Supporting Amnesia in Log-Based Recovery Protocols*

Technical Report ITI-ITE-07/01

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

Abstract

Replicated systems are commonly used to provide highly available and fault tolerant applications, based on the use of replication and recovery protocols. Traditionally, the literature has focused on replicated systems which adopt the fail-stop failure model which presents good performance levels for replicated systems managing few state. This paper points out how the crash-recovery with partial amnesia failure model presents a better accuracy for replicated systems with huge state, but how its use has the amnesia phenomenon drawback. Then, the paper analyzes this phenomenon and how to deal with it in a basic configuration using a log-based recovery approach. Analyzing after, how it is supported and managed with other replication configurations.

1 Introduction

Fault tolerance, high availability and performance are success keys in nowadays information systems. Consequently, distributed systems have been widely expanded among organizations and enterprises in order to provide these characteristics.

Particularly, replicated systems are the most common way used to reach these goals, being replicated databases one of the typical applications. Therefore several replication techniques have been largely studied and a wide range of proposals have been implemented. Thus, latest trends in replication techniques are oriented to make use of group communication system semantics. In fact most non-commercial solutions combine eager update propagation with constant interaction [23] using atomic broadcast protocols [14], providing more efficient implementations. A wide number of approaches [19, 15, 22] are described in the literature.

Group communication systems make use of membership mechanisms, which are often provided to the applications built atop of them (e.g. replication protocols). On one hand, this mechanism excludes disconnected, failed or partitioned nodes from the group, notifying the changes to survivor nodes. On the other hand, it allows new incorporation to the group or node reconnection, also notifying the membership changes to the group members.

Usually, these join events may originate outdated nodes, i.e. replicas that have lost some updates, and therefore without the last state. But traditionally, replication protocols do not give great relevance to the outdated nodes recovery, since a full state transfer had been assumed in order to manage such events. So it seems to be needed a new architecture component that knows what to do when: a node failure occurs, a failed node rejoins to the group or a new node is added to the replication system if a lot of state is being maintained in the replicated servers. The functions of this component are: to update replicas with an outdated system state before becoming full-functional system nodes and to colaborate with the replication protocol in order to store and maintain the information used in the recovery processes.

This recovery process of outdated nodes can be carried out in many ways, ranging from the simplest one (a backup transfer) to more complex alternatives. But ideally, this process must be performed without interfering the common work of the replicated system. In this direction, a wide variety of recovery protocols has been presented in the literature scoped on replicated databases, as [19, 17, 5] most of them based on group communication.

In parallel, the traditional failure model adopted, the *crash* or *fail-stop*, which ignores the outdated nodes recovery, presents a good behavior when the replicated system manages few data state, but it is is not as good for replicated systems with large data states where the outdated nodes recovery becomes a key point for building fully-functional fault tolerant systems. Therefore, the use of the *crash-recovery with partial-amnesia* failure model is

*This work has been partially supported by the Spanish MEC grant TIN2006-14738-C02-01.

recommended for replicated systems which manage large states, because it supports the recovery of outdated nodes providing a better recovery performance. But, this assumption gives to the amnesia phenomenon a key role in recovery processes, phenomenon that if it is not correctly managed can lead to diverging states in recovered nodes. Thus, this paper is focused in this amnesia phenomenon, detailing at which levels appears and how it can be treated avoiding its dangerous effects on the recovery processes in log-based recovery strategies. Moreover, in this work it is also analyzed which replication configurations can deal with this problem and which modifications must be adopted to do so.

This paper is structured as follows. Section 2 details the considered system model in this work, and the node system architecture. A comparison between the fail-stop and the crash-recovery with partial amnesia is performed in section 3. Afterwards, the amnesia phenomenon is described in section 4, explaining in the subsequent section 5 how to deal with it in our initial replicated configuration using a log-based recovery strategy. Section 6 details how the amnesia problem can be managed in other replication configurations depending on their characteristics and section 7 presents and overhead analysis. Finally, some related work is given in section 8, and section 9 concludes the paper.

