Replication and Transaction Management in a Temporal Database

Matthew Dillon
Backplane, Inc.
Transaction Consistency

Client #1

Fred's Bonus

BONUS = BONUS + $10

CORRECT RESULT: $25

Client #2

BONUS = BONUS + $15

INCONSISTENT/INCOHERENT (MY BONUS GOT LOST!)

Read BONUS

BONUS = BONUS + $10

$0

$10

$15

Read BONUS

BONUS = BONUS + $15

INCONSISTENT/INCOHERENT (MY BONUS GOT LOST!)

$15

INCORRECT RESULT: $15
Enforcing Consistency with Record Locking

**Client #1**

WITH RECORD LOCKING

- **Read BONUS**
- **BONUS = BONUS + $10**
- **COMMIT**

WITH RECORD LOCKING

- **Read BONUS**
- **BONUS = BONUS + $10**
- **COMMIT**

**CORRECT RESULT:** $25

**ALTERNATIVE RECORD LOCKING**

- **Read BONUS**
- **BONUS = BONUS + $10**
- **COMMIT**

- **FAIL (STALL, RETRY)** (transaction aborted)
- **Read BONUS**
- **BONUS = BONUS + $15**
- **COMMIT**

**CORRECT RESULT:** $25
Deadlock/LiveLock Issues With Record Locking

Client #1

**READ BONUS**

**READ EXPENSES** (BLOCKED)

**ALTERNATIVE RECORD LOCKING**

- Read BONUS
- Update BONUS
- Read EXPENSES
- (fails)

Possible Livelock. If we block instead of fail, guaranteed deadlock

Client #2

**READ EXPENSES**

**READ BONUS** (fails)

**ALTERNATIVE RECORD LOCKING**

- Read BONUS
- Update BONUS
- Read EXPENSES
- Update EXPENSES

If we allow client #2 to use uncommitted data from client #1, and client #1 later fails, client #2 must then fail leading to a possible livelock.
- Fallback to hot backup on failure
- Hot backup may not be so hot
- Commit performance same as single-db case
- Fallforward is more problematic
Asynchronous non-Coherent Replication

- Transactions can be distributed
- Both read-only and modifying transactions scale
- Commit to single peer, replicate to others. Commit does not guarantee correctness
- Conflict resolution must occur after the fact
Fully Synchronous Coherent Replication

- Transactions can be distributed
- Read-only transactions scale
- Modifying transactions must talk to all peers
- Potentially serious locking & timeout issues if a peer fails
- Many implementations require a designated 'master' node
## Temporal Database Table Structure

<table>
<thead>
<tr>
<th></th>
<th>TRANSID</th>
<th>KEY</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSERT</td>
<td>I</td>
<td>0830</td>
<td>BONUS</td>
</tr>
<tr>
<td>UPDATE</td>
<td>D</td>
<td>0831</td>
<td>BONUS</td>
</tr>
<tr>
<td></td>
<td>I</td>
<td>0831</td>
<td>BONUS</td>
</tr>
<tr>
<td>UPDATE</td>
<td>D</td>
<td>0832</td>
<td>BONUS</td>
</tr>
<tr>
<td></td>
<td>I</td>
<td>0832</td>
<td>BONUS</td>
</tr>
<tr>
<td>DELETE</td>
<td>D</td>
<td>0833</td>
<td>BONUS</td>
</tr>
</tbody>
</table>

- INSERT BONUS = $0
- UPDATE BONUS = BONUS + $10
- UPDATE BONUS = BONUS + $15
- DELETE BONUS

- **LOCKLESS TRANSACTIONS ARE POSSIBLE**
- **HISTORICAL AS-OF QUERIES ARE POSSIBLE (INCLUDING META-DATA)**
- **EASIER TO BACKUP, RESTORE, AND ARCHIVE**
- **EASIER TO RECOVER CORRUPTED DATABASE**
- **APPEND-ONLY FILE STRUCTURE POSSIBLE**
- **LOCKLESS TRANSACTIONS CAN EXTEND TO REPLICATED PEERS USING TWO-PHASE COMMIT**
- **INCREMENTAL REPLICATION WITHOUT QUEUES POSSIBLE**

- **UNLESS VACUUMED, TABLE FILES GROW WITH EACH INSERT, UPDATE, OR DELETION**
- **LOTS OF 'DELETED' RECORDS CAN CLUTTER INDEXES AND TABLE DATA**
- **AN UPDATE APPENDS TWO RECORDS INSTEAD OF ONE**
- **A DELETE APPENDS ONE RECORD INSTEAD OF FLAGGING AN EXISTING RECORD**
### Reverse Scan Optimization

