Database replication for commodity
database services

Gustavo Alonso
Department of Computer Science
ETH Zürich
alonso@inf.ethz.ch
http://www.iks.ethz.ch

An appeal to databases

- From Adam Bosworth’s blog:
  http://www.adambosworth.net/archives/000038.html
- What commercial databases should provide (but don’t):
  - Dynamic schema so that as the business model/description of
    goods or services changes and evolves, this evolution can be
    handled seamlessly in a system running 24 by 7, 365 days a
    year
  - Dynamic partitioning of data across large dynamic numbers of
    machines.
  - Modern indexing.
  - Indeed, in these days of open source, I wonder if the software
    itself, should cost at all? Open Source solutions would
    undoubtedly get hacked more quickly to be robust and truly
    scalable across nice simple software.

Background

- "One size fits all: an idea
  whose time has come and
  gone" (M. Stonebraker)
- Limited growth in commercial
  products leading to collections
  of specialized servers (M.
  Kersten, INS-R9905 CWI)
- Several open source projects
  on extracting data from
  commercial engines and
  placing it on open source
  databases
- User requirements:
  - Consistency is good
  - Constant need for new
    functionality
  - Commercial db engines
    evolve too slowly
  - Data blades, extensions,
    additional code impractical
    (impact on running server)
  - Flexible scalability (cost of
    over-provisioning is very
    high)
  - Open source solutions
    (reduced cost, chance to
    tailor)
  - Scale out + specialization
Our solution: open source satellite databases

Databases as commodity service

- Remote applications use the database through a web services enabled JDBC driver (WS-JDBC)

EXTENSIBILITY: open source satellites

- Keyword searches, information retrieval
- Skyline queries, specialized indexes
- Lineage and provenance

Additional, specialized functionality running on open source engines kept consistent with the master database (a commercial engine).

Some comments

- The first goal (autonomic cluster of satellite databases) is more complex and difficult to solve from both the technical as well as the application (business model) point of view
- The second goal (specialized satellites) is easier to solve and the argument for this solution is much simpler to make
- If we can achieve the first goal, the second comes almost for free
How to replicate data?

- Depending on when the updates are propagated:
  - Synchronous (eager)
  - Asynchronous (lazy)
- Depending on where the updates can take place:
  - Primary Copy (master)
  - Update Everywhere (group)

Theory ...

- The name of the game is **correctness and consistency**
- Synchronous replication is preferred:
  - copies are always consistent (1-copy serializability)
  - programming model is trivial (replication is transparent)
- Update everywhere is preferred:
  - system is symmetric (load balancing)
  - avoids single point of failure
- Other options are ugly:
  - inconsistencies
  - centralized
  - formally incorrect

... and practice

- The name of the game is **throughput and response time**
- Asynchronous replication is preferred:
  - avoid transactional coordination (throughput)
  - avoid 2PC overhead (response time)
- Primary copy is preferred:
  - design is simpler (centralized)
  - trust the primary copy
- Other options are not feasible:
  - overhead
  - deadlocks
  - do not scale
The dangers of replication ...

SYNCHRONOUS
- Coordination overhead
  - distributed 2PL is expensive
  - 2PC is expensive
  - prefer performance to correctness
- Transactions last longer (and therefore have more conflicts)
- Communication overhead
  - 5 nodes, 100 tps, 10 w/txn = 5’000 messages per second !!

UPDATE EVERYWHERE
- Deadlock/Reconciliation rates
  - the probability of conflicts becomes so high, the system is unstable and does not scale
- Useless work
  - the same work is done by all nodes
  - administrative costs paid by all nodes

Research
- Much work playing with relaxed forms of consistency:
  - Demarcation Protocol: asynchronous when values within certain range, synchronous to change the range
  - Coordinated propagation: asynchronous but propagation of changes has to be done in certain way to ensure some form of consistency
  - ...
- Many solutions are application specific
  - Static and dynamic web content
  - Wide area data caching
  - Wireless networks
- Unfortunately, most of the existing work on replication has never been implemented
  - Realistic workloads?
  - Overhead at the master?
  - Practical feasibility (overhead of the mechanism)?

