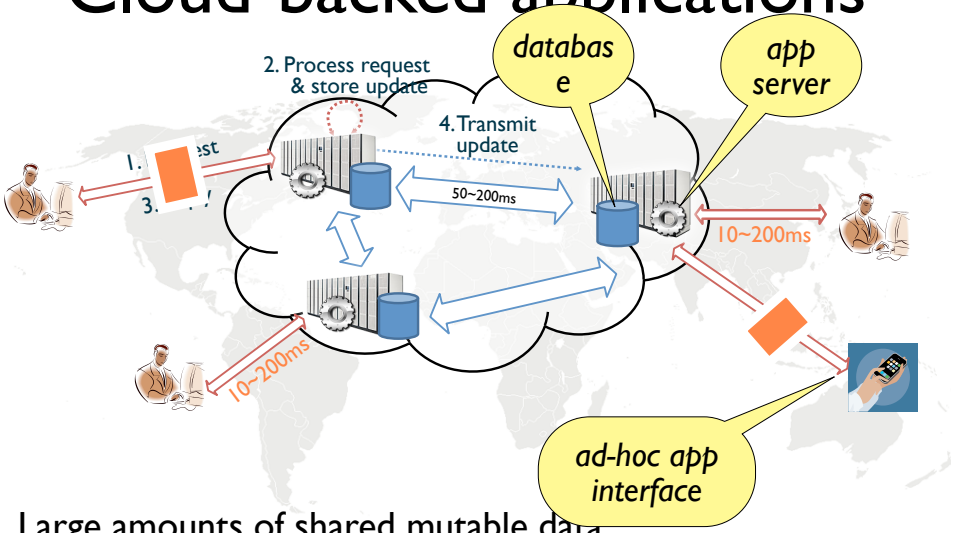
Write Fast, Read in the Past: Causal Consistency for Client-Side Applications

Marc Shapiro
Joint work with:
Marek Zawirski, Inria & LIP6
Annette Bieniusa, U. Kaiserslautern
Valter Balegas, U. Nova Lisboa
Sérgio Duarte, U. Nova Lisboa
Carlos Baquero, U. Minho
Nuno Preguiça, U. Nova Lisboa

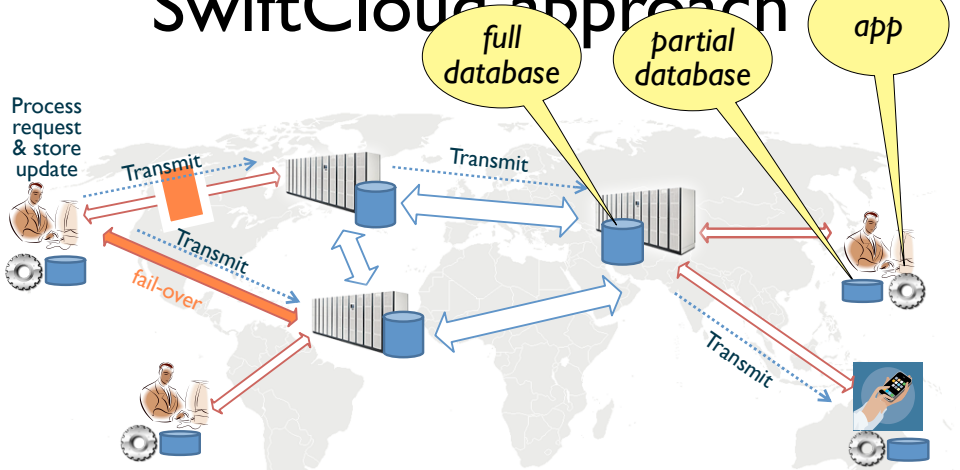
Cloud-backed applications



Large amounts of shared mutable data

Geo-replication: high availability, low latency, fault tolerance

SwiftCloud approach



App + database at/near client
Update shared store locally
Availability + consistency

