10. Replication

CSEP 545 Transaction Processing
Philip A. Bernstein

Copyright © 2005 Philip A. Bernstein

1. Introduction

- Replication - using multiple copies of a server (called replicas) for better availability and performance.

- If you're not careful, replication can lead to:
  - worse performance - updates must be applied to all replicas and synchronized
  - worse availability - some algorithms require multiple replicas to be operational for any of them to be used

Replicated Server

- Can replicate servers on a common resource
- Data sharing - DB servers communicate with shared disk

Replicated Resource

- To get more improvement in availability, replicate the resources (too)
- Also increases potential throughput
- This is what's usually meant by replication
- It's the scenario we'll focus on

Replication

Outlines

1. Introduction
2. Primary Copy Replication
3. Multi-Master Replication
4. Other Approaches
5. Products

Replicated Server

- Can replicate servers on a common resource
- Data sharing - DB servers communicate with shared disk

Replicated Resource

- To get more improvement in availability, replicate the resources (too)
- Also increases potential throughput
- This is what's usually meant by replication
- It's the scenario we'll focus on

Replication

Synchronous Replication

- Replicas function just like non-replicated servers
- Synchronous replication - transaction updates all replicas of every item it updates

Issues

- Too expensive for most applications, due to heavy distributed transaction load (2-phase commit)
- Can't control when updates are applied to replicas
Synchronous Replication - Issues

- If you just use transactions, availability suffers.
- For high-availability, the algorithms are complex and expensive, because they require heavy-duty synchronization of failures.
- ... of failures? How do you synchronize failures?

\[
\begin{align*}
\text{x}_1 & \text{ [x}_0, \text{ ] } \text{ y}_0 \text{ fails } \text{ w}_1 \text{ [y}_1, \text{ ] } \\
\text{x}_2 & \text{ [x}_0, \text{ ] } \text{ x}_0 \text{ fails } \text{ w}_2 \text{ [y}_1, \text{ ] }
\end{align*}
\]

Not equivalent to a one-copy execution, even if \( x_0 \) and \( y_1 \) never recover.

- DBMS products support it only in special situations

A Tomicity & Isolation Goal

- One-copy serializability (abbr: 1SR)
  - An execution of transactions on the replicated database has the same effect as an execution on a one-copy database.
  - Intuition: the execution is SR and in an equivalent serial execution, each transaction reads from the most recent transaction that wrote into any copy of its input.
  - To check for 1SR, first check for SR (using SG), then see if there is equivalent serial history with the above property

- Previous example was not 1SR. It is equivalent to:
  - \( \text{r}_1 \text{ [x}_0, \text{ ] } \text{ w}_1 \text{ [y}_1, \text{ ] } \text{ r}_2 \text{ [y}_1, \text{ ] } \text{ w}_1 \text{ [x}_0, \text{ ] } \)
  - but in both cases, the second transaction does not read its input from the previous transaction that wrote that input.

A Tomicity & Isolation (cont'd)

- Although this is not 1SR
  - \( \text{r}_1 \text{ [k}_0, \text{ ] } \text{ w}_1 \text{ [k}_1, \text{ ] } \text{ r}_0 \text{ [y}_0, \text{ ] } \text{ w}_0 \text{ [x}_0, \text{ ] } \)

These are 1SR

- \( \text{r}_1 \text{ [k}_0, \text{ ] } \text{ w}_1 \text{ [k}_1, \text{ ] } \text{ r}_1 \text{ [k}_0, \text{ ] } \text{ w}_1 \text{ [k}_1, \text{ ] } \)

- \( \text{r}_1 \text{ [k}_0, \text{ ] } \text{ w}_1 \text{ [k}_1, \text{ ] } \text{ r}_1 \text{ [k}_0, \text{ ] } \text{ w}_1 \text{ [k}_1, \text{ ] } \)

The previous history is the one you would expect

- Each transaction reads one copy of its readset and writes into all copies of its writeset.
  - readset (resp. writeset) is the set of data items (not copies) that a transaction reads (resp. writes).

- But it may not always be feasible, because some copies may be unavailable.

A Synchronous Replication

- A synchronous replication
  - Each transaction updates one replica.
  - Updates are propagated later to other replicas.

- Each copy: All transactions update the same copy.

