CRDTs
From sequential to concurrent executions

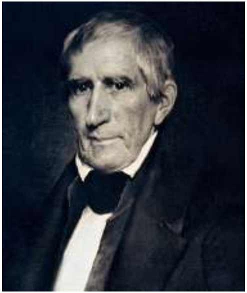
Carlos Baquero

INESC TEC & Universidade do Minho, Portugal

Code Mesh London, November 8th 2018

The speed of communication in the 19th century

W. H. Harrison's death



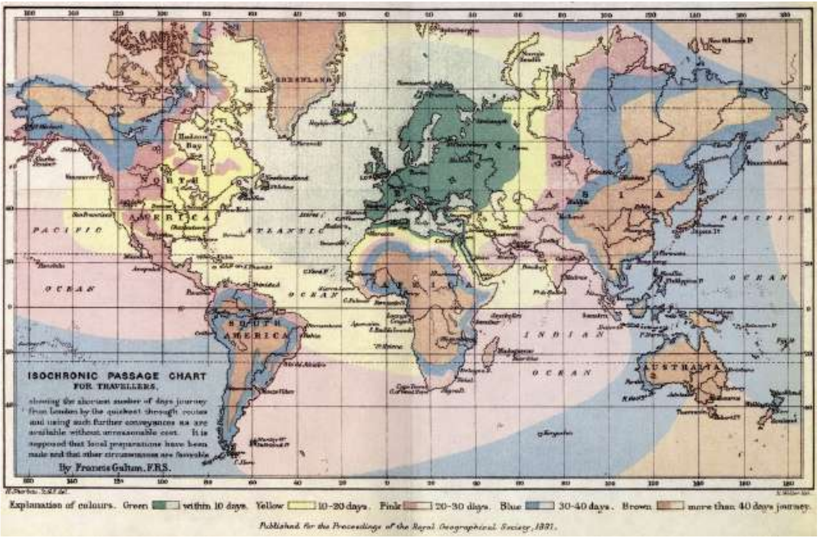
"At 12:30 am on April 4th, 1841 President William Henry Harrison died of pneumonia just a month after taking office. The Richmond Enquirer published the news of his death two days later on April 6th. The North-Carolina standard newspaper published it on April 14th. His death wasn't known of in Los Angeles until July 23rd, 110 days after it had occurred."

Text by Zack Bloom, *A Quick History of Digital Communication Before the Internet*. `https://eager.io/blog/communication-pre-internet/`

Picture by By Albert Sands Southworth and Josiah Johnson Hawes

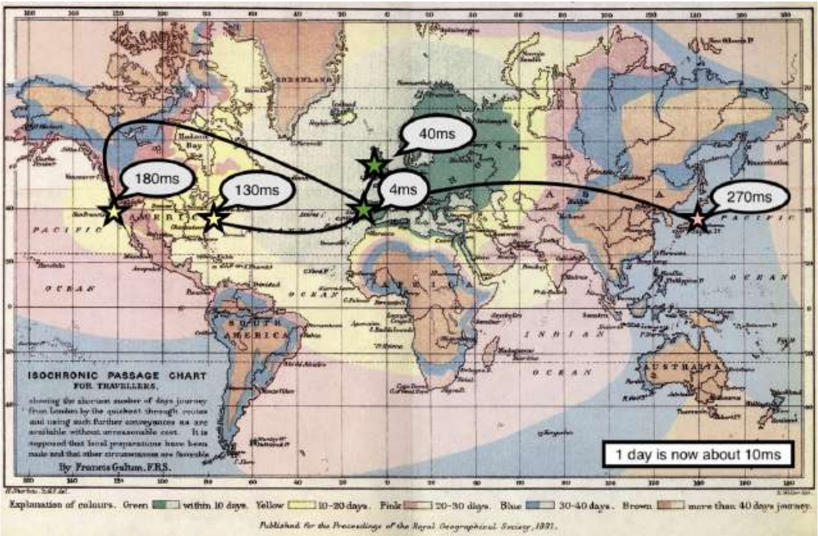
The speed of communication in the 19th century

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 MIMICS DISPLAY GOLDSTON 65535 YAC00036 107 2010.061.08.01.18.452

Propagation Delay

One Way Light Time 390 Sec
6 MIN 30 Sec

Two Way Light Time 779.99 Sec
12 MIN 60 Sec

Distance From Earth 116.917.680 Km

Delay/Disruption Tolerant Networking

`www.nasa.gov/content/dtn`

Latency magnitudes

Geo-replication

- 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)

EC and CAP for Geo-Replication

Eventually Consistent. CACM 2009, Werner Vogels

- In an ideal world there would be only one consistency model: when an update is made all observers would see that update.
- Building reliable distributed systems at a worldwide scale demands trade-offs between consistency and availability.

