Don’t Settle for Eventual: Scalable Causal Consistency for Wide-Area Storage with COPS

Wyatt Lloyd*
Michael J. Freedman*
Michael Kaminsky†
David G. Andersen‡

*Princeton, †Intel Labs, ‡CMU
Wide-Area Storage

Stores:
- Status Updates
- Likes
- Comments
- Photos
- Friends List

Stores:
- Tweets
- Favorites
- Following List

Stores:
- Posts
- +1s
- Comments
- Photos
- Circles
Wide-Area Storage
Serves Requests Quickly
Inside the Datacenter

Web Tier

Storage Tier

A-F

G-L

M-R

S-Z

Replication
Desired Properties: ALPS

- Availability
- Low Latency
- Partition Tolerance
- Scalability

“Always On”
Scalability
Increase capacity and throughput in each datacenter
Desired Property: Consistency

• Restricts order/timing of operations

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

Strong: Impossible [Brewer00, GilbertLynch02]
Sequential: Impossible [LiptonSandberg88, AttiyaWelch94]
Causal: COPS
Eventual: Amazon Dynamo, LinkedIn Voldemort, Facebook/Apache Cassandra
<table>
<thead>
<tr>
<th>System</th>
<th>A</th>
<th>L</th>
<th>P</th>
<th>S</th>
<th>Consistency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scatter</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>Strong</td>
</tr>
<tr>
<td>Walter</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>?</td>
<td>PSI + Txn</td>
</tr>
<tr>
<td>COPS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Causal+</td>
</tr>
<tr>
<td>Bayou</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>Causal+</td>
</tr>
<tr>
<td>PNUTS</td>
<td>✓</td>
<td>✓</td>
<td>?</td>
<td>✓</td>
<td>Per-Key Seq.</td>
</tr>
<tr>
<td>Dynamo</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Eventual</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
Gets-From
Transitivity

New Job!
Causality Is Useful

For Users:

Employment Integrity

For Programmers:

Referential Integrity

Friends

New Job!

Photo Upload

Add to album
Conflicts in Causal

K=1

K=2
Conflicts in Causal

Causal + Conflict Handling = Causal+
Previous Causal+ Systems

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

• Log is single serialization point
  – **Implicitly** captures and enforces causal order
  – Limits scalability OR
  – No cross-server causality
Scalability Key Idea

• Dependency metadata explicitly captures causality

• Distributed verifications replace single serialization
  – Delay exposing replicated puts until all dependencies are satisfied in the datacenter
Put

Client Library

Local Datacenter

Key-Value Store

put
after = ordering metadata

put + ordering metadata

put

put_after

put_after

K:V

Replication Q
Dependencies

• Dependencies are explicit metadata on values
• Library tracks and attaches them to put_afters
Dependencies

- Dependencies are explicit metadata on values
- Library tracks and attaches them to `put_afters`

Client 1

```
put(Key, Val)

```

```
put_after(Key, Val, deps)

```

```
version

```

```
(Thread-Of-Execution Rule)

```

```
K^version

```

```

deps

```
Dependencies

- Dependencies are explicit metadata on values
- Library tracks and attaches them to `put_afters`

Client 2

```
get(K) → value

value, version, deps' ← get(K)
```

```
deps
  . . .
  K

K

version

L

337

M

195

(Gets-From Rule)

(L337)

(M195)

(Transitivity Rule)
```
Causal+ Replication

Key-Value Store

put_after(K,Vdeps)

K:Vdeps

Replication Q

put after
Causal+ Replication

```python
put_after(K, V, deps)
```

```python
dep_check(L337)
```

```python
dep_check(M195)
```

Exposing values after dep_checks return ensures causal+
Basic COPS Summary

• Serve operations locally, replicate in background
  – “Always On”

• Partition keyspace onto many nodes
  – Scalability

• Control replication with dependencies
  – Causal+ Consistency
Remote Datacenter

My Operations

Remote Progress

Remote Progress

New Job!

Remote Progress

Portugal!

Remote Progress

New Job!

Boss

You’re Fired!!

New Job!

You’re Fired!!

New Job!

New Job!
Gets Aren’t Enough

My Operations

- Boss

- New Job!

- Portugal!

- Boss

Remote Datacenter

- Boss

- Portugal!

- New Job!

- Boss

- You’re Fired!!

Remote Progress
Get Transactions

• Provide consistent view of multiple keys
  – Snapshot of visible values

• Keys can be spread across many servers

• Takes at most 2 parallel rounds of gets

• No locks, no blocking

Low Latency
Get Transactions

My Operations

Remote Datacenter

Could Get

New Job!

Portugal!

Boss

Never

New Job!

Portugal!
System So Far

• ALPS and Causal+, but ...

• Proliferation of dependencies reduces efficiency
  – Results in lots of metadata
  – Requires lots of verification

• We need to reduce metadata and dep_checks
  – Nearest dependencies
  – Dependency garbage collection
Many Dependencies

- Dependencies grow with client lifetime
Nearest Dependencies

• Transitively capture all ordering constraints
The Nearest Are Few

• Transitivity capture all ordering constraints
The Nearest Are Few

• Only check nearest when replicating

• COPS only tracks nearest

• COPS-GT tracks non-nearest for transactions

• Dependency garbage collection tames metadata in COPS-GT
Extended COPS Summary

• Get transactions
  – Provide consistent view of multiple keys

• Nearest Dependencies
  – Reduce number of dep_checks
  – Reduce metadata in COPS
Evaluation Questions

• Overhead of get transactions?

• Compare to previous causal+ systems?

• Scale?
Experimental Setup

Local Datacenter

Clients COPS Servers

Replication

Remote DC
COPS & COPS-GT
Competitive for Expected Workloads

All Put Workload – 4 Servers / Datacenter

High per-client write rates result in 1000s of dependencies

Low per-client write rates expected

Max Throughput (Kops/sec)

Average Inter-Op Delay (ms)

People tweeting 1000 times/sec

People tweeting 1 time/sec
COPS & COPS-GT
Competitive for Expected Workloads

Varied Workloads – 4 Servers / Datacenter

Max Throughput (Kops/sec)

Pathological

Expected

Workload
COPS Low Overhead vs. LOG

- COPS – dependencies ≈ LOG
- 1 server per datacenter only

- COPS and LOG achieve very similar throughput
  - Nearest dependencies mean very little metadata
  - In this case dep_checks are function calls
COPS Scales Out

Throughput (Kops)

<table>
<thead>
<tr>
<th>LOG</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>COPS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COPS-GT</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>16</td>
</tr>
</tbody>
</table>
Conclusion

• Novel Properties
  – First ALPS and causal+ consistent system in COPS
  – Lock free, low latency get transactions in COPS-GT

• Novel techniques
  – Explicit dependency tracking and verification with decentralized replication
  – Optimizations to reduce metadata and checks

• COPS achieves high throughput and scales out