Middle-R: A Middleware for Scalable Database Replication

Ricardo Jiménez-Peris

Lsd Distributed Systems Laboratory

Universidad Politécnica de Madrid (UPM)

Joint work with Bettina Kemme, Marta Patiño, Yi Lin and Damian Serrano.
Content

• Motivation
• Snapshot Isolation
• One Copy Snapshot Isolation
• Partial Replication
• Other Issues
Motivation

• Current solutions to data replication exhibit limited scalability due to three main reasons:
  – Restrictive concurrency/replica control.
  – Full replication.
  – Engineering of data replication (e.g. writeset extraction).

• In this talk we address all these issues.
Concurrency and Replica Control Bottlenecks

• 1-copy serializability has been the traditional correctness criterion.

• What is wrong with it?
  – Read-write conflicts $\rightarrow$ restricted potential concurrency
  – And in a replicated setting…

• What is the alternative?
  – Snapshot isolation.
  – Oracle success story.
Serializability

- The result is the same as executing them serially.
- Conflicts: *read/write* and *write/write*
Snapshot Isolation

- Read from a snapshot of the committed data when the transaction starts
- Conflict: **only write/write**

![Diagram showing a conflict scenario with T0, T1, and T2 transactions, illustrating write/write conflicts.](image)
Snapshot Isolation: Atomicity

• Provides to each transaction a snapshot of the committed state of the database as of transaction start.
• Thanks to multi-versioning it avoids read-write conflicts.
• Concurrent writes forbidden: write-write conflicts.
• Splits atomicity it two points:
  – One point at the start of the transaction where all reads happen atomically.
  – Another point at commit time where all writes happen atomically.
Snapshot Isolation: What is lost? Write Skew

We want to enforce the integrity constraint: \( x+y \geq 0 \)

\[ x+y=50+50>100 \]

Initial values

- \( x=50 \)
- \( y=50 \)

- \( x+y=50+50>100 \)

Two transactions:

- T1: Extract 50 Euros from X.
- T2: Extract 80 Euros from Y.

\[ x+y=0+50>0 \]

\[ x+y=-30<0 \]

T1

Initial values:

\( x=50 \)
\( y=50 \)

Final values:

\( x=0 \)
\( y=-30 \)
1-Copy-SI [SIGMOD’05, TODS’09]

• 1-Copy-SI defines 1-copy correctness for SI databases.
• It allows all possible executions in a centralized SI DB and disallows all other executions.
• In practical terms:
  – The DB at each replica provides SI.
  – The system guarantees that read operations see a globally consistent (1-copy equivalent) set of snapshots.
  – Distributed concurrent txns are not allowed to modify the same tuples.
Implementing 1-Copy-SI: Consistency

• To disallow that distributed concurrent txns modify the same tuples:
  – A global validation is needed.

• In order to guarantee that read operations see a globally consistent set of snapshots:
  – The order in which local transactions are started and remote transactions committed at each replica should be constrained.
Implementing 1-Copy-SI: Main Components

- Clients
  - Remote txns
  - Local txns

- Validator
  - Remote txns
  - Local txns

- Replica Control
  - txns
  - upds

BD

- Clients
  - Remote txns
  - Local txns

- Replica Control
  - txns
  - upds

BD
Implementing 1-Copy-SI: Protocol

txn1: w(X)

txn2: commit

Get WS(txn1)

commit(txn1)

Get WS(txn2)

WS1(X)

validate

apply WS1(X)

and commit

txn2: w(X)

commit

abort(txn2)

commit(txn1)

abort
Implementing 1-Copy-SI: Consistency

- Committing all txns in the same order has some cost. Would it be enough to commit write-conflicting txns in the same order?

- It does not guarantee a 1-copy consistent set of snapshots for reads!!!
1-Copy-SI: Replica Control

- One solution:
  - Commit update txns in the same order at all replicas.
- Problem: SI DBs use locking for writes.

  → Deadlocks become possible between the local DBs and the replication logic.

<table>
<thead>
<tr>
<th>Transaction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ti W(x)</td>
<td>Local txn</td>
</tr>
<tr>
<td>Tj W(y)</td>
<td>Local txn</td>
</tr>
<tr>
<td>Tr W(y)</td>
<td>Remote txn successfully validated blocked on W(y)</td>
</tr>
<tr>
<td>Ti finishes execution, waits for Tr to complete to write x</td>
<td></td>
</tr>
<tr>
<td>Tj W(x)</td>
<td>Gets blocked on W(x) waiting for Ti</td>
</tr>
</tbody>
</table>

Tr → Tj → Ti → Tr    Deadlock!!!
1-Copy-SI: Replica Control

• The alternative is to allow non-conflictive update txns to commit in any order (e.g. Tr and Ti in the deadlock example).

• But it is necessary to delay the start of local txns whilst there are holes in the commit order (i.e. the set of committed transactions is not a prefix of the sequence of validated transactions).

