Amnesia Issue in Majority Progress Condition

Rubén de Juan-Marín, Luis Irún-Briz and Francesc D. Muñoz-Escoí

Instituto Tecnológico de Informática, Universidad Politécnica de Valencia
Camino de Vera s/n, 46022 Valencia, Spain

{rjuan, lirun, fmunyoz}@iti.upv.es

Technical Report TR-ITI-ITE-07/12

Amnesia Issue in Majority Progress Condition

Rubén de Juan-Marín, Luis Irún-Briz and Francesc D. Muñoz-Escoí

Instituto Tecnológico de Informática, Universidad Politécnica de Valencia
Camino de Vera s/n, 46022 Valencia, Spain

Technical Report TR-ITI-ITE-07/12

e-mail: {rjuan, lirun, fmunyoz}@iti.upv.es

Abstract

Replication is used for providing highly available and fault-tolerant information systems, which are constructed on top of replication and recovery protocols. Important aspects when designing these systems are the failure model assumed and the progress condition adopted. Replicated transactional systems usually assume the crash-recovery with partial amnesia failure model, and the majority partition progress condition. But, despite the large use of such combination most of these works do not handle accurately a very special phenomenon that can lead to diverging states in different replicas causing, when happening, critical situations.

1 Introduction

Transactional replication has become a key factor in providing fault-tolerant, highly available information systems, providing at the same time good performance levels. Performance can be improved forwarding client requests to their closest replica [16, 17], or by using load-balancing algorithms [1, 13, 18]. And fault tolerance and high availability are reached forwarding such requests to non-failed nodes in a transparent way. In last years these techniques have been making use of Group Communication System (GCS for short) [7] as it is detailed in [20], because they provide communication primitives and membership mechanisms, which are very important in replicated systems.

Important aspects when designing transactional replicated systems are how they manage membership changes –which alter their performance, fault tolerance and high availability support– and the adopted progress condition that must fulfil in order to go on working.

For managing the membership events they make use of *recovery components* which deal with these situations attending the failure model adopted. The most commonly used failure models in transactional systems are *fail-stop* and *crash-recovery with partial amnesia*, as defined in [8], being the last one the most widely used in latest proposals for shortening the recovery times of updated nodes. But, its use arises the amnesia phenomenon [11] which must be correctly managed for solving different state evolutions in different nodes. In relation to the progress condition, most of these works have adopted the majority of alive replicas progress condition –primary partition [7]–.

In this paper we first outline with a simple example (look at Section 5.1) a replicated consistency problem which arises when combining the *amnesia phenomenon* –non-correctly handled– with a specific replicated system composition allowed by the *majority partition* progress condition. A problem that can lead different state evolutions among the members of the replicated system.

Later, we formalize this problem, establishing the replicated system conditions that would generate it, and the properties that must be fulfilled for overcoming it.

Despite being a rare problem, it should be accurately managed for avoiding critical situations. Therefore, we present two different approaches for overcoming it, being each one of them interesting for transactional replicated systems with different characteristics. On one hand, we present a solution for critical systems where already performed and committed work at replicated system level can not be undone nor lost. On the other hand, we propose the use of a technique used in partitionable systems, reconciliation, whose main advantage is its zero overhead in normal work.

The rest of the paper is structured as follows. In Section 2 we detail the system model. Section 3 presents the amnesia phenomenon, how it is manifested and how it can become a problem. Later, we perform a short formalization about progress conditions in Section 4. In Section 5 we outline and formalize the amnesia issue with the progress condition, presenting some solutions in Section 6. Finally, related work is included in Section 7, and Section 8 concludes the paper.

2 System Model

Our model considers a replicated transactional system, which is compound by several replicas, where each replica is located 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 state is fully replicated in each node, so each replica has a copy of the whole state. State changes are performed between the boundaries of transactions.

The replicated system uses a *GCS*, supporting point-to-point and broadcast deliveries. A FIFO and reliable communication is assumed. Transaction updates are propagated to all replicas using atomic broadcast: i.e. total order delivery.

The GCS includes a group membership service, who *knows* in advance the identity of all potential system nodes. These nodes can join the group and leave it either explicitly or implicitly by crashing. The *GCS* provides *Virtual Synchrony*[4] guarantees, thus each time a membership change happens, it supplies consistent information about the current set of reachable members. This information is given in the format of *views*. Sites are notified about a new view installation with *view change events*.

