Certified Mergeable Replicated Data Types

Kartik Nagar

Joint work with Vimala S, Adharsh Kamath and KC Sivaramakrishnan.

*To appear in PLDI 22.*
Outline

• Introduction
  • Replicated Systems
  • MRDT
• MRDT Verification Problem
• Our proposed technique
• Example
• Experimental Results
• Conclusion
Replicated Systems

• Creating multiple replicas of data, independently operated, potentially geo-distributed.

• Many benefits
  • Fault Tolerance
  • Availability
  • Low latency for geo-distributed clients

• How to safely write applications for replicated systems?
  • Programming would be easier if it appears as a single, ‘centralized’ system.
  • Unfortunately, this incurs massive synchronization cost.

• Instead, we have a library of basic replicated data types with slightly ‘weaker’ semantics.
Mergeable Replicated Data Types (MRDTs)

- Version-control model of replication.
- Three-way merge function – 2 concurrent versions and their Lowest Common Ancestor (LCA) between them.
- Consider the counter MRDT:

\[
\text{merge lca } v_1 v_2 = \text{lca} + (v_1 - \text{lca}) + (v_2 - \text{lca})
\]
Set RDT

Desired specification: Add wins
Observed-Remove Set MRDT

Implementation

\[ D_T = (\Sigma, \sigma_0, do, merge) \]

1: \( \Sigma = \mathcal{P}(\mathbb{N} \times \mathbb{N}) \)
2: \( \sigma_0 = \{\} \)
3: \( do(rd, \sigma, t) = (\sigma, \{a \mid (a, t) \in \sigma\}) \)
4: \( do(add(a), \sigma, t) = (\sigma \cup \{(a, t)\}, \bot) \)
5: \( do(remove(a), \sigma, t) = (\{e \in \sigma \mid \text{fst}(e) \neq a\}, \bot) \)
6: \( merge(\sigma_{lca}, \sigma_a, \sigma_b) = \)
   \( (\sigma_{lca} \cap \sigma_a \cap \sigma_b) \cup (\sigma_a - \sigma_{lca}) \cup (\sigma_b - \sigma_{lca}) \)
Observed-Remove Set Specification

Abstract state

\[ I = \langle E, \text{oper}, \text{rval}, \text{time}, \text{vis} \rangle \]

- \( E \) is the set of events
- \( \text{oper}: E \rightarrow Op \)
- \( \text{rval}: E \rightarrow Val \)
- \( \text{time}: E \rightarrow \mathbb{N} \)
- \( \text{vis} \subseteq E \times E \)
Visibility Relation in Abstract State

\[ e_1 \xrightarrow{vis} e_2 \]
Observed-Remove Set Specification

Abstract state

\[ I = \langle E, \text{oper}, \text{ rval}, \text{ time}, \text{ vis} \rangle \]

- \( E \) is the set of events
- \( \text{ oper}: E \rightarrow \text{ Op} \)
- \( \text{ rval}: E \rightarrow \text{ Val} \)
- \( \text{ time}: E \rightarrow \mathbb{N} \)
- \( \text{ vis} \subseteq E \times E \)

Specification

\[ \mathcal{F}_{orset} (\text{rd}, \langle E, \text{ oper}, \text{ rval}, \text{ time}, \text{ vis} \rangle) = \{ a \mid \exists e \in E. \, \text{oper}(e) \}
\]

\[ = \text{add}(a) \land \neg (\exists f \in E. \, \text{oper}(f) = \text{remove}(a) \land e \xrightarrow{\text{vis}} f) \} \]
The problem

Implementation                              Specification

1: \( \Sigma = \mathcal{P}(\mathbb{N} \times \mathbb{N}) \)
2: \( \sigma_0 = \{\} \)
3: \( \text{do}(\text{rd}, \sigma, t) = (\sigma, \{a \mid (a, t) \in \sigma\}) \)
4: \( \text{do}(\text{add}(a), \sigma, t) = (\sigma \cup \{(a, t)\}, \bot) \)
5: \( \text{do}(\text{remove}(a), \sigma, t) = (\{e \in \sigma \mid \text{fst}(e) \neq a\}, \bot) \)
6: \( \text{merge}(\sigma_{lca}, \sigma_a, \sigma_b) = \left((\sigma_{lca} \cap \sigma_a \cap \sigma_b) \cup (\sigma_a - \sigma_{lca}) \cup (\sigma_b - \sigma_{lca})\right) \)

\( F_{orset}(\text{rd}, \langle E, \text{oper}, \text{rval}, \text{time}, \text{vis}\rangle) = \{a \mid \exists e \in E. \text{oper}(e) = \text{add}(a) \land \neg(\exists f \in E. \text{oper}(f) = \text{remove}(a) \land e \xrightarrow{vis} f)\} \)

1. Does the implementation satisfy the specification?
2. Does the implementation ensure convergence?
   • Two replicas which have witnessed the same set of events must have the same state.
Our Contributions

• We propose a simulation-based verification procedure for showing functional correctness and convergence for MRDTs.