- Multi-master: Transactions update different copies.
  - Useful for disconnected operations, partitioned networks.
  - Both approaches ensure that:
    - Updates propagate to all replicas.
    - If new updates stop, replicas converge to the same state.

- Each copy ensures serializability, and often 1SR.
  - Multi-master does not... More later.

2. Primary-Copy Replication

- Designate one replica as the primary copy (publisher).
- Transactions may update only the primary copy.
- Updates to the primary are sent later to secondary replicas (subscribers) in the order they were applied to the primary.

<table>
<thead>
<tr>
<th>T1: Start</th>
<th>Write(x1)</th>
<th>Commit</th>
</tr>
</thead>
<tbody>
<tr>
<td>T2</td>
<td>x1</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>x2</td>
<td>...</td>
</tr>
<tr>
<td>Tn</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Primary Copy

Secondary Copy

Update Propagation

- Collect updates at the primary using triggers or by post-processing the log.

- Triggers
  - On every update at the primary, a trigger fires to store the update in the update propagation table.
  - Post-process ("sniff") the log to generate update propagations.
    - Saves trigger and triggered update overhead during on-line txns.
    - But if log sniffer fails?

- Optionally identify updated fields to compress log.
- Most DB systems support this today.
Update Processing

- At the replica, for each transaction T in the propagation stream, execute a transaction that applies T's updates to the replica.
- Process the stream serially:
  - Otherwise, conflicting transactions may run in a different order at the replica than at the primary.
  - Suppose log contains \( w_1 \cdot c_1 \cdot w_2 \cdot c_2 \). Obviously, \( T_1 \) must run before \( T_2 \) at the replica.
  - So the execution of update transactions is serial.

Update Processing (cont'd)

- To get a 1SR execution at the replica:
  - Update transactions and read-only queries use an atomic and isolated mechanism (e.g., using 2PL).
- Why this works:
  - The execution is serializable.
  - Each state in the serial execution is one that occurred at the primary copy.
  - Each query reads one of those states.

Request Propagation

- An alternative to propagating updates is to propagate procedure calls (e.g., a DB stored procedure call).
- Or propagate requests (e.g., transaction-bracketed stored proc calls).
- Must ensure requests run in the same order at primary and replica (same requirement as updates or procedure calls).
- As for updates, can propagate requests asynchronously, or:
  - can run requests synchronously at all replicas, just as in an update procedure.
  - If supported, it's often an app server (not DB) feature.

Failure & Recovery Handling

- Secondary failure - nothing to do till it recovers:
  - At recovery, apply the updates it missed while down.
  - Needs to determine which updates it missed, just like non-replicated log-based recovery.
  - If down for too long, it may be faster to get a whole new copy.
- Primary failure:
  - Normally, secondaries just wait till the primary recovers.
  - Can get higher availability by electing a new primary:
    - A secondary that detects the primary's failure announces a new election by broadcasting its unique replica identifier.
    - Other secondaries reply with their replica identifier.
    - The largest replica identifier wins.

Failure Handling (cont'd)

- Primary failure (cont'd):
  - All replicas must now check that they have the same updates from the failed primary.
  - During the election, each replica reports the id of the last log record it received from the primary.
  - The most up-to-date replica sends its latest updates to (at least) the new primary.
  - Could still lose an update that committed at the primary and wasn't forwarded before the primary failed.
  - But solving it requires synchronous replication.

Communications Failures

- Secondaries can't distinguish a primary failure from a communication failure that partitions the network.
- If the secondaries elect a new primary and the old primary is still running, there will be a reconciliation problem when they're reunited. This is multi-master.
- To avoid this, one partition must know it's the only one that can operate, and can't communicate with other partitions to figure this out.
- Dynamic solutions are based on Majority Consensus.
Majority Consensus
- Whenever a set of communicating replicas detects a replica failure or recovery, they test if they have a majority (more than half) of the replicas.
- If so, they can elect a primary.
- Only one set of replicas can have a majority.
- Doesn’t work with an even number of copies.
- Use a quorum consensus.
- Provides a weight to each replica.
- The replica set that has a majority of the weights wins.
- E.g., 2 replicas, one has weight 1, the other has weight 2.

