Consistency in large-scale systems

Marc Shapiro, INRIA & LIP6

1. Introduction

Sometime in near future…

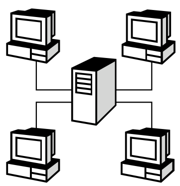
Tomorrow:
- Petascale + distributed convergence
- Fat pipes ↗, core connectivity ↗
- Jitter ↗, latency →

Asynchronous, failure-prone
- FLP impossibility
- Byzantine behaviours

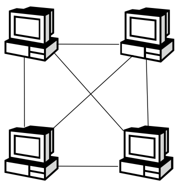
See results from *distributed systems*

Consistent access to mutable information

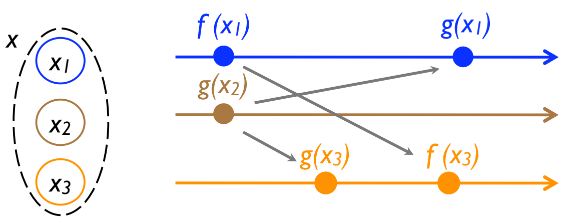
Server-resident repository
- Single, consistent copy
- Remote access
- Centralized management



<u>Replicated data</u>
- Copy in each machine
- Local reads
- *Multi-master* writes
- Consistency: FLP
- High-level: In software



Active replication



Objects $x, y, \ldots$

Replicas, $x_2, x_3, \ldots, y_1, y_2, \ldots$ at sites 1, 2, 3, …

Initiate operation $f(x_1, y_1)$; propagate to other sites; *replay* $f(x_2, y_2)$, $f(x_3, y_3)$, …

Same operations, different orders?

State of the art

Scalable systems:

- Last Writer Wins
- Ad-hoc: Lost work, arbitrary

Strong consistency:

- Maintain unknown invariants
- Single, global execution order
- Replay all operations at all replicas
- Lock-step: No concurrency

Multi-object:

- Total order, total replay across all objects

Our goal: Relaxed but principled

2. CRDTs

Commutativity / Eventual Consistency Theorem

Assuming:

- All concurrent operations commute
- Non-concurrent operations execute in happens-before order
- All operations eventually execute at every replica

Then, if clients stop initiating operations, all replicas eventually *converge* to the same *correct* value

Commutativity

Easy case: disjoint objects

CRDT: Commutative Replicated Data Type

- Trivial: ever-growing set + add
- State of the art: ever-growing ordered set
- Treedoc: efficient, scalable ordered set

Shared ordered-set invariants

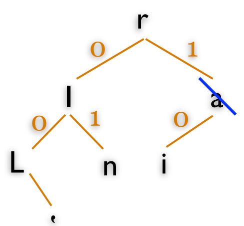
Example: shared buffer "Inria"
buf[0] = "I"; buf[1] = "n"; buf[2] = "r"; ...
Sequence of atoms:

- total order
- identical at all sites
- requires consensus

I < n < r i < a

- partial order
- local

Treedoc ordered set abstraction



= L I n r i a

TID: path = [0|1]*

Contents: infix order

Non-destructive updates

Insert: TIDs do not change

Delete: TIDs do not change; eventually GC leaves

Balancing the tree

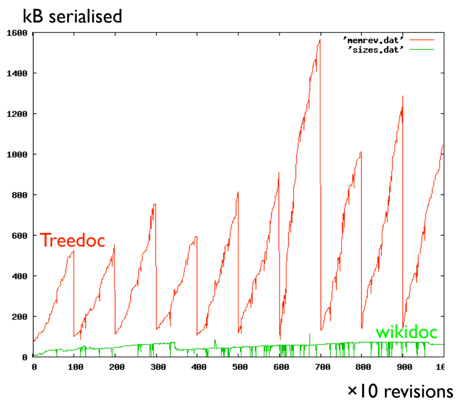
balance

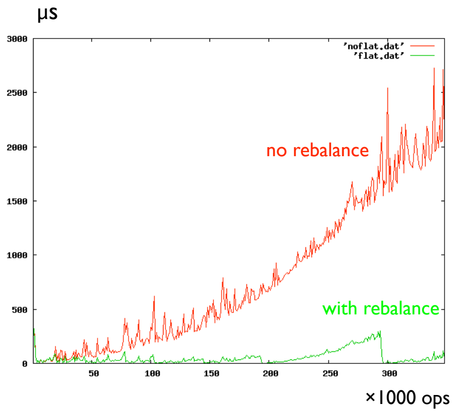
- Does not commute with concurrent updates
- Not essential: *abort* it in the presence of concurrent updates

Requires consensus (2- or 3-phase commit), but off critical path

Garbage collection

Performance





en.wikipedia.org/George_W_Bush

- 150 kB, most frequently revised

3. Optimistic replication

Speculative / optimistic approach to distributed computing

When remote information: wait vs. speculate:

✓Parallel

✓Overcomes latency, failures

✓Reconcile in the background

Maintain invariants

✘ Maintain dependencies

✘ Conflict ⇒ consensus, rollback

✘ Automatic: conservative

Application-specific invariants

Conc. control constraints

Action: reified operation

Constraint: Application-supplied concurrency control specification

Binary relations:

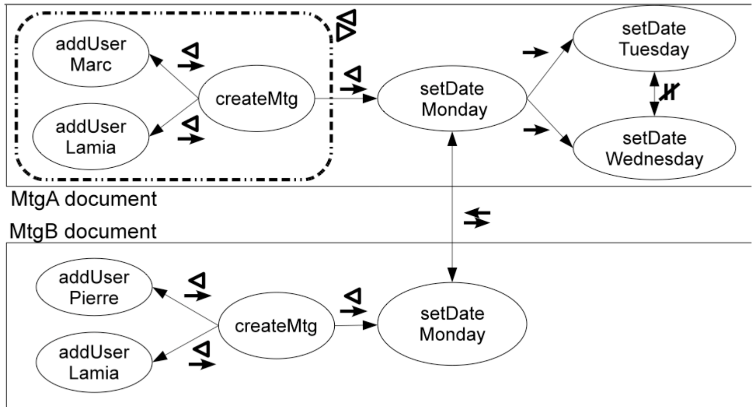
- NotAfter $\alpha \rightarrow \beta$
- Enables (implication) $\alpha \lhd \beta$
- NonCommuting $\alpha \# \beta$

Combinations:

- Antagonistic $\alpha \rightarrow\leftarrow \beta$
- Atomic $\alpha \lhd\rhd \beta$
- Causal dependence $\alpha \rightarrow \beta \wedge \alpha \lhd \beta$

Action-constraint graph ACG

Example ACG: calendar



Schedule: sound cut in the graph

Per site: Scheduling

Sound schedule:

- Path in the ACG that satisfies constraints

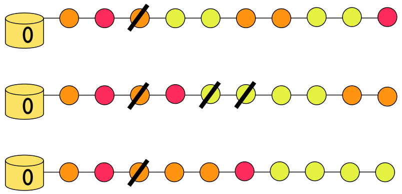
Resolve conflicts:

- Antagonism $\alpha \rightarrow\leftarrow \beta$
- NonCommuting + Dynamic checks $\alpha \# \beta$

Optimal schedule

- Penalise lost work
- IceCube heuristics

Convergence: Eventual consistency



Optimistic: diverge arbitrarily
Common stable prefix
Liveness:

- Every action eventually (aborted or committed) in prefix
- Consensus on next extension of prefix

Telex

Sharing mutable data in decentralised, high-latency environments
Speculative/optimistic execution model
Principled

- Well-defined guarantees
- Tailored to application

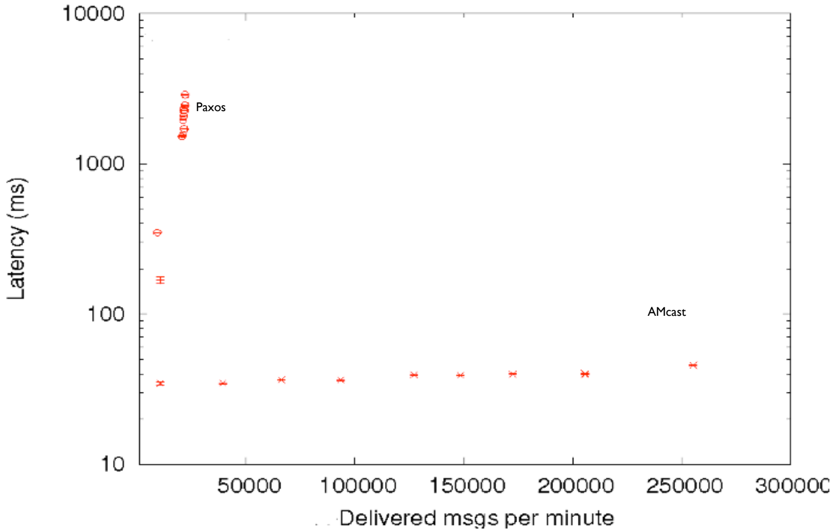
Separation of concerns

- Takes over system issues
- Developer focuses on semantics

At gforge.inria.fr/projects/telex2
BSD licence

4. Partial replication

Total vs. partial order



Partial replication Partial ordering

SOA: total order, replicate at all sites

Partial replication:

- Replicas only at some sites (≥ 1)
- Replay relevant actions only

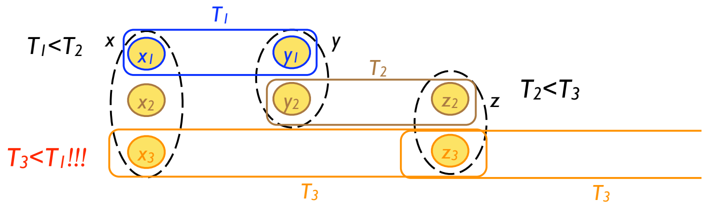
Multi-object transactions:

- Commutative pairs not ordered
- Synchronise only conflicting pairs
- Transitive closure of conflicts

Should scale better

Baseline more complex

Partial replication & consistency



Large-scale distributed system

Many objects

A site replicates only some objects

Occasional multi-object transaction

5. Conclusion

Cluster / distributed system convergence

Petascale for the masses

- Massive computing, storage is available
- Very non-uniform
- Cloud and Edge computing converge
- Asynchronous, failure-prone ⇒ FLP
- Shared data: *Consistency* issue

Application design

Event execution loop + rollback

Spend effort to reduce conflicts!

- Design for commutativity
- Weaken invariants
- Make invariants explicit

Concurrent action pairs

Commute (no constraints)
>> Antagonistic
>> Non-Commuting

System design

- Partial replication: execute subset of operations
- Order only conflicting operations
- Derives from invariants
- Beware transitive conflicts