2 System Model

Our model considers a replicated transactional system, which is compound by several replicas, whose architecture is shown in figure 1, and 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 group communication system (*GCS*). Point-to-point and broadcast deliveries are supported. The minimum guarantee provided is a FIFO and reliable communication.

It is also assumed the presence of 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 group membership service combined with 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* [2] approach. This makes simpler and easier the

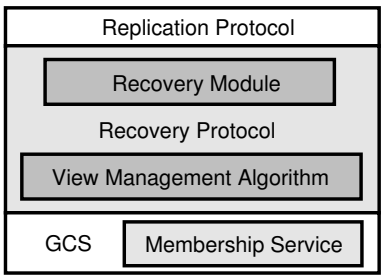

Figure 1. Node Architecture

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 it is detailed in [9]. At the same time the state consistency is ensured because only the primary *subview* is able to work in partition scenarios. Thus, this subview is the only one allowed to generate recovery information, which will be afterwards used for recovery. For similar reasons, a node can not start new transactions until it has not been fully updated.

3 Failure Models

Different papers, as [8, 4] have presented different failure models that can be adopted in replicated systems. Among these failure models it can be pointed out the *fail-stop* and the *crash-recovery with partial amnesia*.

The assumed failure model in a replicated system has a great importance in the way it provides fault tolerance. Traditionally, most replicated systems have adopted the fail-stop failure model, because it is the simplest one to manage. When a replica crashes in these systems, they substitute the crashed replica by a new one, transferring the whole state to the new one. But, this option will not work well for replicated systems which manage large amounts of data state, because will lead to very long transfer periods. Longer periods for preparing a new wholy active replica, will imply in the replicated system the following consequences:

- Longer periods with low performance levels for systems based on active replication. This effect is obvious for replicated systems which do not allow outdated replicas to work. But, it is also present for systems that allow outdated replicas to work, because the rate

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

of work concluded correctly is degraded for being outdated. It must be noticed, that replicated systems based on passive replication do not suffer this effect.

- Longer periods with decreased fault tolerance support. This is due because from the fault tolerance point of view, 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 fullfil the progress condition (i.e. systems based on primary partitions).

In order to avoid all the above presented problems for replicated systems which manage large amounts of data state (i.e. replicated databases), our proposal is to apply recovery processes in the previously crashed nodes, once they reconnect, transferring them only the information they missed. In other words, to use the crash-recovery with partial amnesia failure model.

The problem of this assumption, is that the replicated system must manage the amnesia problem in order to ensure that the achieved state in the outdated node, after applying the recovery process, is consistent. In this scenario, consistent means that the recovered node reaches the replicated system state. Therefore, the replicated system must know which is the last consistent state reached by the crashed node, in order to transfer to it the exact needed information, and avoiding the problems of losing some changes or applying others twice or more.

Following sections detail how the amnesia problem manifests in replicated systems and how it can be managed in a generic way with a log-based strategy. Subsequently, it will be detailed the amnesia support that can provide replicated systems depending on their characteristics.

As previous step, it will be described how the amnesia phenomenon could arise in these systems in order to apply the necessary corrective measures.

4 The Amnesia Phenomenon

As described above, the assumed failure model, crash-recovery with partial amnesia, implies that on reconnections of crashed nodes, they can not remember exactly which was their last state before their crash occurrence. In other words, it can be said that when nodes crash they expect to have a theoretic last state that can differ from their real last state due to amnesia. This phenomenon is due to the fact that nodes have received some messages but their attached changes have not been committed before the nodes crash time. So, their recovery processes must take under consideration their respective actual last state before crashing.

In order to solve this problem the node must know precisely the identity of every committed transaction in its underlying transactional system replica. This is necessary because it is possible for a transaction expected to be committed in the system to have not been actually committed in the underlying transactional system of this replica (e.g. due to overhead or because the node is being recovered and could not apply the currently received replication messages).

The amnesia phenomenon can also be manifested in a different way, when a reconnected node does not remember the received messages before its crash. The idea is that the node receives some messages, and the group communications system assumes that these messages have been delivered. At this point, if the node crashes before committing the updates associated to such messages - storing them persistently -, when the node is reconnected it has lost such updates. Messages that the communications system assumes to be already transferred.

