CRDTs: State-based approaches to high availability

Carlos Baquero
Universidade do Porto

cbm@fe.up.pt

DARE

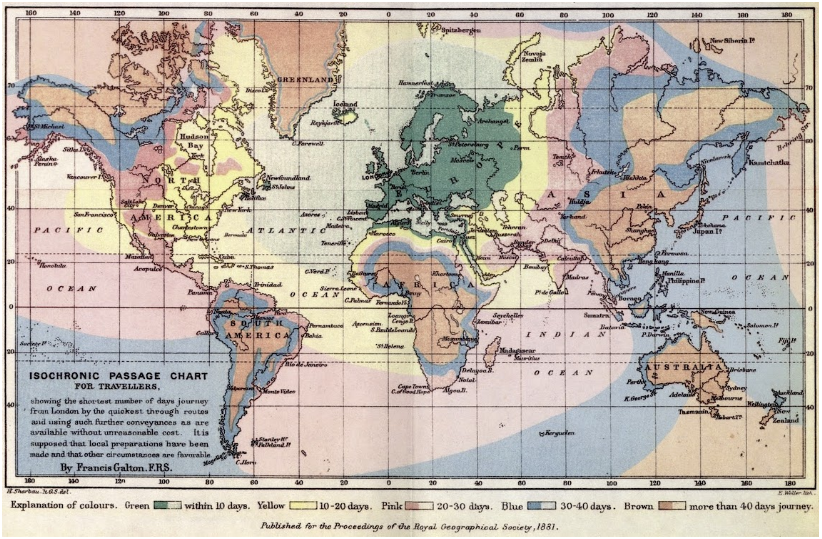
September 12th, 2023. Brussels, BE

Outline

- CAP trade-off and motivation for CRDTs
- Operations vs State
- State based CRDTs
- Delta state based CRDTs
- Decomposing states
- Update propagation

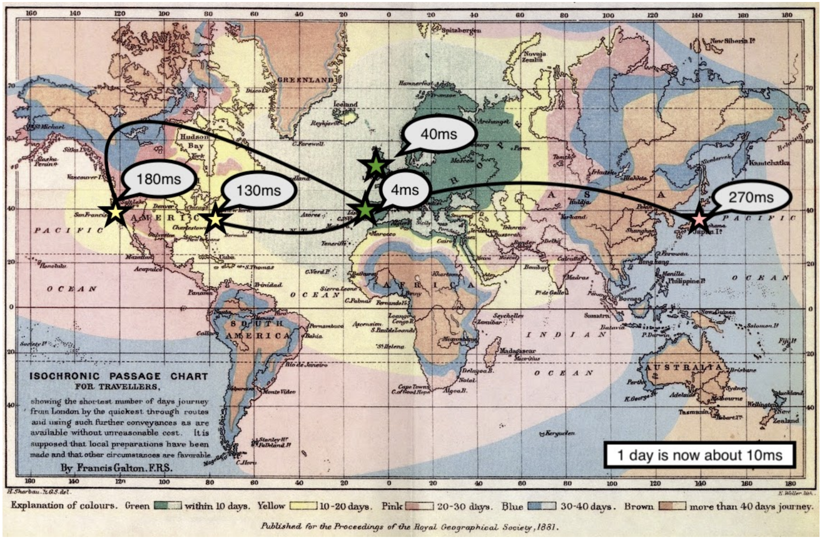
The speed of communication in the 19th century

