High Availability under Eventual Consistency

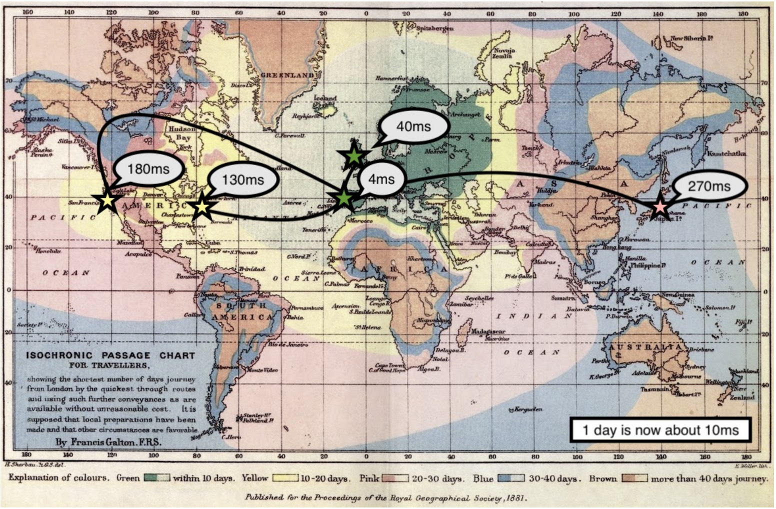
Carlos Baquero
Universidade do Minho & INESC TEC

Shonan PL4DS Meeting 2019

The speed of communication in the 19th century
Francis Galton Isochronic Map

The speed of communication in the 21st century

RTT data gathered via http://www.azurespeed.com

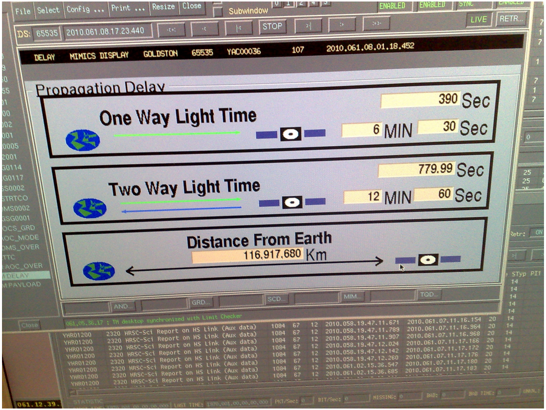


The speed of communication in the 21st century

If you really like high latencies . . .

Time delay between Mars and Earth

`blogs.esa.int/mex/2012/08/05/time-delay-between-mars-and-earth/`



Delay/Disruption Tolerant Networking

`www.nasa.gov/content/dtn`

Latency magnitudes
Geo-replication

Carlos Baquero
Universidade do Minho & INESC TEC

- λ, up to 50ms (local region DC)
- Λ, between 100ms and 300ms (inter-continental)

No inter-DC replication

Client writes observe λ latency

Planet-wide geo-replication

Replication techniques versus client side write latency ranges

Consensus/Paxos $[\Lambda, 2\Lambda]$ (with no divergence)

Primary-Backup $[\lambda, \Lambda]$ (asynchronous/lazy)

Multi-Master λ (allowing divergence)

Multi-Master executions maximize availability and response speeds

From sequential to concurrent executions

Consensus provides illusion of a single replica

This also preserves (slow) sequential behaviour

Sequential execution

$$\textit{Ops } O \qquad o \longrightarrow p \longrightarrow q$$

$$\textit{Time} \qquad \dashrightarrow$$

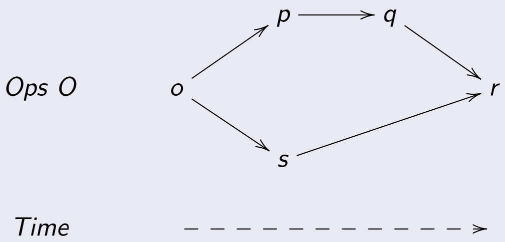
We have an ordered set $(O, <)$. $O = \{o, p, q\}$ and $o < p < q$

Carlos Baquero
Universidade do Minho & INESC TEC

From sequential to concurrent executions

Multi-master can expose concurrency

Concurrent execution



Partially ordered set $(O, \prec)$. $o \prec p \prec q \prec r$ and $o \prec s \prec r$
Some ops in O are concurrent: $p \parallel s$ and $q \parallel s$

Conflict-Free Replicated Data Types (CRDTs)

- Convergence after concurrent updates. Favor AP under CAP
- Examples include counters, sets, mv-registers, maps, graphs
- Operation based CRDTs. Operation effects must commute
- State based CRDTs are rooted on join semi-lattices