The view notification mechanism is extended with node application state information providing the *enriched view synchrony* [3] approach. This makes simpler and easier the support of system cascading reconfigurations. These enriched views (*e-view*) not only inform about active nodes, but they also inform about the state of active nodes: outdated or up-to-date. The use of e-views refines the *primary partition* model into the *primary subview* model, therefore the system only can work when a *progress condition* is fulfilled[1] as detailed in [9]. At the same time the state consistency is ensured because only the primary *subview* is able to work in partition scenarios. For similar reasons, only fully updated replicas can serve client requests.

3 The Amnesia Phenomenon

Replicated transactional systems have usually adopted the fail-stop failure model [8]. The reasons for adopting this failure model are: (a), it is the failure model mainly used in distributed systems; (b), its simplicity. In fact, when a replica crashes it is not recovered but substituted by a new one –transferring to it the whole state–. Therefore, the system must not generate and maintain special information for recovery purposes.

However, assuming this failure model implies some drawbacks when the recovery information to be transferred is large –a common situation in replicated databases. Hence, the larger the information to be transferred the longer it takes to make a replica become active. This will imply, in the replicated system, the following consequences: (a) longer periods with decreased fault tolerance support, since only fully updated replicas can be used to guarantee the correct and consistent state evolution in the replicated system, (b) higher times of unavailability if the replicated system does not fulfill the progress condition (i.e. systems based on primary partitions).

In order to avoid all the above presented issues in replicated systems, researchers have opted for assuming the crash-recovery with partial amnesia failure model [8]. In this case, when a crashed replica

[1]This characteristic prevents the system from working in the starting phase until a primary subview is reached. Therefore, during this initial phase, the recovery protocol must not perform any work.

reconnects the system recovers it transferring only the state it has missed; thus, transferring less information and minimizing the previous issues.

With the assumption of this failure model, the system is forced to determine correctly the subset of information that must be transferred to the recovering node. If it is not correctly determined, the state reached in the recovered node can diverge from the real consistent replicated state, leading to an undesired situation. In [10, 11] we have already described this situation naming it as amnesia phenomenon, which manifests at two different levels:

- *Transport level.* At this level, it implies that the replica does not remember *which messages have been received.* Actually, 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, because message delivery does not really imply correctly processed as demonstrated in [21].

- *Replication level.* The amnesia is manifested here in the fact that the node "forgets" *which were the really committed transactions*. Usually, the internal log used by the underlying databases can be used for solving this.

3.1 A Generic Solution

We have also proposed a generic solution for overcoming this in [10, 11]. The proposed solution consists in forcing each replica to enqueue persistently the broadcast messages as soon as they are delivered, removing them from this queue as soon as they are correctly processed. Then, when a crashed node reconnects, before asking about its missed changes to an updated replica it will check its queue of received and not applied messages (i.e. a log-based solution [5]). Obviously, this is not the unique solution that can be adopted for solving this problem, being possible also to apply version-based techniques [6].

3.2 Amnesia Formalization

As previous step for describing the progress condition issue when the amnesia phenomenon manifests we need to formalize the amnesia problem. To do so, we consider a replicated transactional system, $N = \{n_1, n_2, ..., n_n\}$, compound by n replicas, being $n > 2$ (primary partition assumption [7]). It uses an eager update everywhere protocol based on a GCS which provides an atomic broadcast primitive for spreading messages and virtual synchrony. It also uses constant interaction, broadcasting each transaction updates in a single message.

In this system, we identify each installed view –working view– as $\mathcal{V}_x$, being x the view identifier. $T_x = \{T_{x,1}, T_{x,2}, ..., T_{x,m}\}$ are the transactions delivered (and not aborted in this view –aborted transactions are not considered because they are not relevant for recovering purposes–). As the system uses the atomic broadcast primitive [15] for spreading transactions, all alive nodes deliver the broadcast transactions in the same order, using this order at execution time. This order is being reflected by the second subindex.

