Stronger Consistency for Low-Latency, Geo-Replicated Storage

Michael J. Freedman
Princeton University
Joint with Wyatt Lloyd, David Andersen, Michael Kaminsky
Geo-Replicated Storage
is the backend of massive websites

“Halting is Undecidable”
Geo-Replicated Storage
serves requests quickly
Inside the Datacenter

Web Tier

No durable state
Independent

Storage Tier

Durable
Cooperative

A-F
G-L
M-R
S-Z
Storage Tier Dimensions

Shard Data Across Many Nodes

“Halting is Undecidable”
Storage Tier Dimensions

Shard Data Across Many Nodes

Data Geo-Replicated In Multiple Datacenters
Geo-Replicated Storage Goals

• Serve client requests quickly

• Scale out nodes/datacenter

• Interact with data coherently
Geo-Replicated Storage Goals

√ Serve client requests quickly

√ Scale out nodes/datacenter

• Interact with data coherently
  – Stronger consistency
  – Stronger semantics
ALPS Properties

- Availability
- Low Latency
  \[= O(\text{Local RTT})\]
- Partition Tolerance
- Scalability
Consistency

• Restricts order/timing of operations

• Stronger consistency:
  – Makes programming easier
  – Makes user experience better
Strong Consistency

- Linearizability [Herlihy Wing '90]
  - Total order of operations
  - Order agrees with “real time”

- Intuitively: West coast reads see east coast writes
Consistency with ALPS

<table>
<thead>
<tr>
<th>Consistency</th>
<th>Impossibility Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linearizability</td>
<td>Impossible [Brewer '00, Gilbert Lynch '02]</td>
</tr>
<tr>
<td>Serializability</td>
<td>Impossible [Lipton Sandberg '88, Attiya Welch '94]</td>
</tr>
<tr>
<td>Sequential</td>
<td>This Talk!</td>
</tr>
<tr>
<td>Causal</td>
<td>This Talk!</td>
</tr>
<tr>
<td>“Eventual”</td>
<td>Amazon Dynamo, Facebook/Apache Cassandra</td>
</tr>
</tbody>
</table>
Causality By Example

Remove boss from friends group

Post to friends:
“Time for a new job!”

Friend reads post

Causality (→)
Thread-of-Execution
Reads-From
Transitivty
Users Like Causality
Because sites work as expected

- **Friends**
  - Then
  - **New Job!**
  - Employment retained

- **Boss**
  - Then
  - **Add to Cart**
  - Purchase retained

- **Error**
  - Then
  - 404 – File not found
  - Deletion retained
Programmers Like Causality
Because it simplifies programming

No reasoning about out-of-order operations
Concurrent Writes: Conflicts in Causal
Conflicts in Causal

Causal + Conflict Handling = Causal+

K=3
Previous Causal Systems

- Bayou ‘94, TACT ‘00, PRACTI ‘06
  - Log-exchange based

- Log is single serialization point
  - ✓ Implicitly captures & enforces causal order
  - ✗ Loses cross-server causality

  OR

  Limits scalability
Consistency Challenges

• Strongest forms impossible with ALPS

• Eventual == no consistency

• Log exchange gives causal consistency, but not scalable

• Our work: First scalable causal+
Scalability Key Idea

• Capture causality with explicit dependency metadata

  3 after 1

• Enforce with distributed verifications
  – Delay exposing replicated writes until all dependencies satisfied in DC
Our Architecture

All Ops Local = “Always On”
Our Architecture

Client Library

A-F
G-L
M-R
S-Z

AF
GL
MR
SZ
AF
GL
MR
SZ
Read

Client Library

read

read
Write

write after = write + ordering metadata

Client Library

write

write_after

Replication
Replicated Write

Exposing values after dep_checks return ensures causal

write_after(..., deps)

Locator Key

Unique Timestamp

deps

A

195

L

337

dep_check(A_{195})

dep_check(L_{337})
Basic Architecture Summary

• All ops local, replicate in background
  – “Always On”

• Shard data across many nodes
  – Scalability

• Control replication with dependencies
  – Causal consistency
Challenge: Many Dependencies

- Dependencies grow with client lifetime
Nearest Dependencies

- Transitivity capture ordering constraints
Nearest Dependencies

• Transitivity capture ordering constraints
• Need extra server-side state to calculate
One-Hop Dependencies