<table>
<thead>
<tr>
<th>TRANSID</th>
<th>KEY</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSERT</td>
<td>I</td>
<td>0830</td>
</tr>
<tr>
<td>UPDATE</td>
<td>D</td>
<td>0831</td>
</tr>
<tr>
<td></td>
<td>I</td>
<td>0831</td>
</tr>
<tr>
<td>UPDATE</td>
<td>D</td>
<td>0832</td>
</tr>
<tr>
<td></td>
<td>I</td>
<td>0832</td>
</tr>
<tr>
<td>UPDATE</td>
<td>D</td>
<td>0832</td>
</tr>
<tr>
<td></td>
<td>I</td>
<td>0832</td>
</tr>
<tr>
<td>UPDATE</td>
<td>D</td>
<td>0832</td>
</tr>
<tr>
<td></td>
<td>I</td>
<td>0832</td>
</tr>
<tr>
<td>DELETE</td>
<td>D</td>
<td>0833</td>
</tr>
</tbody>
</table>

- INSERT BONUS = $0
- UPDATE BONUS = BONUS + $10
- UPDATE BONUS = BONUS + $15
- UPDATE BONUS = BONUS + $15
- UPDATE BONUS = BONUS + $15
- UPDATE BONUS = BONUS + $15
- DELETE BONUS

- **IF DATA IS KNOWN TO BE UNIQUE, ONLY ONE RECORD NEEDS TO BE SCANNED**
- **DELETIONS CAN BE PAIRED WITH INSERTS MORE EFFICIENTLY WITH A REVERSE SCAN**

**Freeze Point**
Lockless Transactions

<table>
<thead>
<tr>
<th>TRANSID</th>
<th>KEY</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSERT</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>0830</td>
<td>BONUS</td>
<td>$0</td>
</tr>
</tbody>
</table>

**Insert Bonus = $0**

**Freeze Point A**

**Completed Commits from Other Clients**

<table>
<thead>
<tr>
<th>TRANSID</th>
<th>KEY</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPDATE</td>
<td>D</td>
<td></td>
</tr>
<tr>
<td>0831</td>
<td>BONUS</td>
<td>$0</td>
</tr>
</tbody>
</table>

**Update Bonus = Bonus + $10**

<table>
<thead>
<tr>
<th>TRANSID</th>
<th>KEY</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0831</td>
<td>BONUS</td>
<td>$10</td>
</tr>
</tbody>
</table>

**Update Bonus = Bonus + $15**

**Freeze Point B**

**Completed Commits from Other Clients**

**Client 1**

<table>
<thead>
<tr>
<th>TRANSID</th>
<th>KEY</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPDATE</td>
<td>D</td>
<td></td>
</tr>
<tr>
<td>0832</td>
<td>BONUS</td>
<td>$10</td>
</tr>
</tbody>
</table>

**Update Bonus = Bonus + $10**

<table>
<thead>
<tr>
<th>TRANSID</th>
<th>KEY</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0832</td>
<td>BONUS</td>
<td>$25</td>
</tr>
</tbody>
</table>

**Client 2**

**Simultaneous Transactions by Clients 1 and 2**

- Client queries relative to freeze point A
- Modifications made to temporary tables
- Client commit-phase-1 copies temporary table to data space
- Client commit-phase-1 reruns queries with freeze point temporarily moved
- Any data accessed between freeze point A and data copy indicates conflict
- (Non-replicated) at least one client always guaranteed to succeed
- Commit-phase-2 sets new addition to data space in stone
- Multiple clients can complete commit-phase-2 out of order

**Queries are run twice**
• Transactions are relative to a freeze point.
• Selected quorum must be synchronized to the specified freeze point.
• Commit only needs to occur on a quorum of peers.
• Synchronized freeze point is not updated by commit.
• Remaining peers and snapshots get updated via replication.
• Synchronized freeze point is updated via replication.

• Partitioning a problem in WAN topologies.
Quorum Based Replication (BEFORE)

Synchronized Freeze Point

INSERT I 0830

CLIENT#1 D 0831
I 0831

CLIENT#2 D 0832
I 0832

MISSING

MISSING

INSERT I 0830

CLIENT#1 D 0831
I 0831

MISSING

CLIENT#3 D 0833

NOTE: CLIENTS MADE NON-CONFLICTING COMMITS
SYNCHRONIZED FREEZE POINT UPDATED WITH QUORUM
Out of Order Quorum-Based Replication (AFTER)

- ONLY NON-CONFLICTING TRANSACTIONS WILL REPPLICATE OUT OF ORDER
- GENERALLY HARMLESS
- NO EFFECT ON REVERSE SCAN OPTIMIZATION
- TRANSACTION ID'S NOT MONOTONIC (SOLUTION: INDEX TRANSACTION ID'S)
Optimizing the Replication

- **DIRECT COPY FROM A SINGLE SOURCE UP TO THE EXISTING SYNCHRONIZATION POINT**
- **USE THE CLOSEST SOURCE, OR THE MOST COMPLETE SOURCE?**
- **QUORUM BASED REPPLICATION FOR THE REMAINDER**
Replicating a Historical Database

