A Comprehensive Study of CRDTs

Marc Shapiro, Nuno Preguiça, Carlos Baquero, Marek Zawirski

Presented by :
Devashish Purandare
UC Santa Cruz
Oct 22nd 2018

Formal Definitions

- Process is made up of atoms and objects
- Atom : immutable, identified by content, lower case
- Object : mutable, replicated, Capitalized
- Clients perform operations by querying and modifying object state

State Based vs Operation Based

- Updates happen at source
- Modified payload is sent to replicas
- The system transmits state to other replicas

- System transmits operations
- Payload and initial operations are state based
- Updates cannot have side effects but can compute results
- They are communicated to replicas asynchronously
- Executed locally at source

CvRDTs vs CmRDTs

- State Based CRDT
- Object takes values in a semilattice, after merge(x,y)
- Merge is idempotent and commutative
- The set advances towards a Least Upper Bound (LUB)

- Needs reliable update mechanism that delivers all updates at every replica
- Order of operations can be specified
- If updates are concurrent, they have to commute
- Causal delivery is sufficient

Basic CRDTs

- Counters
- Registers
- Sets
- Graphs

Counters: Op-based

Specification 5 op-based Counter

```
payload integer i
    initial 0
query value () : integer j
    let j = i
update increment ()
    downstream ()                ▷ No precond: delivery order is empty
        i := i + 1
update decrement ()
    downstream ()                ▷ No precond: delivery order is empty
        i := i − 1
```

Counters: State Based Increment Only

- G-Counter : increment only counter
- State based counters can have issues
- Merge can converge on the wrong value, or not be idempotent
- Solution, vector clocks!
- Payload is a vector of integers, with each replica having an entry
- Count is the sum of all entries

Modifications to Counters

- PN Counter :
- Combines 2 G-Counters as a P (increment) and N (decrement)
- Non Negative Counter :
- A client cannot create more decrements than increments (strong condition)

Registers

- Register is a memory cell storing an opaque atom or object
- Assign to update
- Value to query
- Non concurrent assigns have to be sequentially ordered
- If non concurrent updates don’t commute, we need to resolve it
- We pick a winner (LWW) or both get to stay (MV)

Last-Writer-Wins Register

- The total order is decided by the timestamp
- Timestamps are unique, totally ordered and causal
- Assign generates a new timestamp
- Operation happens only if downstream timestamp is lower than the operation

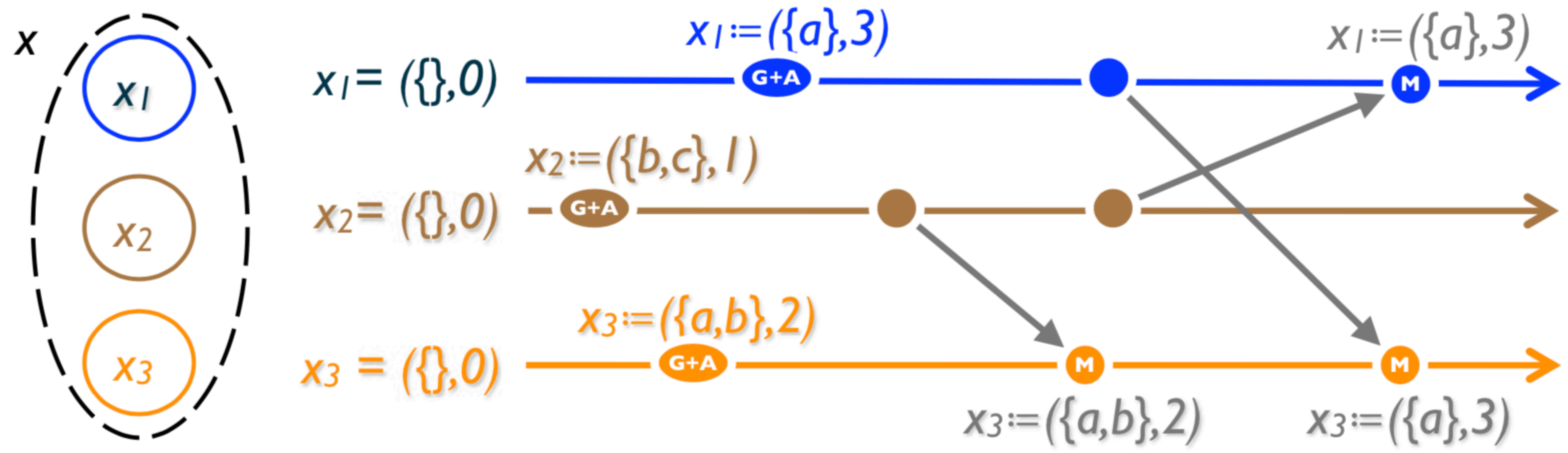

Figure 8: LWW-Set (state-based). Payload is a pair (set, timestamp)