Thus the amnesia problem arises at two different levels:

- *Transport level*, at this level, it 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.

- *Replication level*, the amnesia is manifested here in the fact that the node "forgets" *which were the really committed transactions*.

Depending on the recovery policy used, log-based or version-based, the information that must be maintained to solve the amnesia problem differs. Thus, extra information is needed to be maintained to this particular end, being afterwards used in the amnesia recovery process. The next section provides ways to avoid amnesia problems at recovery time for log-based recovery policies.

5 Log-Based Amnesia Recovery

In the basic considered replicated systems each transaction is only processed in one replica, the replica that serves the client request, and only its associated updates are broadcast among all alive replicas. In this scenario, log-based recovery protocols use as recovery information the broadcast replication messages missed by crashed nodes. So, the solutions provided for log-based protocols must maintain these messages as long as they are not applied by all replicas.

The amnesia recovery in an outdated node has to be done before recovering the missed messages lost during its failure

period, as it is explained in [11] in order to guarantee the state consistency.

In section 4 it has been detailed that the amnesia can be manifested at two different levels: transport and replication. Therefore, the log-based recovery protocols must manage the information necessary for performing the recovery in an adequate way for both levels.

5.1 Transport Level

The amnesia at the transport level implies that received messages non-persistently stored are lost when the node crashes. If this occurs, the amnesia recovery could not be performed using a log-based approach. So the system must ensure these messages are available for recovery purposes by storing them in a persistent way.

The simplest, easiest option is that each node manages its own amnesia recovery information persitently. This storage of received messages must be kept until the respective owner transaction is either committed or aborted[2]. Moreover, if the messages must be maintained after its transaction commit, they must be marked in some way to remember that they have been already applied.

In addition, the permanent storing of a received message must be performed atomically with the group communications system message delivery. Under this principle, if the node is not able to persistently store the message, the group communication system (GCS) does not consider this message as delivered (thus ensuring that the delivery is coupled with the persistent storage).

Another approach, that we can think about, will consist in storing persistently messages in their replica senders, but this option presents several drawbacks. One of them implies that senders must maintain for each sent message a list of replicas that have acknowledged the commit of this message, being only possible to delete the message when all replicas have acknowledged it. Other important drawback is that the recovery process only can be performed when all senders are alive, a condition that can delay a lot of time the recovery process, situation highly discouraged. In spite of having several problems which discard its use in many transactional replicated systems, there are some replicated systems which their work way fits well with this solution. This is the case of hot passive on-processing [10] or voting replication systems [23].

5.2 Replication Level

The amnesia problem relates at the replication level to the fact 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. Thus, the information about the success of such commit must be also stored because it is needed by the recovery amnesia process in order to know which are the messages that must be applied (as we discussed previously). So, at the replicated system level, the problem is to know if a "commit" was successfully applied before the failure or not.

The mechanism for generating this information consists in maintaining some information about the last committed transaction for each open connection. Thus, when a transaction commit is performed in the replica, the system must write this information in a single atomic step, as part of the transaction itself. Thus, on commit success, the system contains the identity of this last committed transaction. Afterwards, when the connection is closed, the entry corresponding to this connection is erased.

Then this generated information is useful in the recovery process to check if messages marked as not committed have been really committed. When the node becomes alive again and starts its amnesia recovery it will check if there are messages marked as not committed, but its owner transaction is marked as committed in the replica.

A similar problem arises regarding the state associated to not committed messages (messages belonging to not yet committed transactions), since it is lost at the crash instant, since the replication system is also a transactional system. Therefore, these messages applied by the replication system but not committed must be again reapplied.

There are three possible scenarios where messages maintained as non-committed belong to a transaction whose owner connection does not have an entry in the table of committed transactions:

- *The node did not start to apply this connection and its transactions before the node crash.* Then, these messages must be applied in the amnesia recovery process.
- *The connection is closed before committing some of its related transactions (implying that these transactions will be aborted) but the node crashes before the system erases those messages.* Thus, all the messages will be reapplied in the amnesia recovery process but they will be aborted again as it has happened in the normal work way.
- *The transaction was really committed, the system has not marked yet its messages as committed, and it has already deleted its table connection entry.* This scenario must be avoided, because it will lead the system to apply twice a transaction if a recovery process is performed. So, when the "remove connection" message is applied, the removal of the corresponding entry in the system must be done after the protocol considers committed the transaction.

