CRDV: Conflict-free Replicated Data Views

Nuno Faria, José Pereira | nuno.f.faria@inesctec.pt, jop@di.uminho.pt | INESCTEC & University of Minho

SIGMOD 2025

Motivation

- Conflict-free Replicated Data Types are commonly used to model distributed data, as they guarantee convergence of replicas using expressive resolution rules.
- Due to their object-based model, adoption in distributed relational systems is not straightforward. Current approaches include:
  - Embedding **blobs** and using **custom code** to read/write – supports many types and rules but does not integrate with the query language and optimizer.
  - Modeling **tables as convergent maps** – compatible with the relational model but limited in the types and rules supported, often just *last-writer-wins*.

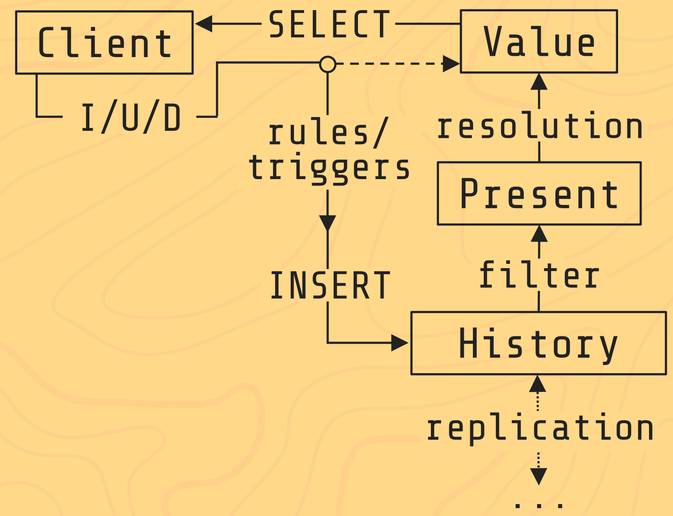

Figure 1: CRDV architecture.

Conflict-free Replicated Data Views

- Brings convergent replicated data to the relational model using **native features** such as views, rules/triggers, and asynchronous replication.
- Seamlessly integrates with the **query engine** and the **local transactional isolation**, while supporting **complex data types** and **conflict resolution rules**.

Architecture

- **History** – stores a log of local and remote writes, using a replicated table.
- **Present** – filters obsolete *History* rows with a view and optional materialization.
- **Value** – handles concurrent conflicting versions in the causal present, using views expressing conflict resolution rules.

Writing

- Writes to *Value* are redirected as Inserts to *History*, with rules/triggers, containing the updated data and additional metadata.
- On commit, rows are asynchronously replicated to the other sites.

```
CREATE RULE update_rule AS
 ON UPDATE TO Value DO INSTEAD
 INSERT INTO History
 SELECT k, v, 'add', siteId(), t.lts, t.pts
 FROM nextTimestamp() AS t;
```

Figure 2: Example of an Update rule (data: *k*, *v*).

Filtering

- *Present* removes obsolete versions based on causality, with vector clocks. Three options:
  - View that filters *History* at runtime (*no-mat)*.
  - Materialization with a table (*sync*).
  - View that combines recent writes with a materialized snapshot (*async*).

Reading

- *Value* views take the *Present* data and apply conflict resolution rules.

Present					
k	**v**	**op**	**site**	**lts**	**pts**
k1	1	add	1	[10,1]	(123,1)
k2		rmv	1	[13,1]	(125,1)
k3	3	add	1	[14,1]	(132,1)
k3		rmv	2	[9,2]	(130,1)
k4	4	add	1	[15,1]	(129,1)
k4	40	add	2	[9,3]	(132,1)

```
CREATE VIEW ValueAw AS
 SELECT k, v
 FROM Present
 WHERE op = 'add';
```

k	v
k1	1
k3	3
k4	4
k4	40

k	v
k1	1
k3	3
k4	22

```
CREATE VIEW ValueAwAvg AS
 SELECT k, avg(v)
 FROM ValueAw
 GROUP BY k;
```

Figure 3: Reading in CRDV.

