The DLR-ORECOP Real-Time Replication Control Protocol

Anis Haj Said, Bruno Sadeg, Bechir Ayeb*, Laurent Amanton
University of Le Havre
LITIS Laboratory
25 rue Philippe Lebon, 76058 Le Havre Cedex, France
{anis.haj-said, laurent.amanton, bruno.sadeg}@univ-lehavre.fr

* University of Monastir
PRINCE Laboratory
Boulevard de l’environnement, 5031 Monastir, Tunisia
bechir.ayeb@fsm.rnu.tn

Abstract

Many real-time applications need data services in distributed environments. Providing such data services is a challenging task due to long remote data accessing delays and stringent time requirements of real-time transactions. In this paper, we use data replication in distributed real-time database systems in order to improve meeting these stringent requirements and fault tolerance. We propose an optimistic replication control protocol, called DLR-ORECOP (Dynamic Level of Replication with Optimistic REplication COntrol Protocol), which finds a trade-off between updating replicas and meeting user transactions deadlines. We introduce a list, called List of Available Copies (LAC), associated with each data item in the database which contains the identifiers of the most updated replicas. Fault tolerance is provided by building LACs dynamically according to transactions executions and system load, giving then the real-time database a dynamic level of replication.

The experimental results show that among the replication control protocols evaluated, DLR-ORECOP provides the best performance for a variety of workloads and system configurations. In addition, the experimental results show that DLR-ORECOP performances are not significantly affected by the loss of update messages.

1 Introduction

In recent years, researches in real-time database systems have emerged to allow real-time systems to manage a large number of data. This research area is increasingly interested to distributed real-time database systems, since real-time applications are mostly distributed by nature. For example, in naval combat control systems, data are scattered through different combat ships, submarines, satellites and the army control center. Several papers have proposed many features in order to meet both real-time transactions and data requirements in a distributed environment. Some researches have addressed the transaction commit processing in distributed real-time database systems (DRTDBS). The main goal is to design distributed commit protocols which take into account the real-time transactions requirements. OPT [8] [7] and PROMPT [11] are two phase based commit protocols (2PC) [13] [5] proposed to this purpose.

Distributed applications frequently use data replication as a means to achieve a higher level of performance, availability, reliability and fault tolerance. Consequently, the management of replicated data has emerged as a problem of great practical importance in recent years. Several replication models have been proposed. We distinguish three main models, namely Eager Replication (also called synchronous model) [6] [3], Lazy Replication (also called asynchronous model) [6] [15] and On-Demand Replication [2] [1] [12] [9] [18]. These replication models adopt different approaches to manage data replication. Eager Replication [6] [3] uses a pessimistic concurrency control protocol to forbid the access to replicas of modified data and synchronises all replicas before it commits. In Lazy Replication [6] [15], transactions are executed without worrying about the freshness of the handled data and the changes introduced by transactions are propagated only after the transaction has committed. This leads to minimal overhead but inconsistencies among replicas often arise. In order to ensure data consistency, database systems use reconciliation protocols based on stamping technics. Thus, transactions which have used dirty data are discarded and restarted. Eager and Lazy Replication control protocols are not suited to the real-time context because they don’t take into account the temporal constraints on data and transactions in real-time databases. Regarding On-Demand replication protocols [2] [1] [12] [9] [18], updates are performed...
when a transaction needs accessing a replica. On-Demand protocols are generally used with primary-copy architecture where the location of the most updated replica is static. Another approach based on a similar correctness criterion such as Epsilon-Serializability [17] can also be fulfilled. Epsilon-Serializability is based on the fact that in distributed real-time database systems, timeliness of results can be more important than their correctness [4].

Most of the researches conducted on replication in DRTDBSs have addressed the control of real-time data replication [2] [1] [12] [9] [18] [20] [17]. However, non real-time data are as important as real-time data because both are involved in real-time transactions, and then they are important for the system performance. In addition, due to the different characteristics of data in RTDBS, a suitable replication control protocol for real-time data is not necessarily suitable for non real-time data. Indeed, generally update transactions are periodic which gives them a static behaviour and their execution is not distributed since only one site performs updates for one real-time data and then propagates it to the other replicas. Whereas, user transactions are not necessarily periodic and they are split into subtransactions according to data availability, thus a distributed execution is necessary.