$\forall\, n_y \in \mathcal{V}_x$ we denote as T^D_{x,n_y} the transactions subset of T_x really delivered to n_y and, respectively, T^C_{x,n_y} the transactions subset of T_x really committed in n_y; fulfilling $T^C_{x,n_y} \subseteq T^D_{x,n_y}$. Virtual synchrony [7] ensures that $T^D_{x,n_y} = T_x$. View transitions are represented as $\mathcal{V}_x \rightarrow \mathcal{V}_{x+1}$.

Then $\forall\, \mathcal{V}_i \rightarrow \mathcal{V}_{i+1}$ triggered by a node crash, it will be at least one node $n_l : n_l \in \mathcal{V}_i \setminus \mathcal{V}_{i+1}$. Considering that $T_i = \{T_{i,1}, T_{i,2}, ..., T_{i,m}\}$ is the transactions set delivered and committed in the replicated system during $\mathcal{V}_i$, it can be assumed that $\forall\, n_k \in \mathcal{V}_i \cap \mathcal{V}_{i+1}$:

$$T_i = T^D_{i,n_k} = T^C_{i,n_k} = \{T_{i,1}, T_{i,2}, ..., T_{i,m}\}$$

While $\forall\, n_l \in \mathcal{V}_i \setminus \mathcal{V}_{i+1}$, without a *successful delivery* primitive [21] it might happen the following $T_i = T^D_{i,n_l} \neq T^C_{i,n_l}$, where:

$T^C_{i,n_l} = \{T_{i,1}, T_{i,2}, ..., T_{i,m-s}\}$, being $0 \leq s \leq m$

In spite of assuming that $s \in \{0, .., m\}$ for simplicity reasons in this paper, it is also possible sometimes that $s > m$ due to workload reasons.

When n_l reconnects to the system, it triggers a new view $\mathcal{V}_{i+x}$, being $x > 1$. Later, the system must update n_l through the recovery process, transferring to it its lost transactions, which are:

- Transactions *forgotten* from its last seen view, $\mathcal{V}_i$: $T^F_{i,n_l} = T_{i,m-s+1}, ..., T_{i,m}$
- Transactions *missed* during its disconnection: $T^M_{n_l} = T_{i+1} \cup ... \cup T_{i+x-1}$

Then, for solving the amnesia phenomenon –*forgotten state*– when recovering n_l the two following properties must be provided:

- *Property 1*: n_l must remember its last committed transaction, $T_{i,m-s}$;
- *Property 2*: the replicated system must maintain and provide a way for obtaining the transactions subset T^F_{i,n_l} or their associated updates.

Once this *forgotten state* has been updated in the recovering replica, the recovery protocol can start with the recovery process itself, transferring *missed* data: $T^M_{n_l}$.

Notice that our generic solution, outlined in Section 3.1 and presented in [10, 11], fulfills both properties. This is due to the fact that the persisted queue contains the messages associated to T^F_{i,n_l}.

4 Progress Condition

Progress condition is the condition that must be fulfilled by a replicated system to be enabled to work. Usually, replicated systems have adopted the primary partition condition [7]. So, in this case the replicated system is allowed to work if a majority of its replicas is alive. In [9] it has been demonstrated how this progress condition can refer either to a majority of updated nodes –more restrictive– or a majority of alive nodes detailing the differences between them.

4.1 Progress Condition Formalization

Considering a replicated transactional system, $N = \{n_1, n_2, ..., n_n\}$, compound by n replicas –with $n > 2$–, we represent with N_x that it has a working view, $\mathcal{V}_x$, while with N^*_x that it has not any working view, being $\mathcal{V}_x$ the last working view installed in the system. Minority partitions are represented by $\mathcal{V}^*_y$, where y is the last working view seen by the members of this partition.

Thus, we can say that it is in a working view, N_x, if it has a $\mathcal{V}_x : card(\mathcal{V}_x) >= \lfloor \frac{n}{2} \rfloor + 1$. Contrarily, we say that it is in a non-working view N^*_x. Minority partitions, $\mathcal{V}^*_x$, always fulfil that $card(\mathcal{V}^*_x) < \lfloor \frac{n}{2} \rfloor + 1$.

