A Cost Analysis of Solving the Amnesia Problems

R. de Juan, L. Irún, F. D. Muñoz

Instituto Tecnológico de Informática

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

Technical Report TR-ITI-ITE-08/08

A Cost Analysis of Solving the Amnesia Problems

R. de Juan, L. Irún, F. D. Muñoz

Instituto Tecnológico de Informática

Technical Report TR-ITI-ITE-08/08

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

April 21, 2008

Abstract

Replicated systems in order to provide more efficient recovery processes have adopted the crash-recovery with partial amnesia failure model. But, this assumption forces to deal with the amnesia phenomenon which implies that not committed state is lost at crash time. So, if this phenomenon is not accurately managed in recovery processes it can lead to state inconsistencies in the replicate state. A general solution that consists in persisting messages atomically in the delivery process has been proposed for overcoming this problem, demonstrating also its validity. But, its use implies a cost: the overhead introduced for persisting messages atomically in the delivery process. This paper analyzes this overhead simulating the proposed solution for a transactional replication protocol based on certification and demonstrates how this overhead can be minimized using solid memories, making it acceptable.

1 Introduction

Transactional replication has become a key factor in providing fault-tolerant, highly available information systems, providing at the same time good performance levels. Performance can be improved forwarding client requests to their closest replica [13, 18], or by using load-balancing algorithms [1, 10, 19]. And fault tolerance and high availability are reached forwarding such requests to non-failed nodes in a transparent way. In last years these techniques have been making use of Group Communication System (GCS for short) [4] as it is detailed in [22], because they provide communication primitives and membership mechanisms, which are very important in replicated systems.

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

In regard to the progress condition, most of these works have adopted the majority of living replicas progress condition –primary partition [4].

For managing the membership events they make use of *recovery components* which deal with these situations attending the failure model adopted. The most commonly used failure model in transactional systems has been the halt-crash failure model [5] –similar to the fail-stop proposed by [21]. It is used due to its simplicity; crashed replicas are discarded and substituted by new ones.

But, this simplicity implies a high cost in the recovery process because it is necessary to transfer the whole state to the replicas that substitute the crashed ones, being very inefficient when talking about systems managing large states. In order to provide more efficient recovery processes protocols have adopted the *crash-recovery with partial amnesia*, as defined in [5]. When this failure model is adopted crashed replicas can recover maintaining some of its previous state –persisted one– reducing then the amount of information to transfer in the recovery process being therefore more efficient.

The problem of adopting the *crash-recovery with partial amnesia* is that in can lead to a consistency state problem as demonstrated in [8] if the recovery process does not manage accurately the amnesia phenomenon in the recovering replica –it is possible that not all delivered messages have been correctly processed before crashing. Moreover, another consistency state problem appears when the amnesia phenomenon is combined with a specific sequence of membership change events tolerated by the primary partition progress condition as shows [6], which will force to stop the work of the recovery process because the consistency of the replicated state is not guaranteed.

For this reason, both papers proposed a generic solution for overcoming this inconsistencies consisting in forcing each replica to persist messages atomically in the delivery process. Obviously, this solution for overcoming these consistency problems introduces some overhead in the work performance of the replication protocol when it is applied. But, this overhead is not always constant. It can vary depending on different characteristics of the replicated system. Therefore, the goal of this paper is to present how the overhead behaves when different characteristics of the replicated system vary: workload, message size, number of replicas, message processing time in replicas,... simulating it in a replication protocol based on certification [23]. Also the simulation considers different storage engines –which provide different transfer speeds– in order to compare them, noticing how with solid memories it is maintained in acceptable levels.

This article is structured as follows. First, section 2 presents the basic system model considered. On the sequel, section 3 outlines the basic solution while section 4 presents the overheads introduced. Later, section 5 introduces the simulation that has been performed for analysing the behaviour of the overhead. This behaviour is subsequently explained in section 6. Related work is detailed in section 7 while section 8 concludes the paper.

2 System Model

A replicated transactional system composed by several replicas –each replica in a different node– is considered. The nodes belong to a partially synchronous distributed system: their clocks are not synchronized but the message transmission time is bounded. The state is fully replicated in each node, so each replica has a copy of the whole state. State changes are performed between the boundaries of transactions.

