DATABASE REPLICATION
A TALE OF RESEARCH ACROSS COMMUNITIES

Bettina Kemme
Dept. of Computer Science
McGill University
Montreal, Canada

Gustavo Alonso
Systems Group
Dept. of Computer Science
ETH Zurich, Switzerland

© ETH, Zurich and McGill University, Montreal

VLDB 2010 - Singapore
Across communities

Postgres-R (Dragon project)
Protocols [Kemme, Alonso, ICDCS’98]
Implementation [Kemme, Alonso, VLDB2000]
Postgres-R

- Intro to Replication
- Postgres-R
- In perspective
- Systems Today
- The next 10 years
A brief introduction to database replication
- Scalability
- Fault-tolerance
- Fast access
- Special purpose copies
Primary Copy vs. Update Everywhere
Eager (synchr.) vs. Lazy (asynchr.)
Theory of replication 10 years ago

<table>
<thead>
<tr>
<th></th>
<th>PRIMARY COPY</th>
<th>UPDATE EVERYWHERE</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAGER</td>
<td><img src="image" alt="Eager Icon" /></td>
<td><img src="image" alt="Update Icon" /></td>
</tr>
<tr>
<td>LAZY</td>
<td><img src="image" alt="Lazy Icon" /></td>
<td><img src="image" alt="Update Icon" /></td>
</tr>
</tbody>
</table>
### Replication in practice 10 years ago

<table>
<thead>
<tr>
<th>EAGER</th>
<th>PRIMARY COPY</th>
<th>UPDATE EVERYWHERE</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Ghost" /></td>
<td><img src="image" alt="Boo" /></td>
<td><img src="image" alt="Devil" /></td>
</tr>
<tr>
<td>LAZY</td>
<td><img src="image" alt="Smiley" /></td>
<td><img src="image" alt="Don't Forget" /></td>
</tr>
</tbody>
</table>

- PRIMARY COPY: Boo!
- UPDATE EVERYWHERE: Devil with trident and text 'Don’t Forget!'
Replication in 2000
- Read-one / Write All
- Distributed locking (writes)
- 2 Phase Commit
The Dangers of Replication and a Solution

Jim Gray (Gray@Microsoft.com)
Pat Helland (PHelland@Microsoft.com)
Patrick O’Neil (POneil@cs.UMB.edu)
Dennis Shasha (Shasha@cs.NYU.edu)

Abstract: Update anywhere-anytime-anyway transactional replication has unstable behavior as the workload scales up: a ten-fold increase in nodes and traffic gives a thousand fold increase in deadlocks or reconciliations. Master copy replication (primary copy) schemes reduce this problem. A simple analytic model demonstrates these results. A new two-tier replication algorithm is proposed that allows mobile (disconnected) applications to propose tentative update transactions that are later applied to a master copy. Commutative update transactions avoid the instability of other replication schemes.

1. Introduction

Eager replication delays or aborts an uncommitted transaction if committing it would violate serialization. Lazy replication has a more difficult task because some replica updates have already been committed when the serialization problem is first detected. There is usually no automatic way to reverse the committed replica updates, rather a program or person must reconcile conflicting transactions.

To make this tangible, consider a joint checking account you share with your spouse. Suppose it has $1,000 in it. This account is replicated in three places: your checkbook, your spouse’s checkbook, and the bank’s ledger.
Response Time and Messages

centralized database

T=

replicated database

T=

update: 2N messages

2PC
... and that’s not all

- Network becomes an issue
  - Messages = copies × write operations

- Quorums?
  - Reads must be local for complex SQL operations
  - Different in key value stores (e.g., Cassandra)
Our goal

Can we get scalability and consistency when replicating a database?
Postgres-R in detail
Fundamentals

- Exploitation of group communication systems
  - Ordering semantics
    - Affect isolation / concurrency control
  - Delivery semantics
    - Affect atomicity
Key insight in Postgres-R

**BEFORE**

Scheduler

Scheduler

**AFTER**

Pre-Ordering Mechanism

Scheduler

Scheduler
The devil is in the details

- Total order to serialization order
- Provide various levels of isolation / atomicity degrees
- Read operations always local
- Propagate changes on transaction basis
- No 2PC
  - Rely on delivery guarantees
  - Return to user once local replica commits
- Determinism
- Propagate changes vs. SQL statements
Distributed locking

Did we really avoided the dangers of replication?

