Improving Recovery in Weak-Voting Data Replication*

Luis H. García-Muñoz, Rubén de Juan-Marín, J. Enrique Armendáriz-Íñigo, and Francesc D. Muñoz-Escoí

Instituto Tecnológico de Informática - Universidad Politécnica de Valencia
Camino de Vera, s/n - 46022 Valencia, Spain
{lgarcia,rjuan,armendariz,fmunyoz}@iti.upv.es

Abstract. Nowadays eager update everywhere replication protocols are widely proposed for replicated databases. They work together with recovery protocols in order to provide highly available and fault-tolerant information systems. This paper proposes two enhancements for reducing the recovery times, minimizing the recovery information to transfer. The idea is to consider on one hand a more realistic failure model scenario –crash recovery with partial amnesia– and on the other hand to apply a compacting technique. Moreover, it is provided amnesia support avoiding possible state inconsistencies –associated to the failure model assumed– before starting the recovery process at recovering replicas.

1 Introduction

Database replication consists in maintaining identical copies of a given database at multiple network nodes. This improves performance, since clients access their local replica or are forwarded to the less loaded one; and availability: whenever a node fails, its associated clients are silently redirected to another available one. Replication protocols can be designed for eager or lazy replication [1], and for executing updates in a primary copy or at all node replicas [2]. With eager replication we can keep all replicas exactly synchronized at all nodes, but this could have an expensive cost. With the lazy alternative we can introduce replication without severely affecting performance, but it can compromise consistency. Many replication protocols are based on eager update everywhere with a *read one, write all available* (ROWAA) approach. As we have briefly highlighted before, these replication protocols provide high availability. However, only a few of them deal with the possible reconnection of the failed node, which is managed by recovery protocols [3,4,5,6].

The aim of the recovery protocols is to bring failed or temporarily disconnected nodes back into the network as fully functional peers, by reconciling the database state of these recovering nodes with that of the active nodes. This could be done by logging transactions and transferring this log to recovering nodes so they can process missed transactions, or transferring the current state of the items that have been updated in the database since the recovering node failed.

* Work supported by FEDER, the Spanish MEC grant TIN2006-14738-C02 and the Mexican DGEST and SES-ANUIES.

M. Xu et al. (Eds.): APPT 2007, LNCS 4847, pp. 131–140, 2007.
© Springer-Verlag Berlin Heidelberg 2007

This paper is focused in the recovery protocol for eager update everywhere replication protocols, proposing some optimizations to the work presented in [6]. These enhancements include amnesia support, and a better performance reducing the amount of data to save in the actions done before recovering and the amount of data to transfer at recovering time. The main idea in the last case is to compact recovery data eliminating redundant information.

The rest of this paper is distributed as follows. Section 2 provides the system model. Section 3 deals with the basic recovery protocol. Section 4 explains the necessary actions for the amnesia support. Next, Section 5 relates the process of compacting recovery information. Later, Section 6 shows the simulation results followed by the related works in Section 7. In the final Section 8, we provide our conclusions.

2 System Model

The basic recovery protocol has been designed for database replicated systems composed by several replicas –each one in a different node–. These nodes belong to a partially synchronous distributed system: their clocks are not synchronized but the message transmission time is bounded. The database state is fully replicated in each node.

This replicated system uses a group communication system (*GCS*) [7]. Point-to-point and broadcast deliveries are supported. The minimum guarantee provided is a FIFO and reliable communication. A group membership service is also assumed, that *knows* in advance the identity of all potential system nodes. These nodes can join the group and leave it, raising a *view change event*. Therefore, each time a membership change happens, i.e. any time the failure or the recovery of one of the member nodes occurs, it supplies consistent information about the current set of reachable members as a view. The group membership service combined with the *GCS* provides *Virtual Synchrony* [7] guarantees, which is achieved using *sending view delivery* multicast [7] enforcing that messages are delivered in the view they were sent. A *primary component* [7] model is followed in case of network partitioning.

The replicated system assumes the *crash-recovery with partial-amnesia* [8] model. This implies that an outdated node must be recovered from two "lost of updateness": forgotten state and missed state. This assumption supports a more realistic and precise way to perform the recovery process. So the assumed model allows to recover failed nodes from their previous crashing state maintaining their assigned node identifiers.