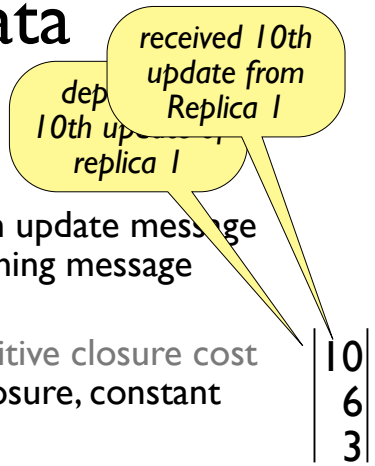
Requirements & challenges

Scalability with #items, #clients
- Partial replication of data
- … and of metadata
- Small, bounded metadata

Availability ⇒ causal consistency
- Eventual exactly-once delivery
- Causal order
- … despite partial replication
- … despite transient failures
- … despite permanent failures

Convergence

Metadata



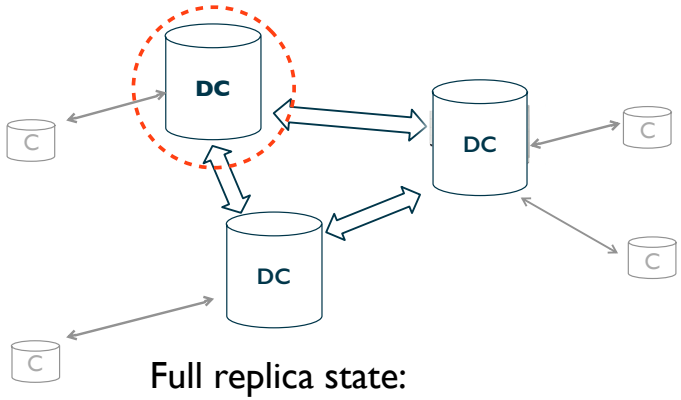
Tracking causality
- Dependence: piggy-backed on update message
- Delivery: compare with incoming message

Representations:
- Graph: worst-case cost, transitive closure cost
- Vector: compact transitive closure, constant cost + optimisations

Issue: size(vector) = O (#masters)

Log: pruning not contingent on clients

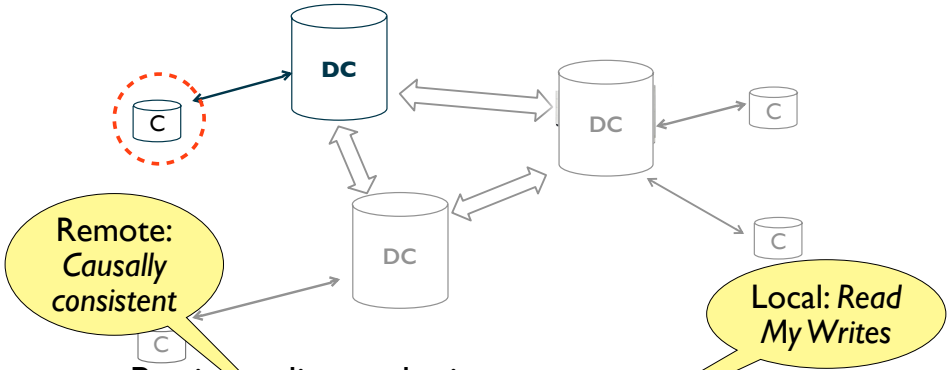
Causality + scalability: DC



Full replica state:
- Causally consistent
- Transitively closed

Vector clock
- size = O(#DCs)

