Implementing δ-CRDTs in VeriFx

DARE 2023

Diogo Barreto & Diogo Paulico

NOVA School of Science and Technology – FCT NOVA

CRDTs

- Conflict-free Replicated Data Types
  - Allow replicas to update data structures independently, syncing them eventually
  - Can be used in distributed systems that require high availability but can tolerate Eventual Consistency
  - Avoids the need for application-specific synchronization methods

CRDTs: Two designs

- Operation-based
  - Replicas that execute an operation send a message to all replicas that encompasses the operation to transform the old state into new state
  - All other replicas receive the message and apply the operation, generating the new state locally
  - This requires reliable exactly-once broadcast
- State-based
  - Operation that changes state (mutator) is executed on the local replica
  - The mutator returns the new full state which is sent to other replicas
  - This mitigates the need for reliable exactly-once broadcast
  - Communication overhead of shipping entire state

δ-CRDTs

- These mitigate the communication overhead of regular state-based CRDTs
- Replicas only need to propagate a representation of the effects of recent operations
- These representations are the δ between the old (X) and the new state (X'), such that $X' = X \sqcup \delta$
- Replicas update their local state by joining deltas with their own state

δ-CRDTs: Classes

- Anonymous δ-CRDTs:
  - δ-mutators do not use a global replica identifier
  - Example: Grow-only Set (GSet)
- Named δ-CRDTs:
  - Each replica only changes a specific part of the state
  - δ-mutators use a global replica identifier
  - Example: Positive-negative Counter (PNCounter)
- Causal δ-CRDTs:
  - The state of the replica considers causality by relying on *dots*
  - The *dots* related with each operation can be seen as pairs containing a unique replica identifier and the operation number
  - Example: Enable-wins Flag (EWFlag)

Causal CRDTs

- Causal Context:
  - Each event is assigned a unique identifier (a pair) known as *dot*
  - Each dot maintains a unique replica identifier and the event number
  - The set of all dots is known as the Causal Context

- Dot Store:
  - A dot store keeps the relation between the dots and the data maintained by the CRDT
  - Multiple types of Dot Stores are already defined, namely: DotSet, DotFun, and DotMap

Causal CRDTs

- To merge two replicas both the Dot Stores and the Causal Contexts need to be merged
- Causal Context:
  - To merge two replicas' Causal Contexts a union of both contexts should be performed
- Dot Store:
  - The merge of the Dot Stores is dependent on the type of store
    - E.g.: when merging Dot Sets, we need to discard the dots present in only one Dot Set, but whose dot is also present in the Causal Context of the other

Causal CRDTs

$$\mathsf{Causal}\langle T : \mathsf{DotStore}\rangle = T \times \mathsf{CausalContext}$$

$$\sqcup \;:\; \mathsf{Causal}\langle T\rangle \times \mathsf{Causal}\langle T\rangle \rightarrow \mathsf{Causal}\langle T\rangle$$

$$\begin{aligned} \textbf{when} \quad & T : \mathsf{DotSet} \\ (s, c) \sqcup (s', c') = {} & ((s \cap s') \cup (s \setminus c') \cup (s' \setminus c), c \cup c') \end{aligned}$$

$$\begin{aligned} \textbf{when} \quad & T : \mathsf{DotFun}\langle \_ \rangle \\ (m, c) \sqcup (m', c') = {} & (\{k \mapsto m(k) \sqcup m'(k) \mid k \in \mathsf{dom}\, m \cap \mathsf{dom}\, m'\} \cup \\ & \{(d, v) \in m \mid d \notin c'\} \cup \{(d, v) \in m' \mid d \notin c\}, c \cup c') \end{aligned}$$

Causal CRDTs

$$\textsf{Causal}\langle T : \textsf{DotStore}\rangle = T \times \textsf{CausalContext}$$

$$\sqcup \;:\; \textsf{Causal}\langle T\rangle \times \textsf{Causal}\langle T\rangle \to \textsf{Causal}\langle T\rangle$$

$$\begin{aligned} &\textbf{when} \quad T : \textsf{DotSet} \\ (s, c) \sqcup (s', c') &= ((s \cap s') \cup (s \setminus c') \cup (s' \setminus c), c \cup c') \end{aligned}$$

$$\boxed{\begin{aligned} &\textbf{when} \quad T : \textsf{DotFun}\langle\_\rangle \\ (m, c) \sqcup (m', c') &= (\{k \mapsto m(k) \sqcup m'(k) \mid k \in \textsf{dom}\, m \cap \textsf{dom}\, m'\} \cup \\ &\quad \{(d, v) \in m \mid d \notin c'\} \cup \{(d, v) \in m' \mid d \notin c\}, c \cup c') \end{aligned}}$$

In DotFun the principle is the same as in DotSet, i.e., to perform the merge we discard the map entries that are not present in the other Dot Store but whose dot is in both Causal Contexts

VeriFx

- VeriFx Language
  - Designed to simplify the task of designing, implementing and verifying RDTs (Replicated Data Types)
  - High-level programming language, capable of automating proofs
  - Allows verifying real-world implementations rather than formalizations

Implementation

- Our implementation on VeriFx was based on the δ-CRDTs presented and specified in the paper “Delta State Replicated Data Types”[1] as well as the respective C++ implementations[2] (with some adaptations)
- The correctness proofs already available for verifying CRDTs from the DARE examples and exercises were used to validate our implementation

