Reviewing Amnesia Support in Database Recovery Protocols

R. de Juan-Marín, L. H. García-Muñoz, J. E. Armendáriz-Íñigo and F. D. Muñoz-Escoí

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

{rjuan, lgarcia, armendariz, fmunyoz}@iti.upv.es

Technical Report TR-ITI-ITE-07/11

Reviewing Amnesia Support in Database Recovery Protocols

R. de Juan-Marín, L. H. García-Muñoz, J. E. Armendáriz-Íñigo and F. D. Muñoz-Escoí

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

Technical Report TR-ITI-ITE-07/11

e-mail: {rjuan, lgarcia, armendariz, 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. An important aspect when designing these systems is the assumed failure model. Replicated databases literature last trends consist in adopting the crash-recovery with partial amnesia failure model because in most cases it shortens the recovery times. But, despite the large use of such failure model we consider that most of these works do not handle accurately the amnesia phenomenon. Therefore, in this paper we survey some works, analyzing their amnesia support. In this study, we focus on primary component membership systems. The same principles could be applied for partitionable or mobile systems, but we have not surveyed them.

1 Introduction

Database replication has become a key factor in providing fault tolerance, high availability and for increasing the performance of information systems. On one hand, performance can be improved when clients access the closest replica to them [24, 25, 23], or by using load-balancing algorithms [26, 13, 2]. On the other hand, replicas may fail or may disconnect; therefore, fault tolerance and high availability are reached forwarding client requests to non-failed nodes in a transparent way.

Latest trends in full database replication techniques –managed by replication protocols [24, 25, 23, 26, 13, 2]– make use of a Group Communication System (GCS for short) [9] as it is detailed in [28]. These GCSs offer different services to the systems built atop of them. They provide several communication primitives, such as the atomic broadcast [18] allowing a more efficient implementation of replication protocols. Moreover, GCSs make use and provide membership mechanisms. The membership service keeps track of the active and connected nodes. Hence, reporting changes on the system configuration (i.e. failure or join of a replica). Being useful for determining if the replicated database progress condition is fulfilled: a majority of alive replicas –when a primary component membership [9] is used.

An important aspect in replicated database systems is how they manage crash node occurrences –which degrade their performance, fault tolerance and high availability support– and node connections or reconnections in order to maintain their original support. These systems have a special component named *recovery component* which deals with these situations –in a coordinated way with the consistency management performed by the replication protocol–, and the way it handles them depends on the adopted failure model. The most commonly used failure models in replicated databases are *fail-stop* and *crash-recovery with partial amnesia*, as defined in [10].

The first one makes replicated systems discard crashed replicas, substituting them by new ones. Thus, when connecting a new node to the replicated system, the recovery protocol must first transfer the whole database to the new node –recovering node– before becoming fully operational, but this is impractical for large databases.

In order to avoid such drawback, last replicated database proposals [21, 19] have adopted the *crash-recovery with partial amnesia* failure model. In this case crashed nodes are not discarded. The replicated system waits for their reconnection in order to start the recovery process. In this case the recovery protocol transfers to each recovering node only the database information lost during its disconnection. Once the recovering node has updated all its lost information, it becomes a fully operational node. So, in this approach the recovery process does not need to transfer the whole database but only the subset lost by the recovering node, shortening the recovery process and diminishing its associated problems.

But when assuming this second failure model, another problem appears: the amnesia phenomenon [12]. In this case, the problem relies on the difficulty that some protocols have to establish the correct subset of information to transfer when recovering. It appears because in some protocols the last assumed state in the recovering node is not really the last one, since such recovering replica may have lost some information that other replicas propagated to it before crashing.

In this paper, we briefly describe some database recovery protocols proposed in the literature [21, 19, 22, 8, 3, 20, 5, 7] and surveyed in [14] emphasizing how they face the amnesia phenomenon. This study also gives special importance to the database replication protocols they are designed for, because as it was depicted in [14], the recovery protocols are very dependent from the characteristics of the replication protocols used, and the information that these replication protocols store.

The rest of the paper is structured as follows. In Section 2 we detail some important GCS aspects from the recovery point of view that will be used later. Section 3 presents the amnesia phenomenon, how it is manifested and how it can become a problem. Section 4 outlines the recovery protocols, and how they support the amnesia phenomenon, while in Section 5 we categorize the analyzed recovery techniques commenting if they provide basic amnesia support, or how they can be improved to support it when they do not. Finally, section 6 concludes the paper.

2 Group Communication System Issues

Before surveying the recovery protocols and the replication protocols they are designed for, we will present some issues dealing with replication aspects that would be used later in the study.

First of all, it must be introduced the *group communication system*. GCS provides some communication primitives that are used by replication protocols to perform their work. Primitives can vary from a point-to-point communication to total order broadcast.