A replication control protocol for DRTDBS must find a trade-off between ensuring replicas consistency and meeting transactions deadlines. This leads us to wonder about the level of replication of the database. Is fully replicated database the best approach? Is there a level of replication of the database that gives the best system performance? Is the level of replication should change according to the system load? What replication model could manage a partially replicated database? The answers to these questions are not given by the existing replication strategies. The common drawback of the different replication strategies cited above is that the number of reads and writes performed by transactions are fixed in advance (one, all, all available, a quorum). With these strategies, sites do not have any particular information about the state of the database at other sites.

In our work, we focus on replication of non real-time data handled by real-time transactions in DRTDBS. We present DLR-ORECOP, a replication control protocol which can be integrated into two-phase based real-time commit protocols. The main contributions of DLR-ORECOP is (1) to propagate optimistically updates and (2) to provide a dynamic level of replication of the database. In addition, DLR-ORECOP tolerates different kinds of faults that may occur during its execution.

The remainder of this paper is organized as follows. Section 2 describes our models and assumptions. In section 3, we present DLR-ORECOP, a replication control protocol designed for DRTDBS. Section 4 shows how our proposed replication control protocol allows the system to tolerate faults. In section 5, we describe our simulation model and parameters, and we discuss the results of two experiments with our simulator. Finally, Section 6 concludes the paper.

## 2 MODELS AND ASSUMPTIONS

The database is modelled as a collection of data items fully replicated at each node of the distributed system. The database includes real-time and non real-time data. With a real-time data is associated a validity interval. A real-time data value is deemed useful only during its validity interval and gets out of date with the passage of time. A non real-time data denotes a data whose value will not change with time, i.e. its validity interval is infinite.

We associate with each non real-time data a list, called List of Available copies and denoted LAC, which includes a subset of site identifiers which have got the most updated replicas, according to the latest site that updated the non real-time data item.

We consider that the DRTDBS manages both real-time and non real-time transactions. For real-time transactions, we focus only on transactions with firm deadlines [16]. We distinguish two types of real-time transactions:

**Update transactions**: generally, they are periodic sensor transactions and they update data which reflect the state of the environment. Update transactions should refresh variant data before exceeding their validity intervals.

**User transactions**: some transactions launched by users or triggered by some events must be performed before exceeding their deadline, otherwise the result will not have any significance for the system. User transactions can read both real-time and non real-time data, and can write only non real-time data. Writes on real-time data can be performed by update transactions only.

We consider that a user transaction may arrive at any site of the system and predeclares its data needs. If all data needed by the transaction are up-to-date at this site, then the transaction is executed locally, otherwise the transaction is split into subtransactions according to the location of up-to-date data. We assume that user transactions arrive in the system according to a Poisson process with an average rate \( \lambda \).

The distributed system consists of a set of sites interconnected by high-speed network. Sites exchange messages through the network and some of these messages could be lost. A probability of message loss, denoted \( P_{\text{loss}} \), is associated with the distributed system. In order to control the distributed execution of transactions, we use, among two-phase based commit protocols, the PROMPT commit protocol [11]. PROMPT is the main commit pro-
tocol used in DRTDBS and the most performant [11].

3 DLR-ORECOP Protocol

The originality of our protocol consists of finding a trade-off between ensuring the database consistency and meeting user transactions deadlines. This trade-off is ensured by introducing a new entity, called List of available copies (LAC) which allows the database to have a dynamic level of replication. LACs are built dynamically according to the commit protocol processing and the transactions execution outcome. In the best case, a LAC includes all sites identifiers (all replicas of non real-time data item are up-to-date), but it may include only a subset of sites identifiers (only a subset of replicas are up-to-date and here the database seems to be not fully replicated). DLR-ORECOP can be classified in the category of optimistic protocols. The optimism is based on the assumption that most real-time transactions commit before their deadline. Based on this assumption, we allow each cohort to optimistically propagate its data updates without waiting for the global decision of the coordinator.

The protocol is executed in four phases (Figure 1). The first phase consists of a distributed locking phase. It begins when a write operation on a non real-time data item is executed. The second phase begins after the completion of the subtransaction processing. It consists of propagating optimistically updates performed by the subtransaction and building new LACs. The cohort breaks off sending update messages from the beginning of the voting phase of the commit protocol. The third phase, consists of broadcasting new LACs, built at the previous phase, to the remainder sites of the system (except the coordinator). The latter phase is a validation phase (applying data and LACs updates). This phase is integrated in the decision phase of the commit protocol. In this phase, sites decide to validate or nullify updates received from cohorts. In the following, we describe briefly the different phases of DLR-ORECOP protocol. For more details on the DLR-ORECOP processing, see [10]

