Conflict-free Replicated Data Types

Shapiro et al. TR 2011

Project updates due tmrw

- By tomorrow at 6pm Vancouver time:
    1. Send me a project update (2 page max)
    2. Email me to schedule a chat (unless we already have regular meetings)

(This is 10% of your mark, please don’t be late)

CAP and CRDTs

- Paper: *CAP is a problem. We have a solution.*
- Breakout discussion:
  - So, do CRDTs “solve CAP”? Why or why not?

CAP and CRDTs

- *So, do CRDTs "solve CAP" ? Why or why not?*
- Doesn't solve it because the C in CAP is not the C (SEC) that CRDTs provide!
- Perhaps a solution to a "theoretical" notion of CAP. Mathematical guarantee that does not reflect reality.
- Makes assumptions that may not be true in practice: eventual delivery
- Performance is also a practical issue
- But, makes progress on CAP! Allows automagic convergence without user involvement. Perhaps useful in contexts where strong consistency is not necessary
- **Start of paper: you need to choose AP, so how much C can you get?** CRDTs meaningful, but paper overclaiming?
- A point on the spectrum of C, so a solution if you consider a well defined set of choices for C,A,P
- CRDTs designed for apps that are okay with a window of inconsistency
- CRDTs provide semantics that withstand weaker consistency (more resistant to async of the underlying network/world)
- Similarities with multi-core data structures (e.g., GC must avoid inconsistencies)

SE versus SEC

- Defined used linear temporal operators (circle, square), part of *temporal logic*
  - *Square: always*
  - *Rhombus: eventually*
- *Netw assumption: Eventually all updates delivered to all replicas*
- EC: eventual consistency (def 2.2)
  - *Always the case (square), if two replicas have same set of states, then eventually (rhombus) always (square) their state will be the same.*
  - There is an unspecified/unbounded delay between when the two replicas' sets synchronize and when they achieve identical states
  - (There is no simple m that automates the merging)
- SEC: **strong** eventual consistency (def 2.3)
  - *There is no delay: when two replicas have identical sets of states, they (at that instant) have identical states.*
  - *Instantaneity is achieved with the **m** function (merge)*

CRDT "*object*" defn

- Assumptions for both state based an op based CRDTS
  - Eventual delivery of all updates
  - Termination of all the operations
- Applications use CRDTs through the exposed API (**ADT abstraction**)
  - Set: add/remove/union/intersection
  - Graph: add_v/add_arc/remove_v/remove_arc
  - Counter: increment/decrement
  - List: add/remove/len

CvRDT "*object*" defn

- CvRDT (state based / con**v**ergent)
  - Object states (values), **order** on states of the object
  - State **m**erge method: m(s1,s2) => s3 **(LUB)**
  - **U**pdate method: monotonic, non-decreasing
- How is a CvRDT object implemented?
  - Record states resulting from updates to the object
  - Share (send) all the states to all the other nodes
  - When you receive states, merge them with whatever you have locally
  - Merge is a compaction routine m(m(m(s1,s2),s3),s4) => s5 (LUB of s1,s2,s3,s4)
  - Propagation of the LUB is critical, but can be summarized with merge

CmRDTs

- **Requirement: causally-ordered broadcast protocol**
- CmRDT (operation based / co**mm**utative)
  - Operation
    - t: prepare-update (side-effect-free) method — runs once, at the source of the operation
    - u: effect-update method (side-effects)
    - At the source the execution requires u to immediately follow t: (t,u) applied as a unit at the source
    - At other nodes, only u is applied
  - P: pre-condition/guard that constrains when you can apply **u**, the update (receiving nodes)
    - P eventually enabled
  - Commutativity of operations: (t,u) and (t',u')
    - Order of applying commutative updates doesn't matter => identical to a merge behaviour, since LUB is the same regardless of ordering of the set of inputs

CmRDTs

- Why the separation of t (side-effect free, prepare update) from u (side-effect, update)?

CRDT examples

- Vector clocks are CRDTs
- Counters: set of increments, and a set of decrements. Merge: \sum over the increments - \sum over decrements
- Graph with sets of vertices (V) and arcs (A)
  - Define commutativity on V, and separately on A, and between operations on V and ops on A

CRDT tradeoffs

- Error handling: how do “ask” or tell an application about a conflict that I want the application to resolve?
  - You can’t (or shouldn’t): conflict resolution must happen inside of the CRDT and it must be consistent across all replicas
- CvRDTs are space inefficient
  - You want CmRDTs in practice. But, CmRDTs have strong networking requirements
- Network eventually delivers all states/operations: still reliant on the network to satisfy this condition

Next: Optimistic Replication (OR)

- In some ways, a pre-cursor to CRDTs. A broad area of distributed coordination algorithms that "assume the best"
  - Similar to optimistic concurrency control mechanism like software transactional memory
- *Optimistic replication deploys algorithms not seen in traditional "pessimistic" systems.*
- *Instead of synchronous replica coordination, an optimistic algorithm propagates changes in the background, discovers conflicts after they happen and reaches agreement on the final contents incrementally*