Managing Update Conflicts in Bayou

a Weakly Connected Replicated Storage System
Purpose

- Database infrastructure for portable computers
- Maximal availability achieved weak consistency
- Honest service model
Assumptions

- Collaborative Applications
- Arbitrary Weak Connectivity (mobile computing) (PDA’s)
- Occasional, pair-wise communication
- Applications are aware of weakly consistent paradigm
Meeting Room Scheduler
Sample Bayou Application

- Reserve a room
- Provided calendar of reservations (possibly out of date)
- Requested -> Tentative -> Reservation
  - Reservation may differ from request
- Can see correct data for each user on same replica
Bibliographic Database

Sample Bayou Application

• Cooperatively manage databases of bibliographic entries

• A user can freely read and write any replica, such as one that resides on his laptop.

• Papers keyed by authorYear[letter if necessary]; keys are initially tentative

• User or application must check to see if tentative key has changed
Bayou System Model

• Data is fully replicated at each server
• Clients interacts via API
  • read/write
  • Writes contain conflict resolution plan
• Replication is automatic
• Communicate with any one server at a time
• Client and server could be co-resident on a host
Bayou System Model

• Server:
  • ordered log of writes
  • data resulting from the execution of these writes
  • Undo method
  • Each server executes an assigned write & deconflicts as necessary
  • All writes are available for read
  • Convergence is certain * ** ***
    • * eventually
    • ** in the absence of new writes
    • *** so long the network is not permanently partitioned

Figure 1. Bayou System Model
Conflict Detection & Resolution

• What is a conflict? (Dependency Checks)
  • Scheduler: Two meetings are scheduled for the same room & overlap in time
  • Bibliography:
    • Two entries have the same key but describe different publications
    • Two entries have different keys but describe the same publication

• What is the resolution? (Merge Procedures)
  • Scheduler: Move the meeting to a different room or time
  • Bibliography:
    • Assign one entry a new key
    • Merge the two entries
Dependency Checks

• A single query, which returns expected results
• Write-write conflicts are easy
• Read-Write: specify the expected values of data items which the update depends on
  • Similar to optimistic concurrency control
• Scheduler: Two meetings are scheduled for the same room & overlap in time
• Bibliography:
  • Two entries have the same key but describe different publications
  • Two entries have different keys but describe the same publication
Merge Procedures

- Interpreted, deterministic program
  - Success: resolved conflict
  - Failure: Log the error
- Scheduler: Move the meeting to a different room or time
- Bibliography:
  - Assign one entry a new key
  - Merge the two entries
Consistency

- Write are initially tentative and eventually committed
- Writes are monotonically increasing at each server to produce an order
  - Clocks don’t need to be synchronized
  - Having close to real clock times makes behavior seem more reasonable
- Write are performed in the same order at all servers
Write Stability

- When a write is stable it will not be executed again
- Write become stable when the primary server commits it
- Clients can ask if a write is stable
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.day AND start < a.time + 60min AND end > a.time) |
                CONTINUE;
                # no conflict, can schedule meeting at that time |
                newupdate = |insert, Meetings, a.day, 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

Receive_Writes (writeset, received_from) |
    IF (received_from = CLIENT) |
        # Received one write from the client, insert at end of WriteLog |
        # first increment the server’s timestamp |
        logicallclock = MAX(systemclock, logicallclock + 1); |
        write = First(writeset); |
        write.WID = [logicallclock, myServerID]; |
        write.state = TENTATIVE; |
        WriteLog_Append(write); |
        Bayou_Write(write.update, write.dependency_check, write.mergeproc); |
    ELSE |
        # Set of writes received from another server during anti-entropy, |
        # therefore writeset is ordered |
        write = First(writeset); |
        insertionPoint = WriteLog_IdentifyInsertionPoint(write.WID); |
        TupleStore_RollbackTo(insertionPoint); |
        WriteLog_Insert(writeset); |
        # Now roll forward |
        FOREACH write IN WriteLog AFTER insertionPoint DO |
            Bayou_Write(write.update, write.dependency_check, write.mergeproc); |
            # Maintain the logical clocks of servers close |
            write = Last(writeset); |
            logicallclock = MAX(logicallclock, write.WID.timestamp); |
        |
    |
Figure 5. Applying Sets of Bayou Writes to the Database

Bayou_Write (update, dependency_check, mergeproc) |
    IF (DB_Eval (dependency_check.query) <= dependency_check.expected_result) |
        resolved_update = Interpret (mergeproc); |
    ELSE |
        resolved_update = update; |
    DB_Apply (resolved_update); |
    |
Figure 2. Processing a Bayou Write Operation
Implementation Issues

- Writes bloated by large amount of repeated code
Access Control

• Can’t assume any connection
• Two untrusted machines must trust each other without third party
• Read/write privileges for entire data collection
• Privilege to be a server
Performance Bibliographic Database

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>Total</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>
## Performance

### Bibliographic Database

**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></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Server running on a Sun SPARC/20 with Sunos</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Undo all (avg. per Write)</td>
<td>0</td>
<td>31</td>
<td>70</td>
<td>330</td>
<td>866</td>
</tr>
<tr>
<td></td>
<td>.62</td>
<td>(6)</td>
<td>(20)</td>
<td>(155)</td>
<td>(195)</td>
</tr>
<tr>
<td>Redo all (avg. per Write)</td>
<td>0</td>
<td>237</td>
<td>611</td>
<td>2796</td>
<td>7838</td>
</tr>
<tr>
<td></td>
<td>4.74</td>
<td>(85)</td>
<td>(302)</td>
<td>(830)</td>
<td>(1094)</td>
</tr>
<tr>
<td><strong>Server running on a Gateway Liberty Laptop with Linux</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Undo all (avg. per Write)</td>
<td>0</td>
<td>47</td>
<td>104</td>
<td>482</td>
<td>1288</td>
</tr>
<tr>
<td></td>
<td>.94</td>
<td>(3)</td>
<td>(7)</td>
<td>(15)</td>
<td>(62)</td>
</tr>
<tr>
<td>Redo all (avg. per Write)</td>
<td>0</td>
<td>302</td>
<td>705</td>
<td>3504</td>
<td>9920</td>
</tr>
<tr>
<td></td>
<td>6.04</td>
<td>(91)</td>
<td>(134)</td>
<td>(264)</td>
<td>(294)</td>
</tr>
</tbody>
</table>


## Performance Bibliographic Database

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>Read: 1 tuple 100 tuples</td>
<td>27 (19)</td>
<td>38 (5)</td>
<td>23 (4)</td>
</tr>
<tr>
<td>Read: 206 (20)</td>
<td>358 (28)</td>
<td>244 (10)</td>
<td></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>Write: 207 (37)</td>
<td>372 (17)</td>
<td>223 (40)</td>
<td></td>
</tr>
</tbody>
</table>
Optimistic Distributed Database

My Thoughts

• Great model for mobile computing in 1995
• Perfect for Single writer / Multi reader
• Good for session consistency
• Probably not appropriate for high connectivity systems
• Not so great with high churn systems
• Challenging for the developer
  • Developers risk passing nuisances onto user