ASYNCHRONOUS REPLICATION CONFLICT CLASSIFICATION, DETECTION AND RESOLUTION FOR HETEROGENEOUS DATA GRIDS[1]

Eva Kühn, Angelika Ruhdorfer and Vesna Šešum-Cavic
Vienna University of Technology, Institute of Computer Languages
Space Based Computing Group, 1040 Vienna, Argentinierstraße 8, Austria

Keywords: Asynchronous replication, conflict detection and resolution, relational database, coordination space, data grid.

Abstract: Data replication is a well-known technique in distributed systems, which offers many advantages such as higher data availability, load balancing, fault-tolerance, etc. It can serve to implement data grids where large amounts of data are shared. Besides all advantages, it is necessary to point to the problems, called replication conflicts that arise due to the replication strategies. In this paper, we present an infrastructure how to cope with replication of heterogeneous data in general for conflict detection and resolution and we illustrate its usefulness by means of an industrial business case implementation for the domain of relational databases and show further extensions for more complex resolution strategies. The implementation deals with the special case of asynchronous database replication in a peer-to-peer (multi-master) scenario, the possible conflicts in this particular domain and their classification, the ways of conflict detection, and shows some possible solution methods.

1 INTRODUCTION

Replication is a practical and efficient technique, commonly used in a distributed environment. The benefits of replication include: improving data availability, data access performance and load balancing, providing fault-tolerance by maintaining copies of data at different locations, etc. (Budiarto et al, 2002; Dunlop et al, 2003; Gustavsson and Andler, 2005). Generally, there are two approaches of data copying: synchronous and asynchronous replication (IBM DB2 Universal Database, 2002). The decision which type of replication scenario is appropriate to apply depends on the system's properties. The problem is that many systems are not highly available and show inconsistent states, include an enormously high number of sites and have not high performance communication links between the single sites. Such kind of system is a candidate for asynchronous replication. The main goal is that all data eventually becomes consistent. Therefore we propose the following methodology that can be used for the realization of asynchronous data replication in general:

- classify all conflicts that might occur
- develop strategies for conflict prevention
- explain how all conflicts can be detected
- define strategies for conflicts resolution.

In this paper, we will apply these steps to the special case of relational database replication. There are different replication topologies for asynchronous replication: master-copy, multi-tier, update anywhere and multi-master (peer-to-peer) (IBM DB2 Universal Database, 2002). The most complex

[1] The work was supported by the Austrian Federal Ministry for Transport, Innovation, and Technology under the FIT-IT Semantic Systems project Triple Space Computing (TSC 809249).

topology is peer-to-peer with asynchronous replication. Each node in a peer-to-peer topology publishes and subscribes to the same data. All nodes are equal peers with equal ownership of the data. The procedures for asynchronous replication are: Dump and Reload, Snapshot Replication and Incremental Refresh, where either *the changed tuples* are transmitted to the target sites or the transactions causing the tuples to be changed, are transferred and applied at the target sites (*transaction-based replication*). Examples for replication facilities of the state of the art in databases systems can be found in (Garmany and Freeman, 2003; Oracle 9i, 2002; Chigrik, 2000; IBM DB2 Universal Database, 2002; IBM DB2 Data Propagator, 2003).

2 CONFLICT CLASSIFICATION

The following assumptions will be taken into consideration: asynchronous replication in relational databases in multi-master scenario (peer-to-peer), supporting transaction-based replication with incremental refresh mode. When we talk about the SQL-operations, it must be clear that only *write* operations (INSERT, UPDATE, DELETE) which change tuples will be considered, because they have a crucial impact on a successful synchronization. We consider each SQL-operation of a transaction of being decomposed into a series of so-called *single-operations*. A single-operation is an operation that involves one tuple only. Two operations of transactions that are executed on different sites are called *conflicting operations*, if they cause a replication conflict. This might the case if they operate on the same tuple or if they transform some tuples which were previously different into the same tuple. There are many different types of conflicts: Insert/Insert, Insert/Update, Insert/Delete, Update/Update, Update/Delete, Delete/Delete. More detailed information about conflict classification can be seen in Kühn at al., 2007; Ruhdorfer, 2005.

Example: Let us assume that there are two sites: A and B, and two operation: *op1* and *op2*. Further, we assume that *op1* is Insert operation and *op2* is also Insert operation. Then *op1/op2* means that *op1* was executed and committed at site A and *op2* was concurrently executed and committed at site B. Afterwards op1 is replicated to site B and op2 to site A and executed there respectively, causing the conflict that can be described as: tuples are entirely the same (PRIMARY KEY and all the columns are the same).