[2] i.e. discarding them on transaction aborts, or maintaining them for committed transactions *only* when failed nodes exist.

As a result, the amnesia recovery stage will just consist in reapplying the messages marked in the log recovery as "not applied" or "not committed", first checking against the replica if they were not really committed.

6 Replicated Systems Characteristics and Amnesia

Once in the previous section has been described how the amnesia problem manifests in a replicated system, in this section it will be described how their specific characteristics affect the way in which the replicated system must manage the amnesia problem. The characteristics set considered is:

Characteristics Set
Active, Update Everywhere or Passive Replication
Eager or Lazy Replication
Voting or non-Voting Technique
Constant or Linear Interaction
Communication Guarantees

Table 1. Characteristics Set.

In following subsections are detailed their effects on the amnesia support.

6.1 Active, Update Everywhere or Passive Replication

The adoption of an active [22] (*AR*), update everywhere [23] (*UER*) or passive [22] (*PR*) replication policy will not have any effect in how the replicated system must manage the amnesia problem. It is due to the fact that the amnesia phenomenon does not depend in how many replicas can serve client requests, but in how the changes are spread and stored at each replica.

6.2 Eager or Lazy Replication

The selection between eager (*ER*) or lazy (*LR*) replication [13] has a great effect in the amnesia phenomenon and its management.

In fact there are some configurations in the eager approach that are managing indirectly the amnesia phenomenon. This is the case of eager passive replication systems, also known as hot passive replication in [21], based on on-processing synchronization [10]. These systems block the client answer until the primary receives the process acknowledge from all secondary replicas. Therefore, the replicated system work way controls which slaves have committed which transactions, providing the necessary information for managing the amnesia problem at the replication level. The other eager passive replication techniques, on-delivery and on-reception synchrony [10] will not provide this intrinsic amnesia support, being necessary in these cases to adopt our amnesia support proposal.

Another case is eager update everywhere replication protocols as [18] which commit a transaction once all active replicas have locally committed it. This work way also provides the needed information to control the amnesia phenomenon at the replication level.

But not all eager solutions provide this intrinsic amnesia replication level support, only those whose transaction replication work includes the processing in all alive replicas into the transaction boundaries. Eager solutions that do not provide this behavior must adopt a system similar to our proposal either for replication and transport level.

This is the case of active replication protocols, which always work in an eager way. When these active replication protocols are based on total order broadcast they rely on a fully deterministic transactions execution as it is commented in [22]. Therefore, they do not provide a support for amnesia, making necessary to add an extra support.

The lazy replication approaches need always to adopt our amnesia support proposals or similar ones in order to support our assumed failure model. In these solutions, their work way does not provide the information needed for providing the amnesia support.

Anyway, it must be noticed that from a recovery point of view using a lazy replication technique will have a disruptive behavior in regard to the replicated consistency state because it will not be guaranteed at any time that all the information needed to reach the last consistent state in the system is available. Therefore, it is not ensured the success of these recovery processes. In fact, a priory some lazy replication protocols have had to implement some eager replication level to provide fault tolerance as it does [16].

Only lazy hot passive replicated systems, based on asynchronous and reliable communication, which propagate changes before answering to the client can either provide amnesia support using our amnesia support proposal, and ensure the replicated consistency correctness. But in order to provide this support, the client must remember which requests have not been already answered.

6.3 Voting or non-Voting technique

The use of a replicated system based on a voting termination technique (*VT*) [23] can provide to the recovery protocol information if a node has not committed a transaction before its failure, because the system needs to know the decisions adopted in all nodes. Thus, its use provides information that is necessary for overcoming the amnesia problem implying a partial implementation of our proposal. Whereas if a non-voting technique (*NVT*) is adopted the replicated system will have to adopt the whole system proposed in this

work.

6.4 Constant or Linear Interaction

The use of constant (*CI*) or linear (*LI*) interaction [23] in the replication protocols will not have great effect on the amnesia level support.