3 Basic Recovery Protocol

Our basic proposal is inspired in the recovery protocol presented in [6]. It has been designed for *eager update everywhere* database replication protocols and proposes the use of *DB-partitions* (see below). It was originally designed for providing recovery support for the *ERP* and *TORPE* [6] replication protocols. Such protocols use a voting termination approach [2], and can be considered as weak voting replication protocols [9]. This basic recovery protocol can be outlined as follows:

– The system has a database table named $MISSED$, which maintains all the information that will be needed for recovery purposes. Each time a new view is installed

a new entry is inserted in the $MISSED$ table if there are failed nodes. Each entry in $MISSED$ table contains: the view identifier, the identifiers of crashed nodes in this view –$SITES$–, and the identifiers list of data items modified during this view –OID_LIST–. The two first ones are set at the beginning of the view, while the last one grows as long as the view passes.

- When a set of crashed nodes reconnects to the replicated system, the recovery protocol will choose one node as the *recoverer* with a deterministic function. Then in a first step the *recoverer* transfers the metadata recovery information to all reconnected nodes. This metadata information contains: the identifiers of modified items, and the crashed node identifiers in each view lost by the oldest crashed node being recovered. The per-view metadata generates a DB-partition during the recovery process; i.e., such items will be blocked while they are being transferred to the recovering node, logically partitioning the database. These *DB-partitions* are also used in order to block in each replica the current user transactions whose modified items conflict with its *DB-partitions*. Subsequently, the *recoverer* starts to recover each *recovering* node view by view. For each lost view, the *recoverer* transfers the state of the modified items during this view. And, once the view has been recovered in the *recovering* node, it notifies the recovery of this view to all alive nodes. The recovery process ends in each *recovering* node once it has updated all its lost views.
- As a transaction broadcast is performed spreading two messages –*remote* and *commit*–, it is possible that a reconnected node receives only the second one, without any information about the updates to be committed. In this case the replication protocol will transfer the associated writesets to these nodes. This behavior implies that transaction writesets are maintained in the sender node until the *commit* message is broadcast.

But this recovery protocol presents the following two problems:

- Amnesia phenomenon. Although we are assuming the *crash-recovery with partial amnesia* [8] failure model, many systems do not handle it in a perfect way. This problem arises because once the replication protocol propagates the *commit* message associated to one transaction, and it is delivered, the system assumes that this transaction is being committed locally in all replicas. But this assumption even using strong virtual synchrony [7] is not always true. It is possible that a replica receives a transaction *commit* message, but before applying the commit the replica crashes, as it is commented in [10] –the basic idea is that message delivery does not imply correct message processing–. The problem will arise when this crashed node reconnects to the replicated system, because it will not have committed this transaction and the rest of the system will not include among the necessary recovery information the updates performed by this transaction, arising then a problem of replicated state inconsistency.
- Large $MISSED$ table and redundant recovery information. If in the system there are long-term crashed nodes –meaning nodes failed during many views– and there are also high update rates it is possible that the $MISSED$ table enlarges significantly with high levels of redundant information, situation that is strongly discouraged. Redundant recovery information will appear because it is possible that the

same item has been modified in several views where the crashed nodes set is very similar. In this case if an item is modified during several views, only knowing the last time –meaning the last view– it was updated is enough. Therefore, it will be interesting to apply algorithms that avoid redundant recovery information, because the larger $MISSED$ tables the greater the recovery information management overhead becomes.

In the following section we will present and study different approaches for solving these problems improving the basic recovery protocol.

4 Amnesia Support

In order to provide amnesia support different approaches can be considered. These approaches can be classified depending on which recovery information they use. On one hand, there are the ones using the broadcast messages –log-based– [3,4] and, on the other hand there are the ones using the information maintained in the database –version-based– [5,6].

But before describing how the amnesia support can be provided in the basic recovery protocol, it must be considered how this amnesia phenomenon manifests. In [11], it is said that the amnesia phenomenon manifests at two different levels:

- *Transport level*. At this level, amnesia implies that the system does not remember *which messages have been received*. In fact, the amnesia implies that received messages non-persistently stored are lost when the node crashes, generating a problem when they belong to transactions that the replicated system has committed but which have not been already committed in the crashed node.
- *Replica level*. The amnesia is manifested here in the fact that the node "forgets" *which were the really committed transactions*.

Hereafter we detail a log-based solution for the amnesia problem. There are other amnesia supporting techniques –e.g., a version-based approach [5]– but are not presented here to due space constraints.