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

The GCS includes a group membership service, who *knows* in advance the identity of all potential system nodes. These nodes can join the group and leave it either explicitly or implicitly by crashing. The *GCS* provides *Virtual Synchrony*[2] 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*.

3 Amnesia Solution Outline

The crash-recovery with partial amnesia failure model implies that when a crashed node reconnects it may not remember exactly which was its last state before crashing. In other words, at reconnection time it expects to have a theoretic last state which in fact is different to its real state due to amnesia. That happens because the node has delivered some messages M^F but their associated updates have not been committed –and therefore persisted– before the node crash. This can happen for example due to workload reasons in this replica. Then all this non persisted state is lost at crash time. So, if the recovery process does not consider this forgotten state in the replica being recovered state inconsistencies could arise between the state reached in this replica after the recovery process and the state in other replicas.

The basic properties that must fulfil a recovery protocol which does not transfer the whole state to overcome this problem were presented in [7] and can be stated as follows:

- *Property 1*: a crashed replica must remember its last committed transaction, knowing therefore its last committed message;
- *Property 2*: the replicated system must maintain and provide a way for obtaining the updates associated to forgotten messages M^F, knowing then in an implicit way the changes attached to forgotten transactions.

Moreover, if the replicas that ensure the continuity between two consecutive majority partitions suffer from amnesia it can imply that the state from which will start to work the new majority partition would be different to the last state reached in previous working view. This will imply therefore that the majority partition progress condition can not ensure the consistency continuity between consecutive working partitions under this specific sequence of events.

As it was demonstrated in [6] only modifying partially the second property presented in [7] –making it more restrictive– was enough to overcome this problem too:

- *Property 2′*: each replica must maintain and provide a way for obtaining the forgotten messages –or transactions– or associated updates, instead of trusting in "the replicated system".

The basic solution for solving the two inconsistency problems related to the amnesia phenomenon described in papers [8], [6] consists in: keeping track of the last committed message and persisting the broadcast messages atomically in the delivery process at each replica. Therefore properties *Property 1*, *Property 2′* are fulfilled.

This solution allows in the recovery process of crashed replicas to check which messages delivered before crashing have not been correctly processed in the crashed replica. So, those that have not been correctly processed before crashing can be applied in this recovery step overcoming in this way the amnesia phenomenon at each replica. Then both previous problems are solved in a straightforward way.

4 Amnesia Solution Overheads

Obviously, forcing the system to persist the messages in the delivery process implies to introduce an overhead in the overall performance. As a minimum, this overhead would be equal to the cost of persisting in physical storage the messages. This cost will therefore depend on the size of the message to store and in the transfer write rate of the used device. Basically, it can be said that the faster the storage engine is, the better the system will behave.

But in some situations this overhead will be higher than the cost of persisting the message. This happens when the persisting process becomes a bottleneck, in other words, when the rate of incoming messages to persist is higher than the speed at which the storage engine persists them.

The messages that must be persisted in the system for overcoming the amnesia problems are the update transactions –assuming a message per transaction– that must be broadcast –update transactions that have not been aborted locally. Therefore, the rate of incoming messages depends on the transactions per second workload that can process the replicated system, the % of read transactions of this workload and the local abort rate of update transactions. High workloads, low % of read transactions and low rates of update transactions locally aborted can convert the persisting process in a bottleneck.

It must be noticed that from a persisting point of view high workloads and low rates of local aborts are bad news but from a replication point of view are good ones. So, the ideal storing engine must be able to deal with high workloads and really low local aborts of update transactions.

Forcing the system to persist messages atomically in the delivery process implies that the replicas can not deliver the message until they know that all living replicas have persisted the message. Thus, it is necessary that replicas exchange messages in order to notify themselves that they have persisted the message before delivering them. Obviously, this extra messages round implies another overhead in the system. But, it must be noticed that this message is really small because is simply a control message. Moreover, as some GCS use internally an ack for confirming the reception of the message, this ack can be delayed in order to inform also that the message has been persisted.

