A PROTOCOL TO CONTROL REPLICATION IN DISTRIBUTED REAL-TIME DATABASE SYSTEMS*

Anis Haj Said, Bruno Sadeg, Laurent Amanton
LITIS, UFR of Sciences and Techniques, 25 rue Philippe Lebon BP 540, 76 058, Le Havre Cedex, France

Béchir Ayeb
PRINCE, Faculty of Sciences, Boulevard de l'Environnement, 5000, Monastir, Tunisia

Keywords: Real-time database, distributed system, transaction, replication , replication control protocol, data consistency.

Abstract: Data replication is often used in distributed database systems to improve both fault tolerance and performance purposes. However, such systems must ensure replicas consistency. In this paper, we discuss the contributions of data replication in distributed real-time database systems (DRTDBS) and we then present RT-RCP, a replication control protocol we designed for DRTDBS. We introduce a new entity called List of available copies *(LAC)* which is a list related to each data item in the database. A LAC of a data item contains all or a part of references of its updated replicas. This allows the database to have a dynamic level of replication.

1 INTRODUCTION

Data replication has been studied in many research areas, especially in distributed database systems for both fault tolerance and performance purposes. The main interest of data replication in DRTDBS is to increase data availability in order to enhance the chances of transactions to meet their deadines. However, allowing transactions to deal with different copies of the same data can generate inconsistencies (Gray et al., 1996) which brings us to be circumspect about data consistency.

Several replication control protocols have been proposed in order to manage replicas. We distinguish two main models (Gray and Reuter, 1993) : eager update model (Bernstein et al., 1987) in which a transaction synchronises with all copies before it commits, and lazy update model (Pu and Leff, 1991) in which changes introduced by a transaction are propagated to other sites only after the transaction has committed.

In addition, we make out two architectures which determine the transactions execution placement (Wiesmann et al., 2000). The *primary copy* architecture in which each data has a primary copy at a specific site (server), and the *update everywhere* architecture in which transactions can be executed at any site and then updates are transmitted to all other sites. Some studies are interested to replication in DRTDBS. They take into account data and transactions specifications. The replication approaches proposed either use a primary copy to deterministically apply updates to replicated data (Wei et al., 2004) or use distributed concurrency control protocols and distributed commit protocols to order updates so that one copy serializability (Xiong et al., 2002) or some similar correctness criterion such as epsilon-serializability (Son and F.Zhang, 1995) can be fulfilled.

Most reaserches conducted on replication in DRTDBS have adressed replication control in update transactions. Whearas user transactions are as important as update transactions because both affects the system performance measurment since they are both real-time transactions. Our work focusses on replication control in user transactions processing.

The remainder of this paper is organized as follows: Section 2 describes our RTDBS model whereas in section 3 we discuss about the use of existing replication protocols in DRTDBS. In section 4 we present our proposed protocol RT-RCP. Finally, Section 5 concludes the paper.

*This work is supported by the Haute-Normandie region **CPER** project "*Logistic Transport Network and Information Technics*".

Haj Said A., Sadeg B., Amanton L. and Ayeb B. (2008).
A PROTOCOL TO CONTROL REPLICATION IN DISTRIBUTED REAL-TIME DATABASE SYSTEMS.
In *Proceedings of the Tenth International Conference on Enterprise Information Systems - DISI*, pages 501-504
DOI: 10.5220/0001712805010504

2 MODELS AND ASSUMPTIONS

The database is modelled as a collection of data items fully replicated at each node. It deals with variant data (also called real-time data), with which are associated validity intervals and invariant data (also called non real-time data) which values will not change with time, i.e. their validity intervals are infinite.

Regarding transactions, we focus only on firm real-time transactions. (Ramamritham et al., 2004). We distinguish two types of real-time transactions:
- Update transactions : These transactions refresh variant data before exceeding their validity interval.
- User transactions : Generally, they are launched by users or triggered by system events. These transactions can perform reads on variant data and/or reads/writes on invariant data.