• We mechanize and automate the complete verification process using F*.

• We propose a new, weaker notion of convergence modulo observable behavior which permits more efficient MRDT implementations.

• We have built a library of efficient and verified MRDTs for common data structures such as set, map, queue, flag, etc.
Replication-aware Simulation Relation \(^1\)

- \(\mathcal{R}_{\text{sim}}(I, \sigma)\) relates an abstract state \(I\) with concrete state \(\sigma\).
  - \(\mathcal{R}_{\text{sim}}\) is the glue relating the concrete and abstract states, as well as the implementation and specification

- Verification using \(\mathcal{R}_{\text{sim}}\) is done in two steps:
  1. We show that \(\mathcal{R}_{\text{sim}}\) holds in all executions in an inductive fashion.
  2. We show that \(\mathcal{R}_{\text{sim}}\) is sufficient to discharge the specification and convergence requirements.

---

OR-Set MRDT Simulation Relation

\[ R_{sim}(I, \sigma) \iff (\forall a, t. (a, t) \in \sigma \iff \\
(\exists e \in I.E \land I.\text{oper}(e) = \text{add}(a) \land I.\text{time}(e) = t \land \\
\neg(\exists f \in I.E \land I.\text{oper}(f) = \text{remove}(a) \land e \xrightarrow{\text{vis}} f))) \]
Verification using $R_{sim}$: Step-1

We show that $R_{sim}$ holds inductively at every step in every execution

1. Verifying operations

2. Verifying merge
Verification using $\mathcal{R}_{sim}$: Step-2

We show that $\mathcal{R}_{sim}$ is sufficient to prove specification and convergence

3. Verifying specification

4. Verifying convergence
Store Properties

\( \Psi_{ts} \) asserts increasing timestamps according to the visibility relation

\[
\Psi_{ts}(I) \quad \forall e, e' \in I.E. \ e \xrightarrow{I.vis} e' \Rightarrow I.time(e) < I.time(e') \\
\quad \land \forall e, e' \in I.E. \ I.time(e) = I.time(e') \Rightarrow e = e'
\]

\( \Psi_{lca}(I_l, I_a, I_b) \quad I_l.E = I_a.E \cap I_b.E \\
\quad \land I_l.vis = I_a.vis |_{I_l.E} = I_b.vis |_{I_l.E} \)

\( \Psi_{lca} \) asserts that events in LCA are present in both the branches, with the same visibility relation

We assume the store properties while proving \( \mathcal{R}_{sim} \)
Example: Verifying $R_{sim}$ for OR-Set MRDT

Simulation Relation:

$$R_{sim}(I, \sigma) \iff (\forall a, t. (a, t) \in \sigma \iff (\exists e \in I.E \land I. oper(e) = add(a) \land I. time(e) = t \land 
\neg(\exists f \in I.E \land I. oper(f) = remove(a) \land e \xrightarrow{vis} f)))$$

Certified Mergeable Replicated Data Types
Example: Verifying $R_{\text{sim}}$ for OR-Set MRDT

Simulation Relation:

$$R_{\text{sim}}(I, \sigma) \iff (\forall a, t. (a, t) \in \sigma \iff (\exists e \in I.E \land I.\text{oper}(e) = \text{add}(a) \land I.\text{time}(e) = t \land \neg(\exists f \in I.E \land I.\text{oper}(f) = \text{remove}(a) \land e \xrightarrow{\text{vis}} f)))$$

Store Property:

| $\Psi_{\text{lca}}(I_l, I_a, I_b)$ | $I_l.E = I_a.E \cap I_b.E \land I_l.\text{vis} = I_a.\text{vis}|_{I_l.E} = I_b.\text{vis}|_{I_l.E}$ |
|----------------------------------|------------------------------------------------------|
Efficient OR-Set implementations

**Space-efficient version**
- Keeps a single version of an element
- Otherwise, it is the same as the original OR-Set MRDT.

**Space & time-efficient version**
- Stores the set internally as a Binary Search Tree instead of a list
- Much better performance for $rd$ queries.
- We can only guarantee convergence modulo observable behavior.