GANYMED: efficient conventional replication

Consistency vs. Performance
- We want both:
  - Consistency is good for the application
  - Performance is good for the system
- Then:
  - Let the application see a consistent state ...
  - ... although the system is asynchronous and primary copy
- This is done through:
  - A middleware layer that offers a consistent view
  - Using snapshot isolation as correctness criteria
Two sides of the same coin

**SNAPSHOT ISOLATION**
- To the clients, the middleware offers snapshot isolation:
  - Queries get their own consistent snapshot (version) of the database
  - Update transactions work with the latest data
  - Queries and updates do not conflict (operate on different data)
  - First committer wins for conflicting updates
- PostgreSQL, Oracle, MS SQL Server

**ASYNCH – PRIMARY COPY**
- Primary copy: master site where all updates are performed
- Slaves: copies where only reads are performed
- A client gets a snapshot by running its queries on a copy
- Middleware makes sure that a client sees its own updates and only newer snapshots
- Updates go to primary copy and conflicts are resolved there (not by the middleware)
- Updates to master site are propagated lazily to the slaves

Ganymed: Putting it together

- Based on standard JDBC drivers
- Only scheduling, no concurrency control, no query processing...
- Simple messaging, no group communication
- Very much stateless (easy to make fault tolerant)
- Acts as traffic controller and bookkeeper
- Route queries to a copy where a consistent snapshot is available
- Keep track of what updates have been done where (propagation is not uniform)

Where are we different?

- Consistency:
  - Clients see a consistent database
  - Clients see only one database not a master and some replicas
  - This is extremely important in practice
- Simplicity:
  - This is not a parallel database (each transaction or query runs on a single database)
  - In doubt, send it to the master
  - General approach (update extraction is through triggers or sql propagation, not through the log –can be done and is more efficient but we do not want to go down that path yet)
- Middleware approach through standard JDBC driver
  - Applications do not have to change
  - The middleware layer gives extensibility, something most database replication systems lack
- Applicable to commercial engines and open source (cross replication)
Experiments

- TPC-W ordering, shopping and browsing traces
- PostgreSQL, Oracle, DB2
- 100 clients running the traces
  - Clients send both updates and reads
  - Clients block if master is slow applying the writes
- Measured
  - Throughput
  - Response time
  - … for:
    - Database alone (base line)
    - Database with Ganymed but no satellites (overhead)
    - Database with Ganymed and satellites (1-6) (gain if any)

Linear scalability (PostgreSQL)

- Improvements in response time (!!!)

Fault tolerance (slave failure)
Fault tolerance (master failure)

GANYMED: Heterogeneous master and satellite databases

Satellite databases

- A satellite database is an open source replica of a commercial engine
- Basic idea remains the same
  - Commercial engine is the main copy
  - Satellites contain snapshots
  - Ganymed provides consistent snapshots to the clients
- On a first approximation, satellites are full copies used for executing queries
- Using only generic solutions, not system specific tools
- The challenges with commercial engines are:
  - Update extraction without introducing too much overhead
  - SQL dialects and query optimizations

Oracle master – PostgreSQL satellites
Oracle master – PostgreSQL satellites

Updates through SQL (Oracle-Postgres)

Updates through SQL (Oracle-Postgres)

DB2 master – PostreSQL satellites
Critical issues

- By combining a commercial master with open source satellites we obtain a very powerful system
- More work needs to be done (in progress)
  - Trigger based = attach triggers to tables to report updates (low overhead at slaves, high overhead at master)
  - Generic = propagate update SQL statements to copies (high overhead at slaves, no overhead at master, limitations with hidden updates)
  - Update propagation = tuple based vs SQL based
  - SQL is not standard (particularly optimized SQL)
  - Understanding workloads (how much write load is really present in a database workload)
  - Replicate only parts of the database (table fragments, tables, materialized views, indexes, specialized indexes on copies ...)