Francis Galton Isochronic Map


ISOCHRONIC PASSAGE CHART
FOR TRAVELLERS,
showing the shortest number of days journey from London by the quickest through routes and using such further conveyances as are available without unreasonable cost. It is supposed that local preparations have been made and that other circumstances are favorable.
By Francis Galton, F.R.S.
Explanation of colours. Green ▨ within 10 days. Yellow ▨ 10-20 days. Pink ▨ 20-30 days. Blue ▨ 30-40 days. Brown ▨ more than 40 days journey.
Published for the Proceedings of the Royal Geographical Society, 1881.
H. Sharbau. R.G.S. del.
E. Weller lith.
GREENLAND
NORTH AMERICA
SOUTH AMERICA
EUROPE
AFRICA
ASIA
AUSTRALIA
PACIFIC OCEAN
ATLANTIC OCEAN
INDIAN OCEAN
London
Spitzbergen
Novaja Zemlia
New Siberia Is.
Iceland
Hudson Bay
Kamtchatka
Japan Is.
Philippine Is.
New Zealand
Madagascar
Kerguelen
C. Horn

The speed of communication in the 21st century

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



40ms
180ms
130ms
4ms
270ms
1 day is now about 10ms
ISOCHRONIC PASSAGE CHART FOR TRAVELLERS,
showing the shortest number of days journey from London by the quickest through routes and using such further conveyances as are available without unreasonable cost. It is supposed that local preparations have been made and that other circumstances are favorable.
By Francis Galton, F.R.S.
Explanation of colours. Green within 10 days. Yellow 10-20 days. Pink 20-30 days. Blue 30-40 days. Brown more than 40 days journey.
Published for the Proceedings of the Royal Geographical Society, 1881.

Communication networks will keep expanding

Earth $\leftrightarrow$ Mars link, depends on planets positions:

- One way: 3 to 21 minutes
- Round trip: 6 to 42 minutes

(Image by tj.blackwell – CC BY-NC 2.0.)

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)
Multi-Master	λ	(allowing divergence)

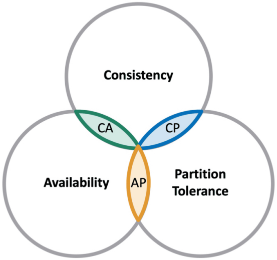
EC and CAP for Geo-Replication

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 the three properties – data consistency, system availability, and tolerance to network partition – only two can be simultaneously achieved.



(Image from https://hazelcast.com/glossary/cap-theorem/)

High Availability – Eventual Consistency

A special case of weak consistency. After an update, if no new updates are made to the object, eventually all reads will return the same value, that reflects the last update. E.g: DNS.

This can later be reformulated to avoid quiescence.

From sequential to concurrent executions

Consensus provides illusion of a single replica

This also preserves (slow $\approx$ 200ms) sequential behaviour

Sequential execution

$$\begin{tikzcd}
\mathit{Ops}\ O & o \arrow[r] & p \arrow[r] & q \\
\mathit{Time} & {} \arrow[rrr, dashed] & & & {}
\end{tikzcd}$$

We have a total order on the set of operations $(O, <)$.
$O = \{o, p, q\}$ and $o < p < q$

From sequential to concurrent executions

EC Multi-master is fast, any replica can locally accept updates
But can expose concurrency

Concurrent execution

$$\begin{tikzcd}
 & & p \arrow[r] & q \arrow[dr] & \\
 \mathit{Ops}\ O & o \arrow[ur] \arrow[dr] & & & r \\
 & & s \arrow[urr] & & \\
 \mathit{Time} & {} \arrow[rrr, dashed] & & & {}
\end{tikzcd}$$

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
- State based can emulate operation based – and vice-versa

Operation-based CRDTs, effect commutativity

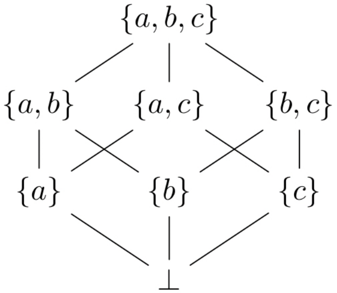
- In some datatypes all operations are commutative.
- PN-Counter: $\text{inc}(\text{dec}(c)) = \text{dec}(\text{inc}(c))$
- G-Set: $\text{add}_a(\text{add}_b(s)) = \text{add}_b(\text{add}_a(s))$

For more complex examples (e.g. sets with add and remove) operations need to generate "special" commutative effects. Here we will only cover examples of state-based CRDTs as they are more common in practice.