![Diagram showing efficient OR-Set implementations](image-url)
## Peepul: Library of Verified MRDTs in F*

<table>
<thead>
<tr>
<th>MRDTs verified</th>
<th>#Lines code</th>
<th>#Lines proof</th>
<th>#Lemmas</th>
<th>Verif. time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Increment-only counter</td>
<td>6</td>
<td>43</td>
<td>2</td>
<td>3.494</td>
</tr>
<tr>
<td>PN counter</td>
<td>8</td>
<td>43</td>
<td>2</td>
<td>23.211</td>
</tr>
<tr>
<td>Enable-wins flag</td>
<td>20</td>
<td>58</td>
<td>3</td>
<td>1074</td>
</tr>
<tr>
<td></td>
<td></td>
<td>81</td>
<td>6</td>
<td>171</td>
</tr>
<tr>
<td></td>
<td></td>
<td>89</td>
<td>7</td>
<td>104</td>
</tr>
<tr>
<td>LWW register</td>
<td>5</td>
<td>44</td>
<td>1</td>
<td>4.21</td>
</tr>
<tr>
<td>G-set</td>
<td>10</td>
<td>23</td>
<td>0</td>
<td>4.71</td>
</tr>
<tr>
<td></td>
<td></td>
<td>28</td>
<td>1</td>
<td>2.462</td>
</tr>
<tr>
<td></td>
<td></td>
<td>33</td>
<td>2</td>
<td>1.993</td>
</tr>
<tr>
<td>G-map</td>
<td>48</td>
<td>26</td>
<td>0</td>
<td>26.089</td>
</tr>
<tr>
<td>Mergeable log</td>
<td>39</td>
<td>95</td>
<td>2</td>
<td>36.562</td>
</tr>
<tr>
<td>OR-set ($\S 2.1.1$)</td>
<td>30</td>
<td>36</td>
<td>0</td>
<td>43.85</td>
</tr>
<tr>
<td></td>
<td></td>
<td>41</td>
<td>1</td>
<td>21.656</td>
</tr>
<tr>
<td></td>
<td></td>
<td>46</td>
<td>2</td>
<td>8.829</td>
</tr>
<tr>
<td>OR-set-space ($\S 2.1.2$)</td>
<td>59</td>
<td>108</td>
<td>7</td>
<td>1716</td>
</tr>
<tr>
<td>OR-set-spacetime</td>
<td>97</td>
<td>266</td>
<td>7</td>
<td>1854</td>
</tr>
<tr>
<td>Queue</td>
<td>32</td>
<td>1123</td>
<td>75</td>
<td>4753</td>
</tr>
</tbody>
</table>
Verified Queue MRDT

At-least-once dequeue semantics
Specification of the Queue MRDT

\[\text{match}_I(e_1, e_2) \iff I.\text{oper}(e_1) = \text{enqueue}(a)\]
\[\land I.\text{oper}(e_2) = \text{dequeue} \land a = I.\text{rval}(e_2)\]

- **AddRem** \((I)\) : \(\forall e \in I.E. I.\text{oper}(e) = \text{dequeue} \land I.\text{rval}(e) \neq \text{EMPTY} \implies \exists e' \in I.E. \text{match}_I(e', e)\)

- **Empty** \((I)\) : \(\forall e_1, e_2, e_3 \in I.E. I.\text{oper}(e_1) = \text{dequeue} \land I.\text{rval}(e_1) = \text{EMPTY} \land I.\text{oper}(e_2) = \text{enqueue}(a) \land e_2 \xrightarrow{\text{I.vis}} e_1 \implies \exists e_3 \in I.E. \text{match}_I(e_2, e_3) \land e_3 \xrightarrow{\text{I.vis}} e_1\)

- **FIFO** \(_1\) \((I)\) : \(\forall e_1, e_2, e_3 \in I.E. I.\text{oper}(e_1) = \text{enqueue}(a) \land \text{match}_I(e_2, e_3) \land e_1 \xrightarrow{\text{I.vis}} e_2 \implies \exists e_4 \in I.E. \text{match}_I(e_1, e_4)\)

- **FIFO** \(_2\) \((I)\) : \(\forall e_1, e_2, e_3, e_4 \in I.E. \lnot (\text{match}_I(e_1, e_4) \land \text{match}_I(e_2, e_3) \land e_1 \xrightarrow{\text{I.vis}} e_2 \land e_3 \xrightarrow{\text{I.vis}} e_4)\)
Merge performance of Peepul and Quark\textsuperscript{1} Queues

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{merge-performance.png}
\caption{Comparison of merge performance between Peepul and Quark Queues.}
\end{figure}

\textsuperscript{1} Kaki et. al. Mergeable Replicated Data Types. OOPSLA 19
Performance of different OR-Sets

Table 3.