3. Multi-Master Replication
- Some systems must operate when partitioned.
  - Requires many updatable copies, not just one primary.
  - Conflicting updates on different copies are detected late.
- Classic example: salesperson’s disconnected laptop.
  - Customer table (updated).
  - Orders table (inserts mostly).
  - Customer log table (append only).
- Use a multi-copy algorithm, with multiple masters.
  - Each master exchanges updates (“gossips”) with other masters when it reconnects to the network.
  - Conflicting updates require reconciliation (i.e., merging).
- In Lotus Notes, Access, SQL Server, Oracle, ...

Example of Conflicting Updates

A Classic Race Condition

<table>
<thead>
<tr>
<th>Replica 1</th>
<th>Primary</th>
<th>Replica 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initially x=0</td>
<td>Initially x=0</td>
<td>Initially x=0</td>
</tr>
<tr>
<td>T1: X=1</td>
<td>T2: X=2</td>
<td>T2: X=1</td>
</tr>
<tr>
<td>Send (X=1)</td>
<td>Send (X=1)</td>
<td>Send (X=2)</td>
</tr>
<tr>
<td>X=1</td>
<td>X=2</td>
<td>X=1</td>
</tr>
<tr>
<td>X=1</td>
<td>Send (X=2)</td>
<td>Send (X=2)</td>
</tr>
<tr>
<td>X=2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Replicas end up in different states.

Thomas Write Rule
- To ensure replicas end up in the same state:
  - Tag each data item with a timestamp.
  - A transaction updates the value and timestamp of data items (time stamps monotonically increase).
  - An update to a replica is applied only if the update’s timestamp is greater than the data item’s timestamp.
  - You only need the latest update of data items that were recently updated whose older update could still be floating around the system.
- A linest multi-master product uses a variation of this.
- Robert Thomas, ACM TODS, June ’79.

Thomas Write Rule ⇒ Serializability

<table>
<thead>
<tr>
<th>Replica 1</th>
<th>Primary</th>
<th>Replica 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1: read x=0 (TS=0)</td>
<td>Initially x=0, TS=0</td>
<td>T1: read x=0 (TS=0)</td>
</tr>
<tr>
<td>T2: X=1, TS=1</td>
<td>T2: X=2, TS=2</td>
<td>T2: X=2, TS=2</td>
</tr>
<tr>
<td>Send (X=1, TS=1)</td>
<td>Send (X=1, TS=2)</td>
<td>Send (X=1, TS=2)</td>
</tr>
<tr>
<td>X=1, TS=1</td>
<td>X=2, TS=2</td>
<td>X=1, TS=1</td>
</tr>
<tr>
<td>X=2, TS=2</td>
<td>Send (X=2, TS=2)</td>
<td>Send (X=2, TS=2)</td>
</tr>
</tbody>
</table>

Replicas end in the same state, but neither T1 nor T2 sees the other’s output, so the execution isn’t serializable.

Multi-Master Performance
- The longer a replica is disconnected and performing updates, the more likely it will need reconciliation.
- The amount of propagation activity increases with more replicas.
  - If each replica is performing updates, the effect is quadratic.
Microsoft Access and SQL Server

- Multi-master replication without a primary
- Each row R of a table has 4 additional columns
  - globally unique ID (GUID)
  - generation number, to determine which updates from other replicas have been applied
  - version number = the number of updates to R
  - array of [replica, version number] pairs, identifying the largest version number it got for R from every other replica
- Use Thomas’ write rule, based on version numbers
  - Access uses replica ID to break ties. SQL Server 7 uses subscriber priority or custom conflict resolution.

Generation Numbers (Access/SQL cont’d)

- Each replica has a current generation number.
- A replica updates a row’s generation number whenever it updates the row.
- A replica knows the generation number it had when it last exchanged updates with R.
- A replica knows the generation number it received every time it exchanged updates with another replica.
- So, when exchanging updates with R, it should send all rows with a generation number larger than what it had when it last exchanged updates with R.

Duplicate Updates (Access/SQL cont’d)

- Some rejected updates are saved for later analysis.
- To identify duplicate updates to discard them:
  - When applying an update to x, replace x’s array of [replica, version#] pairs by the update’s array.
  - To avoid processing the same update via many paths, check version number of arriving update against the array.
- Consider a rejected update to x at R from R’, whose
  - [R’, V] describes R’ in x’s array, and
  - V is the version number sent by R’.
  - If V = V’, then R saw R’’s updates.
  - If V < V’, then R didn’t see R’’s updates, so store it in the conflict table for later reconciliation.