For formalization reasons, we use two different view counters: one for total installed views –first subindex–, and another one for working installed views –second subindex. The first subindex is used for noticing that membership changes also occur in non-majority partitions, installing "views", although usually authors only use the view concept for partitions which fulfil the progress condition. So, this first counter is increased in any members group view change –but it has not any purpose in a real system–, while the second one is only increased when a new working view is installed –being the counter that must be used in a real system–. Possible view transitions are shown in table 1.

5 Amnesia Consistency Problem in Progress Condition

As we have said combining the amnesia problem –which appears when the replicated system adopts the crash-recovery with partial amnesia failure model– with the replicated system progress condition –primary partition– can lead the replicated system to state inconsistencies. The problem is that the system is unable to guarantee the correct system data state progress. This inconsistency problem can be seen with the following example.

TRANSITION CASES
Node Addition
T1: $\mathcal{V}_{x,j} \cup \mathcal{V}^*_{k,l} \rightarrow \mathcal{V}_{x+1,j+1}$
T2: $\mathcal{V}^*_{x,j} \cup \mathcal{V}^*_{k,l} \rightarrow \mathcal{V}^*_{max(x,k)+1,max(j,l)}$
T3: $\mathcal{V}^*_{x,j} \cup \mathcal{V}^*_{k,l} \rightarrow \mathcal{V}_{max(x,k)+1,max(j,l)+1}$
Node Removal
T4: $\mathcal{V}_{x,j} \rightarrow \mathcal{V}_{x+1,j+1}$
T5: $\mathcal{V}_{x,j} \rightarrow \mathcal{V}^*_{x+1,j}$
T6: $\mathcal{V}^*_{x,j} \rightarrow \mathcal{V}^*_{x+1,j}$

Table 1: View Transitions.

5.1 A Problem Sample

Consider that the information system of a hospital is compound by three replicas, $\alpha = \{R_1, R_2, R_3\}$, and all the hospital terminals work against it. All three replicas are fully updated –with the same state– and working at the instant t_0. Then, a doctor introduces a first patient diagnosis in the system through T_1 –including the necessary analysis that need to be performed for refining it–, being delivered and committed in all replicas. After performing and studying these analysis, the doctor introduces in the system that the patient has forbidden to eat some particular food –becuase its ingestion can derive in severe health patient consequences– through T_2. T_2 is delivered to all replicas, but only committed in R_1. This is due because R_2 and R_3 nodes crash before being able to commit T_2, moreover, R_2 and R_3 lose the T_2 associated message because the replication protocol does not persist it. R_2 and R_3 crash implies that the hospital information system does not fulfil the primary partition progress condition, so it stops working. Once the hospital IT staff has repaired R_2 and R_3, these replicas are reconnected to the system, but in this view change it also crashes R_1. Then the information system fulfils the progress condition, but it arises a consistency problem, R_2 and R_3 have not seen the T_2 changes. So, as they fulfil the progress condition they can go on working, but if they work they will start from the state reached after committing T_1 and not T_2 –the last really committed transaction in the replicated transactional system– leading to a diverging state evolution to R_1 state –which is the correct one.

It must be said that this situation or another combination of events that leads to a similar situation is very improbable in a replicated system. And this probability diminishes as long as the number of replicas increases. But, it must be correctly managed in order to avoid undesired situations in the replicated consistent state. In the previous example, the inconsistency can imply that the patient eats something that it has forbidden, causing severe damages in his health.

As previous step to presenting possible solutions that can be applied for solving this problem, we will formalize it.

5.2 Problem Formalization

Assume a replicated transactional system, $N = \{n_1, n_2, ..., n_n\}$, compound by n replicas, being $n > 2$.

$\forall\, T5$ transitions triggered by node crash/es it will be at least one $n_l : n_l \in \mathcal{V}_{x,j} \setminus \mathcal{V}^*_{x+1,j}$.

