Performance and Forgiveness

June 23, 2008

Margo Seltzer
Harvard University
School of Engineering and Applied Sciences

Margo Seltzer

Architect

Outline

- A consistency primer
- Techniques and costs of consistency
- When weaker forms of consistency makes sense.
- Case Study
  - Amazon Dynamo
  - Google Single Sign On
- Conclusions

A Consistency Primer

- Transactions: the gold standard
- Degrees of isolation
- Distributed systems
  - Consistency models
  - Read/write consistency

Transactions

- The gold standard: ACID
- **Atomicity**: Multiple operations are all or nothing.
- **Consistency**: Data consistency maintained even in the face of concurrency.
- **Isolation**: Data behave as if each transaction runs single-threaded.
- **Durability**: Changes persist in the face of any kind of failure.

Distributed Transactions

- Requires distributed locking.
- Commit protocol (2 phase commit) is expensive.
  - One site (leader) says “commit.”
  - Broadcast “prepare” to everyone.
  - Everyone does something durable.
  - Everyone responds to leader.
  - Leader then make commit durable, releases locks, and broadcasts, “commit,” to everyone.
  - Everyone makes commit durable, releases locks.

Is all this necessary?

- Transactions are serializable (degree 3).
- Read committed (degree 2)
  - Provides cursor stability
- Read uncommitted (degree1)
  - Allow read of dirty data

May make the locking problem go away,
but doesn't do much for cost of commit.

Moving to Distributed Data

- How do distributed systems provide high availability?
  - Replication
- Key question: how consistent is the data at different replicas?

Consistency Models

- Perfect consistency: all replicas are guaranteed to return the same data at all times.
- Eventual consistency: if no updates take place for a long time, all replicas will eventually become consistent.
- Causal consistency: writes that depend upon one another are seen in the same order by everyone.

Read/Write Consistency

- Read consistency
  - What you read will be internally consistent, but may be out of date.
- Write consistency
  - Sets of logical writes appear atomically.

Outline

- A consistency primer
- Techniques and costs of consistency
- When weaker forms of consistency makes sense.
- Case Study
  - Amazon Dynamo
  - Google Single Sign On
- Conclusions

Real Distributed Transactions

- Writing applications is easy!
- Recovery is tricky (e.g., leader elections, leases, etc).

- Requires distributed locking.
- Expensive commit processing

Settle for read/write consistency instead.
Forget about true distributed locking.

Replicas and Commits

- Fully synchronous: an updater waits until it knows that everyone has seen and done everything.
- Semi-synchronous: an updater waits until it knows that “enough” participants have seen and done everything.
- Probabilistically synchronous: an updater waits until it knows that “enough” participants have seen everything.
- Asynchronous: updaters send and pray.

Fully Synchronous

Semi-synchronous

Partially Synchronous

Asynchronous

Outline

- A consistency primer
- Techniques and costs of consistency
- When weaker forms of consistency makes sense.
- Case Study
  - Amazon Dynamo
  - Google Single Sign On
- Conclusions

The Real World

- If the user can’t see inconsistency, it doesn’t really exist.
- If an application can resolve inconsistency, don’t sweat it.
- If the cost/benefit ratio makes it too expensive, don’t worry about it.
- If you can recreate it, don’t sweat it.

Outline

- A consistency primer
- Techniques and costs of consistency
- When weaker forms of consistency makes sense.
- Case Study
  - Amazon Dynamo
  - Google SSO
- Conclusions

Dynamo

- Amazon's highly available key/value store.
- Used for things like:
  - Preferences
  - Shopping carts
  - Best-seller lists
  - Session management
- Designed for reliability over consistency.
- Strict SLA.

Dynamo: Amazon's Highly Available Key-value Store, SOSP 2007

Dynamo Implementation

- Key/data store.
- DHT provides distribution, partitioning, replication.
- All data are versioned (vector clock).
- Applications handle multiple inconsistent versions.
- Four possible storage engines on local site:
  - **Berkeley DB (native key/data store)**
  - Berkeley DB Java Edition (native key/data store)
  - MySQL (RDBMS)
  - Persistently backed in-memory buffer

Data Integrity Architecture

- Availability trumps everything.
  - Runs on 10,000s of servers
  - System unavailability costs real dollars
  - Components fail
  - Service has strict 99.9%-ile SLA
- Single key writes
  - Do not overwrite; make new copies
  - No need for transactions
- If multiple copies disagree, vote.

Trade-offs

- Vector clocks detect causal orderings, but …
  - Can still get inconsistencies (parallel operations)
  - Application reconciles
- No transactional updates, but …
  - Use multiple nodes for read/write, but
  - Fewer than that required for quorum
  - Get response from healthiest nodes, not necessarily the “preferred nodes.”