At this point it only must be remarked that the adoption of a linear replication approach will difficult the transaction messages management because the messages belonging to a transanction can be spread among several views, and they can also be interleaved with the messages owned by other transactions, forcing to persist messages belonging to non yet committed transactions with our approach. Obviously this problem only appears when the system adopts a log-based recovery strategy as it is commented in [11].

6.5 Communication Guarantees

Until this moment, it has been considered that all replicated systems use a total order broadcast (*TOB*) for propagating changes[3], with virtual synchrony [4, 7]. Therefore it is ensured that all replicas have delivered the same sequence of messages before a membership change occurs.

But what does it happen when we try to relax these communication guarantees? Will it affect the amnesia problem and the way it must be managed? In the following paragraphs we pretend to evaluate it.

First of all, we will consider communication primitives which provide reliable communication but which allow different delivery orders in different replicas, as it occurs when causal or FIFO (*R-FIFO*) delivery orders are used. The problem of these communication configurations is that they do not guarantee, in most cases, a consistent replicated state, allowing diverging state evolutions in different replicas, situation that can not be accepted. Only the intrinsic characteristics of some replicated systems allow to use more relaxed delivery orders without damaging the replicated consistent state. This is the case of *PR* (also known in the literature as primary copy systems) where *R-FIFO* is enough to guarantee the replicated state consistency, because in these systems there is only one sender. These relaxed orders can also be applied in *AR* or *UER* if they use *commutative updates* [13]. In these cases, where relaxing the total order delivery does not affect badly the replicated consistent state correctness our amnesia support proposal can be applied.

Another possible communications relaxation should be to use non-reliable communications, but under these conditions it is impossible to build any kind of fault tolerant system, because there is no guarantee about which messages have been received by which replicas. Thus, these systems must use a protocol which provides reliability. Therefore, the study of non-reliable communication primitives must not be considered.

[3]In passive replicated systems a FIFO order is enough.

7 Overhead

After detailing how different replication characteristics can affect the amnesia support in a transactional replicated system, we analyse the overhead introduced for providing amnesia support, detailing first the basic processing time (*BPT*) and after the processing time supporting amnesia (*PTSA*). Both processing times will be expressed for each propagated transaction in terms of spread time (st), persisting time (pt) and processing time (Pt).

The st depends on: the communication guarantees provided, messages number to spread for transaction and the message size. In regard to the communications, we consider two possibilities: *TOB* for *UER* and *R-FIFO* for *PR*. For *TOB* we assume the *fixed sequencer* in the broadcast-broadcast (*BB*) variant [12] implementation, which uses two reliable broadcasts for message propagation. On the other hand, the *R-FIFO* as it is considered for *PR* can be implemented using only one reliable broadcast for message propagation because there is only one sender. Then, if α is the cost of a reliable broadcast, the *TOB* has a cost of 2α for spreading a message while the *R-FIFO* cost is α.

For *CI* replication protocols, we will consider that only one message is broadcast per transaction, while *LI* protocols spread as messages as update operations contains the transaction, n. In this context, message size (S) is important because some *GCS* have a maximum message size bound to spread, S_M. Thus, messages greater than S_M must be spread sending $\lceil \frac{S}{S_M} \rceil$ messages. That can occur in *CI* protocols because they transfer the resulting transaction writeset, while messages in *LI* protocols are always smaller because they only transfer one update operation.

The message persisting time is expressed as $\beta + \gamma(S/k)$, where β is the upper bound time for write disk accesses, γ is the storing time of a block size message k, and S is the message size. It is considered that only one write disk access is needed for message. Then, the pt in *CI* protocols where only one message is broadcast is $\beta + \gamma(S/k)$, while in *LI* protocols is $n(\beta + \gamma)$, being n the number of update operations and assuming that their message size are always smaller than k.

For Pt, we consider the π value which depends on the replication system load and in the consistency provided by the replication protocol. Moreover, in *CI* protocols it will depend on the message size to apply (writeset) while on *LI* protocols will depend on the number of update operations and their complexity. Thus π is very difficult to model, anyway we consider that any transaction always fullfill the rule $Pt > pt$.

