Database Replication with Oracle 11g and MS SQL Server 2008

Flavio Bolfing
Software and Systems
University of Applied Sciences Chur, Switzerland
www.hsr.ch/mse

Abstract
Database replication is used widely in nowadays applications. The need of high data availability, high performance and fault tolerance makes it inevitable to use database replication. Therefore a lot of commercial products are having standardized solutions for replication. Examining two well known commercial products on their solutions for replication, conflict behaviour and usability is the main goal of this article.

1. Introduction and Overview
As it is a matter of common knowledge, one of the main goals of a database system is to avoid data redundancy. The reasons hereby are to avoid inconsistency caused by data updates, saving data storage and speed-up the data lookup. To ensure no data redundancy, it is common to use normalizations to avoid all these problems.

By using database replication you intentionally create data redundancy and thereby having a lot of advantages. With this data redundancy you ensure better data availability which implies a better fault tolerance in case of a failure. You also increase the performance by having better response times due to geographic and topological complexity. Another advantage of using replication is the fact that you can also use offline applications like mobile databases to be updated. Last but not least you can also balance the load distribution to several nodes. Like everything, replication has also some negative aspects. This is expressed by higher system complexity, need for more disk space and higher update effort.

This paper is separated into two main parts. In the first part the theoretical part of database replication is described. There is no reference to any platform based product. In the second part the commercial products MS SQL Server 2008 and Oracle 11g are analyzed towards their ability to handle replication and conflict strategies [1].

2. Replication theory
2.1. Conflict of goals
The availability, the performance and the consistency of data are the main requirements of database replication. These 3 requirements are in conflict to each other because a change for the benefit of one of the criterion implies a minimisation at the expense of the other criteria. This is often illustrated in a conflict of goals triangle [2].
As mentioned before it is a goal to increase the data availability by intentionally creating redundancy. With these redundant nodes you have in case of a failure and the loss of a node still data available. This redundancy is at the expense of consistency. Then the more redundant database you run, the more complex it is to keep all the redundant databases updated. With more redundancy you also have more overhead and therefore less performance. In contrary if you want to have better data consistency you may implement a propagation strategy which can have impacts to the performance and also availability. We see that you never can satisfy every of the 3 goals. To use replication efficiently you have to deal with these conflicts carefully.

### 2.2. Consistency of replicated data

The main problem with replication of data is to ensure consistency in every particular copy of the database. To ensure that replicas are consistent, we need to make sure that conflict operations (e.g. a read and write operation at the same data acting concurrently) are done in the same order at every replica. Traditionally replication models have been classified into strong and weak consistency. Thereby strong consistency models ensure that all of the replicas have exactly the same content before any transaction can be performed. In an unreliable network like the internet it is sometimes impossible to use strong consistency because with a large amount of replicas, latency can be so high so that you will never have synchronous replicas in appropriate time. Therefore the strong consistency model is only suitable for systems with few replicas. As an alternative it is possible to use weak consistency. There it is not necessary that all of the replicas have the same content to ensure transactions. You only need to ensure that the replicas sometime converge to a consistent state which is not bounded to a specific period of time. This consistent state is defined by synchronization points. To accomplish that, weak consistency models use synchronization variables. With the synchronization variables you also establish a critical section. Inside of this section you let inconsistency of the shared data happen provided that concurrent read/write actions are not permitted. To ensure that, weak consistency uses locks. Furthermore weak consistency has to fulfil some conditions [3]:

- Accesses to synchronization variables associated with a data store are sequentially consistent.
• No operation on a synchronization variable is allowed to be performed until all previous writes have been completed everywhere.
• No read or write operation on data items are allowed to be performed until all previous operations to synchronization variables have been performed.

3. Replication strategies

3.1. Overview

There exist several approaches to classify replication strategies. One very famous classification of replication strategies is based on contrasting propagation strategy with the ownership strategy. Thereby the propagation strategy is either eager or lazy propagation of updates, whereas ownership strategy defines how updates are controlled. There you differentiate between object master and object group. With object master all updates emanate from a master copy of the object, while with object group updates can emanate from any object to the others (see figure 2) [6].