The information maintained in order to perform the amnesia recovery process will be the broadcast replication messages. In this replication protocol two messages for each propagated transaction: *remote* and *commit*. The amnesia recovery must be performed before starting the recovery of missed updates –the latter will be done by the basic recovery protocol–. The amnesia recovery process will consist in reapplying the messages belonging to non really committed transactions.

A transport-level solution consists in each node storing persistently the received messages, maintaining them as long as the associated transaction, t, has not been committed and discarding them as soon as t its really committed in the replica. But, the message persist process must be performed atomically inside the delivery process as already discussed in [10] with its "successful delivery" concept. Moreover, messages belonging to aborted or rolled-back transactions must be also deleted.

Once the amnesia phenomenon is solved at transport level, it is necessary to manage the amnesia problem at replica level. At this level the amnesia implies that the system

can not remember which were the really committed transactions. Even for those transactions for which the "commit" message was applied, it is possible for the system to fail *during* the commit. Then the amnesia recovery process in a replica will consist in reapplying (and immediatly deleting, in the same transactional context) the received and persistently stored messages in this replica that have not been already deleted, because it implies that the corresponding transactions have not been committed in the replica. These messages are applied in the same order as they were originally received.

It also must be noticed, that in this process is not needed to apply the *remote* messages whose associated *commit* messages have not been received, because it implies that they have been committed in the subsequent view, and therefore their changes are applied during the recovery of its first missed view.

Finally, once the amnesia recovery process ends, the basic recovery protocol mechanism can start.

5 Compacting Recovery Information

In order to increase the performance at the moment of determining and transferring the necessary information for the synchronization of recovering nodes, we propose some modifications based on packing information that enhance the basic recovery protocol described in [6]. This could be done by compacting the records in the $MISSED$ table, and with this, minimize the items to transmit and to apply them in the recovering node, reducing thus the transmission and synchronization time.

These item identifiers can be packed due to the fact that the recovery information only maintains the identifiers of updated items. The state of these items is retrieved by the *recoverer* from the database at recovering time. Moreover, if a *recovering* node, k, has to recover the state of an item modified in different views lost by k it will receive as many times the item value, but transferring its state only once is enough. As a consequence, it is not relevant to repeat the identifier of an updated item across several views, being only necessary to maintain it in the last view it was modified.

We consider that the actions for the amnesia support are performed during the execution of user transactions. Whenever one (or more than one) node fails, the recovery protocol starts the execution of the actions to advance the recovery of failed nodes. To this end, when a transaction commits, the field which contains the identifiers of the updated items, OID_LIST, will be updated in the following way:

1. For each item in the $WriteSet$, the OID_LIST is scanned to verify if the item is already included in it or not. If it is not, it is included and is looked for in previous views OID_LIST, eliminating it from the OID_LIST in which it appears, compacting thus the OID_LIST, i.e. the information to transfer when a node recovers.
2. If as a result of this elimination, an OID_LIST is emptied, the content of the field $SITES$ is included into the field $SITES$ of the next record, and the empty record in the table $MISSED$ can be eliminated.

When a node reconnects to a replicated system, the new view is installed and the actions for the amnesia recovery are performed locally at the recovering node. This is

a lightweight process (i.e. only a few stored messages have to be processed) in comparison to the database state recovery process itself. The other nodes know who is the recovering node, and every one performs locally the next actions:

1. The $MISSED$ table is scanned looking for the recovering node in the field $SITES$ until the view that contains the recovering node is found. The items for which the recovering node needs to update its state are the elements of OID_LIST of this view and the subsequent views.
2. At the recoverer node, the recovery information is sent to the recovering node according to the basic protocol.
3. Once the recovering node has confirmed the update of a view, the node is eliminated from the $SITES$ field in this view, and if it is the last item, also the record that contains this view is eliminated.
4. If a recoverer node fails during the recovering process, then another node is elected to be the new recoverer, according to the basic protocol. And it will create the partitions pending to be transferred, according to the previous points, and then it will perform the item transfer to recovering nodes, again as in the basic protocol.

It is important to note that in a view change consisting in the join and leave of several nodes, we must first update the information about failed nodes, and later execute the recovery process.