3.1 The distributed locking phase

We assumed that the database is fully replicated at each site. In order to ensure the database consistency, sites must propagate updates performed by local transactions. During transaction processing, accessing replicas of modified data must be avoided until their update. In fact, when a transaction needs to perform a write operation on a non real-time data item, it must at first hold a write lock on the local copy and on all its replicas. To this purpose, we use a distributed locking approach by implementing distributed 2PL as a concurrency control protocol. Thus, if the lock is granted by all sites, then the transaction can proceed. If not, it can be delayed and the request is repeated some time afterwards. When a replica of a data item is write-locked, its LAC is modified and contains only the identifier of the site in which the transaction that holds the write lock is executed. At this time, any incoming transaction at any site of the system which needs an access to this data is automatically launched on the site whose identifier is contained in the LAC related to that data item. In fact, when a transaction is launched at one site, it refers to the LACs of the needed data in order to locate the most recently updated replicas. Here, the site which holds write locks behaves like a primary copy.

3.2 Optimistic updates propagation

Once the transaction processing is finished, the cohort sends a WORKDONE message to its coordinator and waits for the beginning of the voting phase which should be initiated by the coordinator. The duration of the waiting time depends on the reception time of the last WORKDONE message by the coordinator. This time can be relatively long since it depends on sites workload, length of the execution times of subtransactions and messages delivery time. During this period, if no active abort process [11] has been launched by other cohorts, we optimistically assume that the transaction is normally executed and will be successfully committed. Based on this assumption, we allow each cohort, which has successfully completed its subtransaction processing, to optimistically propagate prepared data updates and then waits for acknowledgements. The identifier of each sender of an acknowledgement message is added to the LACs associated with updated data items. Thus, a LAC built by the cohort contains sites identifiers in which replicas will be certainly updated if the transaction commits. A cohort stops propagating updates as soon as it receives the PREPARE message from the coordinator. It sends YES vote to the coordinator and waits for the reply message which is the decision of the outcome of the global transaction. Thus, updates propagation do not require any additional time which can cause the global transaction missing its deadline.

3.3 LACs propagation

At this stage, the cohort has entered a prepared state wherein it cannot unilaterally commit or abort the transaction, but has to wait for the final coordinator decision. When the coordinator initiates the voting phase, the transaction abort becomes unlikely since all cohorts have successfully performed their subtransactions and then survived to each possible concurrency conflict. While the transaction waits for the global decision, it broadcasts new LACs, built at the previous phase, to the other sites. The main goal is to allow sites to make a suitable decision when an incoming transaction needs an access to an updated data.

3.4 Global decision propagation

The end of the prepared state takes place when the global decision sent by the coordinator is received by the cohort. Since the outcome of subtransactions depends on
the decision of the coordinator, the validation of updates also depends on this decision. Based on this fact, sites nullify all data updates made by subtransactions if the coordinator decide to abort the global transaction. Otherwise, sites validate updates as usual. According to DLR-ORECOP protocol, some replicas do not receive updates yet. A replica is available if its site identifier is included in its LAC. Then, releasing locks do not affect the accurateness of transactions results. In fact, the access to stale data is indirectly forbidden by LACs.

4 Fault Tolerance in DLR-ORECOP

In this section, we show the usefulness of LACs in order to tolerate faults. The main drawback in using data replication is the heavy load generated when ensuring replicas consistency. If cohorts need to synchronise all replicas before committing a transaction, it leads to an extension of the execution time of transactions and then increases the probability to miss their deadlines. This problem becomes more important when failures while transitting update messages occur. In the following, we show that using LACs allows the system to relax this constraint in order to enhance the chances for transactions to meet their deadline and to tolerate faults.

4.1 Tolerating replica inconsistencies

As we have shown in section 3, the optimistic propagation phase is interrupted when the voting phase is initiated by the coordinator. The main purpose of the updates propagation interruption is to avoid that ensuring replicas consistency affects meeting transaction deadlines. Despite the inconsistencies that occur between replicas, the system performance is not affected since transactions use only up-to-date replicas. This is guaranteed by LACs which contain only site identifiers that have confirmed their replicas updating (by sending an acknowledgement for update message (UM-ACK)). Thus, when a site refers to LACs, it launches transactions at the suitable sites in which replicas are certainly up-to-date.