![Figure 2: Ownership strategies in replication](image)

Table 1: Classification of replication strategies based on propagation and ownership

<table>
<thead>
<tr>
<th>Ownership</th>
<th>Propagation</th>
<th>Lazy</th>
<th>Eager</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group</td>
<td>N transactions</td>
<td>N object owners</td>
<td>one transaction</td>
</tr>
<tr>
<td>Master</td>
<td>N transactions</td>
<td>one object owner</td>
<td>one transaction</td>
</tr>
</tbody>
</table>

Another approach to classify replication strategies is to differentiate them about their consistency definition. The strategy is build-up on the previous strategy but differs between strong and weak consistency. It also gives examples on strategies which are used in practice [2].
3.2. Propagation strategies for updates
Besides consistency, replication differs also in the propagation strategy for updates which is used. Replication can use two different propagation strategies for updates: Synchronous (also called eager) and asynchronous replication (lazy replication). Besides the different update strategy, the two also differ in their conflict resolution. Eager replication does not have to have a conflict resolution, while lazy replication must have one. As an example you can think of the situation that a record is changed on two nodes simultaneously. Thereby the eager replication based system would detect this conflict before confirming the commit and abort one of the two transactions. In contrary a lazy system would allow per definition both transactions to commit, but would run conflict resolution logic during resynchronization between the nodes.

3.3. Asynchronous replication (lazy)
Asynchronous replication updates the replicas not all at the same time. The propagation of the updates to the other nodes takes place after the effective transaction has been done (which means after the commit) by synchronizing each other. Thereby the synchronization of the nodes happen either after a certain interval or after e certain point of time. The big advantage of this strategy is by increasing the performance. It is for sure that with this strategy sometimes conflicts can occur, which implies a good conflict resolution strategy.

With the primary-copy approach, updates are only possible on the master copy of the replicas. From there the updates are propagated to the slaves. In contrary, the update-anywhere approach can be defined as a strategy, where every copy on every node can process updates. It is possible that two nodes want to update the same object and race each other to propagate their own transaction at the other nodes. Thereby the replication mechanism must detect this and solve the problem with his conflict strategy.
3.3.1. Synchronous replication (eager)
Synchronous replication, also known as eager replication, updates all the replicas before the transaction commits. That means that all replicas were kept exactly synchronized at all nodes by updating them as part of one atomic transaction. Therefore inconsistency can not exist with eager replication. Through locking of the data on every replica, conflicts can be prevented (depending on the locking strategy) and result in waiting loops or deadlocks. Reading actions on connected nodes give current data. If the nodes are disconnected eager replication may give stale data. Simple eager replication systems prohibit updates if one of the nodes is not connected. Higher developed systems allow updates even with disconnected nodes among members of the same cluster. Even if all of the replicas are connected, updates may fail due to the fact of existing deadlocks to prevent serialization errors.

3.4. Ownership strategies
The most commonly used ownership strategies are merge replication, ROWA and primary copy. They were described in the following section.

3.4.1. Merge replication (update anywhere / multi-master)
With the merge replication strategy updates to data can be performed anywhere in the system. These updates were propagated mostly asynchronous. The consistency is weak and depends on the frequency of the synchronization. The conflict resolution is the most difficult part in this strategy. To discover a conflict, every update gets a timestamp of its most recent update and the new value. If the timestamp of the replica which has to be updated is different, a conflict exists. To resolve this conflict you can use predefined strategies of the database systems or user defined routines. You may also solve the conflicts manually.

3.4.2. Primary copy (master-slave)
With the primary copy strategy every update should be first sent to the primary copy node. This primary copy node is also called the publisher and has several subscribers. If an update has to be propagated it can either be with lazy or with eager propagation. In the case of eager propagation, the update will first be done on the publisher and then propagated to all of his subscribers. As soon as all of the subscribers have done the update and sent a acknowledge, the publisher can send the commit to the user. In the other case, primary copy with lazy propagation, the user gets the commit as soon as the update has been finished on the publisher and before the propagation to all the other subscribers. The propagation itself is achieved either with a trigger or log based. In primary copy, reading actions first have to ensure that the local copy is synchronized. This happens by comparing the version number of the publisher with the one of the local subscriber. The local copy can only be read if they are identical. In the case of failure of the publisher, primary copy knows 2 different approaches. One of them is called static and prohibits updates as long as the publisher is not online. In contrary the dynamic approach looks that one of the subscribers takes over the role of primary node.