SQL is not SQL

```sql
SELECT * FROM (SELECT i_id, i_title, a_fname, a_lname,
SUM(ol_qty) AS orderkey
FROM item, author, order_line
WHERE i_id = ol_i_id AND i_a_id = a_id
AND ol_o_id > (SELECT MAX(o_id) - 3333 FROM orders)
AND i_subject = 'CHILDREN'
GROUP BY i_id, i_title, a_fname, a_lname
ORDER BY orderkey DESC)
WHERE ROWNUM <= 50
```

Amongst the 3333 most recent orders, the query performs a TOP-50 search to list a category’s most popular books based on the quantity sold.

Current version does very basic optimizations on the slave side. Further work on optimizations at the middleware layer will boost performance even more.

Optimizations can be very specific to the local data.

Use of MAX leads to sequential scan in PostgreSQL, change to:

```sql
SELECT * FROM (SELECT i_id, i_title, a_fname, a_lname,
SUM(ol_qty) AS orderkey
FROM item, author, order_line
WHERE i_id = ol_i_id AND i_a_id = a_id
AND ol_o_id > (SELECT MAX(o_id) - 3333 FROM orders)
AND i_subject = 'CHILDREN'
GROUP BY i_id, i_title, a_fname, a_lname
ORDER BY orderkey DESC)
WHERE ROWNUM <= 50
```

Virtual column specific to Oracle.
In PostgreSQL = LIMIT 50

In PostgreSQL = LIMIT 1
GANYMED: The easy part (but the most profitable?)

Understanding workloads

<table>
<thead>
<tr>
<th>Workload</th>
<th>TPC-W</th>
<th>Updates</th>
<th>Read-only</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ordering</td>
<td>3.21 %</td>
<td>96.79 %</td>
<td>1 : 30.16</td>
<td></td>
</tr>
<tr>
<td>Shopping</td>
<td>10.77 %</td>
<td>89.23 %</td>
<td>1 : 8.29</td>
<td></td>
</tr>
<tr>
<td>Ordering</td>
<td>24.34 %</td>
<td>75.66 %</td>
<td>1 : 3.11</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Workload</th>
<th>POSTGRES</th>
<th>Non-Optimized SQL</th>
<th>Optimized SQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>COST</td>
<td>Ratio (avg) updates : read only</td>
<td>Ratio (total) updates : read only</td>
<td>Ratio (total) updates : read only</td>
</tr>
<tr>
<td>Browsing</td>
<td>7.50 : 50.11</td>
<td>7.50 : 1511.32</td>
<td>6.92 : 10.39</td>
</tr>
<tr>
<td>Shopping</td>
<td>6.38 : 49.35</td>
<td>6.38 : 409.11</td>
<td>6.28 : 6.59</td>
</tr>
<tr>
<td>Ordering</td>
<td>7.70 : 36.28</td>
<td>7.70 : 112.85</td>
<td>6.23 : 3.28</td>
</tr>
</tbody>
</table>

A new twist to Moore’s Law

- What is the cost of optimization?
  - SQL rewriting = several days two/three (expert) people (improvement ratio between 5 and 10)
  - Ganymed = a few PCs with open source software (improvement factor between 2 and 5 for optimized SQL, for non-optimized SQL multiply by 10-100)

- Keep in mind:
  - Copies do not need to be used, they can be kept dormant until increasing load demands more capacity
  - Several database instances can share a machine (database scavenging)
  - We do not need to replicate everything (less overhead for extraction)

Specialized satellite

- We used a satellite to implement a keyword search over TPC-W
- Extra table (keyword, book-id, weight) and an index over the table
- Keywords obtained from i_desc field in item table
- Weight correlated to the last 3333 orders in order_line table (dynamic)
- Tested with DB2, 100 TPC shopping clients, and three satellites (two for queries, one for keword search)
Specialized satellites

- Significant gains in performance
  - Ganymed becomes much simpler:
    - Routing of queries to specialized engines is easier because the queries are distinct (data is not at the master)
    - No optimization, SQL dialect problems
  - Many interesting, useful applications
    - Each satellite a different data schema over the same data
    - Testing new data organizations
    - Specialized indexes, tables
    - No more index recommendations, just build all (in satellites)
    - Derived data (aggregated, materialized, summarized, histograms, etc.) consistent with master
    - ...