CAP theorem. PODC 2000, Eric Brewer

Of three properties of shared-data systems – data consistency, system availability, and tolerance to network partition – only two can be achieved at any given time.

CRDTs provide support for partition-tolerant high availability

From sequential to concurrent executions

Consensus provides illusion of a single replica

This also preserves (slow) sequential behaviour

Sequential execution

Ops O $o \longrightarrow p \longrightarrow q$

Time $\dashrightarrow$

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

From sequential to concurrent executions

Consensus provides illusion of a single replica

This also preserves (slow) sequential behaviour

Sequential execution

Ops O $o \longrightarrow p \longrightarrow q$

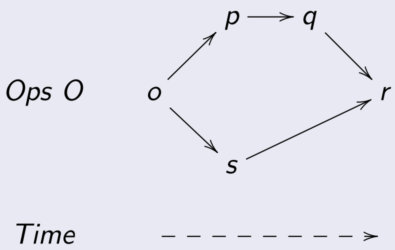
Time $- - - - - - - - \dashrightarrow$

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

From sequential to concurrent executions

EC Multi-master (or active-active) 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 $q \parallel s$

Design of Conflict-Free Replicated Data Types

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 $|\{\text{inc} \mid \text{inc} \in O_i\}| - |\{\text{dec} \mid \text{dec} \in O_i\}|$

CRDTs are efficient representations that follow some general rules

Design of Conflict-Free Replicated Data Types

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 $|\{\mathsf{inc} \mid \mathsf{inc} \in O_i\}| - |\{\mathsf{dec} \mid \mathsf{dec} \in O_i\}|$

CRDTs are efficient representations that follow some general rules

Design of Conflict-Free Replicated Data Types

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 $|\{\mathsf{inc} \mid \mathsf{inc} \in O_i\}| - |\{\mathsf{dec} \mid \mathsf{dec} \in O_i\}|$

CRDTs are efficient representations that follow some general rules

Principle of permutation equivalence

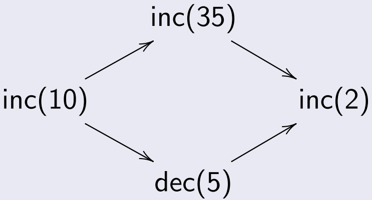
If operations in sequence can commute, preserving a given result, then under concurrency they should preserve the same result

Sequential

inc(10) → inc(35) → dec(5) → inc(2)

dec(5) → inc(2) → inc(10) → inc(35)

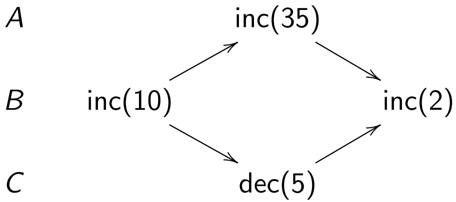
Concurrent



You guessed: Result is 42

Implementing Counters

Example: CRDT PNCounters



A inc(35)

B inc(10) inc(2)

C dec(5)

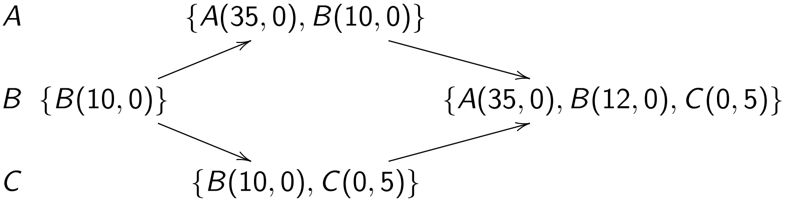
Lets track total number of incs and decs done at each replica

$$\{A(\text{incs}, \text{decs}), \ldots, C(\ldots, \ldots)\}$$

Implementing Counters

Example: CRDT PNCounters

Separate positive and negative counts are kept per replica



A $\{A(35,0), B(10,0)\}$

B $\{B(10,0)\}$ $\{A(35,0), B(12,0), C(0,5)\}$

C $\{B(10,0), C(0,5)\}$

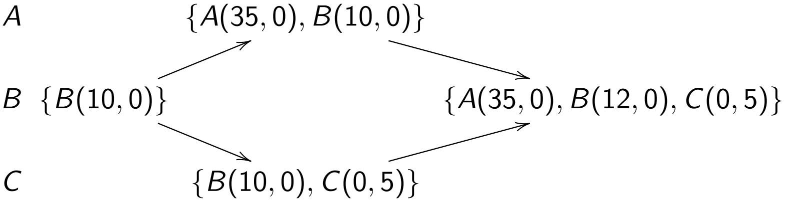
Joining does point-wise maximums among entries (semilattice)

At any time, counter value is sum of incs minus sum of decs

Implementing Counters

Example: CRDT PNCounters

Separate positive and negative counts are kept per replica



Joining does point-wise maximums among entries (semilattice)

