Conflict Free Replicated Data Types

Distributed Systems Spring 2020

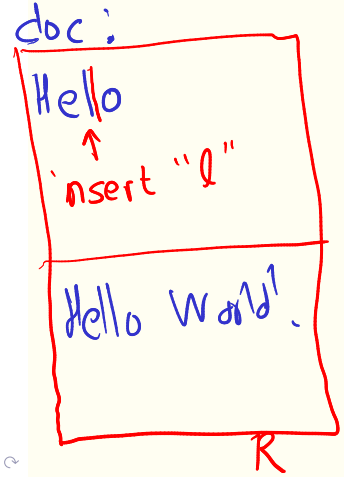
Lecture 16

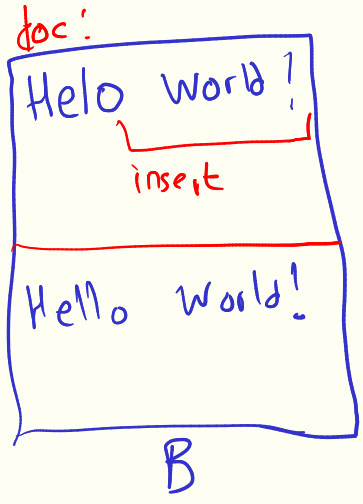
Strong Eventual Consistency

- Two nodes can receive the same set of updates in a different order
- Their view of the shared state is identical
  - No Conflicts!
- Any conflicting updates are merged automatically

- Possible to implement without centralized server/leader
- Local execution can proceed without waiting for other processes
- Intermittent connectivity useful for mobile devices, offline operations, etc

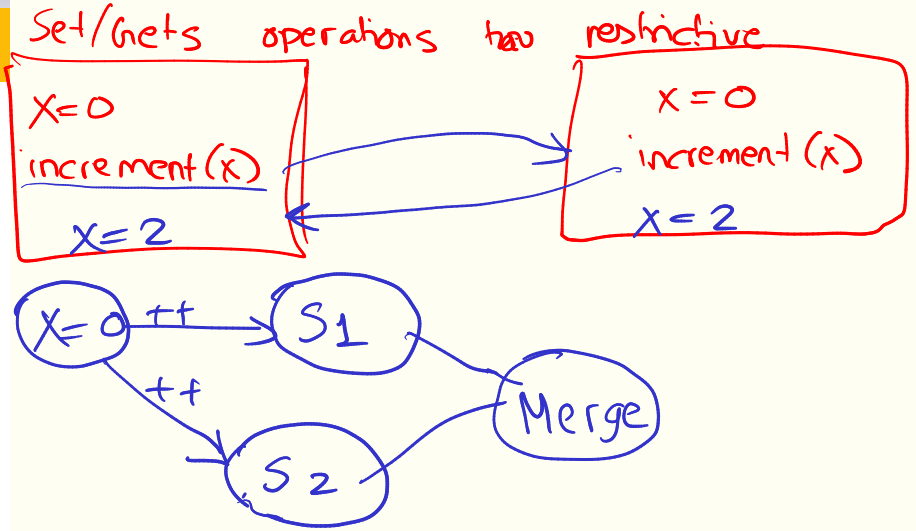
How do collaborative editors like Google Doc work?