• Small superset of nearest dependencies

• Simple to track:
  – Last write
  – Subsequent reads
Scalable Causal+
From fully distributed operation
Geo-Replicated Storage Goals

• ALPS
  – Serve client requests quickly
  – Scale out nodes/datacenter

• Interact with data coherently
  – Causal consistency  COPS [SOSP ’11]
  – Rich data model
  – Read-only transactions
  – Write-only transactions  Eiger [NSDI ‘13]
Column-Family Data Model
Widely-used hierarchical structure

<table>
<thead>
<tr>
<th>Profile</th>
<th>Friends</th>
<th>Count</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ada</td>
<td>Ada</td>
<td>1/1/54</td>
<td>631</td>
</tr>
<tr>
<td>Ada</td>
<td>Alan</td>
<td>1/1/54</td>
<td></td>
</tr>
<tr>
<td>Ada</td>
<td>Alonzo</td>
<td>9/1/36</td>
<td></td>
</tr>
<tr>
<td>Ada</td>
<td></td>
<td>6/6/38</td>
<td></td>
</tr>
<tr>
<td>Alan</td>
<td>Ada</td>
<td>1/1/54</td>
<td>457</td>
</tr>
<tr>
<td>Alan</td>
<td>Alan</td>
<td>9/1/36</td>
<td></td>
</tr>
<tr>
<td>Alan</td>
<td>Alonzo</td>
<td>323</td>
<td></td>
</tr>
<tr>
<td>Alan</td>
<td></td>
<td>1/1/37</td>
<td></td>
</tr>
<tr>
<td>Alonzo</td>
<td>Ada</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Alonzo</td>
<td>Alan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Alonzo</td>
<td>Alonzo</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Alonzo</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Age</th>
<th>Town</th>
<th>Friends</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>197</td>
<td>London</td>
<td>Ada</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Alan</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Alonzo</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>Princeton</td>
<td>Ada</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Alan</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Alonzo</td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>Princeton</td>
<td>Ada</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Alan</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Alonzo</td>
<td></td>
</tr>
</tbody>
</table>
## Column-Family Data Model

Widely-used hierarchical structure

<table>
<thead>
<tr>
<th>Profile</th>
<th>Friends</th>
<th>Count</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Age</td>
<td>Town</td>
<td>Ada</td>
<td>Alan</td>
</tr>
<tr>
<td>Ada</td>
<td>197</td>
<td>London</td>
<td>-</td>
</tr>
<tr>
<td>Alan</td>
<td>100</td>
<td>Princeton</td>
<td>1/1/54</td>
</tr>
<tr>
<td>Alonzo</td>
<td>110</td>
<td>Princeton</td>
<td>-</td>
</tr>
</tbody>
</table>
# Column-Family Data Model

Now with causal consistency

<table>
<thead>
<tr>
<th>Profile</th>
<th>Friends</th>
<th>Count</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Age</td>
<td>Town</td>
<td>Ada</td>
<td>Alan</td>
</tr>
<tr>
<td>Ada</td>
<td>197</td>
<td>-</td>
<td>1/1/54</td>
</tr>
<tr>
<td>Alan</td>
<td>100</td>
<td>Princeton</td>
<td>1/1/54</td>
</tr>
<tr>
<td>Alonzo</td>
<td>110</td>
<td>Princeton</td>
<td>-</td>
</tr>
</tbody>
</table>

Then
### Read-only transaction

Consistent view across many keys/servers

<table>
<thead>
<tr>
<th>Profile</th>
<th>Friends</th>
<th>Count</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Age</td>
<td>Town</td>
<td>Ada</td>
<td>Alan</td>
</tr>
<tr>
<td>Ada</td>
<td>London</td>
<td>-</td>
<td>1/1/54</td>
</tr>
<tr>
<td>Alan</td>
<td>Princeton</td>
<td>1/1/54</td>
<td>-</td>
</tr>
<tr>
<td>Alonzo</td>
<td>Princeton</td>
<td>-</td>
<td>9/1/36</td>
</tr>
</tbody>
</table>

Consistent view across many keys/servers includes:
- Read-only transaction
## Write-only transaction