• Replicas will therefore need to alternate between parallel commitment of update transactions and start of new transactions.
1-Copy-SI: Validation

- Validation of updates is required to be done at the global level.
- It can be centralized or decentralized: Middle-R has decentralized validation that is inherently fault-tolerant.
- In the decentralized case, txn updates sent for validation are sent in total order to guarantee validation determinism.
The Limits of Full Replication

- Updates impose a fraction of remote (not productive) work
- The more copies, the more remote work
  - Full replication can scale to 20-30 replicas depending on the workload.
- Alternative: partial replication
Partial Replication

- Distributed transactions might be needed!!
Hybrid Replication

- Transactions can be executed either at partial replicas or at full replicas.
- Full replicas prevent distributed transactions.
- Partial replicas provide scalability.
Full vs. Hybrid vs. Partial

20% updates
Replication degree 10
Issues with Partial Replication

• Hybrid replication prevents distributed transactions but full replicas become a bottleneck.
• Pure partial replication scales well but distributed txns are possible.
• How to provide 1CSI for a partially replicated system?
  – When a txn becomes distributed it will get different snapshots at different replicas.
  – How to provide a consistent snapshot for distributed txns?
  – Can validation be partial (i.e. non-global)? → Not in general.
• If a site stores a replica of an object, it must receive writesets from objects in the same sub-graph
• But, if transaction access patterns are unknown every replica needs all the writesets
Preserving Snapshots

• Global consistent snapshot:
  – Distributed transactions must be executed in the proper snapshot in the participants.

• Session consistency:
  – Snapshots must be updated with former updates before executing txns for transaction session consistency.
Preserving Snapshots

• Dummy transactions
  – Set of transactions that are begun for each snapshot
  – A transaction is assigned to a snapshot when starts
  – An operation of a transaction must get the snapshot before being executed at a site.
    • Transaction aborts may occur because of unavailable snapshots
  – Garbage collector closes unused dummy transactions based on piggybacked info
Scalability: Evaluation Setup

• Using TPC-W: benchmark that simulates an Amazon-like online book store
  – Database backend: PostgreSQL 7.2
  – Servlet server: Tomcat 5.5.9
  – TPC-W tables are partitioned by tuples

• Homogeneous sites running Fedora Core 3
  – Two processors AMD Athlon 2GHz, 1 GB of RAM
  – 100Mb Ethernet
12 Replicas: Throughput

Throughput (12 Sites)

- Throughput (WIPS)
- Load (tps)

Graph showing the throughput (WIPS) as a function of load (tps) for different replication factors (r = 1, r = 2, r = 4, r = 6, Full). The graph illustrates the performance improvement with increasing replication.
12 Replicas: Response Time

Response Time (12 Sites)

Response Time (ms)

Load (tps)

Full
r = 6
r = 4
r = 2
r = 1
Scale Out

Analytical vs. Empirical Scale Out

Number of Sites (N)

Scale Out

Analytical Full
Analytical r=2
Analytical r=P/2
Empirical Full
Empirical r = 2
Empirical r = N/2
**Database Replication Architectural Approaches**

**White Box**
- **Pros:**
  - It takes advantage of many optimizations performed within the database kernel.
- **Cons**
  - It requires access to source code.
  - It is tightly integrated with the implementation of the regular database functionality.

**Black Box**
- **Pros:**
  - The replication is performed outside the database.
- **Cons:**
  - The database is used exclusively based on its user interface.

**Proposed approach:** Combine advantages of both approaches in a **Gray Box** approach.
Writeset Extraction

• The writeset can be obtained in a standard way by using triggers (black box - standard).

• The writeset in some DBs can be obtained through proprietary log mining facilities (black box – non-standard).

• A function can be implemented in the DB to extract the writeset (gray-box) either in binary or in textual form (SQL update where primary key statement).
Cost of Writeset Extraction

Commercial DB
Triggers, log, no extraction

MySQL
SQL extraction service, no extraction
Cost of Writeset Extraction

PostgreSQL

Binary ws service, trigger, no extraction
Towards MiddleCloud

• High availability does not only require fault-tolerance but online recovery.
• Availability is not the right measure, but performability.
• We have developed online recovery for our replication protocols that is able to provide a peak performance close to 90-95% the one of the system without recovery.
Towards MiddleCloud

• Elasticity:
  – Self-provisioning:
    • Adding new replicas with disrupting the peak performance nor violating consistency.
    • Detecting when the load has lowered to the point one replica can be decommissioned.
  – Self-configuration of partial replicas.
    • Deciding where to create replicas dynamically.
Towards MiddleCloud: A PostgreSQL Cloud

• Replication in across data centers requires special care.
• Group communication across WANs performs badly.
• Centralized validator outperforms other architectural approaches.
• Sticking to one WAN interaction per update txn and local txn execution is crucial.
• Interconnection of clusters require hierarchical replication protocols aware of the network topology (local sites vs. remote sites).
Multi-tier Replication

- High availability for multi-tier architectures introduces interesting problems.
- The application server caches data items.
- Current applications servers (AS) when combined with SI databases do not work!!!
- We have developed SI-aware caching that provides end-to-end SI consistency.
- We have developed a replication solution for J(2)EE that replicate pairs of application servers and databases guaranteeing 1CSI.
Questions?