Considering that $T_j = \{T_{j,1}, T_{j,2}, ..., T_{j,m}\}$ is the transactions set delivered and committed in the replicated system during $\mathcal{V}_{x,j}$, it can be assumed that $\forall\, n_k \in \mathcal{V}_{x,j} \cap \mathcal{V}^*_{x+1,j}$:

$$T_x = T^D_{j,n_k} = T^C_{j,n_k} = \{T_{j,1}, T_{j,2}, ..., T_{j,m}\}$$

While $\forall\, n_l \in \mathcal{V}_{x,j} \setminus \mathcal{V}^*_{x+1,j}$, as it has been formalized in subsection 3.2, it might happen the following: $T_j = T^D_{j,n_l} \neq T^C_{j,n_l}$, where:

$$T^C_{j,n_l} = \{T_{j,1}, T_{j,2}, ..., T_{j,m-s}\}, \text{ being } 0 \leq s \leq m.$$

Due to the *amnesia phenomenon* we will distinguish minority partitions, $\mathcal{V}^*_{z,j}$ –which is used in a generic way–, between $\check{\mathcal{V}}^*_{z,j}$ and $\hat{\mathcal{V}}^*_{z,j}$. First ones, are minority partitions whose last seen view is j, but they can not ensure that they do not have the amnesia phenomenon in relation to this view because all their n_l nodes that have seen the j view fulfil that $n_l : n_l \in \mathcal{V}_{x,j} \setminus \mathcal{V}^*_{x+1,j}$. While second ones, $\hat{\mathcal{V}}^*_{z,j}$, are minority partitions that have seen also the j view, and at least one of their nodes that has seen j fulfills $n_m \in \mathcal{V}_{x,j} \cap \mathcal{V}^*_{x+1,j}$.

Later, if in the first transition of type $T3$ to a new working view, $\mathcal{V}_{k,j+1}$ –recall that the last installed working view in the system was $\mathcal{V}_{x,j}$–, the new installed view fulfils the following:

$\mathcal{V}_{k,j+1} = A \cup B$ where:

- $A = \{n_l \in \mathcal{V}_{x,j} \setminus \mathcal{V}^*_{x+1,j} : n_l \notin \hat{\mathcal{V}}^*_{z,j} : x+1 < z < k\}$, are the nodes that were alive in the last working view, but crashed –so they were not alive in any $\mathcal{V}^*_{x+1,j}$– triggering the view change that lead $N_j \rightarrow N^*_j$, and did not belong to any minority view that can recover the whole j view.
- $B = \{n_k \notin \mathcal{V}_{x,j} \cap \hat{\mathcal{V}}^*_{z,j} : x < z < k\}$, are the nodes that did not belong to the last working view, and that have not recovered the whole j view in any minority partition.

Then, the new reached majority is enabled to go on working. But, in this situation a problem can arise if the s term for A nodes fulfils that $s > 0$. This is because this new installed majority will be unable to reach the last consistent replicated state –the one reached after applying $T_{j,m}$–, due to the fact that:

- $\forall\, n_l \in A$ it is fulfilled that $T^F_{j,n_l} \neq \emptyset$
- $\forall\, n_s \in B$ it is either fulfilled that $T^D_{j,n_k} = T^C_{j,n_k} = \emptyset$ or $T^F_{j,n_l} \neq \emptyset$

So, the arising consistency problem conditions are:

- *Condition 1*: $T3$ transition $\rightarrow \mathcal{V}_{k,j+1}$
- *Condition 2*: $\mathcal{V}_{k,j+1} = A \cup B$
- *Condition 3*: $\forall\, n_l \in A$ it is fulfilled that $T^F_{j,n_l} \neq \emptyset$

The properties that must fulfil the replicated system to avoid this possible situation are similar to the ones proposed for solving the general amnesia phenomenon in subsection 3.2: in fact the *Prop. 1* is necessary as it is defined in 3.2, while the *Prop. 2* must be slightly modified to overcome this problem:

- *Property 2*: each node $n_l \in A$ must maintain and provide a way for obtaining its T^F_{j,n_l} transactions subset or associated updates, instead of trusting in "the replicated system".

Our generic solution, outlined in Section 3.1 and presented in [11, 10], fulfils also both properties, as explained in the next section.

6 Solutions

In this section we provide different approaches for solving this problem in replicated systems.

Persisting Messages. This solution is in fact our generic approach presented in Section 3.1. So, the idea consists in storing persistently the delivered messages in each replica as an atomic step of the delivery message, being only possible to delete them once they have been correctly processed in the replica.

