Bayou

A Replicated Storage System
Replicated Storage System

Ingredients:

1. Make storage system deterministic (state machine)
2. Replicate storage system
3. Ensure that all replicas go through the same sequence of writes
4. Vote on replica outputs
Read and write at any replica

Client #1

Replicas

Client #2

w(x=4)  
w(x=2)  
r(x)  
x=4
**Eventual Consistency**

- Replicas execute every write request locally
- The results of each write are immediately available
- All replicas eventually agree on a global ordering of writes
- All replicas eventually execute each write in that order
After this execution, each replica will agree on the same ordering of events

A B C
Anti-entropy in practice

- Every write is time stamped by the server that processes it.
- Writes are ordered by local time stamp during anti-entropy.
- Clock synchronization is not required for consistency.
1. Replica 1 executes $w(x=x+4)$
2. Replica 2 executes $w(x=x*2)$
3. Replicas 1 & 2 participate in an anti-entropy session.
4. Order is decided to be $w(x=x+4) \ w(x=x*2)$
5. Replica 1 executes $w(x=x*2)$
6. Replica 2 rolls-back $w(x=x*2)$
7. Replica 2 executes $w(x=x+4) \ w(x=x*2)$
Tentative

May be re-executed with different results

Stable

Will not be re-executed ⇒ all previous writes are stable

Committed

Guaranteed to be stable
Committed writes cannot have tentative writes before them.
Clients Can Query:

- The results of all writes (tentative and committed)
- The results of only committed writes
- Whether a given write is committed
  - Writes are assigned a globally unique WriteID
When is a write no longer *tentative*?
1. Primary executes and commits \( w(x=x+4) \)
2. Replica executes \( w(x=x^2) \)
3. Primary & Replica participate in an anti-entropy session.
4. Order is decided to be \( w(x=x+4) \) \( w(x=x^2) \)
5. Primary executes \( w(x=x^2) \)
6. Replica rolls-back \( w(x=x^2) \)
7. Replica executes \( w(x=x+4) \) \( w(x=x^2) \)
Managing Conflicting Writes
Bayou Write

<update, dependency check, merge procedure>

Update: What update should be done
  e.g. make a reservation for 7:00pm

Dependency Check: What conditions must be true to execute the update
  e.g. there are no reservations for 7:00pm

Merge Procedure: What to do if our dependency check conditions fail
  e.g. make a reservation for 8:00pm
1. Primary commits \(<\text{res}(7), 7 \text{ is available, res}(8)\>\) and executes \(\text{res}(7)\).

2. Replica executes \(\text{res}(7)\).

3. Primary & Replica participate in an anti-entropy session.

4. Order is decided to be
   \(<\text{res}(7), 7 \text{ is available, res}(8)\>
   \(<\text{res}(7), 7 \text{ is available, res}(9)\>\).

5. Dependency check fails for second write.

6. Primary executes \(\text{res}(9)\).

7. Replica rolls-back \(\text{res}(7)\) and executes \(\text{res}(7), \text{res}(9)\).
Storage Implementation Details

● Write Log
  ○ Truncated for efficiency
  ○ Keeps only what is necessary for anti-entropy

● Undo Log
  ○ Tracks how to undo tentative writes

● Relational Database (Tuple Store)
  ○ Supports full and committed views
  ○ Can be queried using a subset of SQL
  ○ Facilitates speedy dependency checking
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>
# 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 (6)</td>
<td>70 (20)</td>
<td>330 (155)</td>
<td>866 (195)</td>
</tr>
<tr>
<td>Redo all (avg. per Write)</td>
<td>0</td>
<td>237 (85)</td>
<td>611 (302)</td>
<td>2796 (830)</td>
<td>7838 (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 (3)</td>
<td>104 (7)</td>
<td>482 (15)</td>
<td>1288 (62)</td>
</tr>
<tr>
<td>Redo all (avg. per Write)</td>
<td>0</td>
<td>302 (91)</td>
<td>705 (134)</td>
<td>3504 (264)</td>
<td>9920 (294)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Evaluation

**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</td>
<td>27 (19)</td>
<td>38 (5)</td>
<td>23 (4)</td>
</tr>
<tr>
<td>100 tuples</td>
<td>206 (20)</td>
<td>358 (28)</td>
<td>244 (10)</td>
</tr>
<tr>
<td>Write: no 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>
Conclusion

- Highly available and eventually consistent
- Only has weak connectivity requirements
- Novel method for managing conflicts that maintains high availability
- Conflict detection is highly flexible
- Classical battle between consistency, availability, and partitions
Downsides

- Primary failure prevents writes from stabilizing
- Conflicts may not be resolved automatically
- Replicas must be able to undo tentative writes
- Tentative writes take up a lot of space
Thoughts & Questions