Figure 2 shows an example of DLR-ORECOP processing in which inconsistencies between replicas of a data item are allowed. In the example, site 3 is the coordinator of the global transaction, site 2 is a cohort which executes a subtransaction of the global transaction and the remaining sites are passive. Here, after the cohort has sent a WORKDONE message to its coordinator, LACs associated with replicas of the data item locked by the subtransaction executed at site 2 contain only the cohort identifier (S2). The updates propagation phase is broken off by the voting phase initiated by the coordinator and only site 1 is added to the LAC associated with the updated data item. Thus, after the propagation of LACs, each site can locate up-to-date replicas of that data item despite the presence of inconsistencies between them.

4.2 Tolerating missing update messages

LACs are built dynamically by cohorts that hold write locks. The identifier of each site which has received an update message from a cohort, is not automatically added.
to LACs associated with the updated data. It is only done when an acknowledgement of the update message is received by the cohort. Otherwise, the site is considered as not updated yet due to eventual crashes to an ordinary missing messages in the network or also to an overload state. Thus, cohorts are not forced to guarantee the update of all replicas in order to avoid the extinction of transaction execution time.

![Figure 3. Missing update messages](image)

Figure 3 shows an execution of DLR-ORECOP in a case of missing update message. Here, the update message addressed to site 4 is missed. This does not cause errors since the identifier of site 4 could not be added to the LAC of the updated data. Then, the replica of that data item at site 4 is not available for access, thus, no transactions will use it.

4.3 Tolerating missing LAC update messages

Here, after a cohort has sent its vote, it propagates new LACs. If some messages are lost, there is no particular process to recover these messages. In fact, even if a site don’t receive a LACs updating message, its LACs still contain the C_{LD} (the lock holder) which is necessarily up-to-date. Thus, the access to an up-to-date replica is still possible. The process with which LACs are modified allows to have always at least one site in which replicas (associated with these LACs) is up-to-date. This gives to DLR-ORECOP the ability to conceal missing LACs updating messages.

In figure 4, site 4 don’t receives the LACs updating message. The LAC associated with its replica contains only the cohort identifier in which the replica is certainly up-to-date. This information is sufficient for site 4 since it knows at least the location of one up-to-date replica. Thus, any incoming transaction which needs an access to the stale replica on site 4 will be launched at site 2.

5 Simulation Model and Results

5.1 Simulation Model

To evaluate the performance of the DLR-ORECOP protocol, we have developed a simulation model of a distributed real-time database. A summary of the parameters used in the simulation model is presented in Table 1. The database is modelled as a collection of data items fully replicated over NumSites sites. The database contains NbData of non-real-time data and NbRTData of real-time data. Initially the real-time database is fully replicated at each site and all data items of the database are supposed to be initially up-to-date. This implies that before the beginning of the simulation, LACs associated with non-real-time data include all sites identifiers. The physical resources at each site consist of a NbCPU CPU. At each site, there is a single common queue for the CPUs. The communication network is simply modeled as a switch that routes messages and the CPU overhead of message transfer is neglected. However, the transmission time of messages is taken into account and defined by the parameter T_{trans}. Another parameter, P_{loss}, is associated to messages and denotes the probability of a message to be lost in the network.

The DRTDBS manages both update and user transactions. User transactions arrive in a Poisson stream with rate λ, and each transaction has an associated firm deadline, assigned as described below. A user transaction chooses randomly a site in the system to be the site where the transaction is originated. The transaction may be executed locally if, at its arrival date, all local replicas of accessed data are up-to-date. Otherwise, the transaction