This GCS also provides a *membership monitor* which informs group members about membership events –node connections, disconnections, network partitions, etc.–

Additionally, some GCSs provide *virtual synchrony* [9, 4] (or *view-synchronous multicast* according to [17]) since it ensures that all replicas have delivered the same sequence of messages before any replica fails or any replica is added. According to [9], the most relaxed property related to multicast delivery that provides virtual synchrony is *same view delivery*; i.e., that all destinations of each multicast deliver each message when they belong to the same *view* (a view change arises when one process fails or rejoins the group). Virtual synchrony provides a replicated work way which facilitates the recovery protocols implementation.

3 The Amnesia Phenomenon

A first approach to manage failures in a replicated service is the usage of the fail-stop failure model [10]. 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 replicated state 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 (already commented in [12]):

- Longer periods with decreased fault tolerance support. Only fully updated replicas can be used to guarantee the correct and consistent state evolution in the replicated system.
- 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 databases, researchers have opted for assuming the crash-recovery with partial amnesia failure model [10]. In this case, when a crashed replica 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 [12, 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 [29].
- *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 [12, 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 [6]). Obviously, this is not the unique solution that can be adopted for solving this problem, being possible also to apply version-based techniques [7].

3.2 Amnesia Formalization

Before surveying the considered works, we will first formalize the amnesia problem. To do so, we consider a replicated database system, $N = \{n_1, n_2, ..., n_n\}$, compound by n replicas, being $n > 2$ (primary partition assumption [9]). 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 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 [18] 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 [9] 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 $\forall\, n_l \in \mathcal{V}_i \setminus \mathcal{V}_{i+1}$, due to [29] 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 the general case, but $s \geq 1$ when $T^D_{i,n_l} \neq T^C_{i,n_l}$

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 it 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:

- *Prop. 1*: n_l must remember its last committed transaction, $T_{i,m-s}$;
- *Prop. 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 [12, 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 Considered Recovery Protocols

In this section we briefly describe the recovery protocols considered in this study, highlighting only the details that are important from the amnesia support point of view. When detailed, we also include for each recovery protocol some remarks for our study about the replication protocols to which they are associated. For other details we encourage readers to look at the original papers. Note that in most cases the replication and recovery protocols were originally described in different papers (or even no replication protocol was described). As a result, a solution for the amnesia problems was not the target of such papers.

For determining if these recovery protocols provide accurate amnesia support we will study if they fulfill the two properties presented in Section 3.2.

4.1 Protocols by Kemme, Bartoli and Babaoğlu

Multiple recovery protocols for replicated databases are presented in [22]. All of them are proposed for database replication protocols sharing the following characteristics: update everywhere protocol –ROWAA approach [16]–, based on total order broadcast –without a terminating phase– propagating a message per transaction –constant interaction [28]– and virtual synchrony. Moreover, replicated data objects are tagged with version numbers. The provided correctness criterion is *one-copy-serializability*.

These recovery protocols fulfill the following issues:

1. *Single Site Recovery*. The recovering node first brings its own database into a consistent state. To do so the underlying database maintains a log of performed writes during the normal processing, storing the initial and resulting values for each changed data object. Then, once it reconnects it checks this log in order to store in the database the changes of committed transactions that were not already applied in the database.

2. *Data Transfer*. An operating site –recoverer node– must provide the current database state to recovering nodes. Different techniques can be used, from transferring the full database to transferring only the set of updated objects.

3. *Determination of a Synchronization Point*. If transaction processing is allowed during the recovery process it must be ensured that the recovering node will reflect the updates performed by these transactions. This synchronization process can be done in different ways but it depends strongly on the data transfer technique.

On the sequel we describe the recovery protocols proposed in [22], focusing on the data transfer and synchronization points.

4.1.1 Database State Transfer Checking Version Numbers

In this protocol global transaction identifiers are used, marking each data object with the identifier of the last transaction that updated it –allowing later the system to determine the information set to transfer in recovery processes–. The recovering node informs to the recoverer node about its *cover* transaction, (i.e. the transaction with the highest global identifier that successfully committed, $T_{i,m-s}$). Thus, the recoverer can determine the updates lost by the recovering node –information that must be transferred–, including the updates associated to $T^F_{n_l}$. The recovering node can easily determine its *cover* transaction by reviewing its single site recovery log.

The amnesia phenomenon is avoided with this replication protocol because the recovering node tells to the recoverer node which is its last real committed transaction, $T_{i,m-s}$. Thus, the recovery process transfers all data objects that were modified by transactions delivered between the last real committed transaction in the recovering node and the first transaction propagated after it become alive. These lost updates are transferred using a DT. In properties terms:

- *Prop. 1*: It is fulfilled because the recovering node remembers its last really committed transaction.
- *Prop. 2*: Data modified by T^F_{i,n_l} are marked with the associated transaction identifier, so they will be later transferred in the recovery process. Notice that it is possible that this data is included when recovering $T^M_{n_l}$ and not by T^F_{i,n_l}, because it has been modified also during the node disconnection.