State-based CRDTs, Join semi-lattices

- An (partial) ordered set S; $\langle S, \leq \rangle$.
- A join, $\sqcup$, deriving least upper bounds; $\langle S, \leq, \sqcup \rangle$.
- An initial state, usually the least element $\perp$; $\langle S, \leq, \sqcup, \perp \rangle$.
  $(\forall a \in S, a \sqcup \perp = a)$
- Alternative to a (unique) initial state, is a one time init in each replica assigning any element from S.
- Join properties in a semilattice $\langle S, \leq, \sqcup \rangle$:
  - Idempotence, $a \sqcup a = a$,
  - Commutativity, $a \sqcup b = b \sqcup a$,
  - Associative, $(a \sqcup b) \sqcup c = a \sqcup (b \sqcup c)$.
- $\leq$ reflects state evolution/inflation.
- Updates/mutations $m()$ increase information, $x \leq m(x)$.
- In general, queries can return non-monotonic values, and in other domains than S. E.g: Returning a set size.

State-based CRDTs, Join semi-lattices



(b) GSet$\langle\{a,b,c\}\rangle$.

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

$Ops \in \{\mathsf{inc}, \mathsf{dec}\}$

$$\begin{tikzcd}
 & \mathsf{dec} \arrow[r] & \mathsf{inc} \arrow[dr] & \\
\mathsf{inc} \arrow[ur] \arrow[dr] & & & \mathsf{inc} \\
 & \mathsf{inc} \arrow[urr] & &
\end{tikzcd}$$

$Time \quad \dashrightarrow$

State based CRDTs are efficient specialized representations of pologs

Implementing Counters

Example: CRDT PNCounters

$$\begin{tikzcd}
A & & \text{dec} \arrow[r] & \text{inc} \arrow[dr] & \\
B & \text{inc} \arrow[ur] \arrow[dr] & & & \text{inc} \\
C & & \text{inc} \arrow[urr] & & \\
Time & {} \arrow[rrr, dashed] & & & {}
\end{tikzcd}$$

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

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

This is a semi-lattice, with $\perp = \{A(0,0), B(0,0), C(0,0)\}$

Implementing Counters

Example: CRDT PNCounters

$$\begin{tikzcd}
A & & \text{dec} \arrow[r] & \text{inc} \arrow[dr] & \\
B & \text{inc} \arrow[ur] \arrow[dr] & & & \text{inc} \\
C & & \text{inc} \arrow[urr] & & \\
\mathit{Time} & {} \arrow[rrr, dashed] & & & {}
\end{tikzcd}$$

$$\begin{tikzcd}
A & & \{A(0,1), B(1,0)\} \arrow[r] & \{A(1,1), B(1,0)\} \arrow[dr] & \\
B & \{B(1,0)\} \arrow[ur] \arrow[dr] & & & \{A(1,1), B(2,0), C(1,0)\} \\
C & & \{B(1,0), C(1,0)\} \arrow[urr] & &
\end{tikzcd}$$

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

State-based GCounter

$$\begin{aligned}
\mathsf{GCounter} &= \mathbb{I} \hookrightarrow \mathbb{N} \\
\bot &= \{\} \\
\mathsf{inc}_i(m) &= m\{i \mapsto m[i] + 1\} \\
\mathsf{value}(m) &= \sum_{j \in \mathsf{dom}\, m} m[j] \\
m \sqcup m' &= \{j \mapsto \max(m[j], m'[j]) \mid j \in \mathsf{dom}\, m \cup \mathsf{dom}\, m'\}
\end{aligned}$$

- State maps replica ids to integers
- inc increments self entry (i)
- Join is pointwise max
- Similar in structure to a version vector

