Managing Transaction Conflicts in Middleware-based Database Replication Architectures

F.D. Muñoz-Escoí, J. Pla-Civera, M.I. Ruiz-Fuertes, L. Irún-Briz, H. Decker
{fmunyoz, jpla, miruife, lirun, hendrik}@iti.upv.es
Instituto Tecnológico de Informática, Valencia, Spain

J.E. Armendáriz-Iñigo, J.R. González de Mendívil
{enrique.armendariz, mendivil}@unavarra.es
Universidad Pública de Navarra, Pamplona, Spain
CONTENT

1. Problem Overview.
2. Our Solution Proposal.
3. Applicability to Replication Protocols.
4. Performance Results.
5. Conclusions.
1.- Problem Overview

- Most of database replication protocols share these features:
  - **Eager Replication.**
    - Transaction updates are propagated to the replicas within the boundaries of the transaction.
  - **Update Everywhere.**
    - Allows updates to a data item to be performed anywhere in the system.
    - Each transaction has two phases
      - *Read/Write Phase.* Each read and write operation is executed at the transaction master replica.
      - *Commit Phase.* Transaction updates (writeset) are propagated to the rest of replicas.
  - **Constant Interaction.**
    - A constant number of messages is used to synchronize the servers for a given transaction, independently of the number of operations in the transaction.
    - Total order multicast messages are used.
1.- Problem Overview

A general replication protocol for a transaction \( T_i \) could be:

1. \( T_i \) executes its read/write phase at its local replica \( R_i \).
2. At \( T_i \)'s commit phase:
   1. Its write set is obtained (\( WSi \)).
   2. \( WSi \) is multicast to all available replicas (using total-order).
   3. Upon delivery of \( WSi \) at each available replica \( R_k \):
      - If \( T_k \) is not local and it was delivered before \( WSi \) then the following will happen:
        * Either \( T_i \) is rolled back (in a deterministic way or with a weak voting process).
        * Or \( T_i \) is blocked until conflicting transactions are done.
      - If \( T_k \) is local at \( R_k \) and is still in its read/write phase then \( T_k \) will be rolled back.
3. The \( WSi \) is validated so as to check conflicts with other transactions.
   It is decided whether \( T_i \) may commit or not.
1.- Problem Overview

- Our solution is focused on middleware architectures, e.g. MADIS:
  - It offers a JDBC interface to applications.
  - It sees a JDBC interface with the underlying DBMS.
  - It allows several replication algorithms to be plugged in and out at will.

- Some protocols exclusively validate writesets against already delivered (and respectively validated) transactions.

- The conflict property is stable.

- The protocols must wait until the commit phase of local transactions for rolling them back.

- The question is:
  - *Why do not design a conflict detection schema so transactions are aborted a.s.a.p.?*
2.- Our Solution Proposal

User Application

Replication Middleware Implementation

Certificate(T₁, {x})

Abort(T₁)

Commit(T₁)

Abort(Tₐ)

Certificate(Tₐ, {x})

Commit(Tₐ)

Abort(Tₐ)

Write(Tₐ, {x})

Write(T₁, {x})

Commit(T₁)

Abort(Tₐ)

Group Communication System

〈remote, Tₐ, {x}〉

x

T₁

Tₐ
2.- Our Solution Proposal

User Application

Abort(T₁)

Abort(T₁)

Abort(T₁)

Abort(T₁)

Abort(T₁)

Abort(T₁)

Block Detector

AssignPriority(Tᵣ, 1)

Commit(Tᵣ)

Certificate(Tᵣ, {x})

Replication Protocol

Group Communication System

Replication Middleware

Implementation

<x, T₁, Tᵣ>

remote, Tᵣ, {x}

<table>
<thead>
<tr>
<th>OID</th>
<th>Blocking</th>
<th>Blocked</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>T₁</td>
<td>Tᵣ</td>
</tr>
<tr>
<td>y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>z</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3.- Applicability to Replication Protocols

The SI-Rep (Lin et al. 2005) is used to check the usefulness of this solution:

- It follows the schema already shown.
  - “Black box” approach: Validated transactions may be aborted by the DBMS and they must be reattempted.

- Two optimizations added:
  1. It has a pre-certification before the writeset multicast.
     - It allows an earlier abortion of conflicting transactions.
  2. Writesets may be applied in a “disordered” fashion provided that they do not conflict.
     - Remote transactions are applied faster.
     - Useful for high loaded environments and low conflict rates.