This recovery protocol presents the drawback of scanning the entire database when checking for the database subset to transfer, then the following proposal was designed to overcome it.

4.1.2 Restricting the Set of Objects to Check

In order to avoid the full scan on the entire database, and with this the overhead and long locking time that it may cause, the use of a so-called "reconstruction table" is proposed. A record in this table consists of an object identifier and a global identifier informing about the last transaction that updated the object. Each update is recorded in the reconstruction table, unless all sites have successfully performed the update.

In contrast to the previously discussed protocol options, this one only needs to set a single lock on the entire database. Once the incremental data set to be transferred is determined, that lock is replaced by fine-grained object level locks on the respective data items.

This proposed optimization does not handle correctly the amnesia phenomenon. This is because the reconstruction table only is generated when there are failed nodes. Then, only those objects modified in a view with failed nodes are included in this table. It implies that if a failed node has not been able to process correctly a message delivered before crashing –in a view with no failed nodes–, when it reconnects it will have not applied the associated updates. And the reconstruction table will have not stored these updates because they have been performed in a view without failed replicas, being unable the recovery system to transfer the correct information set. Expressing it in properties terms:

- *Prop. 1*: It is fulfilled because the recovering node remembers its last really committed transaction.

- *Prop. 2*: It is not always fulfilled because data modified by T^F_{i,n_l} is only stored in the reconstruction table and marked with the associated transaction identifier if there are failed nodes. Therefore, the system will not have the T^F_{i,n_l} changes in the reconstruction table when a replica crashes in a replicated system where there were not any crashed node.

Therefore, this recovery protocol must be modified in order to support the amnesia phenomenon. One possibility consists in generating the recovery information in the reconstruction table either with or without the presence of failed nodes, ensuring always the *Prop. 2*. Other possibility consists in using our proposed solution in [12] ensuring then always *Prop. 2*.

4.1.3 Filtering the Log

In the previously discussed optimization, locking of non-relevant data is reduced, but locks on relevant data may still last long. To avoid locks, multiple versions of data can be used, e.g., the use of multi-version concurrency control, as in `PostgreSQL`, or `Oracle`. In that case, transactions can continue to update the database while earlier versions that have been missed by the recovering site are transferred to it.

This recovery technique must be combined with one of the previous ones, that will be used for generating the recovery information and determining the subset to transfer, for working. Therefore, its amnesia support depends on the support provided by the combined technique. Hence, if the last one *–Restricting the Set of Objects to Check–* is selected in its original description the amnesia phenomenon will not be managed accurately.

4.1.4 Lazy Data Transfer

Up to this point, all mentioned solutions use view changes as synchronization points. Then any recovering node enqueues all new broadcast transactions for applying them once it has recovered its lost views. This approach, despite being simple, has several drawbacks (detailed in [22]) leading the authors to decouple the synchronization point from the view change.

In this new approach any recovering site discards the messages delivered –instead of enqueuing them– . After the view change –triggered by its reconnection–, the recoverer site starts the transfer. When the transfer is about to complete, the recoverer and the recovering sites agree a delimiter transaction –one of the transactions broadcast in the new view– as synchronization point. Then, the recoverer site transfers all changes performed by transactions with lower identifier than the delimiter transaction one. While, the recovering site starts enqueuing transaction messages with greater identifier than the delimiter transaction one, for applying them once the data transfer is completed.

This recovery proposal differs from the others in the synchronization point, but depends on the previous ones for obtaining the recovery information. Then, it will support the amnesia phenomenon depending on the recovery information generation policy being used.

4.2 Protocols by Holliday

The recovery protocols proposed by J. Holliday in [19], were designed for the replication protocols *Broadcast Writes*, *Delayed Broadcast* and *Single Broadcast* described in [1]. According to the classification in [27], these are eager update everywhere and non-voting protocols. Concurrency control is performed by the DBMS with Strict 2PL. These replication protocols make also use of a GCS which provides atomic broadcast, virtual synchrony and a membership monitor. They provide the *one-copy serializability* correctness criterion.

These replication protocols differ in the number of messages used for propagating transactions. *Single Broadcast* only spreads a message per transaction, *Delayed Broadcast* propagates two messages per transaction –writeset and commit messages–, while *Broadcast Writes* sends a message per write transaction operation –linear interaction–.

On the sequel we will summarize the recovery approaches presented in [19].