3.4.3. ROWA (Read One Write-All)
ROWA is the most commonly used strategy in synchronous replication. In this approach reading actions are preferred. The can be performed at every node and therefore locally. Writing actions are more complicated. A replica can only be updated, if all the other replicas were locked. For that the client sends a locking request to all replicas. If the lock is not accepted by all, the transaction has to wait. If all the replicas accept the lock, the transaction happens on all simultaneously and the commit can be sent to the user. The problem hereby is that updates can only happen if every replica is online. As a solution there exist a write-all-available (ROWA-A) strategy. In this case only the available replicas were updated immediately, while all the others were updated as soon as they are online again.
3.5. Conflicts

Conflicts only exist in asynchronous replication because the updates are not propagated to all replicas at the same time. Some updates were carried out at a later point of time; meanwhile already new updates can be happened. Normally timestamps are used to detect reconcile updates. Every object carries a timestamp of its most recent update. Thereby an update transaction carries the new value and also the old object timestamp with it. If this object timestamp is not different from the one of the local replica, the update is safe. If the timestamp is different, then the node may reject the incoming transaction and submits it for reconciliation. There exist three types of replication conflicts [5]:

- Update conflicts
- Uniqueness conflicts
- Delete conflicts
- Ordering conflicts

3.5.1. Update conflicts

An update conflict occurs when the replication updates a data object which is in conflict with another update at the same time. Two transaction want to update the same data record at almost the same time is an example case. This kind of conflict can be detected from the propagation node by comparing the old timestamp of the replica with the one of the subscriber. If they differ you have an update conflict. To solve these kinds of conflicts every database has its own strategies.

3.5.2. Uniqueness conflicts

A uniqueness conflict exists when the replication of an update violates entity integrity, like primary key or unique constraint. For example if two transactions, originated from different nodes, insert data which uses the same primary key or the same value for one as unique classified attribute, then a uniqueness conflict occurs. As in update conflicts there exists specific strategies to prevent these kinds of conflicts.

3.5.3. Delete conflicts

A delete conflict occurs if two transactions from different nodes happen, with one transaction deleting a data record and the other wants to update or also delete it. In this case there is simply no data record available to update or delete and therefore a conflict.

3.5.4. Ordering conflicts

Ordering conflicts can only occur in replication environments with more than 3 master sites. They can appear if an update is propagated to the master sites with one of them currently blocked. In asynchronous mode all the available master sites were updated while the blocked one is not. The propagation to the blocked master site resumes as soon as the site is reconnected. If in this resume the updates happen in a wrong order, we have an ordering conflict.

4. Mobile database replications

The increase of requirements to databases and the progress in mobile computing lead to a change of the simple model of a central database system. Users nowadays demand for a constant and efficiently availability of their data, which is in the case of mobile databases not transferable to normal database systems with their replication. Mobile databases are widely used nowadays, especially by working in the field. Their databases aren’t attached to a specific location and therefore their position always changes.
Synchronous replication is not suitable for mobile databases where most nodes are usually disconnected. If so, a transaction would be tentative until all missing nodes are connected. Mobile databases therefore need asynchronous replication algorithms, which propagate updates to the others after the transaction commit. The best practice solution for mobile databases is a modified mastered replication scheme, which is called two-tier-replication scheme. With this scheme we assume to have two different kinds of nodes. The first ones are mobile nodes. These nodes are mostly disconnected and not attached to a specific location. They store a replica of the database and may create tentative transactions. It is possible that a mobile node is an object master of some specific data items. Every mobile node is mastered at some base node. The second ones are base nodes. They are normally connected and they also store a replica of the database. Base nodes are usually the master of data items.

Regarding mobile nodes we can find two different versions of data items. The first one is called master version. The master version is the most recent value which has been received from the object master. This version is the true master version at the object master, but disconnected mobile nodes may have older versions. The second one is called tentative version. This is the most recent value created by local updates.

Last but not least we have two different kinds of transactions. Base transaction can only be performed on master data and produce again master data. There can be at most one connected mobile node and may involve several base nodes. Tentative transactions use only local tentative data. They are not processed as a base transaction before the mobile node reconnects to the base node.

5. Experiments
In the experiments a replication with the two commercial products MS SQL Server 2008 and Oracle 11g should be implemented. The examination of their specific conflict behaviour is the main goal of the experiments. Along with the implementation the available replication strategies of the respective product were studied and described.