In the following, we present a data model based on a data classification according to their criticality and characteristics (Shu et al., 2002). Let D denotes the data set of the database. We denote by $D_{critical}$ the set of critical data (handled only by real-time transactions). This set can be devided into two subsets. We denote by $D^{var}_{critical}$, the subset of $D_{critical}$ including variant data and by $D^{invar}_{critical}$ a subset of $D_{critical}$ including invariant data. We call non-critical data, the data items which are not invoked by real-time transactions. $D_{non-critical}$ denotes this set of data.

In addition, we consider a DRTDBS which consists of main memory real-time databases connected by high-speed network. We assume that messages sent over the network have predictable transmission and delivery time. Besides, messages issued from a single node are delivered in the order they are sent.

3 REPLICATION IN DRTDBS

For DRTDBS, a replication control protocol should ensure mainly the availability and the accessibility of critical data in order to allow the system to resume its main functionality, and thus improve its performance which is the ratio of transactions which meet their deadlines. In the same time, data consistency must be guaranteed to avoid the use of stale data by transactions. A simplest idea consists of adopting an eager model for critical invariant data, which ensures data copies consistency. However, using this model can significantly raise the probability of deadlocks, and consequently failed transactions with transaction size and with the number of nodes (Gray et al., 1996). In fact, updating all critical invariant data copies before the transaction commit increases the transaction size especially when the number of nodes in the network is significant. The lazy replication model, where changes introduced by a transaction are propagated to other sites only after the transaction has commited, is often used in many classical database systems.

This results in minimal overhead but inconsistencies among the copies often arise. It is well that inconsitencies created by lazy replication can be very difficult and expensive. In order to solve this problem, database systems use reconciliation protocols based on timestamping technique. These reconciliation protocols generate the discard and then the restart of transactions which have used stale data. Discarding and restarting transactions can significantly affect the system performance. These two models seem to be not suitable to control replication in DRTDBS.

The common drawback of these two techniques is that sites do not have any particular information about the state of the database at the other sites. In fact, for lazy replication, transactions are executed locally without carriyng about the state of replicas of the involved data. Inconsistencies are detected after the completion of the transaction execution. Whereas for eager replication, transactions are forced to update copies of the involved data and then preclude inconsistencies to occur. In the following, we present RT-RCP (Real Time Replication Control Protocol) to manage replicas of data in $D^{invar}_{critical}$. RT-RCP allows some inconsistencies to happen between copies but prevents access to stale copies.

4 THE RT-RCP PROTOCOL

Fully replicating a database leads us to be vigilant about inconsistencies that may arise between critical data copies. In fact, when a transaction updates a data item, its copies must be updated or locked until its update. Otherwise, other transactions can use stale data.

Since the time needed to transmit a message is predictable, a node can estimate the number of updates which are possible to perform without exceeding the transaction deadline. In fact, if the remainder time before the transansaction deadline is greater than the maximum time needed to transmit updates, then data synchronisation is done immediately, otherwise updates are differed after the commit. The main idea of our protocol is to use a mixed strategy in order to update replicas of critical invariant data. Indeed, in RT-RCP, a node where the transaction executes updates synchronously as many nodes in the network as possible and differs the remainder updates after committing the transaction.

Figure 1 shows the use of RT-RCP in order to update the copies of a data item which is involved in a

