Transactions Across Datacenters

(and other weekend projects)

Ryan Barrett
May 27, 2009

> *Of three properties of distributed data systems – consistency, availability, partition tolerance – choose two.*
>
> Eric Brewer, CAP theorem, PODC 2000

> *Scaling is hard.*
>
> Various

Context

- *multihoming* (n): operating from multiple datacenters simultaneously
- ~~serving~~
- ~~data processing~~
- ~~read-only data~~
- read/write data (the hardest kind!)
- Case study: App Engine datastore

What's ahead and takeaways

- Consistency?
- Why transactions?
- Why across datacenters?
- Multihoming
- Techniques and tradeoffs
- Conclusions

Consistency

- Weak
- Eventual
- Strong
- Use case: read after write

Weak consistency

- After a write, reads *may or may not* see it
- Best effort only
- "Message in a bottle"
- App Engine: memcache
- VoIP, live online video
- Realtime multiplayer games

Eventual consistency

- After a write, reads *will eventually* see it
- App Engine: mail
- Search engine indexing
- DNS, SMTP, snail mail
- Amazon S3, SimpleDB

Strong consistency

- After a write, reads *will* see it
- App Engine: datastore
- File systems
- RDBMSes
- Azure tables

What's ahead

- Consistency?
- **Why transactions?**
- Why across datacenters?
- Multihoming
- Techniques and tradeoffs
- Conclusions

Why transactions?

- Cliched example: transfer money from A to B
  - subtract from A
  - add to B
- What if something happens in between?
  - another transaction on A or B
  - machine crashes
  - ...

Why transactions?

- Correctness
- Consistency
- Enforce invariants
- ACID

What's ahead

- Consistency?
- Why transactions?
- **Why across datacenters?**
- Multihoming
- Techniques and tradeoffs
- Conclusions

Why across datacenters?

- Catastrophic failures
- Expected failures
- Routine maintenance
- Geolocality
    - CDNs, edge caching

Why *not* across datacenters?

- Within a datacenter
    - High bandwidth: 1–100Gbps interconnects
    - Low latency: < 1ms within rack, 1–5ms across
    - Little to no cost
- Between datacenters
    - Low bandwidth: 10Mbps–1Gbps
    - High latency: 50–150ms
    - $$$ for fiber

What's ahead

- Consistency?
- Why transactions?
- Why across datacenters?
- **Multihoming**
- Techniques and tradeoffs
- Conclusions

Multihoming

- Hard problem.
- ...consistently? Harder.
- ...with real time writes? Hardest.
- What to do?

Option 1: Don't

- ...instead, bunkerize.
    - Most common
    - Microsoft Azure (tables)
    - Amazon SimpleDB
- Bad at catastrophic failure
    - Large scale data loss
    - Example: SVColo
- Not great for serving
    - No geolocation

Option 2: Primary with hot failover(s)

- Better, but not ideal
    - Mediocre at catastrophic failure
    - Window of lost data
    - Failover data may be inconsistent
- Amazon Web Services
    - EC2, S3, SQS: choose US or EU
    - EC2: Availability Zones, Elastic Load Balancing
- Banks, brokerages, etc.
- Geolocated for reads, not for writes

Option 3: True multihoming

- Simultaneous writes in different DCs
- Two way: hard
- N way: harder
- Handle catastrophic failure, geolocality
- ...but pay for it in latency

What's ahead

- Consistency?
- Why transactions?
- Why across datacenters?
- Multihoming
- **Techniques and tradeoffs**
- Conclusions

Techniques and tradeoffs

	Backups	M/S	MM	2PC	Paxos
Consistency					
Transactions					
Latency					
Throughput					
Data loss					
Failover					

Backups

- Make a copy
- Sledgehammer
- Weak consistency
- Usually no transactions
- Datastore: early internal launch