Atomic update across many keys/servers

<table>
<thead>
<tr>
<th>Profile</th>
<th>Friends</th>
<th>Count</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ada</td>
<td>Ada</td>
<td>631</td>
<td>-</td>
</tr>
<tr>
<td>Alan</td>
<td>Alan</td>
<td>457</td>
<td>Halting</td>
</tr>
<tr>
<td>Alonzo</td>
<td>Alonzo</td>
<td>323</td>
<td>-</td>
</tr>
</tbody>
</table>

### Profile Details

<table>
<thead>
<tr>
<th>Age</th>
<th>Town</th>
<th>Friends</th>
<th>Count</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>197</td>
<td>London</td>
<td>-</td>
<td>1/1/54</td>
<td>7/15/14</td>
</tr>
<tr>
<td>100</td>
<td>Princeton</td>
<td>1/1/54</td>
<td>-</td>
<td>9/1/36</td>
</tr>
<tr>
<td>110</td>
<td>Princeton</td>
<td>7/15/14</td>
<td>9/1/36</td>
<td>-</td>
</tr>
</tbody>
</table>
Eiger Provides

√ ALPS properties
√ Rich data model
√ Causal consistency
  • Read-only transactions
  • Write-only transactions
Reads Aren’t Enough

Asynchronous requests + distributed data = ??
Read-Only Transactions

• Consistent up-to-date view of data across many servers
Read-Only Transactions

• Round 1: Optimistic parallel reads
  – Results include validity time metadata

• Calculate effective time
  – Ensures progress

• Round 2: Parallel read_at_times
  – Only needed for concurrently updated data
Eiger Provides

√ ALPS properties
√ Rich data model
√ Causal consistency
√ Read-only transactions
√ Write-only transactions

But what does all this cost?
Implementation

• COPS [SOSP '11]
  – Built on FAWN-KV (8.5K LOC)
  – 4.5K Lines of C++

• Eiger [NSDI '13]
  – Built on Cassandra (75K LOC)
  – 5K Lines of Java
Experimental Setup

Local Datacenter (Stanford)

Remote DC (UW)

Replication
Facebook Workload Results

TAO: Eventually-consistent, non-transactional, geo-replicated, production storage at Facebook

6.6% Overhead
Eiger Scales

Facebook Workload

Scales out

72% increases

96% increases

384 Machines!
Geo-Replicated Storage

• **ALPS:** Availability, Low latency, Partition tolerance, Scalability

• Causal+ consistency
  – Explicit dependencies, distributed checks
  – Exploit transitivity to reduce overhead

• Stronger semantics
  – Rich data model
  – Read-only transactions
  – Write-only transactions

• Competitive with eventually-consistent baseline
  – Scales to many nodes
http://sns.cs.princeton.edu/
https://github.com/wlloyd/eiger
Save the planet and return your name badge before you leave (on Tuesday)
Read-Only Transactions

• Consistent up-to-date view of data
  – Across many servers

• Challenges
  – Scalability: Decentralized algorithm
  – Guaranteed low latency
    • At most 2 parallel rounds of local reads
    • No locks, no blocking
  – High performance: Normal case - 1 round of reads
Eiger Provides

- ALPS properties
- Rich data model
- Causal consistency
- Read-only transactions
- Write-only transactions
Write-Only Transactions

• Update data atomically across servers
  – Atomic in each datacenter (not globally)
  – Use 2PC variant

• Challenges
  – Scalability
    • Decentralized algorithm
  – Low latency
    • 3 local RTTs
    • No locks or blocking
    • Read-only transactions not blocked, indirected
Evaluation

• Cost of stronger consistency & semantics
  – Vs. eventually-consistent Cassandra
  – Overhead for real (Facebook) workload
  – Overhead for state-space of workloads

• Scalability
Exploring Possible Workloads

- **Dynamic workload generator**
  - Explore all possible workload types

- **Vary workload parameters:**
  - Value size
  - Structure of data (4 variables)
  - Write fraction
  - Write transaction fraction
Dynamic Workload Results

Overhead
- Medium
- Low
- Minimal

Normalized Throughput

Value Size (B)
- 1
- 8
- 64
- 512
- 4096

Write Fraction
- 0
- 0.2
- 0.4
- 0.6
- 0.8
- 1

Structure of Data Variable 1 (Columns/Read)
- 1
- 2
- 4
- 8
- 16
- 32