Design of Conflict-Free Replicated Data Types

Carlos Baquero Universidade do Minho & INESC TEC

A partially ordered log (polog) of operations implements any CRDT

Replicas keep increasing local views of an evolving distributed polog

Any query, at replica i, can be expressed from local polog O_i

Example: Counter at i is $|\{\mathrm{inc} \mid \mathrm{inc} \in O_i\}| - |\{\mathrm{dec} \mid \mathrm{dec} \in O_i\}|$

CRDTs are efficient representations following some **design principles**

Principle of permutation equivalence

E.g. Counters

If operations in sequence can commute, preserving a given result, then under concurrency they should preserve the same result

Sequential

$$\text{inc}(10) \longrightarrow \text{inc}(35) \longrightarrow \text{dec}(5) \longrightarrow \text{inc}(2)$$

$$\text{dec}(5) \longrightarrow \text{inc}(2) \longrightarrow \text{inc}(10) \longrightarrow \text{inc}(35)$$

Concurrent

$$\begin{tikzcd}
 & \text{inc}(35) \arrow[dr] & \\
\text{inc}(10) \arrow[ur] \arrow[dr] & & \text{inc}(2) \\
 & \text{dec}(5) \arrow[ur] &
\end{tikzcd}$$

You guessed: Result is 42

Registers

Carlos Baquero
Universidade do Minho & INESC TEC

Registers are an ordered set of write operations

Sequential execution

$$A \qquad \mathrm{wr}(x) \longrightarrow \mathrm{wr}(j) \longrightarrow \mathrm{wr}(k) \longrightarrow \mathrm{wr}(x)$$

Sequential execution under distribution

$$\begin{tikzcd}
A & \mathrm{wr}(x) \arrow[dr] & & & \mathrm{wr}(x) \\
B & & \mathrm{wr}(j) \arrow[r] & \mathrm{wr}(k) \arrow[ur] &
\end{tikzcd}$$

Register value is x, the last written value

Implementing Registers
Naive Last-Writer-Wins

CRDT register implemented by attaching local wall-clock times

Sequential execution under distribution

$$
\begin{tikzcd}
A & (11{:}00)x \arrow[dr] & & & (11{:}30)? \arrow[dr] & \\
B & & (12{:}02)j \arrow[r] & (12{:}05)k \arrow[ur] & & ?
\end{tikzcd}
$$

Problem: Wall-clock on B is one hour ahead of A

Value x might not be writeable again at A since 12:05 $>$ 11:30

Register shows value v at replica i iff

$$\mathrm{wr}(v) \in O_i$$

and

$$\nexists \mathrm{wr}(v') \in O_i \cdot \mathrm{wr}(v) < \mathrm{wr}(v')$$

Preservation of sequential semantics

Carlos Baquero
Universidade do Minho & INESC TEC

Concurrent semantics should preserve the sequential semantics

This also ensures correct sequential execution under distribution

Multi-value Registers

Concurrency semantics shows all concurrent values

$$\{v \mid \mathsf{wr}(v) \in O_i \wedge \nexists \mathsf{wr}(v') \in O_i \cdot \mathsf{wr}(v) \prec \mathsf{wr}(v')\}$$

Concurrent execution

$$\begin{tikzcd}
A & \mathsf{wr}(x) \arrow[r] \arrow[dr] & \mathsf{wr}(y) \arrow[rr] & & \{y,k\} \arrow[r] & \mathsf{wr}(m) \arrow[r] & \{m\} \\
B & & \mathsf{wr}(j) \arrow[r] & \mathsf{wr}(k) \arrow[ur] & & &
\end{tikzcd}$$

Dynamo shopping carts are multi-value registers with payload sets

The m value could be an application level merge of values y and k

Implementing Multi-value Registers

Concurrency can be precisely tracked with version vectors

Concurrent execution (version vectors)

$$\begin{tikzcd}
A & {[1,0]x} \arrow[r] \arrow[dr] & {[2,0]y} \arrow[rr] & & {[2,0]y, [1,2]k} \arrow[r] & {[3,2]m} \\
B & & {[1,1]j} \arrow[r] & {[1,2]k} \arrow[ur] & &
\end{tikzcd}$$

Metadata can be compressed with a common causal context and a single scalar per value (dotted version vectors)

Carlos Baquero
Universidade do Minho & INESC TEC

Consider add and rmv operations

$$X = \{\ldots\},\ \mathsf{add(a)} \longrightarrow \mathsf{add(c)} \text{ we observe that } \mathsf{a, c} \in \mathsf{X}$$

$$X = \{\ldots\},\ \mathsf{add(c)} \longrightarrow \mathsf{rmv(c)} \text{ we observe that } \mathsf{c} \notin \mathsf{X}$$

In general, given O_i, the set has elements

$$\{e \mid \mathsf{add(e)} \in \mathsf{O_i} \wedge \nexists \mathsf{rmv(e)} \in \mathsf{O_i} \cdot \mathsf{add(e)} < \mathsf{rmv(e)}\}$$