6 Simulation Results

We have simulated the compacting enhancement in order to know which level of improvement provides. We have considered three replicated scenarios with 5, 9 and 25 nodes each one. The replicated database has 100000 data items. All simulations start having all replicas updated and alive. Then, we start to crash nodes one by one –installing a new view each time a node crashes–, until the system reaches the minimum primary partition in each scenario. At this point two different recovery sequences are simulated. In the first one, denoted as order 1, the crashed nodes are reconnected one by one in the same order as they crashed, while in the second, denoted as order 2, they are reconnected one by one but reversing their crash order. In both cases, each time a node reconnects a new view is installed, and immediately the system starts its recovery, ending its recovery process before reconnecting the following one. In any installed view we assume that the replicated system performs 250 transactions successfully, and each transaction modifies 20 database items. All simulation parameters are described in Table 1.

The items in the writeset are obtained randomly with a uniform distribution. We have not used neither a hot spot, as in other previous works [12], nor typical workloads as TPC-W or TPC-C [13]. In both cases, they would be more favorable environments for the compacting method than a uniform distribution, since they suppose more frequent access to a set of items of the database, removing a big amount of items in the compacting proccess. We have also assumed a fast network, and this reduces the performance difference between the normal and compacting recoveries, since it only depends on the

Table 1. Simulator Parameters

Parameter	*Value*	*Parameter*	*Value*
Number of items in the database	100000	Time for a read	4 ms
Number of servers	5, 9, 25	Time for a write	6 ms
Transactions per view	250	Time for an identifier read	1 ms
Transaction length	20 modified items	Time for an identifier write	3 ms
Identifier size	4 bytes	CPU time for an I/O operation	0,4 ms
Item size	200 bytes	Time for point to point message	0,07 ms
Maximum message size	64 Kbytes	Time for broadcast message	0,21 ms
CPU time for network operation	0,07 ms		

Table 2. Recovery times in seconds (N = Nodes, V = Views)

Order	*N*	*V*	*Basic*		*Compacted*		*Order*	*N*	*V*	*Basic*		*Compacted*	
			Avg	*StdDev*	*Avg*	*StdDev*				*Avg*	*StdDev*	*Avg*	*StdDev*
1	5	2	165.8	0.23	161.7	0.21	2	25	5	414.6	0.18	376.1	0.15
2	5	1	82.8	0.20	82.9	0.18	2	25	7	580.4	0.19	502.1	0.14
2	5	3	248.7	0.18	236.7	0.16	2	25	9	746.2	0.19	616.0	0.12
1	9	4	331.6	0.19	308.1	0.18	2	25	11	912.0	0.19	719.2	0.11
2	9	1	82.9	0.17	82.9	0.20	2	25	13	1077.9	0.19	812.6	0.11
2	9	3	248.7	0.17	236.7	0.18	2	25	15	1243.8	0.19	897.1	0.10
2	9	5	414.5	0.18	376.0	0.17	2	25	17	1409.6	0.19	973.6	0.10
2	9	7	580.4	0.18	501.9	0.17	2	25	19	1575.5	0.19	1042.9	0.09
1	25	12	995.1	0.18	767.2	0.12	2	25	21	1741.3	0.19	1105.4	0.08
2	25	1	82.9	0.20	82.9	0.19	2	25	23	1907.2	0.18	1162.0	0.07
2	25	3	248.7	0.19	236.7	0.16							

amount of transferred items. If we had a slow network, such difference would have been bigger. We have made one hundred repetitions for every experiment obtaining with this, the guarantees of a low dispersion (see Table 2).

This simulation has not considered the costs of: managing the amnesia problem, and recovery information compacting. The amnesia problem, as it has been said before, is solved using a log-based approach, persisting the delivered messages during the replication work, and applying those not committed during the amnesia recovery process. Thus, it implies two costs: one in the replication work and another in the recovery work. The first cost is not considered because does not happen in the recovery process. The second one, although appears in the recovery process, is not considered because it is very low compared to the recovery process itself –usually it will consist in applying few messages (writesets) and in our simulation are very small–. The recovery information compacting cost is not taken into account because this work is performed online, therefore its associated overhead penalizes only the replication work performance, but not the recovery.

The simulation results show that the more views a crashed node loses the better the compacting technique behaves, which is a logical result. In fact, when more updates a crashed node misses the probability of modifying the same item increases. Both in the Table 2 and in the Figure 1 we can observe the same behavior. When a crashed node has lost only one view the compacting technique does not provide any improvement because it has been unable to work. But, as long as the crashed node misses more views the compacting technique provides better results.

