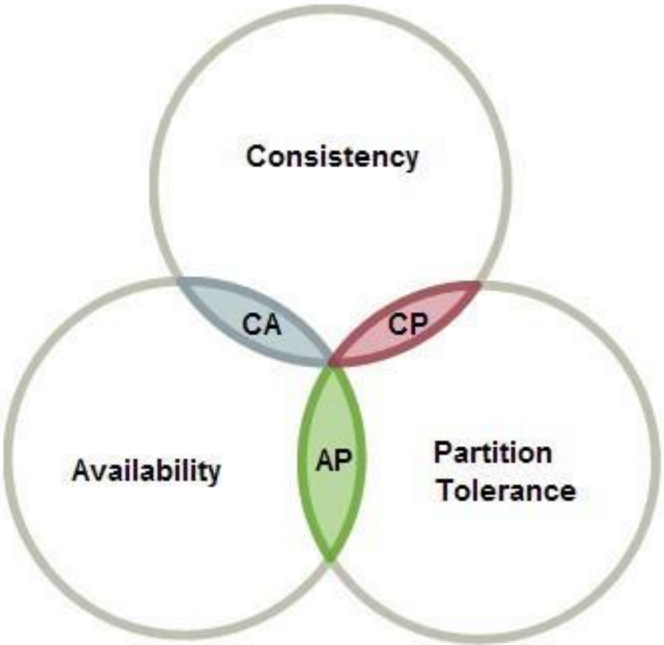
Presenter: Yifei Teng
Part of slides are cited from Nomchin Banga

Road Map

- Eventual Consistency: History and Concepts
- How eventual is eventual consistency?
- Programming eventual consistency
- Stronger than eventual
- Conclusions

CAP Theorem

- Maintaining single-system image has a cost
- Note that you can't "sacrifice" partition toleran
- Consistency-availability tradeoffs
- Consistency-latency tradeoffs



Eventual Consistency

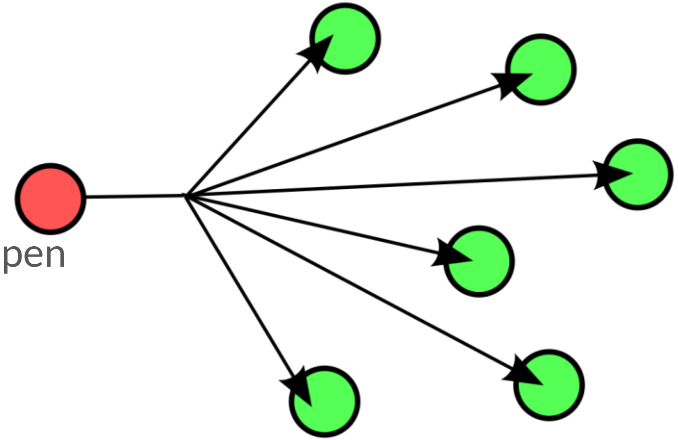
"…changes made to one copy eventually migrate to all. If all update activity stops, after a period of time all replicas of the database will converge to be logically equivalent: each copy of the database will contain, in a predictable order, the same documents; replicas of each document will contain the same fields."

Compare SSI and Eventual Consistency

- The predictable order will not necessarily correspond to the execution order
    - Order confusion
- Eventual consistency doesn't specify windows before converge
    - Arbitrary value
- SSI provides eventual consistency, but not vice versa
    - The "eventual" is immediate

Anti-entropy

- Anti-entropy policy
  - Boardcast is the simpliest one
  - Choose a winning when concurrent writes happen
- Asynchronous process
  - Non blocking anti-entropy



Broadcast

Great Properties

- Easy to implement:
  - does not require writing difficult “corner-case” code to deal with complicated scenarios
- All operations complete locally:
  - Low latency
- Systems can control the frequency of anti-entropy

Safety and Liveness

- Safety – nothing bad happens
    - every value that is read was, at some point in time, written to the database
- Liveness – all requests eventually receive a response
- Eventual Consistency is purely a liveness system.
    - Replicas will converge, but there are no guarantees with respect to what happens

Metrics and Mechanisms

- Metrics
  - Window of consistency: How long for a write to be available to read?
  - Version: How many version old will a returned value be?
- Mechanisms
  - Measurement: How consistent is a store under the workload now?
  - Prediction: How consistent will a store be under a given situation?