3 CONFLICT PREVENTION

Before we start discussion about conflict detection and conflict resolution, it is important to know what techniques/approaches can be used in order to *avoid* replication conflicts. Of course it is much better to avoid conflicts, if this is possible. For example, modification of the database scheme can help where unique numbers for each peer site are added to tables that shall be replicated etc. However, the tradeoff is a change of existing systems that possibly is not acceptable. Another example would be to resign full peer-to-peer replication and to use only one-way read-only replication etc. Replication conflicts can also be prevented by assigning the right to update the data to a single site in one of the following ownership types: static site ownership model, dynamic site ownership model (workflow, token passing), shared ownership model with some strategies for avoiding specific types of conflicts (avoiding uniqueness conflicts, avoiding delete conflicts, avoiding ordering conflicts) (Oracle 9i, 2002; IBM DB2 Redbook, 2002).

4 CONFLICT DETECTION

The process of detecting constraint errors and the process of detecting whether the same tuple was modified concurrently by application programs at more than one peer site during the same replication cycle in a peer-to-peer replication configuration is called *conflict detection*. Commercial databases address this issue differently (Garmany and Freeman, 2003; IBM DB2 Universal Database, 2002). We assume that in the full master replication scenario, the peers communicate directly using a shared coordination space (Kühn, 2001) which provides reliable asynchronous, publish/subscribe based flexible collaboration on shared and distributed data structure, like the communication with near-time event notification, the possibility of reading the same data multiple times and according to different coordination criteria in a flexible way. *Therefore we decided to use a space instead of distributed hash tables, publish/subscribe systems or message queues*.Each database (DB) site is called a peer site. With each DB a gateway process is associated that interfaces both the DB and the space and can be located on another site than the DB. For each table to be replicated, triggers are installed that track every write SQL-operation and store its single operations together with the meta-information necessary for conflict resolution in a log-table. Applications need not be changed. The gateway periodically reads the log-table and publishes the single operations plus meta-information into a space container i.e. a space data structure where the other peers' gateways subscribe; they apply these operations to the destination databases in a transaction. The transaction boundaries are not violated; but with one publication step more than one transaction could be published to the space (called transaction "chunk"). If there is a site failure the gateway can recover the data from the log-table.

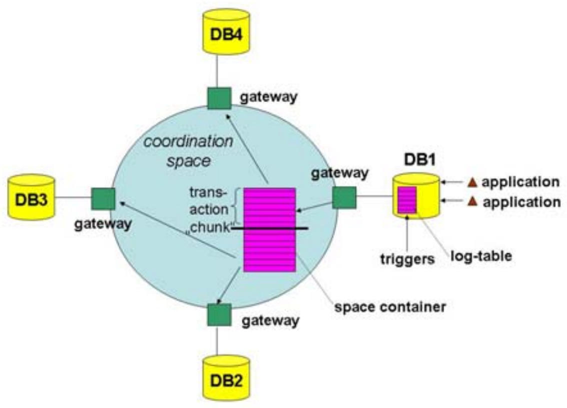

Figure 1: Replication Architecture.

Figure 1 shows only the replication from DB1 to DB2, DB3 and DB4. The full scenario would be symmetric. The architecture is based on the extensible virtual shared memory middleware XVSM (Kühn at al., 2005) that generalizes Linda tuple based communication (Gelernter and Carriero, 1992) by introducing *shared containers* that contain entries and that can be bounded or unbounded. Containers can exhibit different coordination types. XVSM itself supports asynchronous replication and is used to distribute the single operations of the relational database transactions from one database site to one or more target database sites.

What happens when two dependent operations are executed is shown in (Kühn at al, 2007; Ruhdorfer, 2005). Many conflicts can be detected so that the DB returns an error on both sites. However, there are "bad" cases where the conflict can't be detected at both sites. The treatment of these "bad" cases are explained in (Kühn at al, 2007; Ruhdorfer, 2005). Mainly it foresees the support of all old values for update changes.

In that scenarios the environment is limited to two replication sites and at each site a transaction has only one operation. For this simplified, theoretically ideal situation we have explained how to determine whether one of these two transactions causes a conflict and which operations are involved, at each site autonomously. But when we talk about a real IT environment with a lot of different sites and operations, combined in transaction, it is quite difficult to detect the replication conflict exactly. The detection of conflict can be eased if we can assume the existence of transaction counters and state vectors. A transaction counter is a unique number assigned to each transaction. The provision of counters can be done by (1) using a sequencer in the database triggers to mark each database transaction of local applications and (2) using the same sequencer to generate a new number for each transaction from a replication site executed by the gateway and publish this number also to a space container. Each transaction has a state vector attached, which holds the current status of the transaction counters of all sites including the current site before the transaction is executed. The determination of the first operation in a remote transaction that causes a replication conflict is crucial. After that, the next step is to find all the other operations involved in the conflict (see next section).