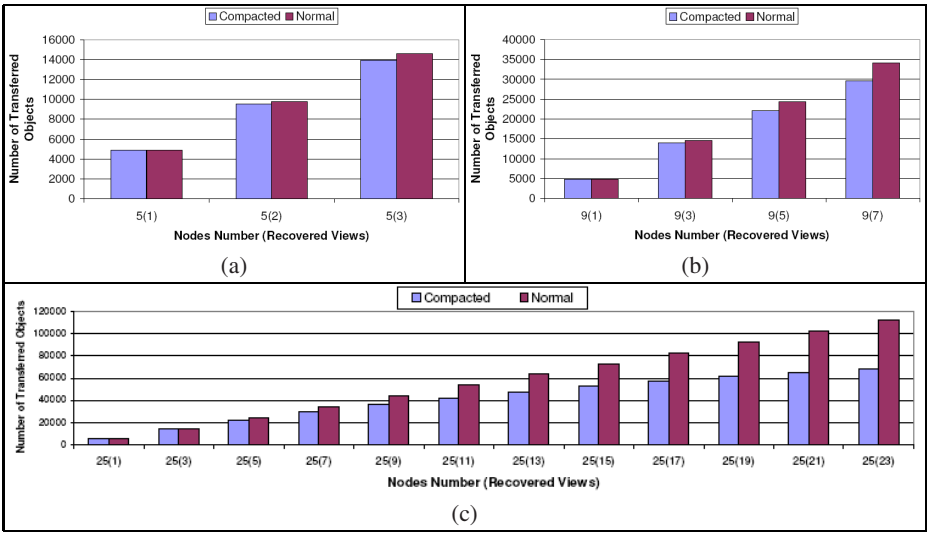

Fig. 1. Item Compactness: (a) 5 nodes, (b) 9 nodes, (c) 25 nodes

It must be also noticed that the basic recovery protocol could arrive to transfer a greater number of items than items has the original database. This occurs because it transfers for each lost view all the modified (and created items in this view) independently they are transferred when recovering other views where these items have been also modified. This situation is avoided by our recovery protocol enhancement. And in the worst case the proposed solution will transfer the whole database because during the inactivity period of the recovered node all the items of the database have been modified.

Obviously, we must say that the improvement provided by our approach depends on the replicated system load activity, the update work rate, and the changed items rate. For the first two ones, we can consider in a general way that when higher they are better our compacting technique behaves. This is because the probabilities of modifying the same item in different views increase. This consideration drives us to the changed items rate, which is really the most important parameter. It tells us if the performed updates are focused in few items or not. Then for our technique it is interesting that changes are focused in as few items as possible. In fact, the worst scenario for our technique will be the one in which all the modifications are performed in different items.

As final conclusion, we can say that our enhanced recovery protocol works better in some of the worst scenarios from a recovery point of view: when the crashed node has lost a lot of updates and the changed items rate is not very high.

7 Related Work

For solving the recovery problem [14] database replication literature has largely recommended the crash recovery failure model use as it is proposed in [3,4,5,6], while process replication has traditionally adopted the fail stop failure model. The use of different approaches for these two areas is due to the fact that usually the first one manages large data amounts, and it adopts the crash recovery with partial amnesia failure model in order to minimize the recovery information to transfer.

The crash-recovery with partial amnesia failure model adoption implies that the associated recovery protocols have to solve the amnesia problem. This problem has been considered in different papers as [10,11,15] and different recovery protocols have presented ways for dealing with it. The *CLOB* recovery protocol presented in [3] and the *Checking Version Numbers* proposed in [5] support amnesia managing it in a log-based and version-based way, respectively.

In regard to the compactness technique, [16] uses it in order to optimize the database recovery. In this case, this technique is used to minimize the information size that must be maintained and subsequently transferred in order to perform the recovery processes. Such paper also presents experimental results about the benefits introduced by using this technique, reaching up to 32% time cost reductions.

The background idea of our compacting technique is very similar to the one used in one of the recovery protocols presented in [5]. This protocol maintained in a database table the identifiers of the modified objects when there were failed nodes. Each one of these object identifiers was inserted in a different row, storing at the same time the identifier of the transaction which modified the object. Therefore, when an object was modified the system checked if its identifier was already inserted in this table. If it has not, the protocol created a new entry where inserted the identifier object and the transaction identifier. If it already existed an entry with this object identifier, the protocol simply updated in this entry the transaction identifier. So, this recovery protocol also avoids redundant information, but it uses a more refined metadata granularity –transaction identifier– than our enhanced protocol –view identifier–.