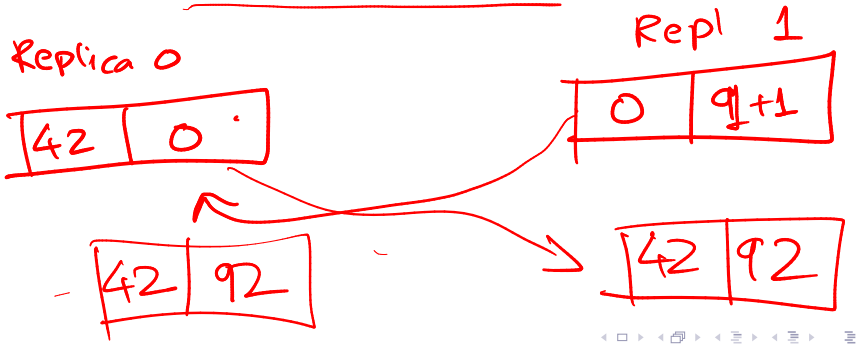
CRDTs

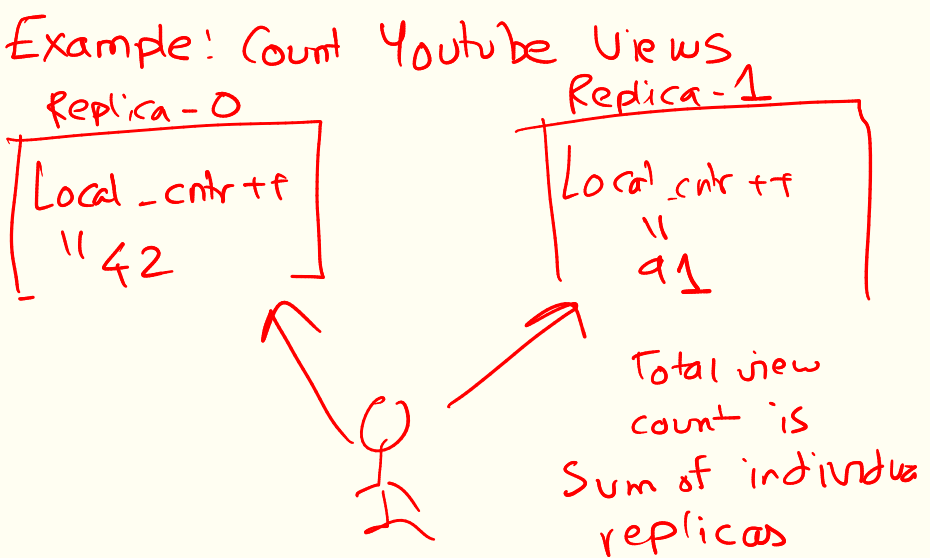
- (Conflict-free/Commutative/Convergent) Replicated Data Types
- Key idea: Use commutative operations for state convergence
- Provably converge to a sequentially consistent result
- States should form a "semi-lattice"
  - Partial order, and Least Upper Bound always exists
  - Values of different replicas *merged* using lattice-join operation
  - Can be implemented with Last-writer-wins, but instead:
  - **CRDT's allow using values and not causal-histories to merge**
- State vs. Operation-based CRDT



Event counting with CRDT's

- Simple growth counter (increment operations are commutative)
- All writes (updates) made only to local copy/version
- Each replica maintains a vector of values $V[1..N]$
  - Similar to vector clock timestamps
- Value of the counter is sum of all local values ($\sum V$)
- Merge is pairwise max of the two vectors

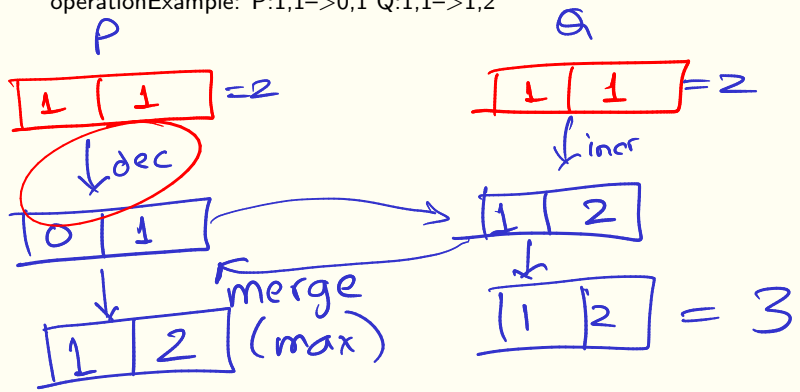




General Counters with CRDT's

- CRDT's work when the states are monotonically increasing
- Value of object itself used for merging, no timestamps needed
- What if we want to support increment and decrement ops?
  - State (value) can decrease, and hence not monotonic!

Since we take max, we dont get sequential consistency if there is a decrement operationExample: P:1,1–>0,1 Q:1,1–>1,2



General Counters with CRDT's

- CRDT's work when the states are monotonically increasing
- Value of object itself used for merging, no timestamps needed
- What if we want to support increment and decrement ops?
  - State (value) can decrease, and hence not monotonic!
- Maintain two monotonically increasing counters
- One (P) for increments, and another (N) for decrements
- Value is $\sum P - \sum N$

Since we take max, we dont get sequential consistency if there is a decrement operationExample: P:1,1–>0,1 Q:1,1–>1,2

