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
The Key-value Abstraction

• (Business) Key $\rightarrow$ Value
• (twitter.com) tweet id $\rightarrow$ information about tweet
• (amazon.com) item number $\rightarrow$ information about it
• (kayak.com) Flight number $\rightarrow$ information about flight, e.g., availability
• (yourbank.com) Account number $\rightarrow$ information about it
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

Replication

Remote DC
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>✔</td>
</tr>
<tr>
<td>Walter</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>?</td>
<td>✔</td>
</tr>
<tr>
<td><strong>COPS</strong></td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Bayou</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
<td>✔</td>
</tr>
<tr>
<td>PNUTS</td>
<td>✔</td>
<td>✔</td>
<td>?</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Dynamo</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
</tr>
</tbody>
</table>

- **A** = Atomic
- **L** = Local
- **P** = Persistent
- **S** = Strong
- **Causal+** = Causal+
- **Per-Key Seq.** = Per-Key Seq.
- **PSI + Txn** = PSI + Txn
- **Strong** = Strong
- **Eventual** = Eventual
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
Causality Is Useful

For Users:

**Friends**

**NEW JOB!**

Employment Integrity

For Programmers:

**Photo Upload**

**Add to album**

Referential Integrity
Conflicts in Causal
Conflicts in Causal

Causal + Conflict Handling = \textit{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
Client Library

```
put = put after = ordering metadata
```

Put

```
K:V
```

Local Datacenter

```
put
put after = ordering metadata
```

Replication Q

```
put
```
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
Dependencies

- Dependencies are explicit metadata on values
- Library tracks and attaches them to put_afters
Causal+ Replication

Key-Value Store

put_after(K, V, deps)
Causal+ Replication

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
Gets Aren’t Enough

My Operations

Remote Datacenter

New Job!

Boss

You’re Fired!!
Remote Datacenter

Boss

You’re Fired!!

My Operations

Boss

New Job!

Portugal!

Boss

Remote Progress

Remote Progress

Remote Progress

Remote Progress

Remote Progress

Remote Progress

New Job!

Boss
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
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

• Transitivity capture all ordering constraints
The Nearest Are Few

- Transitively 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)

COPS
COPS-GT

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)

LOG 1 2 4 COPS 8 16 1 2 4 8 16

COPS-GT
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