| MRDTs | 
|-------|-------|
| verif. eort. | 
| #Lines code | #Lines proof | #Lemmas | Verif. time (s) |
| Increment-only counter | 6 | 43 | 2 | 3.494 |
| PN counter | 8 | 43 | 2 | 23.211 |
| Enable-wins | 20 | 58 | 3 | 1074 |
| | 81 | 6 | 171 |
| | 89 | 7 | 104 |
| LWW register | 5 | 44 | 1 | 4.21 |
| G-set | 10 | 23 | 0 | 4.71 |
| | 28 | 1 | 2.462 |
| | 33 | 2 | 1.993 |
| G-map | 48 | 26 | 0 | 26.089 |
| Mergeable log | 39 | 95 | 2 | 36.562 |
| OR-set (§ 2.1.1) | 30 | 36 | 0 | 43.85 |
| | 41 | 1 | 21.656 |
| | 46 | 2 | 8.829 |
| OR-set-space (§ 2.1.2) | 59 | 108 | 7 | 1716 |
| OR-set-spacetime | 97 | 266 | 7 | 1854 |
| Queue | 32 | 1123 | 75 | 4753 |

Figure 13.

The impact of duplicate elements, we perform an experiment similar to the queue one except that we pick a 50:50 split between add and remove operations. The values added are randomly picked in the range (0:1000). For MRDTs, we pick the space-optimized OR-set (OR-set-space). We report the number of elements in the final set including duplicates. The results are presented in figure 13. Due to the duplicates, the size of the Quark set increases with an increasing number of operations; the growth is not linear due to the stochastic interplay between add and remove. For MRDTs, the set size always remains below 1000 which is the range of the values picked. The results show that MRDTs in MRDTs are much more efficient than in Quark.

7.2.2 MRDT OR-set Performance.

We also compare the overall performance of the three OR-set implementations.

Figure 14.

Running time of OR-sets.

Figure 15.

Space consumption of OR-sets. The OR-set-space line is hidden by the OR-set-spacetime line.
Compositionality

• Generic $\alpha$-map which can be instantiated with any element type $\alpha$.

• Specification of $\alpha$-map uses the specification of $\alpha$ applied to every key.

• We prove the correctness of $\alpha$-map assuming the correctness of $\alpha$.

• We get a whole family of verified map MRDTs!

$$\mathcal{F}_{\alpha\text{-map}}(\text{get}(k, o_\alpha), I) =$$

let $I_\alpha = \text{project}(k, I)$ in $\mathcal{F}_{\alpha}(o_\alpha, I_\alpha)$

$$\mathcal{D}_{\alpha\text{-map}} = (\Sigma, \sigma_0, do, \text{merge}_{\alpha\text{-map}})$$ where

1: $\Sigma_{\alpha\text{-map}} = \mathcal{P}(\text{string} \times \Sigma_\alpha)$

2: $\sigma_0 = \{\}$

3: $\delta(\sigma, k) = \begin{cases} 
\sigma(k), & \text{if } k \in \text{dom}(\sigma) \\
\sigma_0_\alpha, & \text{otherwise}
\end{cases}$

4: $do(set(k, o_\alpha), \sigma, t) =$

let $(v, r) = do_\alpha(o_\alpha, \delta(\sigma, k), t)$ in $(\sigma[k \mapsto v], r)$

5: $do(get(k, o_\alpha), \sigma, t) =$

let $(_, r) = do_\alpha(o_\alpha, \delta(\sigma, k), t)$ in $(\sigma, r)$

6: $\text{merge}_{\alpha\text{-map}}(\sigma_{\text{ica}}, \sigma_a, \sigma_b) =$

$\{ (k, v) \mid (k \in \text{dom}(\sigma_{\text{ica}}) \cup \text{dom}(\sigma_a) \cup \text{dom}(\sigma_b)) \land$

$v = \text{merge}_\alpha(\delta(\sigma_{\text{ica}}, k), \delta(\sigma_a, k), \delta(\sigma_b, k))$}

$\mathcal{R}_{\text{sim-}\alpha\text{-map}}(I, \sigma) \iff \forall k.$

1: $(k \in \text{dom}(\sigma) \iff \exists e \in I.E. \text{ oper}(e) = \text{set}(k, _)) \land$

2: $\mathcal{R}_{\text{sim-}\alpha}(\text{project}(k, I), \delta(\sigma, k))$
Conclusion and Future Work

• We have proposed a technique to verify both the functional correctness and convergence of MRDTs.
• We have successfully applied our technique on a number of challenging MRDTs.
• Our technique supports verification of efficient implementations, as well as compositionality through parametric polymorphism.
• Future work: Applying our technique on more complex MRDTs (e.g. JSON Automerge MRDT)
• Future work: Improve automation
Thank You

Questions?