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Meaning</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>NumSites</td>
<td>number of sites in the distributed system</td>
<td></td>
</tr>
<tr>
<td>NbData</td>
<td>number of non-real-time data in the database</td>
<td></td>
</tr>
<tr>
<td>NbRTData</td>
<td>number of real-time data</td>
<td></td>
</tr>
<tr>
<td>Duration (s)</td>
<td>duration of the simulation</td>
<td>1000</td>
</tr>
<tr>
<td>NbCPU</td>
<td>number of CPU</td>
<td>V</td>
</tr>
<tr>
<td>λ</td>
<td>arrival rate</td>
<td>20</td>
</tr>
<tr>
<td>t_{trans} (ms)</td>
<td>time of transmitting message</td>
<td></td>
</tr>
<tr>
<td>t_{trans} (ms)</td>
<td>probability of message to be lost</td>
<td>10, 15</td>
</tr>
<tr>
<td>t_{rate} (ms)</td>
<td>duration of real operation</td>
<td></td>
</tr>
<tr>
<td>t_{rate} (ms)</td>
<td>duration of write operation</td>
<td>[1, 2]</td>
</tr>
<tr>
<td>N_{read}</td>
<td>number of reads per transaction</td>
<td>[5, 10]</td>
</tr>
<tr>
<td>N_{write}</td>
<td>number of writes per transaction</td>
<td>[1, 2]</td>
</tr>
<tr>
<td>SF</td>
<td>slack factor in deadline assignment</td>
<td>6.0</td>
</tr>
<tr>
<td>Validity</td>
<td>validity of a real-time data</td>
<td>[75, 100]</td>
</tr>
</tbody>
</table>
forks off subtransactions at all the sites where it has to access up-to-date replicas, and the site where the user transaction originates becomes the coordinator of the distributed execution. We consider that subtransactions are executed in sequential manner. Regarding update transactions, the policy used to update real-time data is the More-Less policy [19]. There is no replication control protocol used to update real-time replicas. The same update transaction arrives at all sites at its arrival date and is executed locally at each site. The aim of this assumption is to study only the performance of replication control protocols for non real-time data in a replicated real-time database system.

We consider that user transactions are a succession of read and write operations. A user transaction can read both real-time and non real-time data, and can write only non real-time data. However, update transactions perform only one write operation on one real-time data. The cost of read and write operations are defined respectively by $t_{\text{read}}$ and $t_{\text{write}}$. The number of read and write operations is defined respectively by $N_{\text{read}}$ and $N_{\text{write}}$. The ressource time needed by a user transaction $Tr$ is denoted $R_{Tr}$ and is defined by the following expression

$$R_{Tr} = \sum_{i=1}^{N_{\text{read}}} t_{i, \text{read}} + \sum_{j=1}^{N_{\text{write}}} t_{j, \text{write}}$$

Where $t_{i, \text{read}}$ and $t_{j, \text{write}}$ are, respectively, the duration of the $i^{th}$ read operation and the duration of the $j^{th}$ write operation.

Upon arrival, each user transaction $Tr$ is assigned a firm deadline using the formula

$$\text{Deadline}_{Tr} = \text{ArrivalDate}_{Tr} + SF \times R_{Tr}$$

Where $\text{Deadline}_{Tr}$, $\text{ArrivalDate}$ and $R_{Tr}$ are the deadline, arrival date and resource time, respectively, of transaction $Tr$, while $SF$ is a slack factor that provides control of the tightness/slackness of transaction deadlines. Priorities are assigned to each user transaction so as to minimize the number of killed transactions. In our model, all subtransactions inherit their parent’s priority. The transaction priority assignment used in all of the experiments described here is the widely-used Earlies Deadline First policy [14], wherein transactions with earlier deadlines have higher priority than transactions with later deadlines. The performance metric employed is Success Ratio (SR), defined by the following formula

$$SR = \frac{\text{Number of succeeded transactions}}{\text{Number of Arrived transactions}}$$

These simulation settings will be used in two experiments. The aim of the first experiment is to compare the performance of a set of replication commit protocol in managing data replication involved in user transactions. The second experiment shows the DLR-ORECOP performance in a faulty environment.

5.2 Expt. 1: Benchmark for a set of replication control protocols

Table 1 presents the settings of the simulation model parameters for our first experiment. With these settings, the database is fully replicated and each transaction executes in a sequential manner. The arrival rate $\lambda$ changes and takes values between 0.2 and 2.4. The number of arrived transactions is about 200 when $\lambda = 0.2$, and about 2400 when $\lambda = 2.4$. The parameters $t_{\text{trans}}, t_{\text{read}}, t_{\text{write}}, N_{\text{read}}, N_{\text{write}}$ and $\text{Validity}$ take random values from the corresponding intervals (see Table 1).

Our goal in this experiment is to investigate the performance of the various replication control protocols (Eager Replication, Lazy Replication and DLR-ORECOP) in managing replication of data handled by user transactions. The performance criterion used to compare the three replication control protocols is the success ratio of user transactions. We assume that the lost of messages could not occur during the first experiment, i.e. $\text{P}_{\text{loss}} = 0$.

The experiment results show that in a low system load ($\lambda \leq 1$), the DLR-ORECOP protocol gives a best performance than both Eager and Lazy replication protocols (Figure 5). However, we note that the difference between success ratios given by the three protocols is not substantial. In fact, concurrent accesses to data are unlikely when the system is in low load. Thus, the number of restarted transactions is not important and most of user transactions meet their deadlines. Besides, we observe, that the number of failed transactions grows with Eager Replication. Indeed, the data update phase performed by this protocol causes the failure of some user transactions by exceeding their deadlines.