4.2.1 Single Broadcast Recovery

This recovery approach is designed for replication protocols which broadcast a single message per transaction as [23]. Another author criterion design is to avoid to transfer the whole database in the recovery process if possible, and the selected mechanism for doing so consists in reapplying in outdated nodes the messages that they have lost.

Therefore, this recovery protocol relies on a GCS which provides a log of delivered messages. If the GCS does not provide this log, the recovery protocol must designate some replicas as *loggers*. These *loggers* will have a log where they will store persistently the delivered messages, either those notifying view changes and those broadcasting update transactions –keeping only those associated to committed transactions, deleting aborted ones–.

Then, when a node reconnects –a view change is triggered– it requests a *logger* to be brought up-to-date, informing about its last view –last view in which the recovering node was alive–. Thus, the *logger* transfers to the outdated node the messages broadcast during the views it was crashed –maintaining their original order–. Sometimes the *logger* will not have, due to log storing policies, all the messages necessary in the recovery, thus it will transfer the whole database. It must be remarked that as long as a node is being recovered the replicated system can not work, starting only when the recovery process has been completed.

This protocol does not support the amnesia phenomenon accurately, because when a crashed node reconnects, the system starts to transfer the messages broadcast in the views it was failed. Then, this information does not include messages delivered in the crashed node before crashing but not processed correctly. In properties terms:

- *Prop. 1*: It is not fulfilled because the recovering node only remembers its last seen view, i.e., it does not maintain nor propagate the $T_{i,m-s}$ identifier.
- *Prop. 2*: This property is not fulfilled because messages –recovery information– are stored by view. But it can be easily overcome using messages as basic recovery information unit.

One possibility for handling accurately the amnesia phenomenon in this recovery protocol would be to use message or transaction identifiers –which are equivalent in this replication protocol–. Then the recovery protocol can be modified forcing each replica to mark which is its last really committed transaction. Therefore, when a replica reconnects after a crash, it can inform about its last committed transaction to the *Logger* –instead of using the identifier of its last seen view– for the recovery. Then both properties are ensured.

Another possibility would consist in giving the *Loggers* role to all the replicated system members. Therefore when a node reconnects, it will perform a local recovery step consisting in checking if some of the persisted messages in its local log have not been correctly processed, applying them in this step. In this case both properties are also ensured.

In both cases, notice that it is necessary that *Loggers* store persistently the broadcast messages even when there are no failed nodes. In the second approach, the messages that have been seen by all nodes –because there are not any failed nodes– can be removed from the log –of a replica– as soon as they are correctly processed in this replica. While, in the first approach, these messages can only be removed view per view. And the messages broadcast in a view where there were not failed nodes, only can be removed if in the subsequent view there are not any failed nodes –which is a non sense– or when the nodes whose crash triggered this view change are recovered and the system ensures that they have correctly processed all the messages broadcast when there were not failed nodes.

The second solution provides better recovery support as all replicated members are *Loggers*, and provides a more simple way for managing messages that have been seen by all nodes. Therefore, we encourage its use.

4.2.2 Delayed Broadcast Recovery

The *Delayed Broadcast* replication protocol decouples the writeset broadcast from the commit broadcast for any transaction –weak voting technique [30]–. This behavior raises some problems when recovery is being considered. It might happen that the recovering site was able to deliver the writeset for a particular

transaction, but not its commit or rollback message. So, that writeset was lost when the site failed and should be retransmitted now by the recoverer site if its commit message was delivered whilst the recovering site was crashed. Two possible solutions for the problems caused by the writeset-commit decoupling are presented:

1. *Log Update Method*. In this approach, at each view change, *loggers* must examine their logs or the database state for determining if there exist on progress transactions in the nodes without failure. If there are, the *logger* must mark these transactions in order to copy their writeset message in the log associated to the view when their commit was broadcast. So, when a previously failed node rejoins to the group, the *logger* begins transferring writesets of in progress transactions when the node failed, following with messages of transactions originated and committed while the node was failed. The commit order is the same for all non-aborted transactions. The operations of the aborted transactions are not included in the log since their effects are undone in the nodes without failure.

2. *Augmented Broadcast Method*. This second method gives additional process for managing on-going transactions and requires a change in the lock policy for recovering nodes during the global recovery. The new replication protocol forces to include the writeset in the broadcast commit message for these transactions that have delivered the writeset in a previous view. The nodes that have already seen the first broadcast writeset message ignore the writes included in the commit message, and *loggers* store the augmented commit message. The existence of augmented messages obliges global recovery to change its lock policy as it is described in [19].

In this case, as the policy for determining the start point recovery is the same one as before –the identifier of the last view in which the crashed node was alive–, an accurate amnesia phenomenon support is not provided. Explained in properties terms:

- *Prop. 1*: As before, it is not fulfilled because the recovering node only remembers its last seen view.
- *Prop. 2*: This property is not fulfilled because messages –recovery information– are stored by view.