Problem: Concurrently adding and removing the same element

Concurrent execution

$$
\begin{tikzcd}
A & \mathrm{add}(x) \arrow[r] \arrow[dr] & \mathrm{rmv}(x) \arrow[rr] & & \{?\} \arrow[r] & \mathrm{add}(x) \arrow[r] & \{x\} \\
B & & \mathrm{rmv}(x) \arrow[r] & \mathrm{add}(x) \arrow[ur] & & &
\end{tikzcd}
$$

Carlos Baquero
Universidade do Minho & INESC TEC

Let's choose Add-Wins

Consider a set of known operations O_i, at node i, that is ordered by an *happens-before* partial order $\prec$. Set has elements

$$\{e \mid \mathrm{add(e)} \in \mathrm{O_i} \ \wedge \nexists\ \mathrm{rmv(e)} \in \mathrm{O_i} \cdot \mathrm{add(e)} \prec \mathrm{rmv(e)}\}$$

Concurrency Semantics
Add-Wins Sets

Let's choose Add-Wins

Consider a set of known operations O_i, at node i, that is ordered by an *happens-before* partial order $\prec$. Set has elements

$$\{e \mid \mathrm{add(e)} \in \mathrm{O_i} \ \wedge \nexists\, \mathrm{rmv(e)} \in \mathrm{O_i} \cdot \mathrm{add(e)} \prec \mathrm{rmv(e)}\}$$

Is this familiar?

Concurrency Semantics
Add-Wins Sets

Let's choose Add-Wins

Consider a set of known operations O_i, at node i, that is ordered by an *happens-before* partial order $\prec$. Set has elements

$$\{e \mid \mathsf{add(e)} \in \mathsf{O_i} \ \wedge \nexists\, \mathsf{rmv(e)} \in \mathsf{O_i} \cdot \mathsf{add(e)} \prec \mathsf{rmv(e)}\}$$

Is this familiar?

The sequential semantics applies identical rules on a total order

Let's choose Add-Wins

Consider a set of known operations O_i, at node i, that is ordered by an *happens-before* partial order $\prec$. Set has elements

$$\{e \mid \mathrm{add(e)} \in \mathrm{O_i} \ \wedge \nexists \ \mathrm{rmv(e)} \in \mathrm{O_i} \cdot \mathrm{add(e)} \prec \mathrm{rmv(e)}\}$$

Is this familiar?

The sequential semantics applies identical rules on a total order

$$\{e \mid \mathrm{add(e)} \in \mathrm{O_i} \wedge \nexists \mathrm{rmv(e)} \in \mathrm{O_i} \cdot \mathrm{add(e)} < \mathrm{rmv(e)}\}$$

Equivalence to a sequential execution?
Add-Wins Sets

Can we always explain a concurrent execution by a sequential one?

Concurrent execution

$$
\begin{tikzcd}
A & \{x,y\} \arrow[r] & \mathsf{add}(y) \arrow[r] & \mathsf{rmv}(x) \arrow[r] & \{y\} \arrow[r] \arrow[dr] & \{x,y\} \\
B & \{x,y\} \arrow[r] & \mathsf{add}(x) \arrow[r] & \mathsf{rmv}(y) \arrow[r] & \{x\} \arrow[r] \arrow[ur] & \{x,y\}
\end{tikzcd}
$$

Two (failed) sequential explanations

$$H1 \qquad \{x,y\} \longrightarrow \ldots \longrightarrow \mathsf{rmv}(x) \longrightarrow \{\cancel{x}, y\}$$

$$H2 \qquad \{x,y\} \longrightarrow \ldots \longrightarrow \mathsf{rmv}(y) \longrightarrow \{x, \cancel{y}\}$$

Concurrent executions can have richer outcomes

Concurrency Semantics

Remove-Wins Sets

Alternative: Let's choose Remove-Wins

$$X_i \doteq \{e \mid \mathsf{add(e)} \in \mathsf{O_i} \ \wedge \forall\, \mathsf{rmv(e)} \in \mathsf{O_i} \cdot \mathsf{rmv(e)} \prec \mathsf{add(e)}\}$$

Concurrency Semantics
Remove-Wins Sets

Carlos Baquero Universidade do Minho & INESC TEC

Alternative: Let's choose Remove-Wins

$$X_i \doteq \{e \mid \mathrm{add(e)} \in \mathrm{O_i} \;\wedge \forall\, \mathrm{rmv(e)} \in \mathrm{O_i} \cdot \mathrm{rmv(e)} \prec \mathrm{add(e)}\}$$

Remove-Wins requires more metadata than Add-Wins

Both Add and Remove-Wins have same semantics in a total order

They are different but both preserve sequential semantics

Reference

Conflict-Free Replicated Data Types CRDTs.
Encyclopedia of Big Data Technologies, 2019.
Nuno M. Preguiça, Carlos Baquero, Marc Shapiro.

Open Questions

- Given operations and queries can CRDTs be synthesized?
- Stateful (dis)order tolerant communication protocols
- Integrity guaranties and privacy in data evolution