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

Functional Replicas

Partitioned

Storage Tier

Replication

Remote DC

Scalability:

A-H

I-P

Q-Z
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
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

\[\rightarrow\]

New Job!

Employment Integrity

For Programmers:

Photo Upload

Add to album

Referential Integrity
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

put after = ordering metadata

Local Datacenter

Client Library

Key-Value Store

Put

put after + ordering metadata

Replication Q

put after
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

```
put(Key, Val)
put_after(Key, Val, deps)
```

Client 1

```
Thread-Of-Execution Rule
```

```
version
```

```
K
```
Dependencies

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

```
Client 2

get(K) → value → [deps]

[deps]
K
L337
M195

(value, version, deps') → get(K)

Get-From Rule
Transitivity Rule
```
Causal+ Replication

put_after(K,V,deps)

Key-Value Store
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 Operations
- Remote Progress
- New Job!

Remote Datacenter
- Boss
- Pittsburgh!
- Remote Progress
- New Job!

You’re Fired!!
Remote Datacenter

Boss

Pittsburgh!

New Job!

You're Fired!!

My Operations

Remote Progress

Boss

Remote Progress

Boss

Remote Progress

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

• Transitively 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?
  – High for pathological workloads
  – Minimal for expected workloads

• Compare to previous causal+ systems?
  – COPS $\approx$ LOG with 1 node/dc

• Scale?
COPS Scales Out

Expected Workload

Throughput (Kops)

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

COPS-GT
Future Work

• Richer Data Model
  – COPS on Cassandra

• Put Transactions

• COPS Under Fire
  – Indefinite partitions == partial replication
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