- Clients may talk to a single node.
- Database services at node abstract-out query/commit protocol.
- Replication process is independent from query/commit process.
- Spanning tree protocol reduces the effect of link failures.
- Participation depends on freeze point/synchronization of nodes.
- First-responding-quorum minimizes latency.
- Automatic query restart if link failure interrupts respondent.
- Native replication possible (no queueing).
- Full-on backup links possible.
- High-latency snapshot/backup links possible.
Query Management in a Replicated Environment

CLIENT  PEER-A  PEER-B  PEER-C

BEGIN

QUERY1  QUERY2  QUERY3  QUERY4  QUERY5  QUERY6  QUERY7  QUERY8  QUERY9

COMMIT1

QUORUM1/2  QUORUM2/2

COMMIT1

BEGIN  BEGIN  BEGIN  BEGIN

QUERY1  QUERY2  QUERY3  STALL

QUERY2  QUERY3  QUERY4  FREEZETS

QUERY3  QUERY4  QUERY5  QUERY1

QUERY4  QUERY5  QUERY6  QUERY2

QUERY5  QUERY6  QUERY7  QUERY3

QUERY6  QUERY7  QUERY8  LATENCY

QUERY7  QUERY8  QUERY9  LATENCY

QUERY8  QUERY9  COMMIT1  LATENCY

QUERY9  COMMIT1  COMMIT2  COMMIT1

COMMIT1

MUST ABORT PEERS WHICH HAVE NOT YET ACKED COMMIT-1 AFTER SENDING COMMIT-2

ONLY ONE PEER NEEDS TO RETURN QUERY RESULTS, BUT ALL MUST RECORD QUERIES

CLIENT CAN CONTINUE BASED ON FIRST RESPONSE, COMMIT-2 WHEN QUORUM REACHED

SOME PEERS CAN RETURN FAILURE LEGALLY AS LONG AS QUORUM RETURNS SUCCESS
Temporary Link Failures / Query Restarts Prior to Commit

- Partial results may have to be thrown out if reacquiring from a new peer.
- No hicups, next lowest-latency peer can complete the transaction.
- No locking overhead within transaction body – restarts easy.
Commit Phase – A Quorum is only a Minimum

<table>
<thead>
<tr>
<th>CLIENT</th>
<th>PEER-A</th>
<th>PEER-B</th>
<th>PEER-C</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>QUERY5</td>
<td>QUERY5</td>
<td>QUERY5</td>
<td>QUERY5</td>
</tr>
<tr>
<td>QUERY6</td>
<td>QUERY6</td>
<td>QUERY6</td>
<td>QUERY6</td>
</tr>
<tr>
<td>QUERY7</td>
<td>QUERY7</td>
<td>QUERY7</td>
<td>QUERY7</td>
</tr>
<tr>
<td>QUERY8</td>
<td>QUERY8</td>
<td>QUERY8</td>
<td>QUERY8</td>
</tr>
<tr>
<td>QUERY9</td>
<td>QUERY9</td>
<td>QUERY9</td>
<td>QUERY9</td>
</tr>
<tr>
<td>COMMIT1</td>
<td>COMMIT1</td>
<td>COMMIT1</td>
<td>COMMIT1</td>
</tr>
</tbody>
</table>

- Quorum1/2
- Quorum2/2
- Quorum3/2

- COMMIT2
- AckC1
- AckC1
- AckC2
- AckC2

FINISHED

• Final completion to client can occur after quorum’s worth of Commit-2 acks
• Sending Commit-2 to more than a quorum improves robustness and safety
• Asynchronous background replication is the key to flexibility
• Remember, must throw away AckC1’s after first Commit-2 sent
Failures During the Commit Phase

- Data may not have been committed to a quorum even though we meant to.
- We cannot restart the transaction on B and C due to asynchronous replication.
- Quorum (B and C) may move their synchronization point past the transaction.
- If synch point moved on B and C, replication from Peer-A might not occur.

- B and C know that a transaction was in progress, still have the commit-1 data.
- B and C do not know the transaction ID if they did not get the commit-2.
- Replication failure or recovery conflict can still be detected after the fact.
**Detecting Data Corruption**

- **SOMEHOW CORRUPTED**
  - CLIENT#1
    - D: 0831
    - I: 0831
  - CLIENT#2
    - D: 0832
    - I: 0832
  - CLIENT#3
    - D: 0833

- **SOMEHOW LOST**
  - INSERT
    - I: 0830
  - CLIENT#1
    - D: 0831
    - I: 0831
  - CLIENT#2
    - D: 0832
    - I: 0832

**Synchronized Freeze Point**

- CRC RANGE OF (SORTED) TRANSACTION ID'S AND CHECK AGAINST ALL OTHER COPIES
- UNIQUE DATA NOT UNIQUE
- TABLE CONSTRAINTS FAIL
- INDEXES MISS SOME OF RECORDS
- LOG DOES NOT MATCH DATA
- (RUN TIME) ORDERED QUERY RESULTS DO NOT AGREE