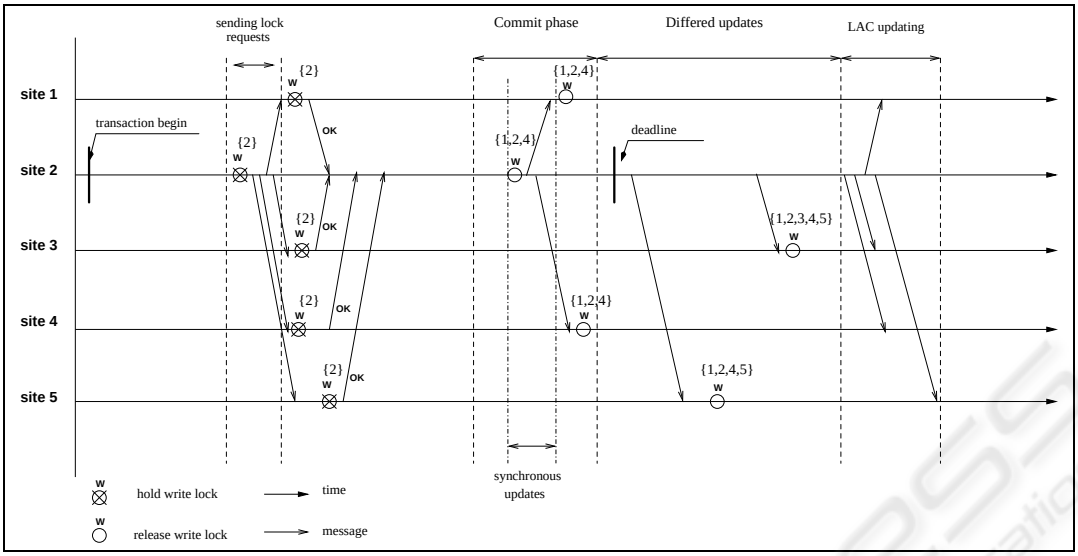

Figure 1: Processing of RT-RCP.

transaction. We consider a distributed system which consists of five sites. Let d a data item in the database involved by the transaction T which is performed on site 2. When transaction T needs a write access to the data item d, it requires write locks on the local data item and its copies. In fact, we use a distributed locking approach. If the lock is granted by all sites, we can proceed. If not, the transaction can be delayed and the request is repeated some time afterwards.

During the commit phase, site 2 updates two copies of d. It can not perform more updates due to transaction deadline which could be exceeded. After the transaction commit, site 2 propagates updates to copies of d which are not updated yet. RT-RCP tries, on the one hand, to respect the transaction temporal constraint and to make available data copies on the other hand.

However, this protocol has a drawback that can be summarized by the following situation.

Let T' a transaction launched on site 4 after the synchronous updates performed by the transaction T. T' is split into subtransactions $T'_1, ..., T'_n$ ($1 < n \leq N$,N is the number of sites). Since the database is fully replicated and sites have no information about the state of the database on the other sites, subtransactions of T' are sent to the participating sites which are choosed randomly. Let T'_i ($1 \leq i \leq n$) be a subtransaction which needs access to the data item d launched on site k, with $1 < k \leq n$. Two cases can arise. - The copy of d on site k is already updated and then the subtransaction T'_i will be executed normally
- The copy of d on site k is not updated yet and then subtransaction T'_i will be blocked until site k receives the new value of d and releases the write lock.

In the latter case in which the subtransaction T' is blocked, the transaction could miss its deadline and as consequence it will be aborted. Whereas subtransaction T' could have more chances to meet its deadline if it was launched on a site in which the data item d is updated. In order to prevent this situation, sites must have knowledge about the state of the database at the other sites of the distributed network. This help sites to make a suitable decision when they choose participant sites to the transaction execution. Associating a list of available copies (LAC) with each data item, allows sites to make a suitable decision when choosing participant sites in a transaction. Each site which holds locks on a data item and its copies must update LACs of this data item at the other sites of the distributed network.

4.1 LACS Updating

LACs must be up-to-date, or at least should not contain wrong informations. Updating LACs is a two step process. The first, consists of joining a new LAC with each data update message. The second begins when all replicas receive updates. We go back to our example to show how LACs are built and updated (Figure 1).In RT-RCP, a LAC is joined with each update message sent by site 2 to other sites. Since site 2 holds write locks on the data item d and it is the site which updates d replicas, then it is the only one which has information about the state of each replica of the data item d.

We assume that intially replicas of d are up-to-date in all sites, thus each LAC related to a copy of the data item d contains the five sites identifiers. Ob-

viously LACs can contain other information. When the transaction T executed on site 2 needs a write access to the data item d, it locks the local copy of d and sends write lock requests to each replica. Once a write lock is hold on a copy, its LAC is modified and it contains only the write lock sender identifier. Each site sends to site 2 a confirmation message when the local copy of d is locked, and LACs are modified to contain only site 2 Identifier. From this stage, site 2 behaves as a primary copy of the data item d since each transaction which needs an access to d is automatically sent to site 2.