Causality + scalability: client



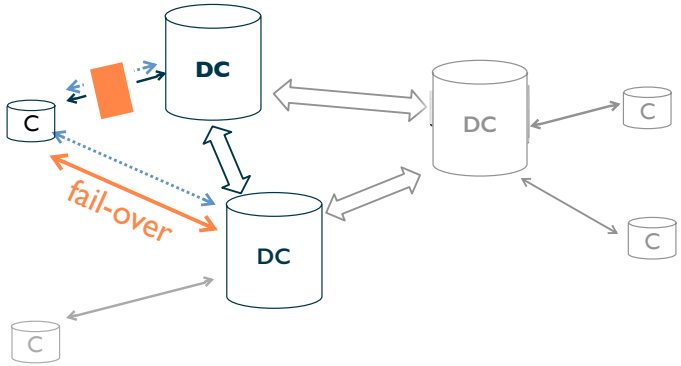
Partial replica: cache interest set

Topology: Communicate with DC only

State = DC state $|_{\text{interest set}}$ ∪ client updates

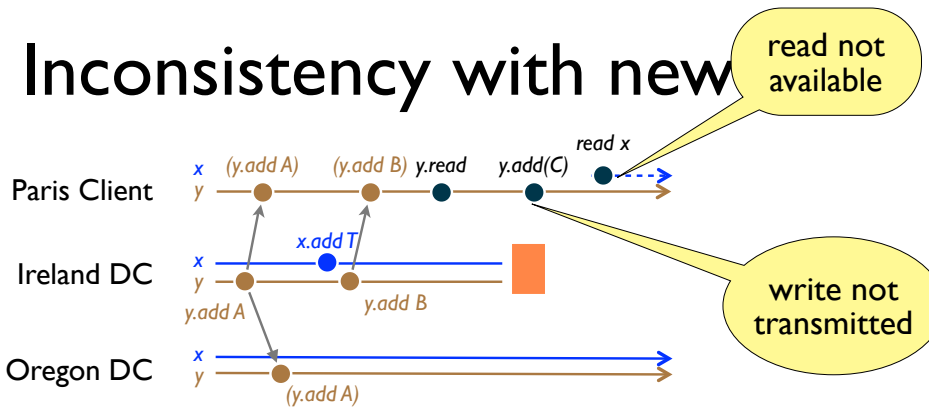
Concurrency: size(vector) = O(#DCs)!

Causal consistency & fail-over



Asynchronous propagation
- At least once: stubborn
- At most once: filter out duplicates: how?
- Consistency with new DC?

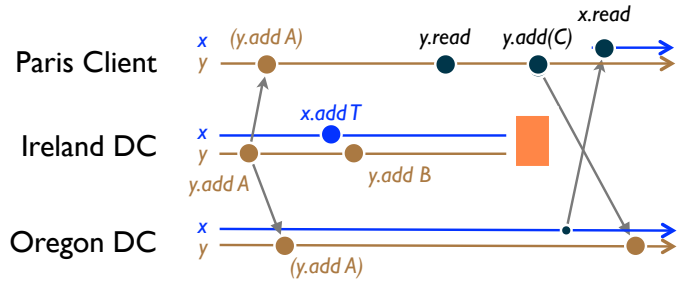
Inconsistency with new



Causal gap at client: *y.add(C)* depends on *x.add(T)*

Oregon cannot satisfy: fail-over impossible; client blocks until Ireland recovers

Solution: reading in the past



Client served only updates that are *K-stable* ⇒ no gaps

- *K=1*: always fresh, fail-over problematic
- *K=N*: fail-over guaranteed, staleness
  - updates possibly buffered indefinitely

Default: K=2, covers common failures

- Also improves throughput

Convergence by construction

Merging concurrent updates:

- Deterministic
- Dependent only on delivery poset
- Not on delivery order, local info

Sufficient conditions:

- Delivered updates commute
- (or) Monotonic semi-lattice

CRDTs

Conflict-free Replicated Data Types (CRDTs)

Converge concurrent updates
Encapsulate replication & resolution
Re-usable data types
Correct by construction

Register
- Last-Writer Wins
- Multi-Value

Set
- Grow-Only
- 2P
- Observed-Remove

Map

Counter
- Unlimited
- Restricted non-negative

Graph
- Directed
- Monotonic DAG
- Edit graph

Sequence

Highly-Available Transactions

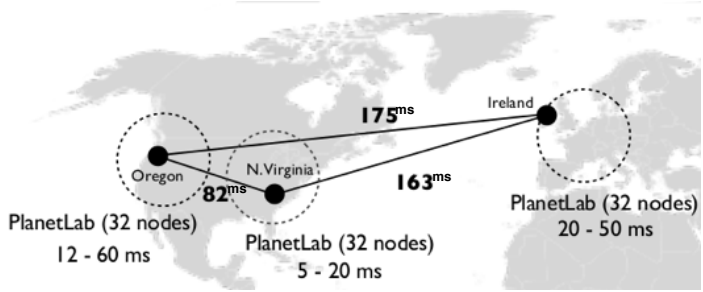
Unit of reading / writing
- Read committed
- Read atomic

