Bayou

All about embracing weak connectivity and disconnected operation

- old world: LANs and continuous connectivity, chatty protocols, strong consistency and coherence guarantees
- new world: mobile devices with spotty wireless connectivity, more efficient protocools, need to reconsider what consistency/coherence guarantees are possible

Several major observations

- best consistency that you can hope for while still letting people make forward progress while disconnected is "eventual consistency"
    - if all updating stops, then eventually all replicas will converge to identical values
    - implies there will be churn at replicas as they wander towards eventual consistency
    - notion of "tentative" vs. "committed" comes out of this – a tentative update might be reordered and thus undo has to happen, might experience a conflict and thus resolution has to happen

- fundamental to weak/disconnected operation are conflicts
    - two writers update a data item in a conflicting manner
    - idea that applications are the best arbitors of what constitutes a conflict and how to resolve it
    - so, replication system propagates updates, applications resolve conflicts when they happen

Basic model of Bayou (and other systems)

- clients introduce state-changing operations, and replicas/servers manage state
- each replica/server defined in terms of state of:
    - "write log" – an ordered log of updates to a DB
    - DB – a database that results from the in-order execution of write log
    - goal: get all replicas to eventually agree on set and order of writes in log
        - Relies on two underlying properties:
            - Total propagation: every server eventually receives every update (perhaps via intermediaries) – epidemic algorithms
            - Consistent ordering: every server can agree on the order of all (non-commutative) updates – primary commit scheme

Basic idea of bayou:

- client is allowed to immediately update its replica
    - but, update is "tentative"- may be reordered with respect to other updates before it becomes committed, or perhaps even rejected if conflict cannot be resolved
- a "write" operation is assigned a monotically increased "accept-stamp" by server
    - total order of writes accepted by server
    - partial order of writes across servers
- "prefix property": enforce accept-stamp order during anti-entropy
    - if server R holds a write stamped Wi that was initially accepted by another server X, then R also holds all writes accepted by X earlier than Wi
    - this property allows the use of version vectors as compact representation of set of writes known to a server: version vector entry is latest accept-stamp known from a certain server
    - also allows incremental update transmission (just move version vector forward, and will eventually propagate all)
- anti-entropy: exchange operations between peers, bringing each other up-to-date by exchanging operations not yet known by other server
    - use vector timestamps to figure out which operations in log to exchange
    - exchanging **operations** is useful: no longer have to worry about death certificates (why? otherwise, cannot tell if update is to a new data element, or an old update from deleted element.)

- "stable write": also known as committed write. One whose position in write-log will never change
    - hence, its side-effect on DB never needs to be undone
    - bayou: uses primary-commit protocol to decide on stability
        - a primary decides on commit order, uses anti-entropy to propagate commit-sequence number
        - write becomes stable at a non-primary replica when it learns its CSN
        - have both commit and accept version vectors
        - first anti-entropy on commit vector, then accept: this preserves prefix property
    - replica can truncate any stable prefix of its log!!

Space of policies for:
- when to begin anti-entropy
- with whom
- write-log truncation policies
    - e.g.: estimate rate at which updates propagate globally, and match truncation rate to it

Weak consistency is visible to application in two ways:

- tentative versus stable commit order
- fact that tentative can be reordered before becoming committed

Conflicts

- may be detected arbitrarily far from user that introduced conflict, and maybe even when no user is present: hence want automatic conflict detection and resolution
- application specifies notion of conflict plus policy for resolving
    - system provides mechanism for detecting conflict and resolving them automatically where possible

Dependency check

- along with each write operation, application provides a query that must resolve correctly before write operation can be safely applied
    - e.g., meeting doesn't conflict with other meeting (room or people)
- if does conflict, apply a merge procedure
    - e.g., alternate meeting times
- if merge procedure cannot complete, log conflict and punt to human
- requirement on dependency check and merge: must be deterministic
    - else may have different answers at different replicas

Other notions of conflicts:

- many systems treat updates as nonconflicting iff
    - u and w update different objects (e.g., files or records)
    - u and w update the same object, but one writer had observed the other's update before making its own.
- hence conflicting update if concurrent writes to same object
    - concurrent can be temporal or causal, depending on system
- bayou:
    - rejects this "blind" view of conflicts, and relies on application-specific semantics
    - hence, two updates may conflict even if they are to different records
        - e.g., DB integrity rules

Note: no real discussion of experience with conflicts in general!!?!