Therefore, the modifications proposed in the previous recovery protocol are also valid for this one. Anyway, it must be pointed out that in this case handling delivered messages correctly is more difficult because the system broadcasts two messages per transaction –*writeset* and *commit* or *rollback*– with its associated complexities.

If the second solution –all nodes are *Loggers*– is selected, when the recovering node performs the additional local recovery step –checking if some of the persisted messages in its local log have not been correctly processed for applying them– it must discard the messages belonging to transactions whose commit message is not also stored in the log. This is because these messages would be later applied in the *Global Recovery* process.

4.2.3 Broadcast Writes Recovery

The *Augmented Broadcast* global recovery method presented for the *Delayed Broadcast* replication protocol could be used also for the *Broadcast Writes* one –which broadcasts a message for each write operation, in other words linear interaction–. Then all writes must be attached to the commit message to be broadcast for on-going transactions, as it does the *Augmented Broadcast*. But in this recovery protocol *loggers* must take special care for removing the logged messages of aborted transactions due to deadlocks, in order to not reapply them in recovering nodes.

As the two other recovery protocols proposed by Holliday it does not handle correctly the amnesia phenomenon problem, because the underlying mechanism for determining the recovery information set to transfer is the same one –to send the identifier of the last view seen by the crashed node–. In properties terms:

- *Prop. 1*: As in two previous ones, it is not fulfilled because the recovering node only remembers its last seen view.
- *Prop. 2*: This property is not fulfilled because messages –recovery information– are stored by view.

Anyway, the proposed solutions for the previous recovery protocol will also work for this one.

4.3 Parallel Recovery by Jiménez, Patiño and Alonso

In [21] the authors presented a recovery protocol whose main goal was to avoid stopping the replicated system work when performing recovery processes.

The replication protocol for which it was designed used a GCS that provided strong virtual synchrony, reliable multicast and a membership monitor. The replicated database was divided into disjoint partitions,and the system forced transactions to access only single partitions. Each partition had a master site –which processed the transactions accessing this partition– and the rest of replicas worked as backups –which only applied updates–, therefore it is a passive replication protocol per partition. And transactions are broadcast using only one message –constant interaction–. The transactional system supports Strict 2PL, providing *one-copy serializability*.

Each node has a log –one per partition– which contains the committed updates in the same order they were applied. Updates are only logged once their commit is confirmed. When a crashed node reconnects to the system it informs about the LSN –*log sequence number*, a global number– of its last committed transaction on each partition. Then the selected recoverer for each partition will collect and transfer from its log the set of messages needed to recover this partition in the outdated node. In order to limit the recovery duration –interesting for long failure times– some form of checkpointing is assumed. Therefore, if it is necessary, the recovering site will first receive a recent checkpoint of the database and later can start applying messages from this checkpoint.

The combination of these two techniques, or the use of the *LSN* of the last committed transaction in the node being recovered allows the protocol to overcome the amnesia problem. The problem of this solution depends on the way in which the checkpoint process is performed, because if the whole data state is transferred the benefits of adopting the crash-recovery with partial amnesia failure model are lost. Expressed in properties terms:

- *Prop. 1*: It is fulfilled with the use of *LSN*.
- *Prop. 2*: This property is fulfilled because nodes store committed updates and combines this with a checkpointing technique when necessary.

4.4 The COLUP Recovery Protocol

A configurable eager/lazy replication protocol with a lazy recovery protocol is proposed in [20]. The replication protocol can be categorized as an update everywhere approach with voting technique, using constant interaction. This protocol defined and provided its own correctness guarantees: *transaction* and *checkout* consistency. These correctness guarantees are somewhat equivalent in some circumstances to *snapshot isolation* and *read committed* respectively.

In this replication protocol each data object is owned by the replica where it was created. For any object, a set of nodes will maintain synchronous copies, while other replicas constitute the set of asynchronous copies. In these last nodes object updates will be eventually received, once they have been committed in synchronous replicas. The owner is responsible of managing object accesses and coordinating the propagation of their last versions.

Conflict transactions are solved in the processing node in an optimistic way, using object versions. To do so, for each accessed object –for those the node does not have a synchronous copy– it calculates the probability of having an outdated version. If the obtained value is higher than an established threshold the node assumes that its object version is obsolete, obtaining from the owner node the last version. Later, in the commit phase it checks for possible conflicts. Aborting the transaction if it has read obsolete values that were updated by other concurrently committed transactions.

In a node crash the ownership of its objects is assumed by an alive and synchronized replica. Then, alive nodes inform the new owner about *previous grants* conceded to these objects by the previous owner. Thus, the new owner can process the requests as if it was the original owner node of the object.