Several observations must be made for table 2. The

Configuration	*BPT*	*PTSA*
C1 - UER, ER, CI, NVT and TOB	$(\lceil \frac{S}{S_M} \rceil + 1)\alpha$	$(\lceil \frac{S}{S_M} \rceil + 1)\alpha + \beta + \gamma \frac{S}{k}$
C2 - UER, ER, LI, NVT and TOB	$2n\alpha$	$n(2\alpha + \beta + \gamma)$
C3 - UER, ER, CI, VT and TOB	$(\lceil \frac{S}{S_M} \rceil + 2)\alpha + \pi$	$(\lceil \frac{S}{S_M} \rceil + 2)\alpha + \pi$
C4 - PR (hot passive on-processing), ER, CI, and R-FIFO	$(\lceil \frac{S}{S_M} \rceil + 1)\alpha + \pi$	$(\lceil \frac{S}{S_M} \rceil + 1)\alpha + \pi$
C5 - PR (hot passive asynchronous), ER, CI, and R-FIFO	$\lceil \frac{S}{S_M} \rceil \alpha$	$\lceil \frac{S}{S_M} \rceil \alpha + \beta + \gamma \frac{S}{k}$

Table 2. Processing Times.

first one is that replication configurations which perform the transaction processing as core part of their propagation process, as C3 and C4, discard pt due to the rule $Pt > pt$. Therefore, π hides the $\beta + \gamma(S/k)$ value. Another consideration is that all configurations perform the message persistency process as an atomic step in the delivery process.

From table 2 can be concluded that *VT* configurations will not have any overhead for supporting amnesia, but from the beginning they have a BPT higher than the others, while the others present an overhead which depends on the persisting time, pt. Overhead which increases with the message size, S, in *CI* and the number of messages, n, in *LI*.

Another conclusion observed is that the overhead introduced for *R-FIFO* configurations is higher than for *TOB* configurations in percentage terms.

At this point it must be remarked the difficulty to compare *CI* configurations with the *LI* configuration. This is because the firsts depend on the size of the writeset to propagate, persist and/or apply, while the second depends on the number of update operations to apply and their complexity, depending then on the transactions characteristics. In a generic way, it can be said that if a transaction contains a lot of update operations but changes few data items the *CI* configuration will introduce less overhead. Contrarily, if the transaction has few update operations but changes a lot of data items the *LI* configuration will introduce less overhead. Thus, only when in our transactional system one of these transaction types predominate can be adopted accurately the configuration that best fits our necessities.

Anyway it must be pointed out that providing amnesia support can only alter the original order cost for the *LI* in regard to the others, depending on the transaction characteristics.

8 Related work

A wide range of proposals for solving the recovery problem [3] have been presented in the literature. Traditionally, among these recovery protocols, the ones oriented for replicated processes have adopted the fail-stop failure model, as [4]. In these cases, they transfer the whole data state to new or "reconnected" nodes, a good approach for systems with few data state. Whereas, for systems which manage large data amounts as, replicated databases, it has been largely recommended the crash recovery as it is proposed in [19, 6, 5, 1, 17] in order to minimize the recovery information to transfer. Few protocols as [20] have adopted the fail-stop failure model for large data state scenarios.

Among these last recovery protocols, some of them as [19, 6, 1] are version-based, while others as [19, 5] are log-based. First ones, which only transfer changed data, are typically useful for long-term outages whilst the latter, which send lost messages, present better performance for recovering short-term failures. Therefore, combining a version-based technique with a log-based one to construct a recovery framework has been proposed in several works as [19, 5] to improve the recovery features, choosing the recovery strategy that presents a lower cost each time an outdated node is detected.

Thus all these recovery protocols that have adopted the crash-recovery with partial amnesia failure model have to solve the amnesia problem. In fact, [5] proposes a solution for the amnesia phenomenon in log-based strategies based on logging received messages. This protocol is focused on replicated systems with the following characteristics: update everywhere, eager and using total order delivery.

In [17] it is presented a different way to deal with the amnesia problem in similar scenarios. In this paper the authors mix checkpointing with message transfer, in other words, they combine as recovery information to transfer a snapshot of the data state and the needed messages from the snapshot instant point. The combination of these two techniques 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 failure model are lost.