5 CONFLICT RESOLUTION

The main goal is that all data at all replicated sites eventually is consistent. Of course, this is still an open issue to work on and the intention of this section is to show *some* strategies and possibilities that can be applied.

Conflict resolution determines which operations are the winners and which ones are the losers, and then takes the right steps in case of a committed loser operation in order to annul its effect ("undo" the operation). The different strategies for conflict resolutions in databases are discussed in (Garmany and Freeman, 2003; Oracle 9i, 2002). Nevertheless, the conflict resolution strategies cannot be generalized – many parameters of the replication environment should be taken into consideration (database schema, type of replication scenario, conflict detection method, and also the semantics of the data to be replicated). When a conflict is detected and the loser operations were identified by the conflict resolution strategy, the appropriate "undo" operations must be applied. As we have mentioned in the section 2, the standard operations for database manipulation – INSERT, DELETE, UPDATE – were transformed into a canonical form, i.e. each operation that affects multiple tuples was split into single operations, affecting only one tuple each. Also, the old data values were included to enhance the conflict detection mechanism. In this way, we deal with “restricted” operations. The “undo” operation means the realization of INVERSE-* operations (INVERSE-INSERT, INVERSE-DELETE, and INVERSE-UPDATE). Every INVERSE-* operation refers to restricted operations only; it replaces the original operation that has already been committed during the previous transaction and was later marked as loser operation by the conflict resolution strategy.

In the following we will develop one resolution strategy for the database replication scenario. If a conflict occurs, the strategy foresees that this site can autonomously resolve the conflict by querying data from the shared space. We assume that the same conflict resolution strategy is used at each site, and that each transaction has a transaction counter and a state vector. When a transaction is transferred to a target site for replication, it contains these two parameters.

Furthermore we require the existence of a global rule for the determination of winner/loser operations that assigns priorities between all sites. Very important aspect is to be able to identify past operations in transactions not originating from the same site, and remaining operations in the current transaction (after the conflicting operation). Let us first define the notions of *anchor-operation* and *anchor-chain*. The anchor-operation is a replicated operation conflicting with another operation, and it belongs to the current transaction, originating from a remote site. The anchor-chain-operations are operations associated with an anchor-operation in that they conflict with the anchor-operation, and they belong to the current or previous transaction. This way, in the series of operations of some transactions, the interdependency between operations can be established.

When a replicated transaction fails to be executed at a target site, conflict detection starts with identification of the anchor-operation and anchor-chain-operations. For this determination, each transaction’s operation must be investigated and analyzed. After the anchor-chain-operations have been identified, the conflict resolution strategy is applied as follows:

1) if the anchor-operation is a winner over all other anchor-chain-operations:
    a. for each $op_i \in$ anchor-chain-operations in buttom-up order do: apply the INVERSE-* operation to op_i
    b. execute op_i
    c. execute the remaining operations of the current transaction
2) else (the anchor-operation is a looser): ignore this operation and execute the remaining operations of current transaction

6 REALIZATION

An implementation was performed in cooperation with industry that implemented this peer-to-peer database replication scenario used by major clients in Austria (Kühn, 2003). However, instead of the resolution strategy described in section 5, a simpler one has been implemented that does not use transaction counters, state vectors, anchor operations, nor anchor-chain-operations.

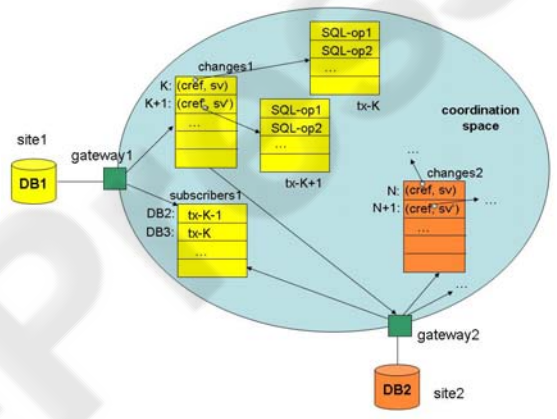

Figure 2: Peer-to-Peer Replication with complex Conflict Resolution Strategy.