At any time, counter value is sum of incs minus sum of decs

Registers

Registers are an ordered set of write operations

Sequential execution

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

Sequential execution under distribution

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

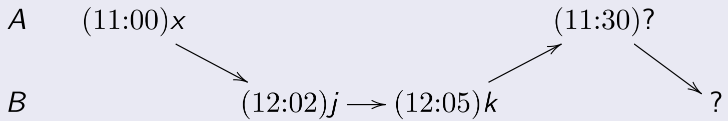
Register value is x, the last written value

Implementing Registers

Naive Last-Writer-Wins

CRDT register implemented by attaching local wall-clock times

Sequential execution under distribution



A $(11{:}00)x$ $(11{:}30)?$

B $(12{:}02)j \longrightarrow (12{:}05)k$?

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

Registers

Sequential Semantics

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

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)

Use case: Registers in Redis CRDB

LWW arbitration

Multi-value registers allows executions leading to concurrent values

Presenting concurrent values is at odds with the sequential API

Redis CRDB both tracks causality and registers wall-clock times

Querying uses Last-Writer-Wins selection among concurrent values

This preserves correctness of sequential semantics

A value with clock 12:05 can still be causally overwritten at 11:30

Sets

Sequential Semantics

Consider add and rmv operations

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

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

In general, given O_i, the set has elements

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

Sets

Sequential Semantics

Consider add and rmv operations

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

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

In general, given O_i, the set has elements

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

Sets

Sequential Semantics

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

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

Sets

Sequential Semantics

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

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

Sets

Sequential Semantics

Consider add and rmv operations

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

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

In general, given O_i, the set has elements

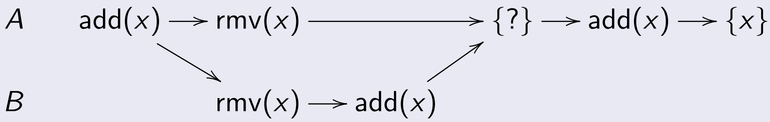
$$\{e \mid \mathsf{add(e)} \in \mathsf{O_i} \wedge \nexists \mathsf{rmv(e)} \in \mathsf{O_i} \cdot \mathsf{add(e)} < \mathsf{rmv(e)}\}$$

Sets

Concurrency Semantics

Problem: Concurrently adding and removing the same element

Concurrent execution



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

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

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

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

Equivalence to a sequential execution?

Add-Wins Sets

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

Concurrent execution

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

Two (failed) sequential explanations

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

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

Concurrent executions can have richer outcomes

Concurrency Semantics

Remove-Wins Sets

Alternative: Let's choose Remove-Wins

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

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

Concurrency Semantics

Remove-Wins Sets

Alternative: Let's choose Remove-Wins

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

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

Choice of semantics

Design freedom is limited by preservation of sequential semantics

Delaying choice of semantics to query time

A CRDT Set data type could store enough information to allow a parametrized query that shows either Add-Wins or Remove-Wins

This flexibility might have a metadata cost

CRDTs in Practice

Implementation styles

- State-based: Full state dissemination; merging of replicas
    - Alternative: Disseminate small state deltas, δ-states
    - States can be merged multiple times
- Operation-based: Reliable dissemination; known membership
    - Operations applied only once

Infrastructure

- Datatype libraries + Dissemination/Gossip Middleware
- Databases with rich APIs and CRDT merge logic

CRDTs in Practice

Use-case	Company/Project	CRDT model
Distributed Applications	Akka	δ State-based
Distributed Applications	Lasp	δ State-based
Distributed Applications	Eventuate	Op-based
P2P Collaborative Editing	IPFS	Op-based
Distributed DB	Riak	State-based
Distributed DB	Redis	Both
Distributed DB	Hazelcast	State-based
Dist. DB, HAT transactions	Antidote	Op-based

Take home message

- Concurrent executions are needed to deal with latency
- Behaviour changes when moving from sequential to concurrent

Road to accommodate transition:

- Permutation equivalence
- Preserving sequential semantics
- Concurrent executions lead to richer outcomes

CRDTs provide sound guidelines and encode policies

Thanks and Questions

Reference

Conflict-Free Replicated Data Types. N. Preguiça, M. Shapiro, C. Baquero. Encyclopedia of Big Data Technologies, Springer Verlag

Thanks to LightKone (`https://www.lightkone.eu`) for support, Redis Labs (`https://redislabs.com`) for their support and inputs on an early version, and my colleagues for early feedback

Glad to address any questions

Carlos Baquero, cbm@di.uminho.pt, @xmal

LIGHTKONE
Lightweight computation for networks at the edge

Universidade do Minho

INESCTEC
TECHNOLOGY & SCIENCE
ASSOCIATE LABORATORY