Probabilistically Bounded Staleness (PBS)

- Provide an expectation of recency for reads of data items
  - 100 milliseconds after a write completes, 99.9 percent of reads will return the most recent version
  - 85 percent of reads will return a version that is within two of the most recent

Eventual Consistency is “good enough

13.6ms

202ms

SimpleDB 500ms

cassandra 200ms

amazon web services S3 12s

Programming Eventual Consistency

- Compensation: a way to achieve safety retroactively
  - Restore guarantees to users
- Evaluate the benefit
  - B: The benefit of weak consistency
  - C: Cost of each compensation
  - R: Rate of anomalies

→ Maximize B - CR

Compensation by Design

- Compensation is error-prone and laborious
- Some researches provide compensation-free programming
    - CALM theorem: consistency as logical monotonicity
    - ACID 2.0: associativity, commutativity, idempotence, and distributed
    - CRDT: commutative, replicated data types

CALM (Consistency as Logical Monotonicity)

- Monotonicity: compute an ever-growing set of facts and do not ever “retract” facts that they emit
- A program satisfies CALM can always be safely run on an eventually consistent store.
- Monotonic operations
  - Initializing variables, add set members, and testing a threshold
- Non-monotonic operations
  - variable overwrites, set deletion, counter resets, and negation

ACID 2.0

- Associativity, commutativity, idempotence, and distributed
- Commutative and associative program can tolerate message re-ordering
- Idempotence allows the use of at-least-once message delivery
- Applying these design patterns can achieve logical monotonicity

CRDT (Commutative, Replicated Data Types)

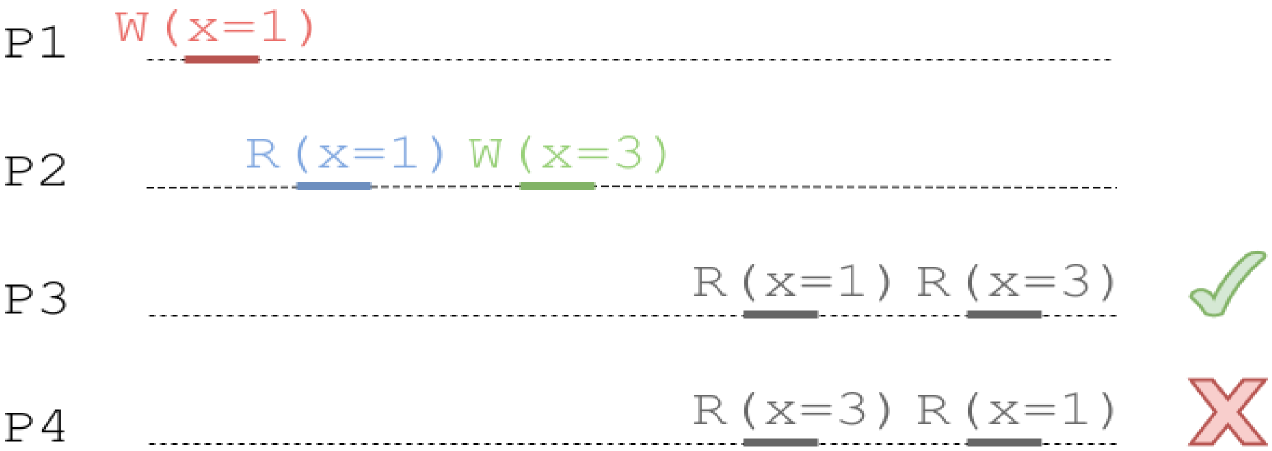
- CRDTs embodies CALM and ACID 2.0 principles
- Any program that correctly uses CRDTs is guaranteed to avoid any safety violations.
- A key property is separating data store and application-level consistency concerns.
    - Enjoy strong application level consistency
    - And the benefits of weak distributed read/write consistency
    - G-Counter is a typical example

Stronger than Eventual

- Compensation requires dealing with inconsistency outside of systems
- CRDT limites the operations an application can employ
- Research shows that no consistency model stronger than causal consistency is available in the presence of partitions
- Causality can be added to eventual consistency

Causal Consistency

- guarantees each process’s write are seen in order, transitive data dependencies hold



Pushing the Limits

- Causality
  - COPS, Eiger provide causality with low latency and high availability.
  - Many eventual consistent applications can be augmented with causality
- Re-architecting weak isolation databases in distributed environment
  - Keep the same ACID properties
  - High availability

Recognizing the Limits

- A fundamental cost to remaining highly available and low latency
- Staleness guarantees are impossible in a highly available system
  - give me the latest value
- Cannot maintain arbitrary global correctness constraints
  - “create an account with ID 50 if the account does not exist”

Conclusions

- Eventual consistency improves availability and performance at the cost of guarantees
- Eventual consistency not perfect for every task, but good enough for many applications.
- And eventual consistency will be admired in the future because of its performance and availability

Thanks You!