State-based PNCounter

$$\begin{array}{rcl}
\mathsf{PNCounter} & = & \mathsf{GCounter} \times \mathsf{GCounter} \\
\bot & = & (\bot, \bot) \\
\mathsf{inc}_i((p, n)) & = & (\mathsf{inc}_i(p), n) \\
\mathsf{dec}_i((p, n)) & = & (p, \mathsf{inc}_i(n)) \\
\mathsf{value}((p, n)) & = & \mathsf{value}(p) - \mathsf{value}(n) \\
(p, n) \sqcup (p', n') & = & (p \sqcup p', n \sqcup n')
\end{array}$$

- Solved through a pair of GCounters (product composition)
  - increments and decrements tracked separately
  - counter value obtained as difference
- In practice a single map to pairs is used (as seen on the example)

Grow only set GSet$\langle E \rangle$

$$\begin{aligned}
\mathrm{GSet}\langle E \rangle &= \mathcal{P}(E) \\
\bot &= \{\} \\
\mathrm{add}_i(e, s) &= s \cup \{e\} \\
\mathrm{elements}(s) &= s \\
\mathrm{contains}(e, s) &= e \in s \\
s \sqcup s' &= s \cup s'
\end{aligned}$$

- Trivial CRDT: state same as sequential datatype
- Add already an inflation
- Merging replica states by set union
- Anonymous CRDT: no need for node ids in the state

How to delete from a set?

Two phase set 2PSet$\langle E \rangle$

$$\begin{array}{rcl}
\mathrm{2PSet}\langle E \rangle & = & \mathcal{P}(E) \times \mathcal{P}(E) \\
\bot & = & \{\} \times \{\} \\
\mathrm{add}_i(e, (s, t)) & = & (s \cup \{e\}, t) \\
\mathrm{rmv}_i(e, (s, t)) & = & (s, t \cup \{e\}) \\
\mathrm{elements}((s, t)) & = & s \setminus t \\
\mathrm{contains}(e, (s, t)) & = & e \in (s \setminus t) \\
(s, t) \sqcup (s', t') & = & (s \cup s', t \cup t')
\end{array}$$

Problems:

- Does not comply with usual sequential behaviour
- Cannot delete and readd elements

Lets try to add unique tags to each added element

Basic add-wins set AWSet$\langle E \rangle$

$$\begin{aligned}
\mathsf{AWSet}\langle E \rangle &= \mathcal{P}(E \times U) \times \mathcal{P}(U) \\
\bot &= \{\} \times \{\} \\
\mathsf{add}_i(e, (s, t)) &= (s \cup \{(e, utag())\}, t) \\
\mathsf{rmv}_i(e, (s, t)) &= (s, t \cup \{u \mid (e, u) \in s\}) \\
\mathsf{elements}((s, t)) &= \{e \mid (e, u) \in s \land u \notin t\} \\
\mathsf{contains}(e, (s, t)) &= e \in \mathsf{elements}((s, t)) \\
(s, t) \sqcup (s', t') &= (s \cup s', t \cup t')
\end{aligned}$$

Problems:

- Removed elements are still kept in s
- How to generate unique tags? random strings?

Dots as Causal Histories

We can use the node id $i \in \mathbb{I}$ and a natural counter $\mathbb{N}$, to create tags of the form $\mathbb{I} \times \mathbb{N}$.

- $\{(a, 1), (a, 2), (b, 1), (b, 2), (b, 3), (c, 1), (c, 2), (c, 3)\}$
- $\{a \mapsto 2, b \mapsto 3, c \mapsto 3\}$

In cases with fixed membership one can compress to a vector

- $[2, 3, 3]$

Causal Context

$$\begin{aligned}
\mathsf{CausalContext} &= \mathcal{P}(\mathbb{I} \times \mathbb{N}) \\
\mathsf{max}_i(c) &= \mathsf{max}(\{n \mid (i, n) \in c\} \cup \{0\}) \\
\mathsf{next}_i(c) &= (i, \mathsf{max}_i(c) + 1)
\end{aligned}$$