The resolution strategy provides:

- manual conflict resolution, providing the administrator all old and new values, and
- locally built-in rules at each site S where for any other site S’ and each possible conflict the looser/winner relationship is explicitly defined plus a resolution action.

In Figure 2 we sketch the implementation of the conflict resolution strategy as proposed in section 5. Basically, the coordination data structures in the space for the replication of one table from site1 (DB1) to site2 (DB2) are shown. The full peer-to-peer replication scenario assumes these for all peer sites in a symmetric way.

7 CONCLUSION

Although asynchronous replication offers well-known advantages in distributed IT systems, many obstacles appear during their realization. As the main goal remains that all data at all peer sites must be consistent at the end, our task is to surpass these problems. In this paper, we focused on a general coordination infrastructure for asynchronous data replication. We showed its application for the replication of data for relational databases. We analyzed conflicts that can appear, classified them, described the ways of appropriate conflict detection and proposed some methods for conflict resolution. The proposed solution works for heterogeneous databases from different vendors.

Our next work will be to extend the coordination data stored in the space infrastructure by using semantics. The objective is to make use of an extension to space based computing called “triple computing” (Riemer at all, 2006) that provides RDF data which can be used to share the rules for conflict resolution.

As conflict resolution for asynchronous replication in general is still an open research issue, our future work will deal with finding a generalized solution for different types of user data, *using the proposed space-based replication pattern* that provides a convenient way to access replication changes and meta-data for coordination, and investigate further resolution strategies. This work is a first step to achieve conflict classification, detection and resolution onto the abstract level.

ACKNOWLEDGEMENTS

We would like to thank the anonymous reviewers for their valuable comments.

REFERENCES

Budiarto, Nishio S., Tsukamoto M., 2002,.Data management issues in mobile and peer-to-peer environments, *Data & Knowledge Engineering 41*, pp. 183–204.

Chigrik A., 2000. Setting Up Transactional Replication: A Step-by-step Guide, *Database Journal*, http://www.databasejournal.com/features/mssql/article.php/1438201.

Dunlop N., Indulska J., Raymond K., 2003. Methods for Conflict Resolution in Policy-Based Management Systems, *EDOC'03*, p.98.

Gelernter D., Carriero N., 1992. *Coordination Languages and Their Significance*, Comm.of ACM, 35, pp.97-107.

Garmany J., Freeman R., 2003. *Oracle Replication. Snapshot, Multi-Master & Materialized Views* Scripts, Rampant TechPress, Kittrell, North Carolina.

Gustavsson S., Andler S.F., 2005. Continuous Consistency Management in Distributed Real-Time Databases with Multiple Writers of Replicated Data, *IPDPS'05 Workshop 2*, p. 137.

IBM DB2 Universal Database, 2002. *Replication Guide and Reference*, Version 8, SC27-1121-00, http://publib.boulder.ibm.com/cgi-bin/bookmgr/library.

IBM DB2 Redbook, 2002. *A Practical Guide to DB2 UDB Data Replication* V8, http://www.redbooks.ibm.com/redbooks/SG246828.html

IBM DB2 *DataPropagator*, 2003.
http://www-306.ibm.com/software/data/integration/replication/

Kühn e., 2001. Virtual Shared Memory for Distributed Architecture, *Nova Science Publishers*, 2001.

Kühn e., 2003. The Zero-Delay Data Warehouse: Mobilizing Heterogeneous Databases, *VLDB Conf.* Berlin.

Kühn e., Riemer J., Joskowicz G., 2005. XVSM (eXtensible Virtual Shared Memory) Architecture and Application, Technical Report TU-Vienna, E185/1.

Kühn e., Ruhdorfer A., Sesum-Cavic V., 2007. Classification and Detection of Resolution Conflicts for Relational Databases, Technical Report TU-Vienna, E185/1.

Oracle 9i 2002. *Advanced Replication Release*, 2 (9.2), Part No.A96567-01, http://www.lc.leidenuniv.nl/awcourse/oracle/server.920/a96567/title.htm,

Riemer J., Martin-Recuerda F., Ding Y., Murth M., Sapkota B., Krummenacher R., Shafiq O., Fensel D., Kuehn e., 2006. Triple Space Computing: Adding Semantics to Space-based Computing,. Proc. 1st Asian Semantic Web Conf., Beijing, China.

Ruhdorfer A., 2005. *Conflict Detection and Conflict Resolution for Asynchronous Database Replication*, Diploma Thesis, TU-Vienna, E185/1.