Update is visible only when
- preceding visible
- all in transaction visible

Single ID, increment
- read in the past

No synchronisation necessary

Experiments

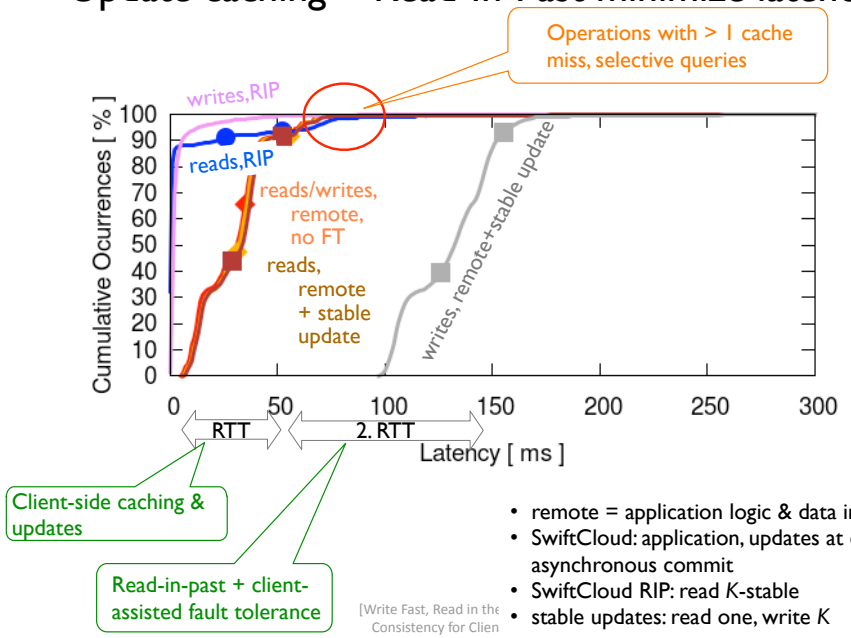


Social networking application
- 90% cache hits

Evaluate
- Availability, responsiveness
- Size of metadata
- Latency, throughput

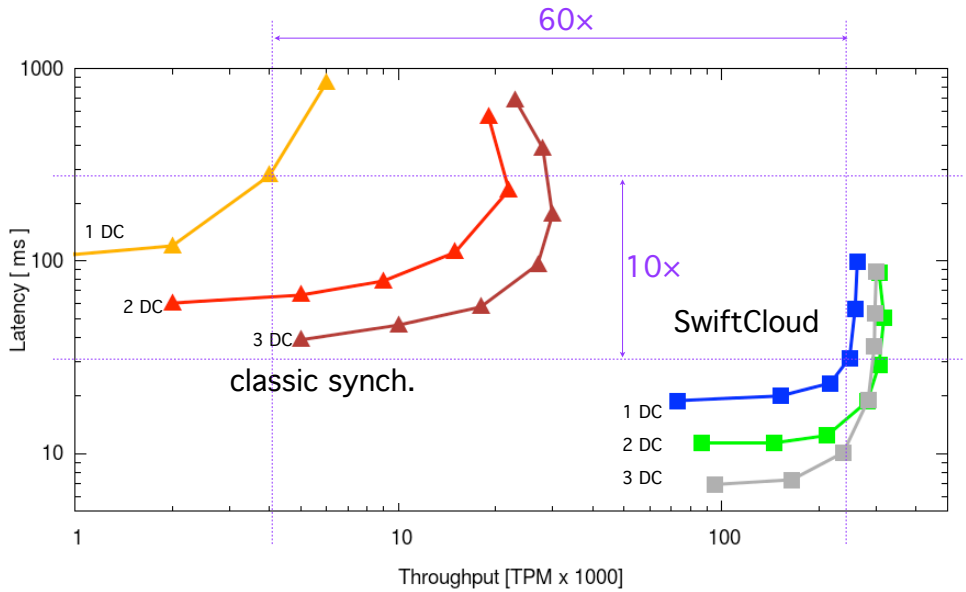
3 DCs in Amazon EC2
1 DC = 1 EC2 medium instance
100 client nodes in PlanetLab
Multiple sessions per node
Cache size: 512 objects