8 Conclusions

In this paper we have reviewed the functionality of the original recovery protocol described in [6]. We have enhanced it providing an accurated amnesia support and incorporating a compacting method for improving its performance.

The amnesia support has been improved using a log-based technique which consists in persisting the messages as soon as they are delivered in each node, in fact they must be persisted atomically in the delivery process.

Our compacting technique avoids that any data object identifier appears more than once in the *MISSED* table. Then, this mechanism reduces the size of recovery messages, both the ones that set up the DB-partitions and the ones which transfer the missed values.

Tests have been made with a simulation model and the advantages of the enhanced recovery protocol have been verified when comparing the results of both protocols. The obtained results have pointed out how our proposed compacting technique provides better results when the number of lost views by a crashed node increases. Thus, our compacting technique has improved the recovery protocol performance for recoveries of long-term failure periods.

References

1. Gray, J., Helland, P., O'Neil, P., Shasha, D.: The dangers of replication and a solution. In: ACM SIGMOD International Conference on Management of Data, pp. 173–182. ACM Press, New York (1996)
2. Wiesmann, M., Schiper, A., Pedone, F., Kemme, B., Alonso, G.: Database replication techniques: A three parameter classification. In: SRDS, pp. 206–215 (2000)
3. Castro, F., Esparza, J., Ruiz, M., Irún, L., Decker, H., Muñoz, F.: CLOB: Communication support for efficient replicated database recovery. In: 13th Euromicro PDP, Lugano, Sw, pp. 314–321. IEEE Computer Society Press, Los Alamitos (2005)
4. Jiménez-Peris, R., Patiño-Martínez, M., Alonso, G.: Non-intrusive, parallel recovery of replicated data. In: SRDS, pp. 150–159. IEEE Computer Society Press, Los Alamitos (2002)
5. Kemme, B., Bartoli, A., Babaoğlu, O.: Online reconfiguration in replicated databases based on group communication. In: Intl.Conf.on Dependable Systems and Networks, Washington, DC, USA, pp. 117–130 (2001)
6. Armendáriz, J.E., Muñoz, F.D., Decker, H., Juárez, J.R., de Mendívil, J.R.G.: A protocol for reconciling recovery and high-availability in replicated databases. In: Levi, A., Savaş, E., Yenigün, H., Balcısoy, S., Saygın, Y. (eds.) ISCIS 2006. LNCS, vol. 4263, pp. 634–644. Springer, Heidelberg (2006)
7. Chockler, G.V., Keidar, I., Vitenberg, R.: Group communication specifications: A comprehensive study. ACM Computing Surveys 4(33), 1–43 (2001)
8. Cristian, F.: Understanding fault-tolerant distributed systems. Communications of the ACM 34(2), 56–78 (1991)
9. Wiesmann, M., Schiper, A.: Comparison of database replication techniques based on total order broadcast. IEEE Trans. Knowl. Data Eng. 17(4), 551–566 (2005)
10. Wiesmann, M., Schiper, A.: Beyond 1-Safety and 2-Safety for replicated databases: Group-Safety. In: Proceedings of the 9th International Conference on Extending Database Technology (EDBT2004), Heraklion - Crete - Greece (2004)
11. de Juan-Marín, R., Irún-Briz, L., Muñoz-Escoí, F.D.: Supporting amnesia in log-based recovery protocols. In: ACM Euro-American Conference on Telematics and Information Systems, Faro, Portugal, ACM Press, New York (May 2007)
12. Kemme, B.: Database Replication for Clusters of Workstations. PhD thesis, Swiss Federal Inst. of Technology, Zurich, Switzerland (2000)
13. The transaction processing performance council, `http://www.tpc.org`
14. Bernstein, P.A., Hadzilacos, V., Goodman, N.: Concurrency Control and Recovery in Database Systems. Addison Wesley, Reading, MA, EE.UU (1987)
15. de Juan-Marín, R., Irún-Briz, L., Muñoz-Escoí, F.D.: Recovery strategies for linear replication. In: ISPA, pp. 710–723 (2006)
16. Civera, J.P., Ruiz-Fuertes, M.I.: García-Muñoz, L.H., Muñoz-Escoí, F.D.: Optimizing certification-based database recovery. Technical report, ITI-ITE-07/04, Instituto Tecnológico de Informática (2007)