Managing Update Conflicts in Bayou, a Weakly Connected Replicated Replicated Storage System

Douglas B. Terry, Marvin M. Theimer, Karin Petersen, Alan J. Demers, Mike J. Spreitzer and Carl H. Hauser

SOSP 1995
Motivation

- Users working concurrently can introduce conflicts for collaborative applications
- Unreliable network connections and high per-minute connection costs mean it is impractical to have continuous connections
- Goal: balance the realistic expectations of poor network connectivity with the desire to maintain weakly consistent, replicated data
Solution: Bayou

- Application-specific determination of how to detect and resolve conflicts
  - Accomplished via dependency checks and merge procedures
- Maintain two states of an update: tentative and committed
- Manage the states to determine when an update can be changed from tentative to committed
- Designed for non-real-time collaborative applications (e.g. meeting room scheduler, mail databases, etc.)
- Ensure that replicas will be eventually consistent
Application Examples

- **Meeting room scheduler**
  - For a given room, any time for a meeting for that room can be assigned to no more than one person
  - User is presented current state of scheduler with respect to his/her copy of the room schedule (which may be out of date if, e.g., it is a local version that has not recently been updated)
  - As the connectivity allows, the schedule is periodically re-read, and any are updated on user’s graphical user interface

- **Bibliographic database**
  - Users add papers to the database as they are found
  - Must assign unique key to each paper
  - Keys are tentative until committed, since there could be a conflict, and uniqueness of keys must be ensured
  - Cannot expect to maintain persistent connection with library network due to, e.g. student hackers
Layout of Bayou’s System Model

- Data collection fully replicated among servers
- Applications communicate with servers through Bayou API
  - Basic operations: read and write
- Access to one server is enough for client to perform read/update operations
- Bayou write operations contain additional information for how to deal with conflicts
- Each write has a globally unique WriteID
- Storage system maintains ordered log of writes
- Writes are passed from one server to another via pairwise contacts (“anti-entropy”)
  - Theory promises that, if some assumptions about no servers being permanently partitioned are met, then write will eventually make it to every server
Dealing with Conflicts

- How conflicts are managed is application-specific
- Dependency checks
  - Used to determine if write-write conflict occurs
  - For each write, a query is associated with it which is checked against current state of connected server
  - If database evaluates SQL-like query as holding true, this indicates a conflict
    - Leads to merge procedure
- Merge procedure
  - Also left as a choice of developer
  - Main objective: when faced with a conflict, how to resolve it
  - Room scheduler example: provide a list of multiple preferred times
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;
    })
)

Figure 3. A Bayou Write Operation
Maintaining Consistency

- Eventually consistent
  - Guarantees that all servers eventually receive all writes
- When write is first accepted, marked as tentative
  - Write is associated with timestamp
- Server maintains log of writes, ordered by timestamp
  - Committed writes are ordered before tentative writes
  - Global order of tentative timestamps ensures agreement for isolated cluster of nodes
  - In some cases, tentative writes need to be undone and re-executed
- Need to determine write is stable so that it can be committed
  - Accomplished using a primary commit scheme
    - One server, designated as primary, has the authority to commit updates
    - Desirable because requiring majority quorum is difficult in face of connectivity issues
    - If primary fails, client can still perform useful read and write operations
      - In this case, writes just remain tentative
Implementing the Storage System

- Design of Bayou leads to certain requirements for the structure of the storage system
- Write log
  - Contains write obtained by a given server
  - Writes are in their global committed or tentative order (based on timestamp)
- Tuple store
  - Implemented as relational database
  - Provides support for SQL-like queries
  - Interesting property: maintains two views of data
    - Committed view: only shows committed writes
    - Full view: shows both tentative and committed writes
- Undo log
  - Allows server to undo effects on tuple store
Figure 4. Bayou Database Organization
Access Control

● Because of poor network connectivity assumption, cannot rely on trusted central authentication server

● Instead, mutual authentication is implemented
  ○ Based on public-key cryptography
  ○ Uses a single trusted signing authority to sign all certificates

● Authorization is granted for entire data collection
  ○ Can be revoked if it is found that certificate has been revoked
Performance