Working in this way it is always ensured that $\forall\, T5$ transition triggered by node crashes –reaching $\mathcal{V}^*_{x+1,j}$– all $n_l \in A$ has persisted its T^F_{j,n_l}. Thus, when they reconnect and start their recovery process they can apply them. So, if in the first transition of type $T3$ to –reaching $\mathcal{V}_{k,j+1}$– it is fulfilled that $\mathcal{V}_{k,j+1} = A \cup B$, then the A nodes in spite of having the $T^F_{j,n_l} \neq \emptyset$, they have permanently stored the messages associated T^F_{j,n_l}. Hence, they are able to reach the last consistent state of the replicated system, avoiding diverging state evolutions, when the amnesia problem is combined with a $T3$ transition.

Obviously, persisting messages as soon as they are delivered implies an overhead during the replication work. A study of this overhead cost is presented in [11]. An overhead that will penalize constantly the replication work in order to avoid problems for situations that will rarely occur.

Mobile approach. So, another possible solution is to do nothing and assume these situations can happen. In this case, the idea is that among the alive nodes that compound the new primary partition –instead of not having the last consistent state– decide a new last consistent replicated state, allowing the system to go on working from this point, the $T_{j,m-s}$ with highest $m - s$ value of A nodes.

Later, when a replica which really reached the last consistent state of the replicated system reconnects, it must undo the transactions not processed in the new consistent replicated state before being recovered.

This solution avoids the overhead of persisting messages and simply implies to undo –in very rare occasions– some transactions –usually very few–. This solution is similar in concept to some approaches used in reconciling processes for partitionable systems [2].

Selecting Alternative. Which solution must be adopted? It depends. The first solution solves the problem ensuring that committed transactions are not lost, but implies a constant overhead during the normal work for solving a problem that will rarely happen. While the second solution avoids the problem without implying any overhead, but some transactions must be undone when this improbable scenario happens. So, it depends on the replicated system tolerance to undo some already committed transactions. If this tolerance is critical we have to select the first approach, while if there are not important problems of undoing some committed transactions, the second one can be adopted.

7 Related Work

As far as we know, there are not published works in replicated systems literature which study possible arising problems when combining the amnesia phenomenon –associated to the *crash recovery with partial amnesia* failure model– with the most extended progress condition in replicated systems.

In relation to the amnesia phenomenon different works have presented several results as [10, 11, 12, 14] where this phenomenon has been studied for specific recovery protocols [12, 14] or in a more generic way as in [10, 11].

It must be also noticed that in [21], authors analyzed the basic phenomenon which underlies behind the amnesia problem. They also proposed in such paper the concept of *successful delivery* that when correctly implemented, it overcomes both the amnesia generic problem and the amnesia issue with the progress condition presented in this paper.

In [19], authors realised that the adoption of the crash-recovery failure model in replicated systems, in spite of being more realistic implied some problems that they solved combining checkpointing and message logs at communications level.

Anyway, in [19, 21] authors do not formalize the amnesia phenomenon and do not study the associated problem when combined with the majority progress condition.

8 Conclusions

In this paper we have shown how combining the amnesia phenomenon, which arises when replicated systems assume the *crash recovery with partial amnesia* failure model, with a particular scenario allowed for the most commonly used progress condition –majority partition– in these replicated systems can lead to diverging replicated state evolutions. We have formalized it, and proposed the properties that must be ensured in order to overcome these undesired situations. Later, we have proposed two different approaches for solving this problem, being interesting each one for different replication scenarios.

This phenomenon in spite of being very rare can cause catastrophic consequences in consistency concerned replicated systems, so in these systems it must be accurately managed.

9 Acknowledgements

Work supported by FEDER and the Spanish MEC grant TIN2006-14738-C02.

References

[1] Cristiana Amza, Alan L. Cox, and Willy Zwaenepoel. A comparative evaluation of transparent scaling techniques for dynamic content servers. In *ICDE*, pages 230–241. IEEE Computer Society, 2005.

[2] Mikael Asplund, Simin Nadjm-Tehrani, Stefan Beyer, and Pablo Galdámez. Measuring Availability in Optimistic Partition-tolerant Systems with Data Constraints. In *International Conference on Dependable Systems and Networks (DSN)*, June 2007.

[3] O. Babaoğlu, A. Bartoli, and G. Dini. Enriched view synchrony: A programming paradigm for partitionable asynchronous distributed systems. *IEEE Trans. Comput.*, 46(6):642–658, 1997.