Example: $\mathsf{next}_a(\{(a, 1), (a, 2), (b, 1), (b, 2), (b, 3)\}) = (a, 3)$

(causal) Add-wins set $\mathsf{AWSet}\langle E\rangle$

$$
\begin{array}{rcl}
\mathsf{AWSet}\langle E\rangle & = & \mathcal{P}(E \times \mathbb{I} \times \mathbb{N}) \times \mathcal{P}(\mathbb{I} \times \mathbb{N}) \\
\bot & = & \{\} \times \{\} \\
\mathsf{add}_i(e, (s, c)) & = & (s \cup \{(e, d)\}, c \cup \{d\}) \textbf{ where } d = \mathsf{next}_i(c) \\
\mathsf{rmv}_i(e, (s, c)) & = & (s \setminus \{(e, d) \mid (e, d) \in s\}, c) \\
\mathsf{elements}((s, c)) & = & \{e \mid (e, d) \in s\} \\
\mathsf{contains}(e, (s, c)) & = & e \in \mathsf{elements}((s, c)) \\
(s, c) \sqcup (s', c') & = & (s \cap s' \cup f(s, c') \cup f(s', c), c \cup c') \\
 & & \quad \textbf{where } f(a, b) = \{(e, d) \mid (e, d) \in a \wedge d \notin b\}
\end{array}
$$

Note 1: For simplicity we assume $(e, (i, n)) \equiv (e, i, n)$.
Note 2: $f(a, b)$ is a filter function that preserves elements in a not known in b.

State based CRDTs

Pros

- Permissive communication model: Drops, Reorder, Duplicates
- Causal consistency by default
- Decouples operation rate from transmission rate
- Can compact operations

Cons

- All metadata makes part of the state
- Requires full state transmission

What if could separate in updates the state that inflates the lattice?

Delta state-based GCounter

$$\begin{array}{rcl}
\mathsf{GCounter} & = & \mathbb{I} \hookrightarrow \mathbb{N} \\
\bot & = & \{\} \\
\mathsf{inc}_i(m) & = & m\{i \mapsto m[i] + 1\} \\
\mathsf{inc}^{\delta}_i(m) & = & \{i \mapsto m[i] + 1\} \\
\mathsf{value}(m) & = & \displaystyle\sum_{j \in \mathsf{dom}\, m} m[j] \\
m \sqcup m' & = & \{j \mapsto \max(m[j], m'[j]) \mid j \in \mathsf{dom}\, m \cup \mathsf{dom}\, m'\}
\end{array}$$

Delta State CRDTs

- State: like normal state-based CRDTs
- Messages: based on *delta-states*, that are states, hopefully small
- Essential difference: delta-mutators, which return delta states, to be
  - joined with current state
  - joined with other delta states, forming *delta-groups* d, to send in messages
- For each mutator m we can define a delta-mutator m^δ such that:

$$m(X) = X \sqcup m^\delta(X)$$

Basic anti-entropy algorithm

```
durable state:
    $X_i \in S$, CRDT state; initially
     $X_i = \bot$

volatile state:
    $D_i \in S$, delta-buffer; initially
     $D_i = \bot$

on operation$_i(m^\delta)$
    let $d = m^\delta(X_i)$
    $X_i \leftarrow X_i \sqcup d$
    $D_i \leftarrow D_i \sqcup d$

on receive$_{j,i}(d)$
    $X_i \leftarrow X_i \sqcup d$
    $D_i \leftarrow D_i \sqcup d$

periodically
    let $m =$ choose$_i(X_i, D_i)$
    for $j$ in neighbors$_i$ do
        send$_{i,j}(m)$
    $D_i \leftarrow \bot$
```

Problems with naive anti-entropy

Naive transitive propagation causes much redundancy

- deltas re-propagated back to where they came from
- delta-group wholly joined to delta-buffer, even if mostly reflected in state

Solution

- avoid back-propagation of delta-groups; tagging origin
- remove redundant state in received delta-groups; how?