Outline

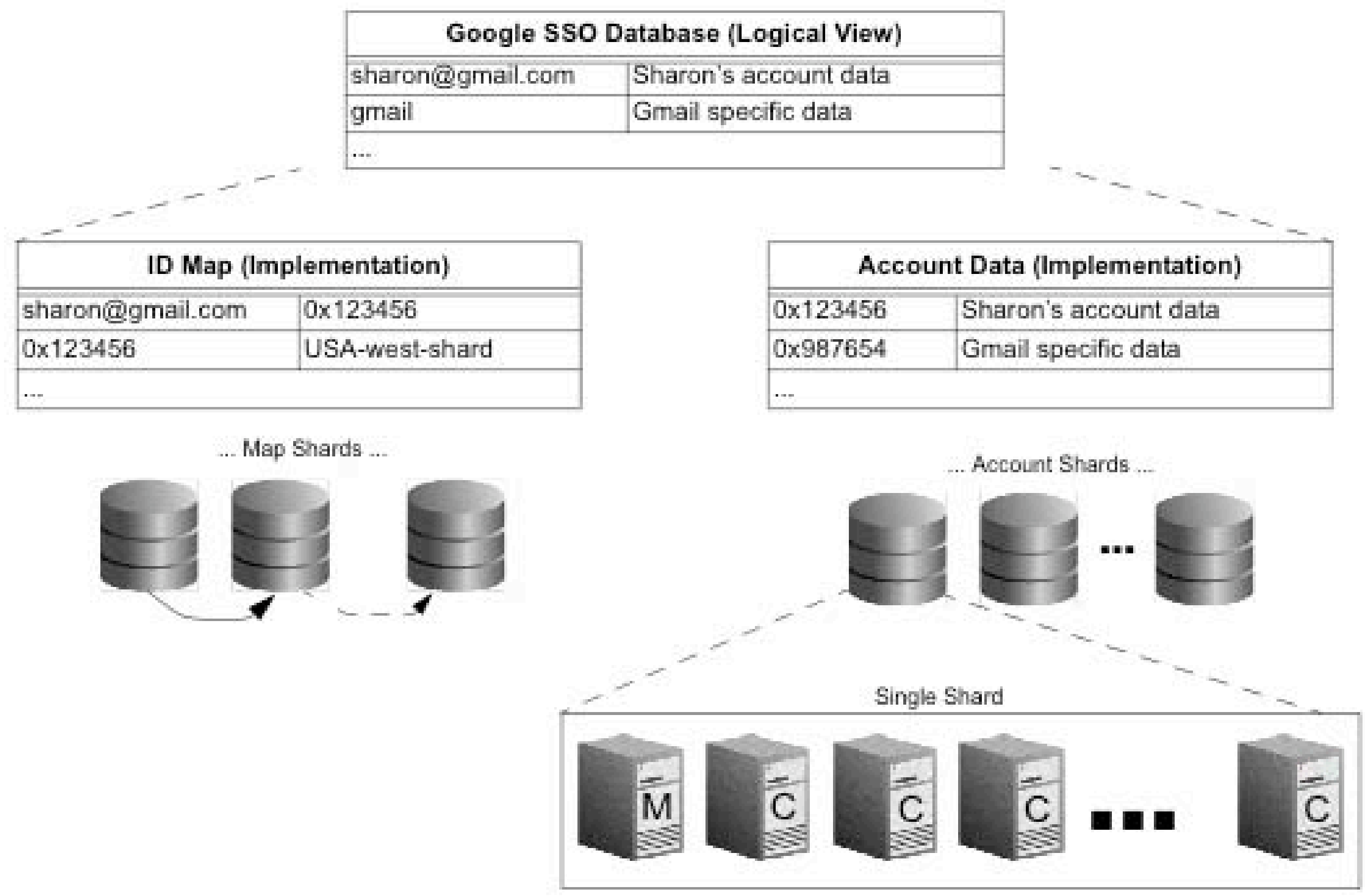
- A consistency primer
- Techniques and costs of consistency
- When weaker forms of consistency makes sense.
- Case Study
  - Amazon Dynamo
  - Google Single Sign On
- Conclusions

Single Sign On (SSO)

- Google Accounts: supports many services.
- SSO reliability sets upper bound on application reliability.
- **Required single-copy consistency.**
- Data partitioned for load balancing.
- Data replicated for availability.

Data Management for Internet-Scale Single-Sign-On, WORLDS 2006

SSO Architecture



From: Data Management for Internet-Scale Single-Sign-On, WORLDS 2006

Data Integrity Architecture

- Master implements quorum protocol.
  - Wait for acks from more than half.
- Majority needed to elect new master.
- Master leases allow consistent reads without read quorum.

Trade-offs

- Consistent reads must go to master, but …
  - Stale reads can happen on replicas.
- Large groups have up to 15 replicas, but …
  - Only 5 of those can become masters.
  - Commit quorum is only 5 (not 15).
  - 10 sites do not contribute to commit latency
- Spread replicas geographically, but
  - Not too far as replicas communicate at commit.

Outline

- A consistency primer
- Techniques and costs of consistency
- When weaker forms of consistency makes sense.
- Case Study
  - Amazon Dynamo
  - Google Single Sign On
- Conclusions

Conclusions

- Consistency is not always a requirement.
- Many agents can make up for inconsistent data:
  - Applications
  - System software
  - Users
  - Customer service
- Make trade-offs explicitly
  - Know the values of the pieces to trade

Thank You!

- For further reading:
  - Life beyond distributed transactions: An Apostate's Opinion, Pat Helland, CIDR
  - http://www-db.cs.wisc.edu/cidr/cidr2007/papers/cidr07p15.pdf

margo@eecs.harvard.edu

margo.seltzer@oracle.com