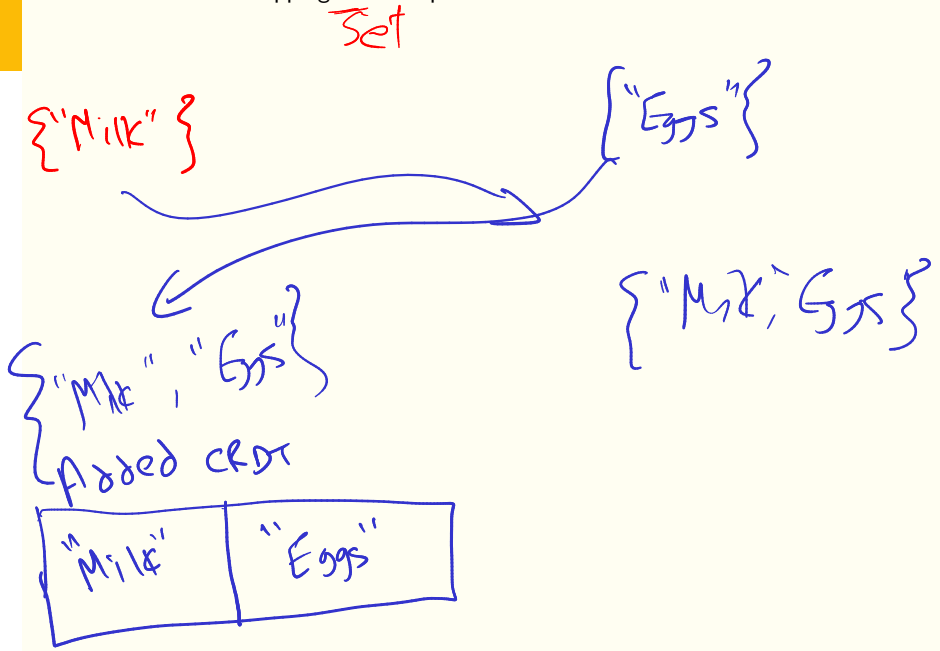
Sets with CRDT's

- Parition-tolerant sets: One each for added and deleted items
- Merge operation is the set union operation
- Actual set is Added−Deleted.
- Can prune sets by applying delete operations, when system is unpartitioned

optimization

Merge: Set Union

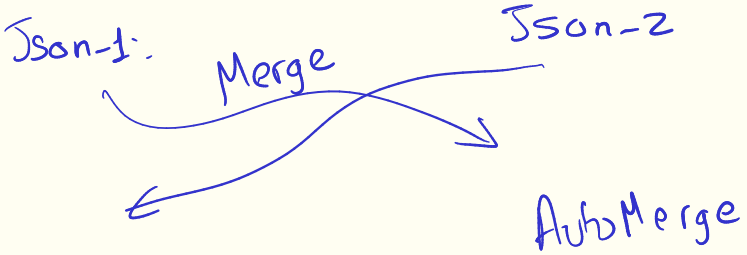
Collaborative shopping list example



Sets with CRDT's

- Parition-tolerant sets: One each for added and deleted items
- Merge operation is the set union operation
- Actual set is Added−Deleted.
- Can prune sets by applying delete operations, when system is unpartitioned
- Deep result: Combining two CRDT's results is still a CRDT
- CRDT's can be combined to form complex CRDT's
  - Sets, maps, counters, graphs, sequences
  - JSON objects
- Popular usecase: Collaborative document editing

Collaborative shopping list example



Ordered Lists with CRDT's

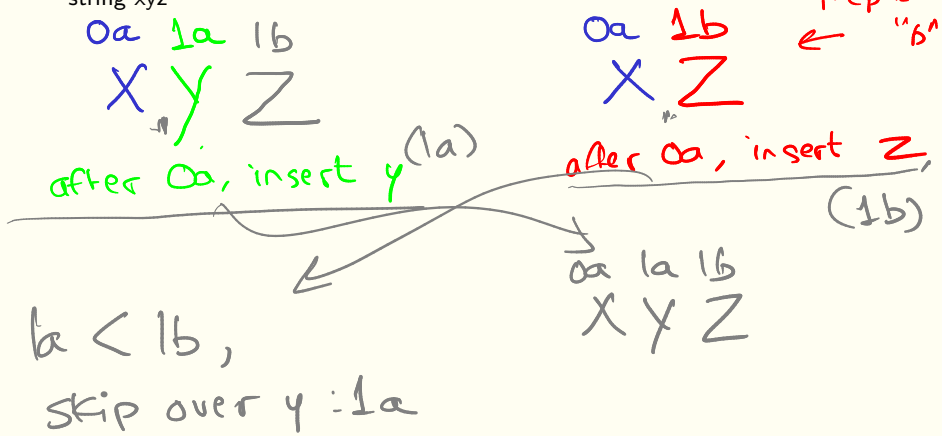
- Attach a unique location-id=(index, node-id) to each element
  - "Hello" → H:0a, e:1a, l:2a,...
- Operation-based: Add/delete operations specified relative to location
  - Insert "!" after 4a
- Merge lists by replaying operations
- If conflict, skip over existing list elements with greater ID

0a 1a 2a 3a 4a 5a

H e l l o !

after 2a, remove char

Example: Both processes start with "x". First process appends y, other process appends z. Because process ids are totally ordered, both processes end up with string xyz



References

- Martin Kleppmann: Automerge, lots of videos on CRDTs, ..
- CRDT papers by Marc Shapiro et.al.