Don’t Settle for Eventual: COPS

W. Lloyd\textsuperscript{1}, M. J. Freedman\textsuperscript{1}, M. Kaminsky\textsuperscript{2}, D. G. Andersen\textsuperscript{3}
\textsuperscript{1}Princeton University, \textsuperscript{2}Intel Labs, \textsuperscript{3}Carnegie Mellon University
MOTIVATION

- CAP Theorem
  - Consistency
  - Availability
  - Partition Tolerance
CAP Theorem

- Consistency
  - Every read receives the most recent write or an error
- Availability
  - Every request receives a (non-error) response – without guarantee that it contains the most recent write
- Partition Tolerance
  - The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes
MOTIVATION

- CAP Theorem
  - Consistency
  - Availability
  - Partition Tolerance
- Weak Consistency Model
  - Eventual Consistency
  - Causal Consistency
  - Causal with Convergent Conflict Handling (Causal+)
**Eventual Consistency**

- If no new updates are made to a given data item, eventually all accesses to that item will return the last updated value.
- No consistency.

\[
\begin{align*}
W(A,1) & \quad R(A)=3 \quad W(A,4) \quad R(A)=2 \\
W(A,6) & \quad R(A)=-1 \quad R(A)=32,768 \\
R(A) & \quad R(A)=-2,147,483,648 \ldots \\
\ldots & \quad (100\text{yrs later}) \\
R(A) & \quad R(A)=6 \\
R(A) & \quad R(A)=7
\end{align*}
\]
Causal Consistency

- Each process can observe those causally related operations in common causal order

Remove advisor from friend list.

Add spring break photos.
Causal Consistency

• Each process can observe those causally related operations in common causal order
• Divergence.

Boyu: I want to append a line to this document.

Jimmy: Boyu wants to append something to this document.

Boyu: I want to append a line to this document.
Causal+ Consistency

- Causally consistent.
- Given a set of conflict operations, all processes handle them in the same way.

Boyu: I want to append a line to this document.

Jimmy: Boyu wants to append something to this document.
Motivation

- CAP Theorem
  - Consistency
  - Availability
  - Partition Tolerance
- Weak Consistency Model
  - Eventually Consistent
  - Causally Consistent
  - Causal with Convergent Conflict Handling (Causal+)
- Existing Causal+ Systems
  - Not Scalable
EXISTING CAUSAL+ SYSTEMS

- Log-exchange.
- Operations written in a log, marked with a version vector.
- Exchange logs, establish causality using version vectors at a single node.
COPS Overview

- Key-value store.
- Clients query local clusters.
- Linearizability within a cluster.
- COPS among clusters.
COPS

- put(key, val)
  - Write to local cluster.
  - Replication between clusters.
    - Version Number.
    - Guarantee causality.
    - Conflict resolution.
Lamport Clocks!
GUARANTEE CAUSALITY

- get(key_1)=val_1 (ver_1)
  get(key_2)=val_2 (ver_2)
  ......
  get(key_N)=val_N (ver_N)
  put(key_x, val_x)

- Send dependencies together with value
- Update value only after dependencies are met

- Basically, a vector clock!
Conflict Resolution

• Last writer wins. (Bring order to the universe.)

• Lamport Clocks!

• <LC(val), cluster_id>
  ○ Total order.
  ○ Guarantees convergent conflict handling.

• Key contribution.
COPS

- put(key, val)
  - Write to local cluster.
  - Replication between clusters.
    - Version Number.
    - Guarantee causality.
    - Conflicts resolution.
- get(key) = val
  - Retrieve the most recent version.
Is COPS ENOUGH?

- A public reference to a set of photos.
  - Check priority.
- Set reference as private.
- Add some photos.
  - Get the photos.
Is COPS enough?

- A private reference to a set of photos.
  
  Get the photos.

- Delete some photos.

- Set reference as public.
  
  Check priority.
COPS-GT

- Supporting Get-Transactions!
- Maintain values of all versions.
- Track down dependencies.
PUT IN COPS-GT

- <key, val, version, dep*>
**Get-trans in COPS-GT**

- get-trans(key_1, key_2, ..., key_N)

- For each i=1, 2, ..., N,
  - get(key_i)=(val_i, version_i, dep*_i)

- If key_i in dep*_j
  - If version_i < dep*_j.key_i.version
    - get(key_i, dep*_j.key_i.version)
Garbage Collection

- Version Garbage (COPS-GT)
  - <key, v1> Old version.
    <key, v2> New trx access new versions.
  - <key, v1> can be cleaned when
    - trx existing when v2 comes in disappear.
  - Trx timeout (5 sec).
Dependency Garbage (COPS-GT)

- `<key2, v2>` depends on `<key1, v1>`
  - If `<key2, v2>` is updated in all clusters
    - `<key1, v1>` is updated in all clusters
    - Newtrx will access v1 or a more recent version.
  - Trx timeout.
Garbage Collection

- Client-side Garbage
  - Keep track of all operations of a client instance.
  - Each operation creates a dependency.
    - get(key_1) = val_1
    - get(key_2) = val_2
    - put(key_3, val_3)
    - put(key_4, val_4)
Garbage Collection

- Client-side Garbage
  - `<key, val>` is updated in all clusters:
    - Remove dependencies.
    - Notify clients.

- Global checkpoint
  - Most recent LC achieved in all clusters
  - Remove older dependencies.
  - Notify clients.
Fault Tolerance

- Client failure.
  - Does not violate consistency.

- Node failure within a cluster.
  - Masked by the replication protocol.

- Cluster failure (or partition).
  - Updates within the cluster. (lost/delayed)
  - Garbage collection fails.
    - Wait for the cluster to come back
    - Reconfigure COPS
EVALUATION

<table>
<thead>
<tr>
<th>System</th>
<th>Operation</th>
<th>Latency (ms)</th>
<th>Throughput (Kops/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>50%</td>
<td>99%</td>
</tr>
<tr>
<td>Thrift</td>
<td>ping</td>
<td>0.26</td>
<td>3.62</td>
</tr>
<tr>
<td>COPS</td>
<td>get_by_version</td>
<td>0.37</td>
<td>3.08</td>
</tr>
<tr>
<td>COPS-GT</td>
<td>get_by_version</td>
<td>0.38</td>
<td>3.14</td>
</tr>
<tr>
<td>COPS</td>
<td>put_after (1)</td>
<td>0.57</td>
<td>6.91</td>
</tr>
<tr>
<td>COPS-GT</td>
<td>put_after (1)</td>
<td>0.91</td>
<td>5.37</td>
</tr>
<tr>
<td>COPS-GT</td>
<td>put_after (130)</td>
<td>1.03</td>
<td>7.45</td>
</tr>
</tbody>
</table>

Table 2: Latency (in ms) and throughput (in Kops/s) of various operations for 1B objects in saturated systems. `put_after(x)` includes metadata for `x` dependencies.
Figure 8: In our experiments, clients choose keys to access by first selecting a keygroup according to some normal distribution, then randomly selecting a key within that group according to a uniform distribution. Figure shows such a stepped normal distribution for differing variances for client #3 (of 5).
EVALUATION
EVALUATION

The diagram illustrates the max throughput (in Kops/sec) as a function of variance for different configurations. The configurations include:

- COPS 1:4
- COPS-GT 1:4
- COPS 1:1
- COPS-GT 1:1

The throughput decreases as the variance increases for all configurations, with COPS-GT 1:4 showing the highest throughput compared to others.
EVALUATION
EVALUATION
**Evaluation**

- How scalable LOG is?
- Only 2 datacenters?
- How about eventually consistent systems?