5.1. Oracle 11g
5.1.1. Replication strategies
With Oracle 11g there are 3 different replication strategies available:

1. Multi-master replication
2. Updateable materialized view
3. Read-only materialized view

Each site in a multi-master replication environment is a master site, and therefore able to update any replicated table. Each site communicates with the other master sites. It is possible to replicate with asynchronous or synchronous propagation. The converge of different data items happens in a specific period of time determined by the administrator. Materialized views are tables based on queries. They contain a complete or partial copy of a master from a certain point of time. Thereby a materialized view can be read-only, updateable or writeable. They provide the following benefits:

- Enable local access, which improves response times and availability.
- Increase the data security by enabling to replicate only a selected subset of the target master's data set.
5.1.2. Test scenario
For the tests a Windows Server 2008 with two Oracle 11g database instances have been created (orcl1 and orcl2). The internal tables from the SCOTT scheme have been installed and used for the replication. To get replication you have to define a replication group and the master definition site. To test the behaviour of Oracle 11g in conflict situations, I used a multi-master replication with one master/subscriber and one table named salgrade for replication.

5.1.3. Conflict resolution in Oracle 11g
Oracle 11g has a lot of inbuilt conflict resolutions:

- Latest time stamp: Most recent update wins
- Overwrite: Overwrites current value with new value
- Discard: Ignores the value
- Additive: Difference of the two values is added to the current value (current value = current value + (new value - old value))
- etc.

In my example I used two different methods for conflict resolution. The first one was nothing, which means that you manually have to resolve the conflict. The second one was the latest timestamp.

![Conflict resolution in MS SQL Server 2008](image)

5.1.4. Experiences
During the tests the following observations were made:

- No conflict resolution: The update conflict was detected correctly with an entry in the error table. The two data items stayed different. With conflict resolution the data update with the latest timestamp won.
- No conflict resolution: The delete conflict was detected correctly with an entry in the error table. The two data items stayed different. With conflict resolution the data update with the latest timestamp won.
- A uniqueness conflict couldn’t be performed because they aren’t allowed in Oracle 11g.
5.2. MS SQL Server 2008

5.2.1. Replication strategies

With MS SQL Server 2008 we have 3 different replication strategies available:

1. Snapshot publication
2. Transactional publication
3. Merge publication (multi-maser)

With a snapshot publication a copy of the entire database is published out to the subscribers. The fact that it is only a snapshot of the database at a certain time implies that further changes are not tracked at the subscribers. Transactional publication monitors the transaction log for changes and then sends these transactions to the distributor. The subscribers either pull the changes or they are pushed to them. Merge replication (or multi-master) is a replication strategy where publisher and subscriber can update the published data independently. Thereby all changes are merged periodically by synchronization. If the synchronization has to update the same data differently, the synchronization will result in a conflict and has to be solved.

5.2.2. Test scenario

For the tests a Windows Server 2008 with two MS SQL 2008 database instances have been created. The AdventureWorks tables have been installed and used for the replication. To test the behaviour of MS SQL 2008 in conflict situations, I used a merge replication with only one table named Person.Address to replicate.

5.2.3. Conflict resolution in MS SQL Server 2008

MS SQL Server 2008 has three different conflict resolution methods:

- The first method states that the publisher always wins.
- The second states that the subscriber always wins.
- In the third case, you are asked to resolve each conflict individually through a SQL Server 2008 interface.

In my example I used the method that the publisher always wins.
5.2.4. Experiences
During the tests the following observations were made:

- The chosen conflict resolution (publisher always wins) was executed correctly.
- The update conflict was detected correctly with the conflict type 1 (Update conflict) and the publisher as winner.
- The delete conflict was detected correctly with conflict type 3 (Update/Delete) and the publisher as winner.
- A uniqueness conflict couldn’t be performed because they aren’t allowed in MS SQL Server 2008 due to the uniqueness violation rule.
- Merge replication is suitable for big amount of replicas

6. Conclusion
The two commercial products are very different. While the goal of the Microsoft database is surely user friendliness with his installation wizards, the Oracle solution is very difficult to install due to the fact that almost everything has to be done manually with scripts. The support of different replication strategies are almost the same at both solutions. Detecting conflicts worked in both products very well and without problems. In the conflict resolution, Oracle is at advantage due to the high amount of inbuilt solutions. Generally spoken replication is relatively easy to implement but you really should avoid conflicts by carefully thinking about the design.
7. References