Postgres-R removed a lot of the overhead of replication, providing scalability while maintaining strong consistency
In perspective
What worked

- Ordered and guaranteed propagation of changes through an agreement protocol external to the engine

- The implementation was crucial to prove the point

- Thinking through the optimizations / real system issues

- Levels of consistency
What did not work

- Modify the engine
  - Today middleware based solutions

- Enforce serializability
  - Today SI and session consistency
  - Data warehousing less demanding
  - Cloud computing has lowered the bar
Systems today
Very rich design space

- Applications (OLAP vs OLTP)
- Data layer (DB vs. others)
- Throughput/Response time
- Staleness
- Availability guarantees
- Partial vs. full replication
- Granularity of changes, operations
### A Suite of Replication Protocols

<table>
<thead>
<tr>
<th>Serializability</th>
<th>Correctness</th>
<th>Local decisions</th>
<th>Mssgs/txn</th>
<th>Problems</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-copy serializeability</td>
<td>write locks abort conflicting local read locks</td>
<td>2 mssgs write set confirm commit</td>
<td>very high abort rate</td>
<td></td>
</tr>
<tr>
<td>Cursor Stability</td>
<td>no lost updates no dirty reads</td>
<td>write locks abort conflicting local read locks</td>
<td>2 mssgs write set confirm commit</td>
<td>inconsistent reads</td>
</tr>
<tr>
<td>Snapshot isolation</td>
<td>allows write skew</td>
<td>first writer to commit wins (deterministic)</td>
<td>1 mssg write set</td>
<td>high abort rate for update hot-spots</td>
</tr>
<tr>
<td>Hybrid</td>
<td>1-copy serializeability</td>
<td>as serializability for update transactions</td>
<td>2 mssgs write set confirm commit</td>
<td>requires to identify the queries</td>
</tr>
</tbody>
</table>

6 versions of each protocol depending on delivery guarantees and whether deferred updates or shadow copies are used (22 protocols) [Kemme and Alonso, ICDCS’98]
Eventually all this moved to a middleware layer above the database

- Middle-R
- All systems out there today
Consistency variations

- Snapshot isolation,
- optimistic cc,
- conflict detection,
- relaxed consistency

- Middle-R
- GORDA project
- Tashkent
- Pronto/Sprint
- C-JDBC
- C. Amza et al.
Architectural variations

- Ganymed
- DBFarm
- SQL Azure
- Zimory (virtualized satellites)

Primary copy
Specialized satellites
Engine variations

- Xkoto (Teradata)
- Galera
- Continuent
- SQL Azure
- Ganymed

MySQL, DB2, SQL Server, Oracle, Heterogeneous
Change unit variations

- Xkoto (Teradata)
  - SQL statement propagation
- Continuent
  - Log capture
- Determinism!
Cloud solutions

- PAXOS

- Cassandra
- PNUTS
- Big Table
- Cloudy
- ...

Key value stores, files, tables
With a lot of related topics

- Consistency (Khuzaima et al., Cahill et al.)
- Applications (Vandiver et al.)
- Agreement protocols (Lamport et al.)
- Determinism (Thomson et al.)
- Recovery (Kemme et al., Jimenez et al.)

...
The next 10 years
Understanding the full picture

- Paxos- Group communication protocols differ in the exact properties they provide
  - Often difficult to understand for outsiders
    - Subtleties in implementation and efficiency
  - Complex implementations
- Adjusting the agreement protocols to the needs of databases
  - Properties that suffice
  - Efficient implementation
- Plenty of use cases (one size does not fit all)
Database and distributed systems

- Databases and distributed systems have converged in practice
  - Many similar concepts
- Research
  - Work still done in separate communities
- Teaching
  - Dire need for joint courses (thanks to Amr El Abbadi and Divy Agrawal from UCSB!)
Thanks

- Andre Schiper, Fernando Pedone, Matthias Wiesmann (EPFL)
- Marta Patiño, Ricardo Jiménez (UPM)
- The PostgreSQL community
- PhD and master students at ETH and McGill who have worked and are working on related ideas
- Many colleagues and friends ...