5 Simulation

For observing how behaves the proposed solution it has been simulated a transactional replication protocol based on certification for a wholly replicated database. It works in an update everywhere approach so all replicas can serve client requests; read transactions are processed only locally while update transactions are broadcast to all nodes –ROWAA approach– using a single message per transaction –constant interaction. As it has been said a read transaction is only processed locally so at commit time if there are no conflicts the node serving the transaction commits it and answers to the client. While an update transaction is first processed locally in the node that is serving the request and at commit time –if it has not been aborted locally– is broadcast using total order to all nodes. In this case it is broadcast both the writeset –WS– and the readset –RS– in order to provide serializability.

The total atomic broadcast is implemented using a sequencer with two reliable broadcasts. In the first broadcast the sender spreads the message to all nodes, the second broadcast –a small control message– is used by the sequencer to notify the delivery order. A reliable point to point communication is used by the nodes in order to notify that they have persisted the message to the other nodes. Note, however, that such additional round –to the two ones used by the basic atomic broadcast considered– only uses small control messages; i.e., they do not carry the request or update-propagation contents of the original message, so their size is small and such message round can be completed faster than the contents-propagation one in the regular case (Considering, e.g., that in database replication protocols the broadcast messages propagate transaction writesets and their size may be as big as several hundred KB). The simulation has used network values appropriate for a 1 Gbps LAN.

In table 1 are listed the values assumed for different parameters in the simulation. The value of some parameters has been varied in order to analyse how behaves the solution.

Parameter	*Value*
Database size	100000 items
Transaction processing time in serving replica	50 ms
Transaction processing time in other replicas	20 ms
Net average delay	0.15 ms
Workload	30, 100, 300 and 500 TPS
Number of nodes	3, 5, 7, 9, 11 and 21
Total order broadcast message size	100, 200, 300 and 500 KB
% of read transactions	0, 10 and 20

Table 1: Simulation values.

Moreover, as the solution consists in persisting the messages broadcast in total order by the replication protocol two different secondary storage systems have been considered. On one hand a hard disk drive of 7200 r.p.m. (a.k.a. HDD) as basic storage system commonly found in low- and middle-range personal computers. On the other hand a solid state disk based on flash memory. There are disks of this kind able to store 16 GB and with a transfer rate of 90 MB/s for less than 400 USD (December 2007 prices). Table 2 summarizes the main performance-related figures of both disks. In the simulation, we consider that there is a disk entirely dedicated to GCS log management, apart from the one being used by the DBMS.

Hard Disk Drive	
Parameter	*Value*
Positioning disk average time	5.5 ms
Rotation disk average time	4.16 ms
Write transfer rate	40 MB/s
Flash Memory	
Parameter	*Value*
Write transfer rate	90 MB/s

Table 2: Storing system values.

The tested configurations in the simulation are the result of combining the workload, number of nodes, the message size and the rate of read transactions. The experiment measures the transaction completion time and consisted in simulating each configuration with each considered storing engine. An additional test without persisting messages has been performed for each configuration. This last one is used as the base level for comparison purposes.

Each test consisted in completing 40000 transactions in the whole system. In the simulation it has been forced that there are not local aborts –so all update transactions must be broadcast– because this is the worst scenario from a persisting point of view. Once the simulation has completed all these transactions, the average results for committed transactions are calculated.

6 Results

For explaining the simulation results, different graphics have been prepared. They present the results without persisting –basic–, persisting in HDD –HDD– and persisting in flash memory –flash–. The percentage of read transactions used in these figures is 10 %, corresponding each one to 3, 9 and 21 replicas respectively.

Figures 1, 3 and 5 show the average completion time and persistence overhead in absolute values with two different graphics. In these figures, (a) graphics show the total cost for basic, HDD and flash storing policies. But, as it is really difficult to see differences in them (b) graphics have been attached. Those show the difference of HDD an flash in regard to the basic one. In both graphics, MS stands for Message Size (in KB) whilst TPS gives the workload in transactions per second. The vertical axis gives times expressed in milliseconds.