Other protocols as [6] avoid the amnesia problem in a version-based strategy. But the first thing to remark in this strategy is that the amnesia at the transport level must not be managed, because its protocols do not use messages to perform the recovery processes. In this case, the recovery processes tranfer to the outdated nodes the changed state from their crash time. Therefore, this strategy only needs to manage the information associated to the amnesia at the replication level. And the solution provided for the log-based amnesia recovery can also be applied in this case, without

the necessity of managing the messages. But it must be noticed that transferring only the state changed during the views where the node was failed will not include the necessary information in some cases. This problem is avoided if the replicated work uses a voting technique or if the information maintained to check the version is a version number, timestamping or the identifier of the last transaction which updated the data item.

9 Conclusions

In this paper, we detail how the crash-recovery with partial amnesia failure model presents some advantages from a recovery point of view for transactional replicated systems which manage large amounts of data state with respect to the most extended one, the fail-stop. And also, we explain its provided advantages for constructing fault tolerant systems under scenarios with huge data amounts.

But, it is noticed that the assumption of this failure model implies to deal with the amnesia problem. Then, this work analyzes how this phenomenon manifests at two different levels in replicated systems build on top of a total order broadcast communication system. And subsequently it is detailed how these systems can manage it, avoiding its related problems for log-based recovery strategies.

It is also studied in this paper, starting from how it is provided the amnesia support in our initial replicated system configuration, how the amnesia can be supported in other replicated systems depending on their characteristics.

Finally, it is performed an analysis of the overhead introduced for providing amnesia support in different replicated system configurations using a log-based recovery strategy.

References

[1] J. E. Armendáriz, F. D. Muñoz, H. Decker, J. R. Juárez, and J. R. G. de Mendívil. A protocol for reconciling recovery and high-availability in replicated databases. *21st International Symposium on Computer Information Sciences*, 2006.

[2] 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.

[3] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison Wesley, Reading, MA, EE.UU., 1987.

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

[5] F. Castro, J. Esparza, M. Ruiz, L. Irún, H. Decker, and F. 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. Muñoz. FOBr: A version-based recovery protocol for replicated databases. In *13th Euromicro PDP*, pages 306–313, Lugano, Sw, 2005. IEEE Computer Society.

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

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

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

[10] R. de Juan-Marín, H. Decker, and F. D. Muñoz-Escoí. An extended hot passive replication classification. In *2nd International Conference on Availability, Reliability and Security*. IEEE, 2007.

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

[12] X. Défago, A. Schiper, and P. Urbán. Total order broadcast and multicast algorithms: Taxonomy and survey. *ACM Comput. Surv.*, 36(4):372–421, 2004.

[13] J. Gray, P. Helland, P. O'Neil, and D. Shasha. The dangers of replication and a solution. In *ACM SIGMOD International Conference on Management of Data*, pages 173–182, 1996.

[14] V. Hadzilacos and S. Toueg. Fault-tolerant broadcasts and related problems. In S. Mullender, editor, *Distributed Systems*, chapter 5, pages 97–145. ACM Press, 2nd edition, 1993.

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

[16] L. Irún-Briz. *Implementable Models for Replicated and Fault-Tolerant Geographically Distributed DataBases. Consistency Management for GlobData*. PhD thesis, Polytechnic University of Valencia, 2003.

[17] R. Jiménez-Peris, M. Patiño-Martínez, and G. Alonso. Non-intrusive, parallel recovery of replicated data. In *SRDS*, pages 150–159. IEEE Computer Society, 2002.

[18] B. Kemme and G. Alonso. A new approach to developing and implementing eager database replication protocols. *ACM Trans. Database Syst.*, 25(3):333–379, 2000.

[19] 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. IEEE Computer Society.

[20] E. Lau and S. Madden. An integrated approach to recovery and high availability in an updatable, distributed data warehouse. In *VLDB*, pages 703–714, 2006.

[21] A. J. Wellings and A. Burns. Programming replicated systems in ada 95. *The Computer Journal, 39(5):361-373*, 1996.

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

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