Bayou: A Weakly Connected Storage System

Robert Grimm
New York University
The Three Questions

- What is the problem?
- What is new or different?
- What are the contributions and limitations?
Bayou from High Above

- Designed for mobile computing
  - Supports read anywhere, write anywhere
  - Makes minimal assumptions about connectivity
- Provides eventual consistency
  - Exposes both tentative and stable data
- Is *not* transparent to applications
  - Writes are <update, dependency check, merge procedure>
- Is centered around an epidemic anti-entropy protocol
  - One-way operation between pairs of servers
  - Propagation of writes, constrained by (accept) order
The Target Environment

- A pretty-bad-case scenario...
  - Mobile computers
  - Expensive connection time
  - Frequent disconnections
  - Little overlap in connection times
- Less bad is ok though...
  - Considerable flexibility in setting anti-entropy policies
    - When to reconcile
    - With which replica to reconcile
    - When to truncate the write log
    - On which server to register new replica
Consistency depends on
* Ordering of operations (e.g., when each meeting is added)
* Detection and resolution of conflicts (e.g., rejecting overlapping meetings)

Traditional solution: lots of clients, one server
* Ordering: one copy, server picks order
* Conflicts: server checks for overlap, returns error

So, why not use this approach?
* Local access on personal devices
* Intermittent connectivity with central infrastructure (internet)
* Intermittent connectivity with other users (ad hoc wireless)
Straw Man: Swap/Sync DBs

- May be resource intensive
  - Notably, requires lot of network bandwidth
- Hard to ensure consistency
  - There is no notion (of ordering) of operations
  - There is very little meaning to data (besides tuple structure)
- Underlying problem: DB is collection of bits
  - Represents a snapshot in time
- Solution: view DB in terms of updates
  - Operational: Read, think, make change
  - Well-ordered: All replicas converge on same snapshot
Towards a More Good Solution

- Maintain an ordered list of updates for each node
  - Enter the write log
- Make sure that every node has the same updates
- Make sure every node applies updates in same order
  - Accept order, causal accept order, total order
- Make sure that updates are deterministic
  - No access to local time, server name, rand(), ...
- Now: sync does not merge databases, but lists
  - Easier than merging of collections of bits
What about Ordering?

- Observation: we very much care about ordering
  - Even for tentative operations
- Read your writes: W→R
  - E.g., change password, log in
- Monotonic reads: W→R1→R2
  - E.g., meetings stay in calendar, listed emails are readable
- Write follows read: W1→R→W2 implies W1→W2
  - E.g., newsgroup reply appears after original post
- Monotonic writes: W1→W2
  - E.g., last text file edit survives
Bayou_Write(
    update = {insert, Meetings, 12/18/95, 1:30pm, 60min, “Budget Meeting”},
    dependency_check = {
        query = “SELECT key FROM Meetings WHERE day = 12/18/95 
            AND start < 2:30pm AND end > 1:30pm”,
        expected_result = EMPTY},
    mergeproc = {
        alternates = {{12/18/95, 3:00pm}, {12/19/95, 9:30am}};
        newupdate = {};
        FOREACH a IN alternates {
            # check if there would be a conflict
            IF (NOT EMPTY ( 
                SELECT key FROM Meetings WHERE day = a.date
                AND start < a.time + 60min AND end > a.time))
            CONTINUE;
            # no conflict, can schedule meeting at that time
            newupdate = {insert, Meetings, a.date, a.time, 60min, “Budget Meeting”};
            BREAK;
        }
        IF (newupdate = {}) # no alternate is acceptable
            newupdate = {insert, ErrorLog, 12/18/95, 1:30pm, 60min, “Budget Meeting”};
        RETURN newupdate;
    })

* Marked by timestamp <local time, accepting node id>
* Lamport clock for causal accept order
Propagating Writes

- Unidirectional, peer-to-peer synchronization
  - By wired/wireless network, floppy disk, USB keychain...
- Updates may appear out of order
  - Need to be merged into log
    - Undo newer updates
    - Insert just received updates
    - Replay the log
- User's view of data may change as result
  - But when everybody has seen all writes, everybody will agree
We Like Short Logs, Step 1

- We need to know when everybody has seen a write
  - Lamport clock preserves causal order, but doesn't provide global consensus
- We need a notion of commitment
  - For entry X to be committed, everyone must agree on
    - The total order of all previous writes
    - The fact that X is next in this total order
    - The fact that all tentative entries follow after X
  - "Any mechanism that stabilizes the position of a write in the log can be used"
Bayou's Commitment

- Each data collection has one *primary* replica
  - Commits all writes for that collection
  - Marks each write with a commit sequence number (CSN)
    - Timestamp really is <CSN, local time, accepting node>
  - Propagates commitments during anti-entropy
- How to ensure that CSN observes causal accept order?
  - Local time is Lamport (logical) time
  - Everybody propagates updates in order
  - As a result, primary sees updates in causal order and commits them in that order
We Like Short Logs, Step 2

- Truncate the log
  - Tentative writes must never be discarded
    - May have to be undone and redone (due to reordering)
  - Committed writes *may* be discarded
    - But other, long disconnected replicas may not have seen them
    - So, keep some amount of history

- Wait: where did truncated writes go?
  - We don't just have a log, but also an actual database
    - Also contains tentative writes (marked as such)
  - We remember the omitted sequence number (OSN)
More on Truncation

- During anti-entropy, we may have to send DB
  - If receiver's CSN is smaller than sender's OSN
    - I.e., if receiver's head of log is before sender's tail of log
  - Sender's DB provides new starting point for receiver
    - Receiver discards committed writes, but keeps tentative ones
Some More Details

- Replicas can be added and removed dynamically
  - Addition/removal is relative to another replica
  - Operations preserve causal order
- Access control performed at granularity of database
  - Based on public/private key cryptography
  - Checked by accepting and by committing replica
    - Accepting replica first-line defense against unauthorized access
    - Committing replica definitive authority
What Do You Think?