Selected Results

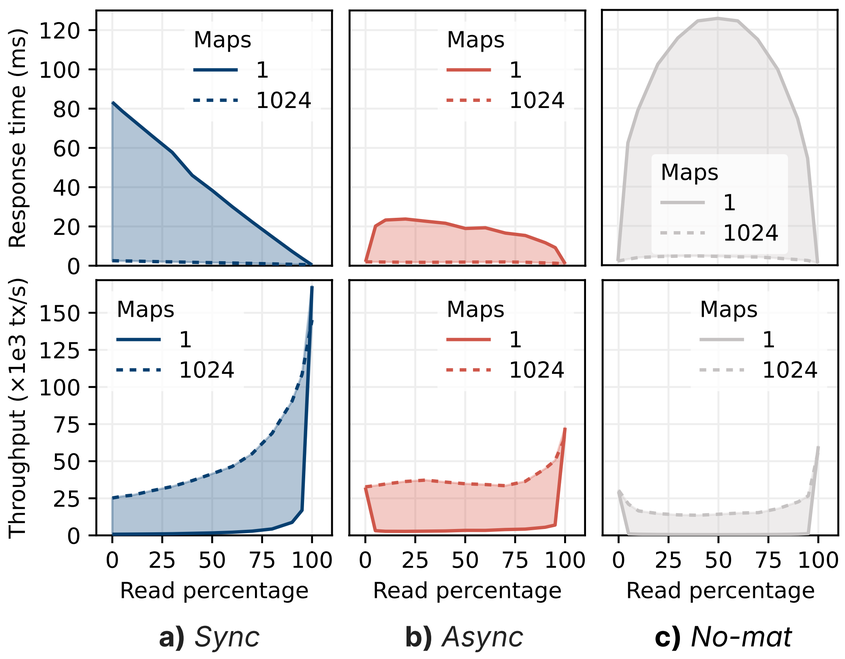

Figure 4: Comparison of materialization strategies.

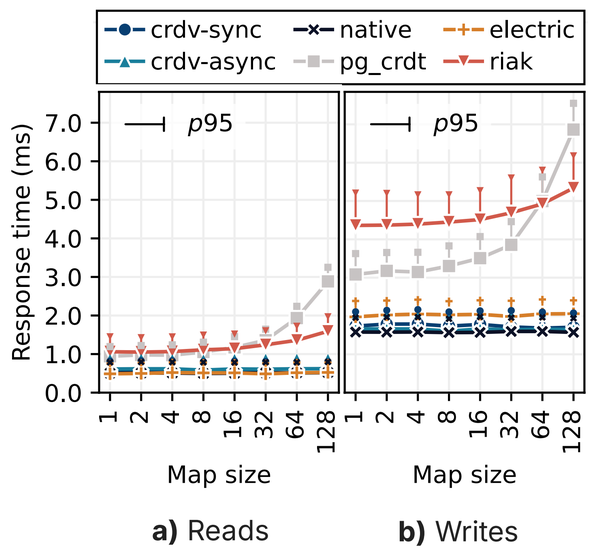

Figure 5: Read and write latency.

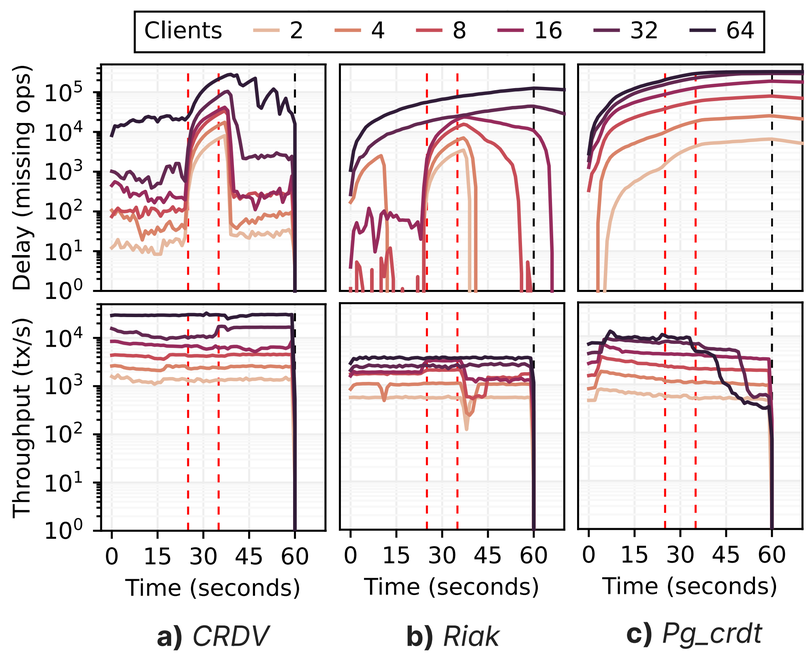

Figure 6: Delay and throughput (3 sites).

INESCTEC

fct Fundação para a Ciência e a Tecnologia

This work is financed by National Funds through the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia, within project LA/P/0063/2020. DOI 10.54499/LA/P/0063/2020