Implemented Causal **CRDTs**

- Enable-wins Flag (EWFlag):
  - A CRDT responsible for maintaining an enable/disable flag across multiple replicas
  - If multiple replicas change the flag concurrently to a distinct state, it should be kept enabled
  - Implemented by relying on a DotSet as DotStore

$$\begin{aligned}
\mathsf{EWFlag} &= \mathsf{Causal}\langle\mathsf{DotSet}\rangle \\
\mathsf{enable}^{\delta}_{i}((s,c)) &= (d, d \cup s) \quad \textbf{where } d = \{\mathsf{next}_i(c)\} \\
\mathsf{disable}^{\delta}_{i}((s,c)) &= (\{\}, s) \\
\mathsf{read}((s,c)) &= s \neq \{\}
\end{aligned}$$

δ-CRDT Enable-wins Flag, as presented in [1]

Implemented Causal **CRDTs**

- Multi-value registry (MVRegistry):
  - If the registry is written concurrently by multiple replicas, all values written can be read after a merge is performed
  - A write operation overwrites all previously written values
  - Implemented by relying on a DotFun as DotStore

$$\begin{aligned}
\mathsf{MVRegister}\langle V\rangle &= \mathsf{Causal}\langle\mathsf{DotFun}\langle V\rangle\rangle \\
\mathsf{write}^{\delta}_{i}(v,(m,c)) &= (\{d \mapsto v\}, \{d\} \cup \mathsf{dom}\, m) \quad \textbf{where } d = \mathsf{next}_i(c) \\
\mathsf{clear}^{\delta}_{i}((m,c)) &= (\{\}, \mathsf{dom}\, m) \\
\mathsf{read}((m,c)) &= \mathsf{ran}\, m
\end{aligned}$$

δ-CRDT Multi-value register, as presented in [1]

Implemented Anonymous CRDTs

- Grow-Only Set (GSet):
  - A replicated set that only supports inserting elements

$$\begin{aligned}\mathsf{GSet}\langle E\rangle &= \mathcal{P}(E)\\ \bot &= \{\}\\ \mathsf{insert}^{\delta}_{i}(e,s) &= \{e\}\\ \mathsf{elements}(s) &= s\\ s \sqcup s' &= s \cup s'\end{aligned}$$

A δ-CRDT grow-only set, as presented in [1]

Implemented Anonymous **CRDTs**

- Two-Phase Set (2PSet):
  - A replicated set that supports inserting and removing elements
  - Comprised of two sets, where one is added elements and the other is removed elements
  - Current set elements are the set difference between these two sets

$$\begin{aligned}
\text{2PSet}\langle E\rangle &= \mathcal{P}(E) \times \mathcal{P}(E) \\
\bot &= (\bot, \bot) \\
\text{insert}_i^{\delta}(e, (s, t)) &= (\{e\}, \bot) \\
\text{remove}_i^{\delta}(e, (s, t)) &= (\bot, \{e\}) \\
\text{elements}((s, t)) &= s \setminus t \\
(s, t) \sqcup (s', t') &= (s \sqcup s', t \sqcup t')
\end{aligned}$$

A δ-CRDT two-phase set, as presented in [1]

Implemented Named **CRDTs**

- Grow-Only Counter (GCounter):
  - A replicated counter that can only be increased
  - Each replica increases an independent counter
  - Total value results from adding value of all replicas

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

A δ-CRDT counter, as presented in [1]

Implemented Named **CRDTs**

- Positive-Negative Counter(PNCounter):
  - A replicated counter that can be increased and decreased
  - Comprised of two GCounters
  - Total value results from subtracting the counter of decreases from the counter of increases

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

A δ-CRDT positive-negative, as presented in [1]

Proprieties to prove

- In δ-CRDTs state evolves by joining the replica's state with a δ, either one generated by itself, or one it received from another replica
- Thus, the join operation must adhere to the following restrictions:
  - If the same δ is joined more than once, the state should remain the same (Idempotence)
  - The order by which δs are joined, should not affect the final state (Commutativity & Associativity)

VeriFx Proof

- The following correctness properties were defined:
  - For all x,y and z that are δ-CRDTs of the same type and whose state is reachable:
    - Joining x with itself must originate x (Idempotence)
    - Joining x with y must generate the same state as joining y with x (Commutativity)
    - Joining x with y and then joining the resulting state with z must originate the same state as joining y with z and then joining the resulting state with x (Associativity)
- VeriFx will then try to find a counterexample that violates the correctness properties, if none is found, the implementation is correct

Results

δ-CRDTs	Type	Correct	Time
DeltaGSet	Anonymous	✓	6.2s
Delta2PSet	Anonymous	✓	6.2s
DeltaGCounter	Named	✓	5.3s
DeltaPNCounter	Named	✓	7.6s
DeltaMVRegister	Causal	✓	6.0s
DeltaEWFlag	Causal	✓	5.9s

Implementation: Github repository

https://github.com/Diogo-Paulico/dare_delta-crdts

References

[1] Paulo Sérgio Almeida, Ali Shoker, Carlos Baquero - Delta State Replicated Data Types, https://doi.org/10.48550/arXiv.1603.01529

[2] https://github.com/CBaquero/delta-enabled-crdts

[3] Kevin De Porre, Carla Ferreira, Elisa Gonzalez Boix - VeriFx: Correct Replicated Data Types for the Masses, https://arxiv.org/abs/2207.02502