When a node recovers from a failure, it sends a message to the node that managed its owned objects in order to synchronize the activity in both nodes. In this process, the recovering node updates in a version-based way the state either of its owned objects and the objects for which it is a synchronized replica. During this process, the recovering node may receive requests for objects that were updated during the

failure interval. In order to handle this situation, the recovering node must consider each object of which it is owner like an asynchronous replica until it is updated by a synchronous replica.

This recovery protocol provides accurate amnesia support because as soon as a node reconnects it starts to obtain the last state of its owned and synchronized objects in a version-based way. The objects that are maintained in this replica asynchronously are updated using the basic mechanism provided by the replication protocol. This protocol does not fulfill in a direct way the properties presented in section 3.2, but in an indirect way:

- *Prop. 1*: It is not fulfilled, but in fact it works as if it has not committed any transaction: all synchronized objects are transferred immediately in the recovering process, and non-synchronized are updated using the replication mechanism.
- *Prop. 2*: This property is fulfilled because it uses all the database as source of recovery information.

4.5 CLOB: Short-Term Failure Recovery

CLOB (Configurable LOgging for Broadcast protocols) described in [5] is defined as a framework for reliable broadcast protocols that are used as a basis for database replication. Its aim is to log messages in the broadcast protocol core, providing with this automatic recovery for short-term failures, but discarding the log and using a version-based recovery protocol (e.g. [7]) for long-term outages.

In order to do so the recovery protocol has two logs: one for missed messages, another for received messages. In the first one, each node stores any message it delivers when there are failed nodes, maintaining them as long as there is any failed node that has not received them. In the second one, each node stores any received message, removing it as soon as it is correctly processed. So, when a crashed node reconnects –and the system uses the log recovery–, it first checks the log of received messages in order to process its last received messages that were not correctly processed before crashing. Later, it asks for its missed messages, and applies them.

Notice that if the outage period exceeds a given threshold, the reliable broadcast service will notify the replication protocol about that, and the logs will not be used.

The CLOB recovery protocol manages accurately the amnesia phenomenon because it considers a persistent log where each replica stores its delivered messages as soon as they are received. And these messages are only deleted once they are correctly processed. Then, when a crashed node reconnects, only needs to check this log and reapply the messages it contains. Talking about properties:

- *Prop. 1*: It is fulfilled in an indirect way. All messages maintained in the queue represent delivered transactions non correctly processed, so instead of knowing its last really committed transaction it has the T^F_{i,n_l}.
- *Prop. 2*: As it has been said above, each node stores persistently its own T^F_{i,n_l}.

4.6 Protocol by Armendáriz

In [3] three replication protocols are considered –*BRP*, *ERP* and *TORPE* –, and a recovery protocol that can be applied on *ERP* and *TORPE* is proposed. These two replication protocols are categorized for being eager update everywhere and sending a constant number of messages per transaction. They make use of a GCS which provides reliable broadcast, a membership monitor and virtual synchrony. The correctness guarantees provided by these protocols were *one-copy serializability*, provided thanks to the use of underlying DBMS which ensured serializability.

The main idea for the recovery protocol proposed in [3] is to store in a database table –in all alive replicas– the identifiers of objects modified when there are failed nodes, grouping them per views. Then, when a failed node reconnects, it informs about the last view in which it was alive. Later, a recoverer node transfers to the recovering node the identifiers of modified objects during its disconnection, and later transfers their values.

The recovery protocol proposed by Armendáriz for the replication protocols ERP and TORPE can not manage accurately the amnesia problem. In this case, the problem resides in the fact that this recovery

protocol assumes that any delivered message is correctly processed, but this assumption, as demonstrated in [29], is not correct. So, all generated recovery information does not contain all the information that would be needed for supporting amnesia. Expressing all this in properties terms:

- *Prop. 1*: It is not fulfilled because the recovering node only remembers its last seen view.
- *Prop. 2*: This property is not fulfilled because the recovery information is grouped by view. And either it presents the problem of being generated only when there are failed nodes.

In [15] it is provided amnesia support to this recovery protocol. The adopted solution is the same one as we propose in Section 3.1, to log persistently the delivered messages.

5 Amnesia Support Recovery Observations

We have seen in the study how a correct amnesia support depends on the combination of an adequate recovery information generation policy and an accurate way for notifying the last really committed changes in the node that must be recovered.

On the sequel, we will present some observations obtained from the performed study. These observations are grouped first by the used technique –version-based or log-based–, and secondly by the granularity used for managing the recovery information.

5.1 Version-based Techniques

Version-based recovery protocols can overcome this problem in different ways, depending on the basic way used for performing the recovery processes.