MV Register

- Merges values by taking a union
- Clients can then reduce them to a single value if required
- Payload is a set of (X, versionVector)
- Assign computes a version vector that dominates other version vectors

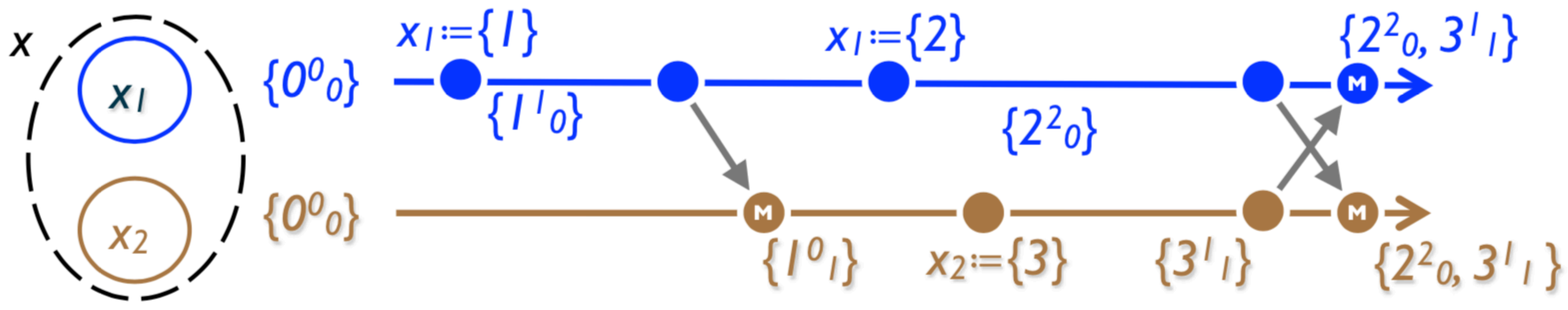

Figure 9: MV-Register (state-based)

Growth Only Set

- Supports add and lookup operations

Specification 11 State-based grow-only Set (G-Set)

payload set A
    initial $\varnothing$
update add (element e)
    $A := A \cup \{e\}$
query $lookup$ (element e) : boolean b
    let $b = (e \in A)$
compare (S, T) : boolean b
    let $b = (S.A \subseteq T.A)$
merge (S, T) : payload U
    let $U.A = S.A \cup T.A$

Two Phase - 2P Set

- Element can be added and removed, but cannot be added again
- Combines 2 G-sets for adding and removing
- Since G-Sets don't support deletion, object cannot be re-added

2P Set Implementations

- State Based
  - Payload is composed of two sets, A and R.
  - LUB is the union
- Op-Based
  - Concurrent adds and removes commute
- U Set
  - U enforces every element to be unique, and ensures that add(e) is delivered before remove (e)

LWW-element-set

- LWW attaches a timestamp to each element
- Add set A and Remove set R contain (element, timestamp) pairs
- Merging takes unions of add and remove sets

PN-Set

- Similarly, we can associate a counter to each element
- Adding an element increments the associated counter
- Removing decrements it
- Element is in the set if the counter is positive
- While PN-set converges, there can be cases where add has no effect

Observe-Remove Set

- Supports adding and removing elements
- Each added element is tagged uniquely
- While removing, unique tags observed at source are removed

