Composition and Decomposition/Fragmenting of CRDTs

Carlos Baquero, Paulo Almeida
HASLab, Minho and INESC Tec

Concordant, Nantes, November 2012

Properties for state convergence
CRDTs and Bloom^L

Carlos Baquero, Paulo Almeida
HASLab, Minho and INESC Tec

- An 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 $\bot$; $\langle S, \leq, \sqcup, \bot \rangle$. $(\forall a \in S, a \sqcup \bot = a)$
- 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, $a \sqcup a = a$,
  - Commutatity, $a \sqcup b = b \sqcup a$,
  - Associative, $(a \sqcup b) \sqcup c = a \sqcup (b \sqcup c)$.
- $\leq$ reflects monotonic state evolution – increase of information.
- Updates must conform to $\leq$.
- In general, queries can return non-monotonic values, and in other domains than S. E.g: Returning a set size.

Abstract State, Concrete State, Ids

- The semilattice relates the abstract states of a CRDT.
- Implementations of CRDTs derive concrete states.
  E.g: Sets are implemented by sequences.
- Several concrete states map to a single abstract state.
- Concrete states can include replica ids and local counters.
- Updates that are static w.r.t $\leq$ can still change concrete states.
- Concrete states are in a pre-order and synch (concrete merge implementation) might not commute.

$A \sqcup B = B \sqcup A$, but allow $a.\mathtt{synch}(b) \neq b.\mathtt{synch}(a)$.

Objects and Literals

Carlos Baquero, Paulo Almeida HASLab, Minho and INESC Tec

- An object has a type T that conforms to a CRDT specification: $T \doteq$ semilattice $\langle S, \leq, \sqcup \rangle$ plus update and query operations.
- A literal is an immutable opaque state with no further structure; a finite bit sequence of known length that is testable for equality.
- Literals can be related in a total order.
- Literals are a special case of CRDTs – constant CRDTs. E.g: $\langle \{\pi\}, =, \texttt{either}, \pi \rangle$ and no update ops

CRDT composition

cartesian product of semilattices

- Let T_0, T_1 be two CRDT types.
- Let $x_0, y_0 : T_0$ and $x_1, y_1 : T_1$ be typed instances:
- Join composition: $(x_0 \times x_1) \sqcup (y_0 \times y_1) \equiv (x_0 \sqcup y_0) \times (x_1 \sqcup y_1)$
- Pointwise order: $(x_0 \times x_1) \leq (y_0 \times y_1) \equiv x_0 \leq y_0$ and $x_1 \leq y_1$

- This generalizes to any finite product/sequence, $T_0 \times \cdots \times T_n$.
- All instances in a given position must match the position type.

CRDT composition

generalizing products as maps

- Let M be a map from literal keys to CRDT objects/instances. $M = \{k_0 \mapsto x_0, \ldots\}$
- The keys are typed such that values for identical keys, in two maps, have identical types:
  if $k_0 \mapsto x_0 \in M_x$ and $k_0 \mapsto y_0 \in M_y$ implies $x_0, y_0 : T_0$.
- Join: Keywise join of values in common keys and union of distinct mappings.
- Order: $x \leq y$ if $keys(x)$ included in $keys(y)$ and $\leq$ in each common key.

Examples are recursive filesystem (or bookmark) tree CRDTs and P-Counters from MaxInt CRDTs. This generalization can subsume composition by cartesian product.

CRDT composition
linear sum

Carlos Baquero, Paulo Almeida
HASLab, Minho and INESC Tec

- Given two ordered disjoint sets $\langle P, \leq_P, \sqcup_P \rangle$ and $\langle Q, \leq_Q, \sqcup_Q \rangle$. Linear sum is denoted $P \oplus Q$.
- If not disjoint can always do disjoint union: $P \uplus Q \equiv \{(p, 0) | \forall p \in P\} \cup \{(q, 1) | \forall q \in Q\}$.
- Make all elements in P ordered as lower than elements in Q.
- Join: $(x_0 \cup x_1) \sqcup (y_0 \cup y_1) \equiv (x_0 \sqcup y_0) \cup (x_1 \sqcup y_1)$
- Order: for $a, b \in P \cup Q$
  $a \leq b \equiv a, b \in P \text{ and } a \leq_P b$ or
  $a, b \in Q \text{ and } a \leq_Q b$ or
  $a \in P \text{ and } b \in Q$.

An example is possibly a 2P-Set where added elements are in P and removed are in Q, with removes dominating adds. In general it is possible to build convergent protocols with a linear order of evolution, E.g. Handoff Counter monotonic protocols.

CRDT composition
lexicographic mapping

Carlos Baquero, Paulo Almeida
HASLab, Minho and INESC Tec

- Pair mapping totally ordered literals to CRDT objects. $k_a \mapsto x_a$
- When joining higher key wins, if equal then join value.
- Join: $k_a \mapsto x_a \sqcup k_b \mapsto x_b \equiv$
  $k_a \mapsto x_a \text{ iff } k_a > k_b$,
  $k_b \mapsto x_b \text{ iff } k_b > k_a$,
  $k_b \mapsto x_a \sqcup x_b \text{ otherwise.}$
- Order: $k_a \mapsto x_a \leq k_b \mapsto x_b \equiv$
  $k_a < k_b \text{ or } (k_a = k_b \text{ and } x_a \leq x_b)$

Examples: Cassandra Counters are maps of site ids to a lexicographic mapping to MaxInt CRDTs. The lexicographic mapping allows the counter value to decrease, by using a higher key.

Invariants on CRDT composition

The Bloom^L CvRDT problem

- After making a composition from basic *fragments*, updates that change several fragments often need to be applied as a group.
- Examples:
  - Bloom^L sets of students and sets of teams.
  - Edge and Vertices dependencies in graph CRDTs.
- Can be addressed by transactions, but there are probably simpler solutions to only address grouping. E.g. Shipping all composed state together and merging together.
- Possibly only the changed fragments of the composed state need sending – using at-least-once reliable channels.

CRDT decomposition (fragmenting)
Decomposing by replica

Carlos Baquero, Paulo Almeida
HASLab, Minho and INESC Tec

- P-Counters (and PN-Counters) are fragmented by replica id.
- Each replica updates a private position.
- Queries report an aggregate, summing all positions.
- Obtained by map compositions and MaxInt objects.
- Cassandra counters are also fragmented by replica id.
- In general no need for grouping across fragments

CRDT decomposition (fragmenting)

Decomposing by element

Carlos Baquero, Paulo Almeida
HASLab, Minho and INESC Tec

- Sets and maps are fragmented by elements and keys.
- In OR-Sets multiple replicas act on a given fragment.
- Tagging by replica based UUIDs tracks causality in each fragment.
- In general no need for grouping across fragments.

- Opt-OR-Sets has two maps: From replica ids and elements.
- I suspect need of grouping of updates on the two maps.

Discussion and Open Questions

- Is there a minimal kernel of composition rules?
- How to obtain lightweight grouping without full SwiftCloud transactions?
- Can a CRDT instance be shared in multiple compositions? Effects on grouping ...
- CRDT hierarchies. E.g: A G-Set can upgrade to a 2P-Set; P-Counter to PN-Counter.
- Are fragments usefull? Less correctness proofs.
- Fragments seem finner grained than grouping.
- Composition and decomposition dual views of the same thing?