When the system load increases ($\lambda$ increases), the success ratio given by the three protocols decreases (Figure 5). The DLR-ORECOP performances are still better than Eager and Lazy Replication performances. Concurrent accesses become more frequent, what causes more restarts of transactions. This affects considerably the three protocols performances and especially Lazy Replication, because concurrent accesses to data is detected later than DLR-ORECOP and Eager Replication. In fact, concurrent accesses are detected at the latter phase of the Lazy Replication protocol, which is the data update phase. This causes exceeding restarted transactions deadlines.

The pessimistic concurrency control protocol, 2PL, used with Eager Replication and DLR-ORECOP, allows the system to detect earlier this problem and to prevent the use of stale data. However, the time needed to perform the distributed locking phase of 2PL coupled with the data update phase increase considerably transactions execution time. This decreases Eager Replication performances significantly. DLR-ORECOP minimizes the im-
impact of data updates by integrating this phase of the protocol in the commit protocol. Thus, no extra time is needed for data updates propagation, what gives DLR-ORECOP better performances, even in an overloaded system.

5.3 Expt. 2 : DLR-ORECOP performance in faulty environment

Our goal in the second experiment is to investigate the performance of DLR-ORECOP in the presence of failures during transmitting update messages. As described in the different scenarios in section 4, the lost of update messages can be concealed by DLR-ORECOP. Indeed, there is no process to recover these messages and the transaction continues its execution normally.

In this experiment, only DLR-ORECOP is used as a replication control protocol. We keep the same system settings as in the first experiment except $P_{loss}$ which takes different values. In fact, this experiment consists of six simulations, and for each one, we increase the value of $P_{loss}$ by 0.1 (from 0 to 0.5). Thus, for the first simulation $P_{loss} = 0.0$ and for the last simulation $P_{loss} = 0.5$. We define a new performance criterion, called Relative Success Ratio ($RSR$). It is define by the following formula.

$$RSR(P_{loss} = x) = \frac{SuccessRatio(P_{loss} = x)}{SuccessRatio(P_{loss} = 0)}$$

The experimental results (Figure 6) show that for a normal system load ($\lambda \leq 1$) the loss of update messages do not affect considerably the system performance ($SR > 0.9$ and $RSR > 0.96$). In fact, as explained in section 4, DLR-ORECOP allows inconsistencies between replicas to occur due to message loss, but forbids the access to the stale data by using LACs. As the system is in normal load, and although some replicas are not available, transactions can access up-to-date data without considerably increase the load of sites where these data are located. When the system load increases, the effects of loosing messages became more important especially when $P_{loss}$ increases. Figure 7 shows that for $P_{loss} \geq 0.4$, we have $RSR \leq 0.75$. However, we note that the results given by DLR-ORECOP in an overloaded system with $P_{loss} \leq 0.3$ remains acceptable ($RSR > 0.8$). Tolerating the failure of update messages makes the database seeming to be not fully replicated since only a subset of replicas are updated and the lost messages are not recovered. With this experiment, we note that when a third of update messages are lost ($P_{loss} = 0.3$), the system is not considerably affected and its performance is satisfactory, though non updated data are taken into account in the distributed locking process and in the update process of DLR-ORECOP.

6 conclusion

The DLR-ORECOP finds a trade-off between respecting temporal constraints of user transactions and updating replicas. The integration of the data update process in the commit protocol avoids the use of an extra
phase to propagate updates. Thus, with DLR-ORECOP, updating replicas do not lead to an extension of the execution time of user transactions. This approach gives more chances to these transactions to meet their deadlines.

Fault tolerance in DLR-ORECOP is based on building dynamically the Lists of Available copies (LACs) during the optimistic data updates propagation. The building process of LACs ensures that, even if data update messages are lost, they include only sites identifiers which have reception updates. Thus, DLR-ORECOP tolerates the loss of update messages, and no particular process is used to recover them.

Our experiments show that among Eager Replication, Lazy Replication and DLR-ORECOP, our protocol gives a better system performance. In addition, the experiments show that the system performance when update messages failure occurs depends on both the frequency of these failures and the system workload. The system is not considerably affected when the system is in normal load. However, when the system load increases, the effects of updates messages failures become more consistent when the probability of transmission failure is greater than a third.

References