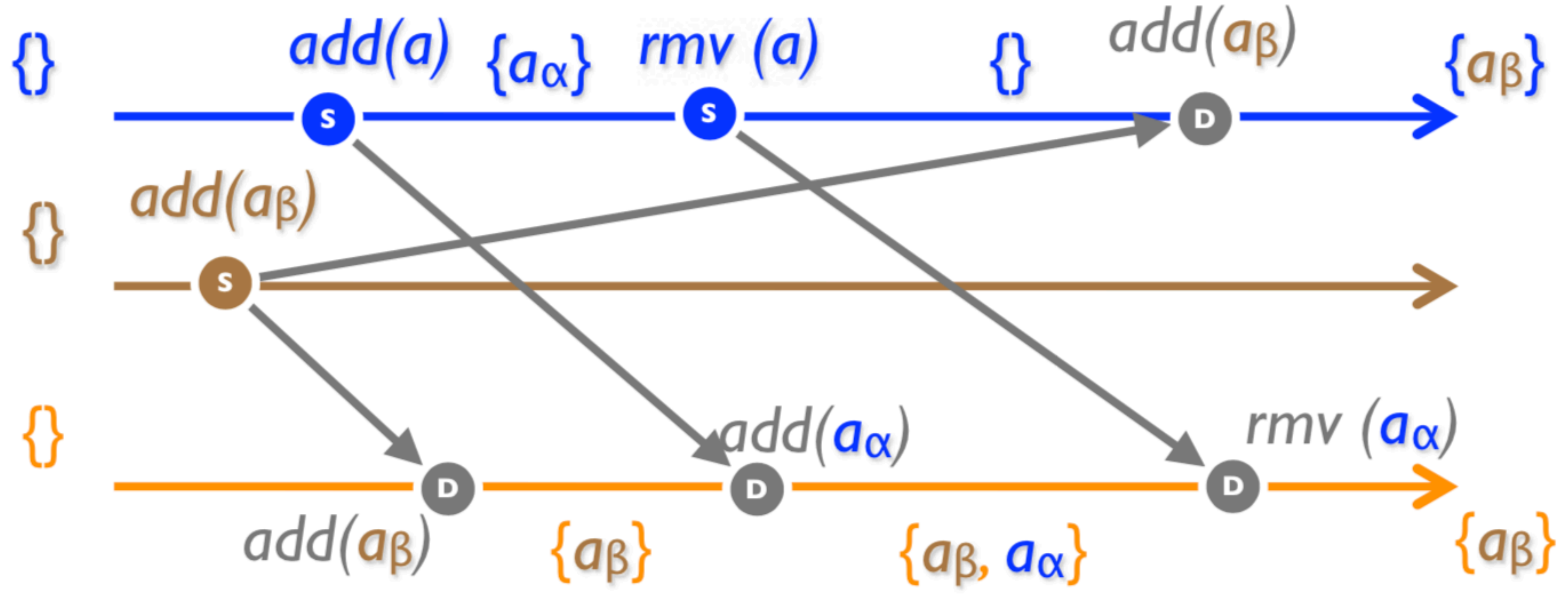

Figure 14: Observed-Remove Set (op-based)

Graphs

- Maintaining a tree or a DAG cannot be done by a CRDT since it requires a global invariant, which is impossible to determine without synchronization

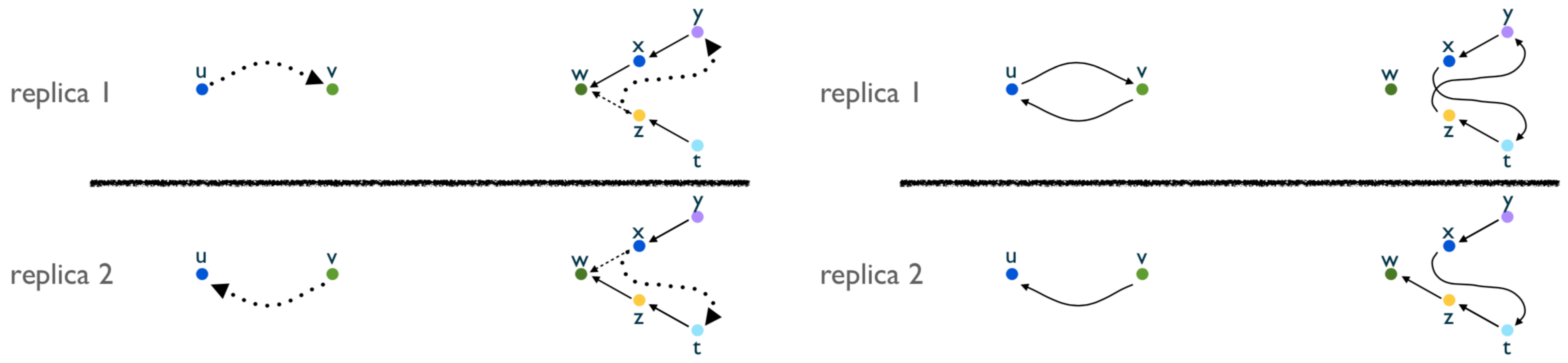

Figure 15: Maintaining strong properties in a graph (counter-example). Left: initial state and update (dashed edges removed, dotted edges added); right: final state.

Monotonic DAGs

- However, certain local conditions can be laid out, such as directionality and monotonicity to allow CRDTs
- DAG is initialised with left and right sentinels $\vdash$ and $\dashv$ and edge $(\vdash, \dashv)$.
- Concurrent addEdge is either independent or idempotent
- However removals can be problematic

Cooperative Text Editing and RGAs

- Users often collaborate on text editing : google docs
- Users can add/remove text
- Users can go offline
- CRDTs can ensure that edits always converge and never conflict, even if users are offline

Replicated Growable Array

- Implements a sequence as a linked list
- AddRight(a,v) adds an element to the right of v
- If inserts happen twice at the same position
  - AddRight(a,v);AddRight(v,b) it adds b to the left of a with higher timestamp
- This is a subclass of Add-Remove partial order

Class Questions and Conclusions

- What is the cost of implementing CRDTs? Are they too expensive?
- Are the few implementations of counters/ registers and sets exhaustive?
- Have better CRDTs been explored?
- What are some practical examples?