Update caching + Read-In-Past minimize latency

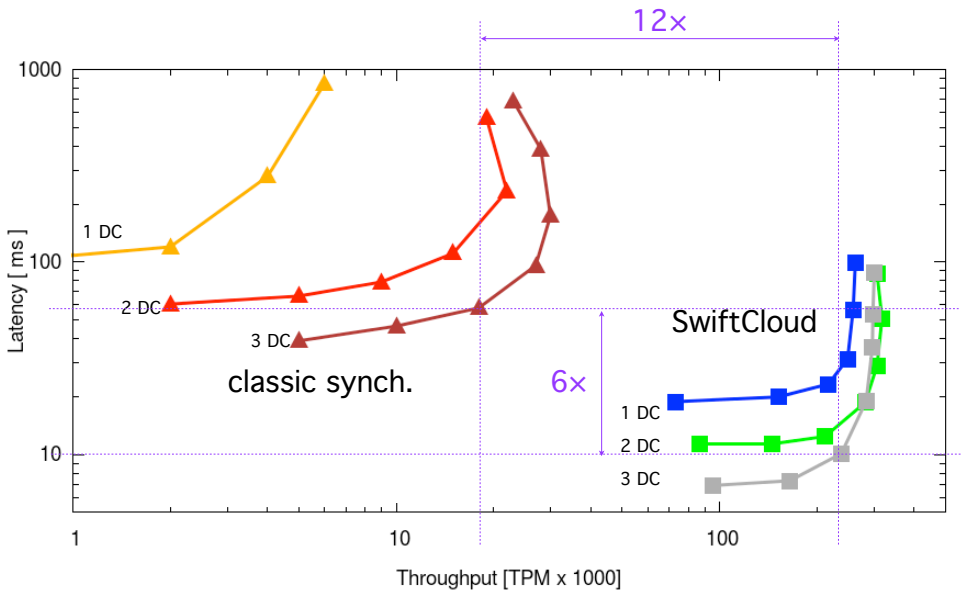


- remote = application logic & data in DC
- SwiftCloud: application, updates at client, asynchronous commit
- SwiftCloud RIP: read *K*-stable
- stable updates: read one, write *K*