[4] Kenneth P. Birman and Robbert Van Renesse. *Reliable Distributed Computing with the ISIS Toolkit*. IEEE Computer Society Press, Los Alamitos, CA, USA, 1993.

[5] F. Castro, J. Esparza, M.I. Ruiz, L. Irún, H. Decker, and F.D. Muñoz. CLOB: Communication support for efficient replicated database recovery. In *13th Euromicro PDP*, pages 314–321, Lugano, Sw, 2005. IEEE Computer Society.

[6] F. Castro, L. Irún, F. García, and F. D. Muñoz. Fobr: A version-based recovery protocol for replicated databases. In *PDP*, pages 306–313, 2005.

[7] G. V. Chockler, I. Keidar, and R. Vitenberg. Group communication specifications: A comprehensive study. *ACM Computing Surveys*, 4(33):1–43, 2001.

[8] Flaviu Cristian. Understanding fault-tolerant distributed systems. *Communications of the ACM*, 34(2):56–78, 1991.

[9] Rubén de Juan-Marín. (n/2+1) alive nodes progress condition. In *Sixth European Dependable Computing Conference, EDCC-6*, 2006.

[10] Rubén de Juan-Marín, Luis Irún-Briz, and Francesc D. Muñoz-Escoí. Recovery strategies for linear replication. In *ISPA*, pages 710–723, 2006.

[11] Rubén de Juan-Marín, Luis Irún-Briz, and Francesc D. Muñoz-Escoí. Supporting amnesia in log-based recovery protocols. In *Euro American Conference on Telematics and Information Systems, EATIS, Faro, Portugal*, 2007.

[12] Rubén de Juan-Marín, María Idoia Ruiz-Fuertes, Jerónimo Pla-Civera, Luis Héctor García-Muñoz, and Francesc D. Muñoz-Escoí. On Optimizing Certification-Based Database Recovery Supporting Amnesia. In *XV Jornadas de Concurrencia y Sistemas Distribuidos (JCSD 07), Torremolinos, Spain*, June 2007.

[13] Sameh Elnikety, Steven Dropsho, and Willy Zwaenepoel. Tashkent+: Memory-aware load balancing and update filtering in replicated databases. In *Proc. EuroSys 2007*, pages 399–412, March 2007.

[14] Luis H. García-Muñoz, Rubén de Juan-Marín, J. E. Armendáriz, and Francesc D. Muñoz-Escoí. Adding amnesia support and compacting mechanisms to replicated database recovery. Technical report, ITI-ITE-07/08, Instituto Tecnológico de Informática, 2007.

[15] V. Hadzilacos and S. Toueg. Fault-tolerant broadcasts and related problems. In S. Mullender, editor, *Distributed Systems*, chapter 5, pages 97–145. ACM Press, 1993.

[16] Yi Lin, Bettina Kemme, Marta Patiño-Martínez, and Ricardo Jiménez-Peris. Middleware based data replication providing snapshot isolation. In Fatma Ozcan, editor, *SIGMOD Conf.*, pages 419–430. ACM, 2005.

[17] Marta Patiño-Martínez, Ricardo Jiménez-Peris, Bettina Kemme, and Gustavo Alonso. Middle-r: Consistent database replication at the middleware level. *ACM Trans. Comput. Syst.*, 23(4):375–423, 2005.

[18] Christian Plattner and Gustavo Alonso. Ganymed: Scalable replication for transactional web applications. In Hans-Arno Jacobsen, editor, *Middleware*, volume 3231 of *Lecture Notes in Computer Science*, pages 155–174. Springer, 2004.

[19] Luís Rodrigues and Michel Raynal. Atomic broadcast in asynchronous crash-recovery distributed systems and its use in quorum-based replication. *IEEE Trans. Knowl. Data Eng.*, 15(5):1206–1217, 2003.

[20] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, and G. Alonso. Understanding replication in databases and distributed systems. In *ICDCS*, pages 464–474, Washington, DC, USA, 2000. IEEE Computer Society.

[21] M. Wiesmann and A. Schiper. Beyond 1-Safety and 2-Safety for replicated databases: Group-Safety. In *9th International Conference on Extending Database Technology*, pages 165–182, 2004.