We need tools to compare states

Join Decompositions

D is a irredundant join decomposition of s :

$$
\begin{array}{ll}
\text{(decomposition)} & \bigsqcup D = s \\
\text{(irredundant)} & \forall d \in D \cdot \bigsqcup (D \setminus \{d\}) \sqsubset s
\end{array}
$$

Join Decompositions

Given $s = \{a, b, c\}$:

- ✓ $D = \{\{a, b, c\}\}$
- ✗ $D = \{\{b\}, \{c\}\}$
- ✗ $D = \{\{a, b\}, \{b\}, \{d\}\}$
- ✓ $D = \{\{a, b\}, \{c\}\}$
- ✓ $D = \{\{a\}, \{b\}, \{c\}\}$

(This still allows join reducible elements, e.g. $\{a, b\}$)

Join-Irreducible Decompositions

- ✗ $D = \{\{a, b, c\}\}$
- ✗ $D = \{\{a, b\}, \{c\}\}$
- ✓ $D = \{\{a\}, \{b\}, \{c\}\}$

Join-Irreducible Decompositions

GCounter $\Downarrow m = \{\{i \mapsto n\} \mid i \mapsto n \in m\}$

GSet $\Downarrow s = \{\{e\} \mid e \in s\}$

Differences and optimal deltas

We knew how to derive standard mutators from delta mutators

Mutator from deltas

$\mathrm{m}(x) = x \sqcup \mathrm{m}^{\delta}(x)$

Now we can derive optimal delta mutators from standard mutators

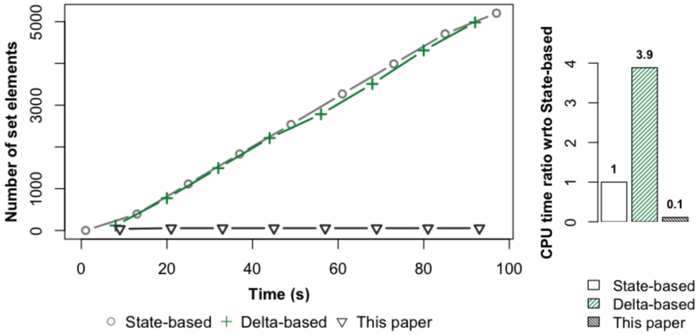
Difference

$\Delta(a, b) = \bigsqcup\{y \in \Downarrow a \mid y \not\sqsubseteq b\}$

Optimal delta mutator

$\mathrm{m}^{\delta}(x) = \Delta(\mathrm{m}(x), x)$

Improving anti-entropy



Improving anti-entropy

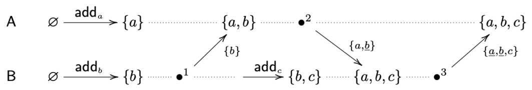


Fig. 4: Delta-based synchronization of a GSet with 2 replicas $\mathsf{A}, \mathsf{B} \in \mathbb{I}$. Underlined elements represent the BP optimization.

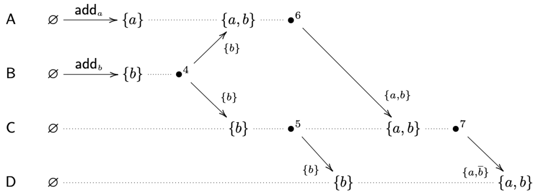


Fig. 5: Delta-based synchronization of a GSet with 4 replicas $\mathsf{A}, \mathsf{B}, \mathsf{C}, \mathsf{D} \in \mathbb{I}$. The overlined element represents the RR optimization.