4. Other Approaches

- Non-transactional replication using time-stamped updates and variations of Thomas’ write rule.
- Directory services are managed this way.
- Quorum consensus per-transaction:
  - Read and write a quorum of copies.
  - Each data item has a version number and time stamp.
  - Each read chooses a replica with the largest version number.
  - Each write increments version number one greater than any one it has seen.
  - No special work needed during a failure or recovery.

Other Approaches (cont’d)

- Read-one replica, write-all-available replicas.
  - Requires careful management of failures and recoveries.
- E.g., Virtual partition algorithm:
  - Each node knows the nodes it can communicate with, called its view.
  - Transaction T can execute if its home node has a view including a quorum of T’s readset and writeset.
  - If a node fails or recovers, run a view formation protocol, much like an election protocol.
  - For each data item with a read quorum, read the latest version and update the others with a lesser version if.

Summary

- State-of-the-art products have rich functionality.
  - It’s a complex world for app designers.
  - Lots of options to choose from.
- Most failover stories are weak.
  - Fine for data warehousing.
  - For 24/7 TP, need better integration with cluster node failover.
5. Products

- All major DBMS products have a rich primary-copy replication mechanism. These are big subsystems.
- Differences are in detailed features:
  - performance
  - ease of management
  - richness of filtering predicates
  - push vs. pull propagation
  - stored procedure support
  - transports (e.g., Sybase SQL Anywhere can use email!)
- The following sum may be an incomplete snapshot of products as of May 2003.

Microsoft SQL Server 2000

- Publication - a collection of articles to subscribe to
- Article - a horizontal table slice or stored proc
  - Customizable table filter (WHERE clause or stored proc)
  - Stored proc may be transaction protected (replicate on commit).
  - Replicates the requests instead of each update.
- Snapshot replication makes a copy
- Transactional replication maintains the copy by propagating updates from publisher to subscribers:
  - Post-processes log to store updates in Distribution DB
  - Distribution DB may be separate from the publisher DB
  - Updates can be pushed to or pulled from subscriber
  - Subscribers can customize propagated updates using stored procedures

SQL Server 2000 (cont’d)

- Immediate updating subscriber - can update replicas
  - Queued updates are synchronized with publisher via 2PC.
  - Triggers capture local updates and forward them to the Subscriber using a not null foreign key for replicated updates from the publisher.
  - Subscriber's local update has before-value of row version-id.
  - Publisher checks that its copy of row has the same version-id.
  - If not, it performs the update and asynchronously forwards it to other subscribers
  - If not, it aborts the transaction (subscriber updated the row lately)
- Access control lists protect publishers from unauthorized subscribers
- Merge replication - described later (in multi-master)

Oracle 9i

- Like SQL Server, can replicate updates to table fragments or stored procedure calls at the multi-master copy
  - Uses triggers to capture updates in a deferred queue
    - Updates are row-oriented, identified by primary key
  - Can optimize by sending keys and updated columns only
  - Group updates by transaction, which are propagated:
    - Either serially in commit order or in parallel, with dependent transaction ordering:
      - Each read(x) reads the "commit number" of x;
      - Updates are ordered by dependent commit number
  - Replicas are implemented as materialized views
  - Replicas are updated in a batch refresh:
    - Pushed from master to snapshots, using queue scheduler
  - Replicas can be updatable (via bar to SQL Server)

Oracle 9i Conflict Resolution

- Conflict resolution strategies (defined per column-group):
  - A delta difference between the old and new values of the originating site to the destination site
  - A hash of the value of the current site and the originating site
  - A unique key generated from the two values
    - The one with the lowest hash value wins
    - The site with the value wins if there is a priority
    - Can apply methods in sequence, e.g., by row ID, then by priority.
  - Can call custom procedures to log, notify, or resolve the conflict:
    - Passed overall update’s before/after value and row’s current value
  - For a given update, if no built-in or custom conflict resolution applies, then the entire transaction is logged.
**IBM DB2**

- Very similar feature set to SQL Server and Oracle
- Filtered subscriber
  - Create snapshot, then update incrementally (push or pull)
- Many table type options:
  - Read-only snapshot copy, optionally with timestamp
  - Aggregates, with cumulative or incremental values
  - Consistent-change data, optionally with row versions
  - "Replica" tables, form utility aster updating
- Interoperates with many third-party DBMS’s
- Captures DB2 updates from the DB2 log
  - For other systems, captures updates using triggers