Cloud and Cluster Data Management

SCALABLE CONSISTENCY AND TRANSACTION MODELS

PART 2

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

Problems with CAP (from D. Abadi)

- **Asymmetry of CAP Properties**
  - Consistency (C) applies system-wide, Availability (A) only during partitions
- **Not Three Independent Choices**
  - In practice, CA and CP are similar, as A is only sacrificed when a partition occurs
- **Misuse of CAP**
  - "CAP forces me to drop consistency for availability" → Not always true

Source: Daniel Abadi, Yale University

Beyond CAP – Additional Costs of Consistency

- **Synchronization Overhead**
  - Synchronization mechanisms add computational and communication overhead.
- **Latency Trade-offs**
  - Geographically distributed replicas introduce delays
  - Higher reliability requires cross-datacenter replication

Source: Daniel Abadi, Yale University

PACELC – A More Complete Trade-off Model

General scheme: P[A|C]/E[L|C]

- **P (Partition)** → Choose between Availability (A) or Consistency (C)
- **E (Else: Normal Operation)** → Choose between Latency (L) or Consistency (C)

- PA/EL
  - Dynamo, SimpleDB, Cassandra, Riptano, CouchDB, Cloudant
- PC/EC
  - ACID compliant database systems
- PA/EC
  - GenieDB
- PC/EL
  - Less common

Source: Daniel Abadi, Yale University

A Case for P*/EC

- Increased push for horizontally scalable transactional database systems
  - cloud computing
  - distributed applications
  - desire to deploy applications on cheap, commodity hardware
- Vast majority of currently available horizontally scalable systems are P*/EL
  - developed by engineers at Google, Facebook, Yahoo, Amazon, etc.
  - these engineers can handle reduced consistency, but it's really hard, and there needs to be an option for the rest of us
- Also
  - distributed concurrency control and commit protocols are expensive
  - once consistency is gone, atomicity usually goes next → NoSQL

Source: Daniel Abadi, Yale University

Key Problems to Overcome

- High availability is critical, replication must be a first-class citizen
- Today's systems generally act, then replicate
  - complicates semantics of sending read queries to replicas
  - need confirmation from replica before commit (increased latency) if you want durability and high availability
  - In-progress transactions must be aborted upon a primary failure
- Want system that replicates then acts
- Distributed concurrency control and commit are expensive, want to get rid of them both

Source: Daniel Abadi, Yale University

Key Idea

- Instead of weakening ACID, strengthen it
- Challenges
  - guaranteeing equivalence to *some* serial order makes active replication difficult
  - running the same set of transactions on two different replicas might cause replicas to diverge
- Disallow any nondeterministic behavior
- Disallow rollbacks caused by DBMS
  - disallow deadlock (restrict locking order)
  - distributed commit much easier if there are no rollbacks

Source: Daniel Abadi, Yale University

Consequences of Determinism

- Replicas produce the same output, given the same input
  - facilitates active replication
- Only initial input needs to be logged, state at failure can be reconstructed from this input log (or from a replica)
- Active distributed transactions not rolled back upon node failure
  - greatly reduces (or eliminates) cost of distributed commit
  - don't have to worry about nodes failing during commit protocol
  - don't have to worry about effects of transaction making it to disk before promising to commit transaction
  - any node that potentially can deterministically roll back the transaction need only send one message
  - this message can be sent in the middle of the transaction, as soon as it knows it will commit

Source: Daniel Abadi, Yale University

Strong vs. Weak Consistency

- Strong consistency
  - after an update is committed, each subsequent access will return the updated value
- Weak consistency
  - the systems does not guarantee that subsequent accesses will return the updated value
  - a number of conditions might need to be met before the updated value is returned
  - **inconsistency window:** period between update and the point in time when every access is guaranteed to return the updated value

Based on: "Eventually Consistent" by W. Vogels, 2008

Eventual Consistency

- Specific form of weak consistency
- "If no new updates are made, eventually all accesses will return the last updated values"
- In the absence of failures, the maximum size of the inconsistency window can be determined based on
  - communication delays
  - system load
  - number of replicas
  - …
- Not a new esoteric idea!
  - Domain Name System (DNS) uses eventual consistency for updates
  - RDBMS use eventual consistency for asynchronous replication or backup (e.g. log shipping)

Based on: "Eventually Consistent" by W. Vogels, 2008

However …

Eventual Consistency and Perpetual Inconsistency are not mutually exclusive!

See Doug Terry paper on consistency of baseball scores.

- Consistent prefix
- Monotonic reads

Models of Eventual Consistency

- Causal Consistency
  - if A communicated to B that it has updated a value, a subsequent access by B will return the updated value, and a write is guaranteed to supersede a causally earlier write
  - access by C that has no causal relationship to A is subject to normal eventual consistency rules
- Read-your-writes Consistency
  - special case of the causal consistency model
  - after updating a value, a process will always read the updated value and never see an older value
- Session Consistency
  - practical case of read-your-writes consistency
  - data is accessed in a session where read-your-writes is guaranteed
  - guarantees do not span over sessions

Based on: "Eventually Consistent" by W. Vogels, 2008

Models of Eventual Consistency

- Monotonic Read Consistency
  - if a process has seen a particular value, any subsequent access will never return any previous value
- Monotonic Write Consistency
  - system guarantees to serialize the writes of one process
  - systems that do not guarantee this level of consistency are hard to program

- Properties can be combined
  - e.g. monotonic reads plus session-level consistency
  - e.g. monotonic reads plus read-your-own-writes
  - quite a few different scenarios are possible
  - it depends on an application whether it can deal with the consequences

Based on: "Eventually Consistent" by W. Vogels, 2008

Participation Question 1

Pick a big data application that uses eventual consistency (e.g., X, Snapchat, Facebook, Instagram,…) and analyze its eventual consistency model (or mix of models).