5.1.1 Transaction identifier

The first one will consist in storing for each object the identifier of the last transaction that modified it. But, this must be done even if there are not failed nodes as it does the *Database State Transfer Checking Version Numbers* presented in [22], because if it is not done the amnesia support is not provided as it happens with *Restricting the Set of Objects to Check* presented also in [22]. Thus, in this case the recovering node only has to inform the recoverer node about the identifier of its last committed transaction. Therefore, properties *Prop. 1* and *Prop. 2* are ensured.

An alternative for this strategy will be to combine it with our amnesia generic solution approach described in Section 3. In this case it would not be necessary to generate this information even when there are not failed nodes. And, then this approach does not need the transaction granularity being enough with the view identifier granularity. It is due to the fact that in this case each replica maitains its own T^F_{i,n_l}, being only necessary to inform the recoverer node about the last seen view in the recovering node.

5.1.2 View identifier

Another possibility is to store for each object the identifier of the last view in which it was modified. The problem of this solution is that the recovery protocols that follow this approach start the recovery process from the first view lost by the recovering node, being impossible then to solve the amnesia problem, associated to the *forgotten state* $-T^F_{i,n_l}-$ because even if *Prop. 1* is ensured, *Prop. 2* is not ensured. It happens in *Protocol by Armendáriz* [3]. This can be solved as follows:

- One option for overcoming this would consist in including in the transfer recovery process the changes performed in the last view seen by the recovering node. So, this solution forces the system to generate recovery information even when there are not failed nodes. But, this approach presents some drawbacks. On one hand, it forces to transfer all the performed changes in a view –most of which will have been already seen by the recovering node– for solving the amnesia problem that will affect usually a very small subset of changes done in such view. On the other hand, it is possible that in very special cases transferring only the changes done in the last view seen by the recovering node is not enough for solving the amnesia problem (e.g. a sequence of very short views in time terms).

- Discarding the previous option, another strategy will consist in combining this strategy with our generic approach –using in each replica a persistent log of delivered messages– as it is done in [15], fulfilling then the properties *Prop. 1* and *Prop. 2*. In this case, it is not necessary for the version-based strategy to generate information when there are not failed nodes, because it is already maintained in the queue.

5.2 Log-based Techniques

In these techniques, recovery protocols use as recovery information the broadcast messages during the replication work. Therefore, the only way for solving the amnesia problem is to maintain in the system the messages that can be affected by the amnesia problem.

5.2.1 Transaction identifier

In this technique, stored messages –all replicas store messages– are not grouped by views, then when a crashed node reconnects it informs about the message corresponding to its last committed transaction. Then, the recoverer node sends to the recovering node the set of messages it has not correctly processed and it has lost. Notice, that this policy will overcome the amnesia phenomenon in all cases, only if logs store messages even when there are not failed nodes. If this behavior is not provided the *Prop. 2* is not ensured when a replicated system transits from a view where all replicas were alive to another where there are failed nodes.

An important aspect of this technique is when messages or updates are stored in the log. If messages are persisted as soon as they are delivered, crashed nodes will have at recovering time the messages they have delivered but not processed correctly –those associated to T^F_{i,n_l}–. Then, they do not have to ask updated replicas for these messages, only for those they have not seen. On the contrary, if messages –or updates– only are logged when they are really committed, crashed nodes will not have the messages necessary for overcoming the amnesia problem at recovering time. So, in this case the information for solving the amnesia phenomenon must be looked for in the recoverer replica.

This is the case of the *Parallel Recovery by Jiménez, Patiño and Alonso* [21] protocol. This protocol also combines this technique with checkpointing for log shortening reasons. It must be noticed that this protocol stores updates once they are committed –non when they are delivered–, so crashed replicas must ask updated replicas for messages delivered but not correctly processed.

5.2.2 View identifier

In this strategy broadcast messages are stored when they are delivered –in the same order delivery– being grouped by views –when there are crashed nodes–. Then, when a crashed node reconnects it informs to the system about its last seen view. At this point, the system starts to send to the recovering node the messages broadcast during the view it was crashed. Therefore, the amnesia problem is not solved as it occurs in all recovery protocols proposed in [19], because it will not contain messages seen by the crashed node but non correctly applied, in other words the recovery process does not transfer the messages corresponding to the transactions set T^F_{i,n_l}. In fact, neither *Prop. 1* and *Prop. 2* are ensured. For solving this problem, two different approaches can be adopted:

- A first proposal for avoiding the amnesia problem in this technique can consist in transferring in the recovery process the messages broadcast during the last view where the crashed node was alive. Then, this solution needs to store broadcast messages even if there are not failed nodes. But, it can be optimized if the recovering node informs about the identifier of the message associated to its last correctly processed transaction. Moreover, it must be noticed that if all nodes store broadcast messages the own crashed node will contain the messages it has received and not correctly applied, obtaining then the second approach.