- Final certification depends on the interval between the snapshot taken and its validation.
  - It ensures a Generalized Snapshot Isolation level.
  - Our solution is complementary to any isolation level; obviously, it does not eliminate the certification for this isolation level.
3. - Applicability to Replication Protocols

A general replication protocol for a transaction $T_i$ could be:

1. $T_i$ executes its read/write phase at its local replica ($R_i$).
2. At $T_i$’s commit phase:
   1. Its write set is obtained ($WS_i$).
   2. $WS_i$ is multicast to all available replicas (using total-order).
   3. Upon delivery of $WS_i$ at each available replica ($R_k$):
      - The Block Detector will detect conflicts with local transactions and abort them by way of priority assignment.
      - It is decided whether $T_i$ may commit or not.
      - If $T_i$ must commit, it will be applied in the database.

- Local certification of $WS_i$ against already certified transactions but still not applied.
- If it is not certified, $WS_i$ will not be multicast.
3.- Applicability to Replication Protocols

Three versions have been implemented in our MADIS middleware:

- SIR: The original SI-Rep (Lin et al. 2005).
- SIR-BD: SI-Rep including our conflict detection technique.
- SIR-SBD: SI-Rep, without optimizations but including our conflict detection mechanism.
4.- Performance Results

- The three implementations have been checked in this system:
  - Cluster with 2, 4, or 8 replicas.
    - Pentium 4 at 2.8GHz, 1GB RAM.
    - Ethernet 1Gb/s.
    - Linux Fedora Core 2.
    - PostgreSQL 7.4.12 as the DBMS.
  - Each replica has 10 concurrent clients. 10 transactions per client without any pause between them.
  - Only one table in the database with 10000 registers:
    - Two fields (key/value).
  - Each transaction firstly reads part of the table (with a `SELECT FOR UPDATE`) and, afterwards, updates each read register.
  - Two kind of transactions, depending on the pause between the two previous statements.
    - 1 sec. pause (*Short transactions*)
    - 4 sec. pause (*Long transactions*)
4.- Performance Results

Long Transactions. 2 Replicas

![Graph showing performance results for long transactions with 2 replicas. The x-axis represents the number of accessed items, and the y-axis represents the transaction length in seconds. Different lines represent SIR committed, SIR-BD committed, SIR-SBD committed, SIR aborted, SIR-BD aborted, and SIR-SBD aborted.]
4.- Performance Results

Long Transactions. 4 Replicas

- SIR c
- SIR-BD c
- SIR-SBD c
- SIR a
- SIR-BD a
- SIR-SBD a
4.- Performance Results

Long Transactions. 8 Replicas

![Graph showing performance results for Long Transactions with 8 replicas, comparing SIR c, SIR-BD c, SIR a, and SIR-BD a.]
4.- Performance Results

- Short transactions follow the same tendency:
  - Transactions are rolled back earlier with our block detector proposal.
  - Hence, the system load is reduced and transactions are committed earlier.

- However, the obtained gain with short transactions is not as noticeable as with the long ones.
  - See figures in the proceedings.
4.- Performance Results Varying Transaction Rate

Long Transactions. 8 Replicas

![Graph showing performance results varying transaction rate with different transaction lengths and TPS values. The graph compares SIR c, SIR-BD c, SIR a, and SIR-BD a.]
5.- Conclusions

- A conflict detection schema has been introduced that enables to:
  - Abort local transactions conflicting with transactions that have been validated (and must be committed).
  - Obtain a row-level granularity for conflict detection.
  - Minimize the effort needed for conflict detection.
  - Be easily implemented over many DBMSs.

- Results show that this kind of detection mechanism may increase the overall system performance.
Managing Transaction Conflicts in Middleware-based Database Replication Architectures

F.D. Muñoz-Escoí, J. Pla-Civera, M.I. Ruiz-Fuertes, L. Irún-Briz, H. Decker
{fmunyoz, jpla, miruiue, lirun, hendrik}@iti.upv.es
Instituto Tecnológico de Informática, Valencia, Spain

J.E. Armendáriz-Iñigo, J.R. González de Mendívil
{enrique.armendariz, mendivil}@unavarra.es
Universidad Pública de Navarra, Pamplona, Spain