WICE - A Pragmatic Protocol for Database Replication in Interconnected Clusters

Jon Grov\textsuperscript{1}  Luís Soares\textsuperscript{2}  Alfrânio Correia Jr.\textsuperscript{2}  José Pereira\textsuperscript{2}  Rui Oliveira\textsuperscript{2}  Fernando Pedone\textsuperscript{3}

\textsuperscript{1}University of Oslo, Norway
\textsuperscript{2}University of Minho, Portugal
\textsuperscript{3}University of Lugano, Switzerland

Presented by Jon Grov, December 19, PRDC 2006
Setting

- Database replication among clusters
- WAN-setting
- Goals:
  - Maximize fault tolerance
  - Reduce query latency
- Full replication: Each server holds a complete copy of the database
Our focus: Concurrency control

- Servers receive *transaction requests*. Example:

  \[ T = r(x) \ r(y) \ w(y) \]

- Each server can receive any type of request
- Transactions may execute concurrently, but we require a total, logical order: Execution should be reproducible in a non-replicated database
Replication strategies

- Primary backup: One node executes all update transactions
- Deterministic execution: Execute all transactions at all sites in the same order
- Distributed coordination: Execute at any site, only updates are distributed. Requires coordination before commit.
- Our interest is in protocols using distributed coordination
Coordination through atomic broadcast

- Basic principle:
  - First, all operations are executed at receiving server
  - Then, updates are distributed to all servers by an *atomic broadcast*, provided by a group-communication service

- Atomic broadcast provides total order
- This order is basis for validation:

  A transaction can commit if and only if all read-operations read the most recently written value according to the total order.
- Message contains updates, read-set and version of reads
- Deterministic certification: If one site successfully certifies transaction, all sites will
- Atomic broadcast must be *uniform* to provide failover
- The delay between an update is initiated until it is applied at remote site determine the abort rate
DBSM with interconnected clusters

- Generic group-communication software in WAN is troublesome
- Using a black-box primitive for uniform atomic broadcast may block application-specific optimizations
Example: Cost of uniformity

- T1: \( x := x + 4 \), T2: \( x := x - 4 \)
- Initially: \( x = 4 \)

Local execution at origination site

site A: T1: read \( x = 4 \), write \( x = 8 \)

site B
Example: Cost of uniformity

- $T_1$: $x := x + 4$, $T_2$: $x := x - 4$
- Initially: $x = 4$

![Diagram showing local execution at origination site and T1's updates being distributed over time.]

- Site A: $T_1$: read $x = 4$, write $x = 8$
- Site B:
Example: Cost of uniformity

- $T_1: x := x + 4$, $T_2: x := x - 4$
- Initially: $x = 4$

Diagram:

- $T_1$: read $x = 4$, write $x = 8$
- Delay before $T_1$'s update is received by all sites
- $T_1$'s updates are being acknowledged...
Example: Cost of uniformity

- T1: $x := x + 4$, T2: $x := x - 4$
- Initially: $x = 4$
Example: Cost of uniformity

- **T1**: \( x := x + 4 \), **T2**: \( x := x - 4 \)
- Initially: \( x = 4 \)
Example: Cost of uniformity

- T1: \( x := x + 4 \), T2: \( x := x - 4 \)
- Initially: \( x = 4 \)

Diagram:

- Site A:
  - T1: read \( x = 4 \), write \( x = 8 \)
- Site B:
  - Delay before T1's update is received by all sites
  - T2: read \( x = 4 \), write \( x = 0 \)
  - Delay before T1's update is known by site B to be stable

- T1 is committed
- T1: commit
- T1: write \( x = 8 \), commit
  - T2: abort due to conflict with T1
Example: Cost of uniformity

- T1: \( x := x + 4 \), T2: \( x := x - 4 \)
- Initially: \( x = 4 \)

At this point, T1's update on x is available at site B...

---

**Local execution at origination site**

**T1's updates are distributed**

**T1's updates are acknowledged**

**T1 is committed**

<table>
<thead>
<tr>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1 : commit</td>
</tr>
<tr>
<td>T1 : write ( x = 8 ), commit</td>
</tr>
<tr>
<td>T2: abort due to conflict with T1</td>
</tr>
</tbody>
</table>

- Delay before T1's update is received by all sites
- Delay before T1's update is known by site B to be stable
Example: Cost of uniformity

- T1: \( x := x + 4 \), T2: \( x := x - 4 \)
- Initially: \( x = 4 \)

At this point, T1's update on \( x \) is available at site B...

...but when relying on a black-box communication-primitive, it is not visible until here

<table>
<thead>
<tr>
<th>Site A</th>
<th>Site B</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1: read ( x = 4 ), write ( x = 8 )</td>
<td>T2: read ( x = 4 ), write ( x = 0 )</td>
</tr>
<tr>
<td>Delay before T1's update is received by all sites</td>
<td>Delay before T1's update is known by site B to be stable</td>
</tr>
<tr>
<td>T1: commit</td>
<td>T1: write ( x = 8 ), commit T2: abort due to conflict with T1</td>
</tr>
</tbody>
</table>

Local execution at origination site<br>T1's updates are distributed<br>T1's updates are acknowledged<br>T1 is committed
Our proposal: WICE

We want to reduce abort rate and simplify deployment by:

- Opening the communication-primitive: Group communication only within clusters – all inter-cluster messaging sent by unicast
- Exploiting tight integration with database system: Remote updates should be available before uniformity
- Perform certification at one central site: Allows explicit certifier placement

Disadvantage: Failure-handling part of the protocol
WICE: Communication pattern

Cluster A

Cluster B
Example: WICE

- T1: $x := x + 4$, T2: $x := x - 4$
- Initially: $x = 4$
Example: WICE

- $T_1: x := x + 4$, $T_2: x := x - 4$
- Initially: $x = 4$
Example: WICE

- $T_1$: $x := x + 4$, $T_2$: $x := x - 4$
- Initially: $x = 4$

![Diagram showing execution and update distribution]

- Local execution at origin site
- $T_1$'s updates are distributed
- $T_1$'s updates are being acknowledged...

- Site A: $T_1$: read $x = 4$, write $x = 8$
- Site B: $T_1$: write $x = 8$
- Delay before $T_1$'s update is received by all sites
Example: WICE

- T1: $x := x + 4$, T2: $x := x - 4$
- Initially: $x = 4$
Example: WICE

- T1: \( x := x + 4 \), T2: \( x := x - 4 \)
- Initially: \( x = 4 \)
Example: WICE

- $T_1$: $x := x + 4$, $T_2$: $x := x - 4$
- Initially: $x = 4$
Experiments

- WICE was simulated along with DBSM.
- Two clusters, each with 3 servers:
  - negligible intra-cluster latency
  - 200 ms inter-cluster latency
  - Unrestricted bandwidth
- 92% of transactions update some objects
- From 60 to 6000 simultaneous clients
Throughput

Cluster A

Cluster B

Number of transactions committed per minute
Abort rate

Cluster A

Cluster B

Fraction of submitted transactions that eventually aborts
Latency

Cluster A

Elapsed time from submit until reply is received

Cluster B
Conclusion

- We have shown one strategy to extend the ideas of group-communication based replication protocols to a WAN-environment
- Our most important point is that updates must be exposed before the transaction is stable
- Further work:
  - Implementation in real system
  - Weaker assumptions on replication degree and stability requirements
Thank you!

Questions?

- The GORDA-project: http://gorda.di.uminho.pt
- My email: jongr@ifi.uio.no