- Size of storage increases as number of tentative writes increases
  - Primarily attributed to increase in overhead cost of access control certificate for tentative writes
- Execution time for Bayou server to undo/redo all tentative writes
  - Cost of redoing is nearly constant time
- Performance of operations between client and server
  - In case of conflict, write is not unique
    - Thus, requires additional time for reassignment within merge procedure
Table 1: Size of Bayou Storage System for the Bibliographic Database with 1550 Entries
(sizes in Kilobytes)

<table>
<thead>
<tr>
<th>Number of Tentative Writes</th>
<th>0 (none)</th>
<th>50</th>
<th>100</th>
<th>500</th>
<th>1550 (all)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write Log</td>
<td>9</td>
<td>129</td>
<td>259</td>
<td>1302</td>
<td>4028</td>
</tr>
<tr>
<td>Tuple Store Ckpt</td>
<td>396</td>
<td>384</td>
<td>371</td>
<td>269</td>
<td>1</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>405</strong></td>
<td><strong>513</strong></td>
<td><strong>630</strong></td>
<td><strong>1571</strong></td>
<td><strong>4029</strong></td>
</tr>
<tr>
<td>Factor to 368K bibtex source</td>
<td>1.1</td>
<td>1.39</td>
<td>1.71</td>
<td>4.27</td>
<td>10.95</td>
</tr>
</tbody>
</table>
Table 2: Performance of the Bayou Storage System for Operations on Tentative Writes in the Write Log
(times in milliseconds with standard deviations in parentheses)

<table>
<thead>
<tr>
<th>Tentative Writes</th>
<th>0</th>
<th>50</th>
<th>100</th>
<th>500</th>
<th>1550</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Server running on a Sun SPARC/20 with Sunos</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Undo all (avg. per Write)</td>
<td>0</td>
<td>0.62</td>
<td>0.7</td>
<td>0.66</td>
<td>0.56</td>
</tr>
<tr>
<td></td>
<td>237</td>
<td>(85)</td>
<td>611</td>
<td>(302)</td>
<td>2796</td>
</tr>
<tr>
<td></td>
<td>4.74</td>
<td>6.11</td>
<td>5.59</td>
<td>5.05</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Server running on a Gateway Liberty Laptop with Linux</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Undo all (avg. per Write)</td>
<td>0</td>
<td>.94</td>
<td>1.04</td>
<td>0.96</td>
<td>0.83</td>
</tr>
<tr>
<td></td>
<td>302</td>
<td>(91)</td>
<td>705</td>
<td>(134)</td>
<td>3504</td>
</tr>
<tr>
<td></td>
<td>6.04</td>
<td>7.05</td>
<td>7.01</td>
<td>6.4</td>
<td></td>
</tr>
</tbody>
</table>

Table 3: Performance of the Bayou Client Operations
(times in milliseconds with standard deviations in parentheses)

<table>
<thead>
<tr>
<th>Server Client</th>
<th>Sun SPARC/20 same as server</th>
<th>Gateway Liberty same as server</th>
<th>Sun SPARC/20 Gateway Liberty</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Read: 1 tuple 100 tuples</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>27 (19)</td>
<td>38 (5)</td>
<td>23 (4)</td>
</tr>
<tr>
<td></td>
<td>206 (20)</td>
<td>358 (28)</td>
<td>244 (10)</td>
</tr>
<tr>
<td>Write: no conflict with conflict</td>
<td>159 (32)</td>
<td>212 (29)</td>
<td>177 (22)</td>
</tr>
<tr>
<td></td>
<td>207 (37)</td>
<td>372 (17)</td>
<td>223 (40)</td>
</tr>
</tbody>
</table>
Strengths

- Eventual consistency guarantees, via pairwise “anti-entropy” communication
- Flexibility for how application developers manage conflict
- Managing conflict occurs at granularity of per-iteration
- Design is such that servers do not need nearly perfect synchronized clocks
  - Good for unreliable networks
Weaknesses

- Application-specific conflict detection and resolution means more work for the application developer, and is possibly prone to human error
- Potential for cascading conflicts (a new write depends on a previous conflict write, etc.)
  - Mitigated by circumstances of application, e.g. if application is not continuously subjected to (possibly conflict-inducing) writes
Conclusion

- Bayou provides a weakly consistent storage system for mobile applications connecting via an unreliable network.
- Conflicts are resolved (or, if unresolved, written to an error log to be dealt with by hand) at the granularity of each individual write.
- Bayou maintains both a full and a committed view of the data.
- Best suited for situations with low likelihood of conflicts occurring.
Questions?