Write Fast, Read in the Past: Causal Consistency for Client-side Apps with SwiftCloud

Presented by Marek Zawirski
Inria / UPMC-LIP6, Paris
(now at Google, Zürich)

Marek Zawirski, Nuno Preguiça, Sérgio Duarte, Annette Bieniusa, Valter Balegas, Marc Shapiro

Challenge: Database Access for Client-side Apps

Limited boundaries of server-side database guarantees

⇒ ad-hoc on the client-side
**Challenge: Database Access for Client-side Apps**

Extended boundaries with **SwiftCloud**

**Stronger than Eventual: Causal Consistency**

Default on client-side: eventual consistency $\Rightarrow$ anomalies

- bob_posts.add("don’t think of visiting Vancouver…")
- bob_posts.add("… just do it! YOLO")
- replies.add("Alice: totally 😊")

**Stronger than Eventual: Causal Consistency**

Default on client-side: eventual consistency $\Rightarrow$ anomalies

- bob_posts.add("don’t think of visiting Vancouver…")
- bob_posts.add("… just do it! YOLO")
- replies.add("Alice: totally 😊")

- read

**Consistent, available and convergent data access**

- **Scalability** with #objects and #clients

- **Fault-tolerance**

-Zawirski et al., Write Fast, Read in the Past: Causal Consistency for Client-side Applications with SwiftCloud
Stronger than Eventual: Causal Consistency

Default on client-side: eventual consistency ⇒ anomalies

Convergent Causal Consistency: No Lost Updates

Causal consistency: reads from causally-closed snapshot

High-level convergent objects\(\text{\textcopyright}C\text{RDTs}\) resolve concurrency
Zawirski et al., Write Fast, Read in the Past: Causal Consistency for Client-side Applications with SwiftCloud

Challenge: Causal Consistency with Partial Replicas

[PRACTI, NSDI’06]
Challenge: Causal Consistency with Partial Replicas

Zawirski et al., Write Fast, Read in the Past: Causal Consistency for Client-side Applications with SwiftCloud

Approach: Cloud-backed Partial Replicas

Data Center full replicas:
- ✓ Provide consistent view
- ✓ Assign small metadata

Inherent trade-offs in the general case:
scalability vs. availability vs. fault-tolerance
Approach: Cloud-backed Partial Replicas

Data Center full replicas:
✓ Provide consistent view  ✓ Assign small metadata

Client reads: cached fragment of cloud version \( \cup \) own log
✓ High availability  ✓ Consistency w/ read your writes
Potential of Cloud-backed Client Replicas

Setup: DCs in 3 AWS EC2 regions, YCSB workload, cache=256 objects

Challenges for the Cloud Approach: Safe DC Failover

Objects in the cache ⇒ immediate, consistent response

Setup: DCs in 3 AWS EC2 regions, YCSB workload, cache=256 objects

Challenges for the Cloud Approach: Safe DC Failover

Objects in the cache ⇒ immediate, consistent response
Challenge for the Cloud Approach: Safe DC Failover

operations with risky dependencies

risky read

Supporting Failover by Conservative Reads

Foreign updates: read version replicated in $K > 1$ DCs
Own writes: read from the log, recover to a new DC

Foreign updates: read version replicated in $K > 1$ DCs
Own writes: read from the log, recover to a new DC
Supporting Failover by Conservative Reads

Foreign updates: read version replicated in $K > 1$ DCs
Own writes: read from the log, recover to a new DC

Experiment: Injection of Short DC Disconnection

Experiment: Injection of Short DC Disconnection
Experiment: Injection of Short DC Disconnection

Trade-off controlled by $K$: **staleness vs. availability**
- Staleness negligible in most $K=2$ setups, < 1% reads
- In cherry-picked unfavorable setup, 1.0–2.5% reads

Challenge for the Cloud Approach: Protocol Retries

Zawinski et al., Write Fast, Read in the Past: Causal Consistency for Client-side Applications with SwiftCloud

Zawinski et al., Write Fast, Read in the Past: Causal Consistency for Client-side Applications with SwiftCloud
Challenge for the Cloud Approach: Protocol Retries

Zawirski et al., Write Fast, Read in the Past: Causal Consistency for Client-side Applications with SwiftCloud

Safe Retries with Decoupled Metadata

Solution: client-assigned timestamps for safety + 1..N DC timestamps for efficient summary
Safe Retries with Decoupled Metadata

\[
x = 0, \quad x = 1
\]

Solution: client-assigned timestamps for safety
+ 1..N DC timestamps for efficient summary

Extension: log pruning independent of client availability

---

Summary

SwiftCloud provides client-side apps:

- **Consistent, available** and convergent object database
- **Scalability**: full replicas at DC back partial at client
  \[\Rightarrow\] small causality metadata (< 15B/update)
- **Fast failover** thanks to conservative reads (< 1% stale)
- **Safe retry** of interrupted transfer and **safe log pruning**
  thanks to decoupled metadata

Research prototype at: [github.com/SyncFree/SwiftCloud](https://github.com/SyncFree/SwiftCloud)

---

Experiment: Size of Metadata on Client-DC Link

- Client-assigned vectors\(^{[\text{PRACTI, NSDI'06}]}\): unbounded overhead
- SwiftCloud’s decoupled metadata: const size

Setup: 3DCs, YCSB B uniform workload

SwiftCloud compared to “Lazy Replication”

- Assume client-side application logic
- Describe causal consistency support
- Support communication with multiple servers
- Use decoupled metadata

<table>
<thead>
<tr>
<th>SwiftCloud</th>
<th>Monolithic DB</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB = RDT objects + global transactions</td>
<td>No client-side replicas</td>
</tr>
<tr>
<td>Supports partial client replicas (\Rightarrow) fast reads and read-your-writes</td>
<td>Stability discussion</td>
</tr>
<tr>
<td>K-stability-driven trade-off</td>
<td>Physical-clock-driven GC</td>
</tr>
<tr>
<td>GC independent of clients</td>
<td>More consistency choices</td>
</tr>
</tbody>
</table>
Challenge for the Cloud Approach: Protocol Retries

Safe Retries with Decoupled Metadata

Solution: client-assigned timestamps for safety + 1..N DC timestamps for efficient summary

Extension: log pruning independent of client availability