Improving anti-entropy

Channel can duplicate and reorder, but does not drop (missing Ack step)

```
inputs:
  $n_i \in \mathcal{P}(\mathbb{I})$, set of neighbors
state:
  $x_i \in \mathcal{L}$, $x_i^0 = \bot$
  $B_i \in \mathcal{P}(\mathcal{L})$, $B_i^0 = \varnothing$        $B_i \in \mathcal{P}(\mathcal{L} \times \mathbb{I})$, $B_i^0 = \varnothing$
on operation$_i(\mathsf{m}^\delta)$
  $\delta = \mathsf{m}^\delta(x_i)$
  store$(\delta, i)$
periodically // synchronize
  for $j \in n_i$
    $d = \bigsqcup B_i$                   $d = \bigsqcup\{s \mid \langle s, o\rangle \in B_i \wedge o \neq j\}$
    send$_{i,j}$(delta, $d$)
  $B_i' = \varnothing$
on receive$_{j,i}$(delta, $d$)
                                     $d = \Delta(d, x_i)$
  if $d \not\sqsubseteq x_i$                  if $d \neq \bot$
    store$(d, j)$
fun store$(s, o)$
  $x_i' = x_i \sqcup s$
  $B_i' = B_i \cup \{s\}$               $B_i' = B_i \cup \{\langle s, o\rangle\}$
```

Algorithm 1: Delta-based synchronization algorithms at replica $i \in \mathbb{I}$: classic version and version with BP and RR optimizations.

Improving anti-entropy

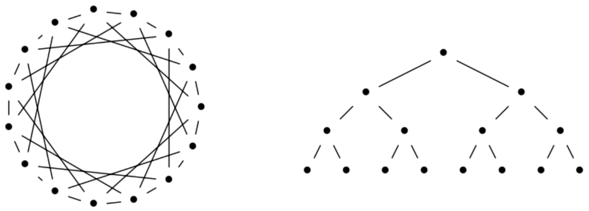


Fig. 6: Network topologies employed: a 15-node partial-mesh (to the left) and a 15-node tree (to the right).

Summary

- CAP trade-off and motivation for CRDTs
- Operations vs State
- State based CRDTs
- Delta state based CRDTs
- Decomposing states
- Update propagation

Bibliography

- N. Preguiça, C. Baquero, M. Shapiro, “Conflict-free Replicated Data Types (CRDTs),” in Encyclopedia of Big Data Technologies, Springer International Publishing, 2018, https://doi.org/10.48550/arXiv.1805.06358
- P. Almeida, A. Shoker and C. Baquero, “Delta state replicated data types,” in Journal of Parallel and Distributed Computing, Volume 111, 2018, Pages 162-173, ISSN 0743-7315, https://doi.org/10.1016/j.jpdc.2017.08.003
- V. Enes, P. Almeida, C. Baquero and J. Leitão, “Efficient Synchronization of State-Based CRDTs,” in 2019 IEEE 35th International Conference on Data Engineering (ICDE), Macao, Macao, 2019 pp. 148-159. doi: 10.1109/ICDE.2019.00022

Topic 1

Asymmetric Merge Functions

- Deltas and classic states are combined by merge functions
- With state growth, deltas become relatively smaller that states
- Merge arguments commute, but assume *merge(big,small)*
- Program merge to be more efficient
- E.g. iterate on the appropriate state
- Build example data types and asymmetric merges
- Measure the complexity/performance gains

Topic 2

Topology Matters

- Deltas can improve data transmission of CRDT states
- Topology of node connections is important
- Ideally topology/overlay formation should be decentralized
- Topology can be static or dynamic
- As long as eventually updates reach all nodes
- Compare different overlay construction strategies
- Measure the impact on data transmission size and speed

Topic 3

Deflation in Isolation

- Mutations in state CRDTs are always inflations
- In PNCounters decrements are separately stored
- Could we increment by deflating increment count?
- Maybe, if we do it very privately
- Keep separate operations or deltas not yet communicated
- Explore mutual cancellations: `inc` vs `dec`; `add(x)` vs `rmv(x)`
- Measure potential metadata size improvements

Practical

Implementing State, Deltas, Join Decompositions, Differences

- You will use your language of choice
- Pick one or more state CRDTs
- Implement them with:
    - Both state and delta mutators
    - A classic join
    - A method/function for textual dumps
    - A method/function for extracting a set with state decompositions
    - A join that accepts a set of states
    - A method/function for computing the state difference