Backups

	Backups	M/S	MM	2PC	Paxos
Consistency	Weak				
Transactions	No				
Latency	Low				
Throughput	High				
Data loss	Lots				
Failover	Down				

Master/slave replication

- Usually asynchronous
  - Good for throughput, latency
- Most RDBMSes
  - e.g. MySQL binary logs
- Weak/eventual consistency
  - Granularity matters!
- Datastore: current

Master/slave replication

	Backups	M/S	MM	2PC	Paxos
Consistency	Weak	Eventual			
Transactions	No	Full			
Latency	Low				
Throughput	High				
Data loss	Lots	Some			
Failover	Down	Read only			

Multi-master replication

- Umbrella term for merging concurrent writes
- Asynchronous, eventual consistency
- Need *serialization* protocol
    - e.g. *timestamp oracle*: monotonically increasing timestamps
    - Either SPOF with master election…
    - …or distributed consensus protocol
- No global transactions!
- Datastore: no strong consistency

Multi-master replication

<table>
<tr><th></th><th>Backups</th><th>M/S</th><th>MM</th><th>2PC</th><th>Paxos</th></tr>
<tr><td>Consistency</td><td>Weak</td><td colspan="2">Eventual</td><td></td><td></td></tr>
<tr><td>Transactions</td><td>No</td><td>Full</td><td>Local</td><td></td><td></td></tr>
<tr><td>Latency</td><td colspan="3">Low</td><td></td><td></td></tr>
<tr><td>Throughput</td><td colspan="3">High</td><td></td><td></td></tr>
<tr><td>Data loss</td><td>Lots</td><td colspan="2">Some</td><td></td><td></td></tr>
<tr><td>Failover</td><td>Down</td><td>Read only</td><td>Read/write</td><td></td><td></td></tr>
</table>

Two Phase Commit

- Semi-distributed consensus protocol
    - deterministic coordinator
- 1: propose, 2: vote, 3: commit/abort
- Heavyweight, synchronous, high latency
- 3PC buys async with extra round trip
- Datastore: poor throughput

Two Phase Commit

<table>
<tr><th></th><th>Backups</th><th>M/S</th><th>MM</th><th>2PC</th><th>Paxos</th></tr>
<tr><td>Consistency</td><td>Weak</td><td colspan="2">Eventual</td><td>Strong</td><td></td></tr>
<tr><td>Transactions</td><td>No</td><td>Full</td><td>Local</td><td>Full</td><td></td></tr>
<tr><td>Latency</td><td colspan="3">Low</td><td>High</td><td></td></tr>
<tr><td>Throughput</td><td colspan="3">High</td><td>Low</td><td></td></tr>
<tr><td>Data loss</td><td>Lots</td><td colspan="2">Some</td><td>None</td><td></td></tr>
<tr><td>Failover</td><td>Down</td><td>Read only</td><td colspan="2">Read/write</td><td></td></tr>
</table>

Paxos

- Fully distributed consensus protocol
- "Either Paxos, or Paxos with cruft, or broken"
    - Mike Burrows
- Majority writes; survives minority failure
- Protocol similar to 2PC/3PC
    - Lighter, but still high latency

Paxos for the Datastore

- Close together? no.
- In the *same* datacenter? no.
- Opt-in? maybe...

Paxos for App Engine

- Coordinate moving state
- ...usually via lock server
- Apps
- Memcache
- Offline processing

Paxos

	Backups	M/S	MM	2PC	Paxos
Consistency	Weak	Eventual		Strong	
Transactions	No	Full	Local	Full	
Latency	Low			High	
Throughput	High			Low	Medium
Data loss	Lots	Some		None	
Failover	Down	Read only	Read/write		

Conclusions

- No silver bullet...
- ...but still worth doing!
- Embrace tradeoffs...
- ...and punt them to application developers!

What's behind (phew!)

- Consistency?
- Why transactions?
- Why across datacenters?
- Multihoming
- Techniques and tradeoffs
- Conclusions

Questions?