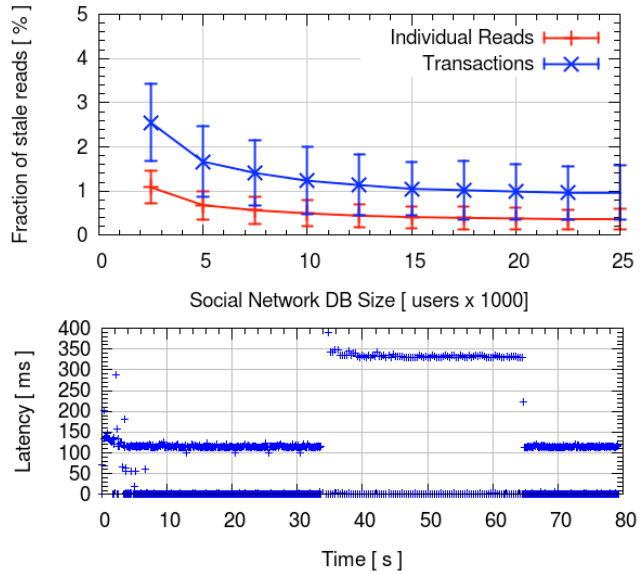
Latency vs. throughput



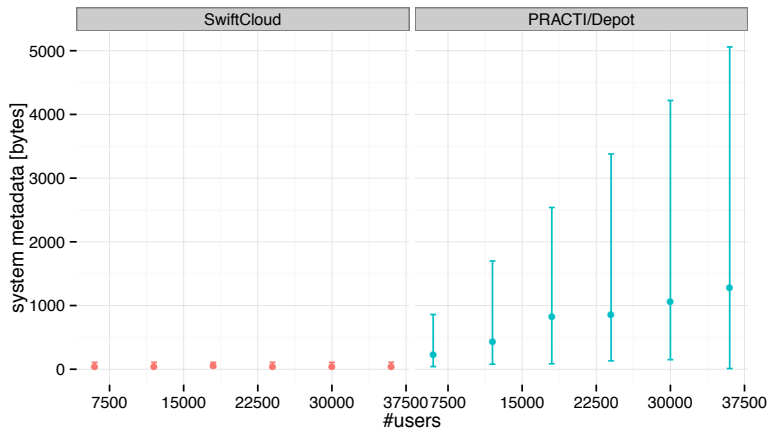
Latency vs. throughput



Staleness for fault tolerance



Metadata overhead



Summary

Fast response, high availability
- Shared store replicated at client
- Full consistency guarantees

Availability: Causal consistency is strongest possible
(+ transactions + red-blue)

Challenges & solutions:
- Large database: partial replicate data, metadata
- Small, bounded metadata: DC-based, pruning
- Fail-over
  - At-most-once: Client ID + merge clocks
  - Avoid gaps: read in the past (*K*-stable)
- Convergence: CRDTs

Current & future work

SyncFree EU project
- Inria, Nova, Minho, Kaiserslautern, UC Louvain, Koç
- Basho, Trifork, Rovio

Objectives
- CRDTs in theory + real apps
- Extreme scalability (DC and clients)
- Transactions, invariants, security
- Languages, patterns, tools
- Crowd-sourced experiments

Internet round-trip times

		RTTS (ms)		
Endpoints	Speed of light	min	mean	max
Berkeley – Canberra	79.9	174.0	174.7	176.0
Berkeley – New York	27.4	85.0	85.0	85.0
Berkeley – Trondheim	55.6	197.0	197.0	197.0
Pittsburgh – Ottawa	4.3	44.0	44.1	62.0
Pittsburgh – Hong-Kong	85.9	217.0	223.1	393.0
Pittsburgh – Dublin	42.0	115.0	115.7	116.0
Pittsburgh – Seattle	22.9	83.0	83.9	84.0

Red/Blue consistency

Mix synchronous (red) and asynchronous (blue) updates

Correctness
- All reads and updates are causally ordered
- Blue updates commute with both red and blue ones
- Red updates are totally ordered with red ones (but not with blue ones)

[Li et al. OSDI 2012]

Eventual Consistency

- Every update eventually reaches every replica at least once
- An update has effect on a replica at most once
- At all times, the state of a replica satisfies the objects' specifications
- Two replicas that received the same set of updates eventually reach the same state

Transactional: replace "update" with "transaction comprising one or more updates".

CRDTs in the wild

Walter [SOSP 2011]: c-set

Riak 2.0: counter, set, map

Facebook Apollo (announced 2014-06)

StateBox (Erlang), KnockBox (Clojure), meangirls (Ruby), riak-java-crdt (Java)

SwiftCloud: geo-replication right to the client machine

Extreme numbers of clients

Large database

Full replication?

Causal consistency for servers and clients

Fault tolerance

Switching servers?

Metadata proportional to # clients?

SwiftCloud: geo-replication right to the client machine

Low latency

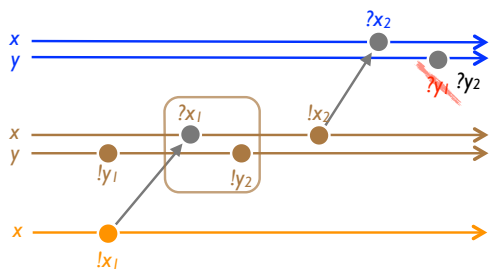
… for **reads**

- Partial replication/caching in clients

… for **writes**

- Weak consistency
- Mergeable writes => CRDT
- + Transactions with parallel snapshot isolation (PSI transactions)

Causal transactions



Read a causally-consistent state
All-or-nothing, isolated updates
No consensus (not ACID)