- The second one consists in applying our proposed generic solution, that in fact is the solution already applied in [29, 5]. In [29], authors proposed the "*successful delivery*" approach. A successfully delivered message implies that it has been correctly processed. Therefore, they proposed that the used

GCS has to deliver the same message to a replica until it is successfully delivered in this replica. In [5], each node stores persistently all its delivered messages, being only removed when they are correctly processed. Obviously, if there are failed nodes, correctly processed messages are not removed but maintained in another log for recovering failed nodes during this view.

6 Conclusions

In this survey we have analyzed how some recovery solutions for replicated databases, which have adopted the crash-recovery with partial amnesia failure model –in order to avoid to transfer the whole database–, manage the introduced amnesia phenomenon problem.

This problem appears because some works assume that all delivered messages are correctly processed, fact that as it is demonstrated in [29] is not true. Then, in most cases their provided recovery solutions do not handle correctly this problem. Among the studied papers only the recovery protocols proposed in [21, 20, 5] and two of [22] manage accurately this problem.

Moreover, for those studied recovery protocols which do not provide accurate amnesia support we have proposed solutions for overcoming this situation. In fact, we always recommend for solving this problem a queue in each replica where it persists broadcast messages as soon as they are delivered, removing from this place as soon as they are correctly processed.

Later, we have categorized the analyzed recovery techniques commenting if they provide accurate amnesia support, and how they can be improved to support when they do not in its original definition.

7 Acknowledgements

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

References

[1] D. Agrawal, G. Alonso, A. El Abbadi, and I. Stanoi. Exploiting atomic broadcast in replicated databases. *LNCS*, 1300:496–503, 1997.

[2] 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.

[3] José Enrique Armendáriz. *Design and Implementation of Database Replication Protocols in the MADIS Architecture*. PhD thesis, Univ. Pública de Navarra, Pamplona, Spain, February 2006.

[4] K. Birman and T. Joseph. Exploiting virtual synchrony in distributed systems. In *11th ACM Symposium on Operating Systems Principles*, pages 123–138, New York, NY, USA, 1987. ACM Press.

[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 *PDP*, pages 314–321, 2005.

[6] 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.

[7] 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.

[8] F. Castro, L. Irún, F. García, and F.D. Muñoz. FOBr: A version-based recovery protocol for replicated databases. In *13th Euromicro PDP*, pages 306–313, Lugano, Sw, 2005.

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

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

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

[12] 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.

[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, J. Enrique Armendáriz-Íñigo, Hendrik Decker, and Francesc D. Muñoz-Escoí. Recovery protocols for replicated databases - a survey. In *Workshop FINA-07, in the AINA-07 Conference*. IEEE-CS Press, 2007. Accepted for Publication.

[15] 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.

[16] Jim Gray, Pat Helland, Patrick O'Neil, and Dennis Shasha. The dangers of replication and a solution. In *ACM SIGMOD International Conference on Management of Data*, pages 173–182, 1996.

[17] Rachid Guerraoui and André Schiper. Software-based replication for fault tolerance. *IEEE Computer*, 30(4):68–74, 1997.

[18] 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.

[19] JoAnne Holliday. Replicated database recovery using multicast communication. In *NCA*, pages 104–107. IEEE Computer Society, 2001.

[20] Luis Irún, F. Castro, F. García, A. Calero, and Francisco Muñoz. Lazy recovery in a hybrid database replication protocol. In *XII Jornadas de Concurrencia y Sistemas Distribuidos*, pages 295–307, 2004.

[21] Ricardo Jiménez, Marta Patiño, and Gustavo Alonso. An algorithm for non-intrusive, parallel recovery of replicated data and its correctness. In *SRDS*, pages 150–159, 2002.

[22] B Kemme, A. Bartoli, and O. Babaoğlu. Online reconfiguration in replicated databases based on group communication. In *Intl.Conf.on Dependable Systems and Networks*, pages 117–130, Washington, DC, USA, 2001.

[23] 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.

[24] Francesc D. Muñoz-Escoí, J. Pla-Civera, María Idoia Ruiz-Fuertes, Luis Irún-Briz, Hendrik Decker, José Enrique Armendáriz-Iñigo, and J. R. Gonzalez de Mendivil. Managing transaction conflicts in middleware-based database replication architectures. In *SRDS*, pages 401–410. IEEE Computer Society, 2006.

[25] 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.

[26] 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.

[27] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, and G. Alonso. Database replication techniques: a three parameter classification. In *SRDS*, pages 206–215, 2000.

[28] 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.

[29] 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.

[30] Matthias Wiesmann and André Schiper. Comparison of database replication techniques based on total order broadcast. *IEEE Trans. Knowl. Data Eng.*, 17(4):551–566, 2005.