At the commit phase, site 2 updates synchronously as many replicas of d as possible. Since messages have predictable transmission and delivery time, site 2 can estimate the number of updates that it can perform without missing the deadline of transaction T. For the data item d, site 2 can update only two copies, then it chooses two sites (site 1 and site 4) for which it will send updates. A new LAC is joined to the message in which appear the three identifiers of the updated sites. The main goal of sending updates before committing the transaction is to avoid overloading site 2 by making available replicas of d.

After the committing phase, site which performs the transaction sends updates and a new LAC to each site in which copy of d is not updated yet. Thus, site 2 sends an update message to site 3 and site 5 and joins to each message a new LAC. Each LAC contains, in addition to the receiver identifier, identifiers of sites in which replicas of d are up-to-date when the message is sent. At this stage of the protocol, all copies of the data item d are updated. However, LACs contains different informations according to the order of receiving updates. Once site 2 has finished updating sites, it brodcasts to all sites a new LAC which contains all sites identifiers.

We note that RT-RCP introduces a new round of messages exchange in order to update LACs. The aim of this messages exchange is to inform sites that a data item is already up-to-date everywhere in the network. Without this messages, the database seems to be not fully replicated. This means that the degree of replication for a data item viewed by each site is not the same. Keeping the system at this state can be benifical when the system is overloaded. Eliminating the differed updates and the final LACs updating step in case of system overload could not affect efficiency of RT-RCP since updates may be done by a new transaction. This can be an interesting issue for DRTDBS since changing the behavior of RT-RCP according to the system load and transactions requirements can greatly improve the performance of DRTDBS.

5 CONCLUSIONS

RT-RCP finds a trade-off between respecting temporal constraints of real time transactions and updating replicas. In fact, RT-RCP is not anxious to update all replicas synchronously. We have a dynamic degree of replication which changes with the system load and the real time transactions requirements. RT-RCP ensures the use of fresh data by real time transactions. Indeed, sites refer to the LAC of each data item in order to make an adequate choice about participant sites in the execution of a transaction. Since a LAC related to a data item is delivred each time by the site which holds write locks on this data item, sites which appear in this LAC contain an updated copy of the data item.

REFERENCES

Bernstein, P. A., Hadzilacos, V., and Goodman, N. (1987). *Concurrency Control and Recovery in Database Systems*. Addison-Wesley.

Gray, J. N., Holland, P., Shasha, D., and O'Neil, P. (1996). The dangers of Replication and a Solution. In *1996 ACM SIGMOD on Management of Data*, pages 173–182, Montreal, Canada.

Gray, J. N. and Reuter, A. (1993). *Transaction Processing:Concepts and Techniques*. Morgan Kaufmann Publisher.

Pu, C. and Leff, A. (1991). Replica Control in Distributed Systems: An Asynchronous Approach. In Clifford, J. and King, R., editors, *Proc. of the ACM SIGMOD on Management of Data*, pages 377–386.

Ramamritham, K., Son, S., and Dipippo, L. (2004). Real-Time Databases and Data Services. *Real-Time Systems*, pages 179–215.

Shu, L., Stankovic, J., and Son, S. (2002). Real-Time Logging and Failure Recovery. In *In IEEE Real-Time Technology and Applications Symposium*.

Son, S. H. and F.Zhang (1995). Real-Time Replication Control for Distributed Database Systems: Algorithms and Their Performance. In *Proc. of the 4th Intl. Conf. DASFAA'95, Singapore*, pages 214–221.

Wei, Y., Aslinger, A. A., Son, S. H., and Stankovic, J. A. (2004). ORDER : A Dynamic Replication Alogorithm for Periodic Transactions in Distributed Real-Time Databases. In *10^{th} Int'l Conf. RTCSA 2004*.

Wiesmann, M., Pedone, F., Schiper, A., Kemme, B., and Alonso, G. (2000). Database Replication Techniques: A Three Parameter Classification. In *Proceedings of 19th IEEE Symposium on Reliable Distributed Systems (SRDS2000)*, Germany.

Xiong, M., Ramamritham, K., Haritsa, J. R., and Stankovic, J. A. (2002). Mirror: a State-Conscious Concurrency Control Protocol for Replicated Real-Time Databases. *Inf. Syst.*, pages 277–297.