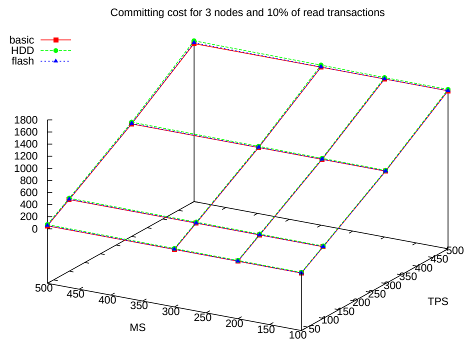
Figures 2, 4 and 6 depict the overhead in % introduced by the proposed solution for 3, 9 and 21 replicas respectively. As in previous figures, they show the results for the *basic* approach –without persisting–, for the *HDD* and *flash* storing engines. In this case the basic graphic only can be used as a reference point as it happened for (b) graphics in figures 1, 3 and 5.

Attending to these graphics on the sequel it will be explained how the proposed solution behaves attending to several characteristics in the replicated system simulated.

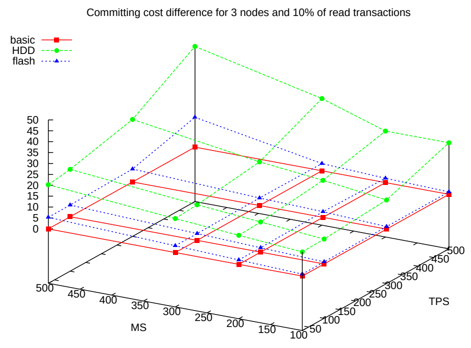
6.1 Workload

For transaction workloads it can be observed from figures in a general way that for higher workloads the proposed solution presents: higher overheads in absolute terms at least for HDD –(b) graphics of figures 1, 3 and 5–, but lower overheads in percentage terms –figures 2, 4 and 6 either when using HDD or flash.

First of all, someone can state that the persisting overhead in absolute terms must not increase with the workload if the size of the messages to store does not increase, contradicting then the trend observed for HDD in (b) graphics. This statement would be right if the write transfer speed of the storing engine was always high enough to not become a bottleneck when the



(a) Committing time (in ms)



(b) Persistence overhead (in ms)

Figure 1: Results for 3 replicas and 10% read-only Txs.

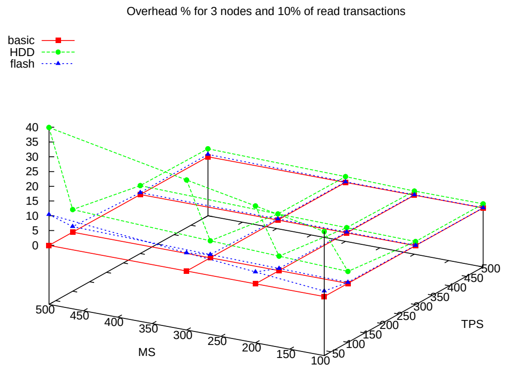


Figure 2: Overhead cost - 3 replicas and 10 % of read Tx.

workload rises. And (b) graphics show that; HDD becomes sooner a bottleneck –increasing then its overhead in absolute terms without increasing the message size– as it is the slowest storing engine tested, while flash memory –which has higher write transfers rates– does not show this trend at least for the tested workload ranges. As it can be inferred this trend manifests better for high message size values –worse cases– as shown HDD tendencies in (b) graphics.

It is also necessary to explain the apparently –at first glance– contradictory behaviour of the overhead in absolute –increase– and percentage –decrease– terms when the workload rises. The explanation for this phenomenon relates to the fact that the overall replicated system becomes a bottleneck for a certain workload, increasing the response time when the workload grows from this threshold level. From this assert, it can be stated that the tested replication protocol behaves in a bottleneck way because its time processing cost increases with workload when messages are not persisted as it can be seen in (a) graphics for the basic approach. Therefore, the persisting overhead decreases in percentage –even when it increases as it happens for HDD solution– when the workload rises because the basic cost of processing transactions grows more.

At this point, it can happen that the replicated system bottleneck hides or avoids to manifest in all its magnitude the persisting overhead. In fact, observing the (b) graphics of figures 1, 3 and 5 and figures 2, 4 and 6 it can be seen how when the number of replicas increases, the overhead increases both in absolute and relative terms –for the same workload and message size– because the effects of the replicated system bottleneck are lower.

Finally, it must be also specified that the persisting overhead depends on the workload of messages to persist which in spite of being related to is different from the workload of incoming requests to the replicated system. The former one depends on the incoming workload, the rate of update transactions of this workload and the rate of transactions aborted locally.

6.2 Message size

For this parameter there can not be observed in figures any unsurprising result; when the message size grows the persisting overhead grows both for HDD and flash storing engines. And, obviously, it manifests in a sharper way for the HDD storing engine than for the flash one, and already expected result as the second one has a higher write transfer.

Moreover, from figures it also can be deduced that the message size has an important effect in the probabilities that the persisting solution becomes a bottleneck. And it affects in the following way, when higher the message size the more probabilities has the persisting solution to become a bottleneck.

6.3 Number of replicas

The number of replicas affect to the persisting overhead in an indirect way. As it has been said previously, when the system has more replicas it can process higher workloads without becoming a bottleneck, then the persisting engine must manage higher workloads without the barrier provided by the replicated system when it acts as a bottleneck so its introduced overhead manifests more in percentage terms.

In regard to the bottleneck phenomenon, the persisting solution has more probabilities of becoming a bottleneck when the system has more replicas as it can manage usually higher workloads, forcing the storage engine to persist a higher rate of incoming messages.

6.4 Storing engines

From the figures it can be said that any storing engine introduces some overhead in the replication work, being lower when higher its write transfer rate is. So, the flash memory introduces lower overheads than the HDD solution for any tested replication configuration both in absolute and percentage terms. In fact, it can be seen how the flash storing engine never becomes a bottleneck as it happens with the HDD solution as graphics (b) from figures 1, 3 and 5 demonstrate for any of the simulated workloads.

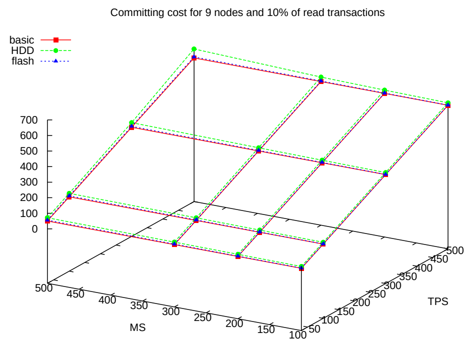
It must be said, that having a fast soring engine is not only interesting because it introduces lower overhead but also for decreasing the probabilities of becoming a bottleneck as it is seen in (b) graphics.

6.5 Other parameters

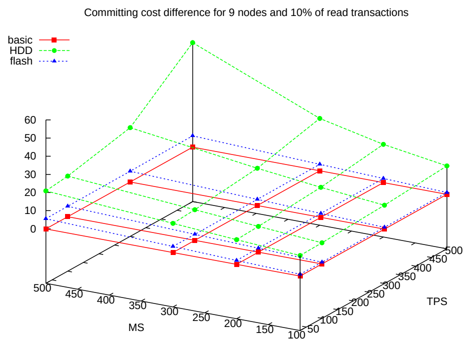
In the simulation other parameters have been considered: % of read transactions and % of local aborts. No graphics associated to them have been included due to space reasons. But, it can be said that the observed evolution was the expected one. The overhead decreased either when the % of read transactions increased or when the % of locally aborted transactions increased, because it implied less messages to persist.

6.6 Summary

From the results obtained in the simulation different conclusions can be stated. The first and obvious one is that any persisting solution introduces some overhead in the system. It is also important to say that this introduced overhead depends on the combination of several static and dynamic characteristics of the replicated system. This overhead in absolute terms increases with the workload, the message size, and % of write transactions, while in % the worst cases are those in which the system is able to process in a fast way the transactions.



(a) Committing time (in ms)



(b) Persistence overhead (in ms)

Figure 3: Results for 9 replicas and 10% read-only Txs.

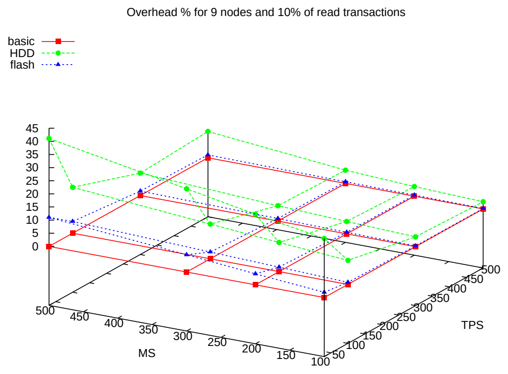
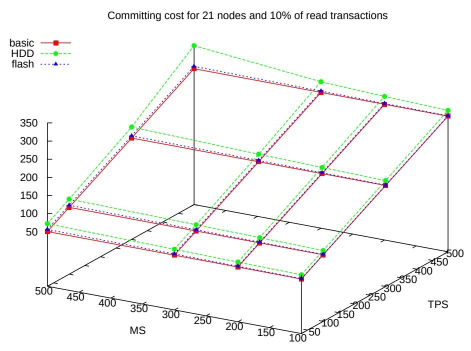


Figure 4: Overhead cost in % - 9 replicas and 10 % of read Tx.

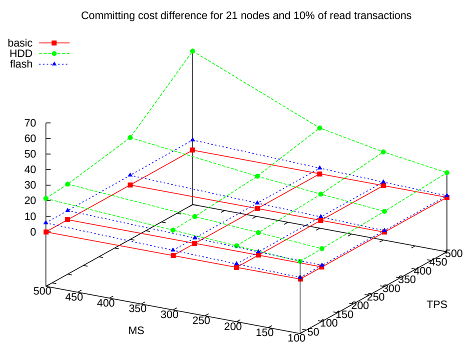
Overhead that would be lower when faster is the used storage engine. And that the worst overhead cases appear when the persisting process becomes a bottleneck because in this case the overhead introduced by the persisting process is higher than the a priory expected cost of persisting –considering the message size and the write transfer speed. Evidently, as it has been said the phenomenon of becoming a bottleneck is more probable for slow storing engines. At this point it also must be remarked how the flash memory does not only become a bottleneck but even it maintains the introduced overhead in a low level range for usual workloads.

Moreover, it has been seen how the worst conditions from the persisting point of view are a system processing high workloads, with great message sizes, low % of read transactions and slow storing engines. In regard to the message size, it must be advanced that the replicated protocols which propagate the writeset and readset would behave worse from the storing point of view than those transferring the operations to perform. Moreover, in the particular case of certification it would behave better if instead of using serializability it would have used snapshot isolation, because in this last case the readset must not be transferred by the replication protocol.

Another obtained conclusion from the simulation is that sometimes when the replicated system gets overloaded it can hide or decrease the phenomenon of saturation in the persisting process. This conclusion can be converted in a guideline to follow when designing a replicated system and this rule will state that the capacity of processing transactions by the replicated system and the capacity of persisting messages by this replicated system must evolve parallely. The idea is that having a replicated system that can manage high workloads is not worthy if the storing process acts as a bottleneck because the latter will decrease the overall performance. And the



(a) Committing time (in ms)



(b) Persistence overhead (in ms)

Figure 5: Results for 21 replicas and 10% read-only Txs.

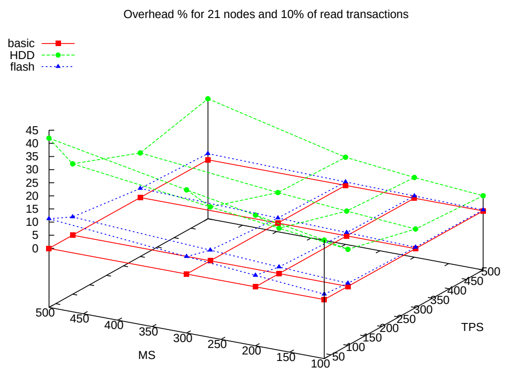


Figure 6: Overhead cost in % - 21 replicas and 10 % of read Tx.

rule also works in the other way, so investing money in fast storing engines is not profitable if the replicated system can not deal with high workloads. In this last case, it only would be interesting if the size of message transactions is quite large. Other parameter to consider is the average time cost of processing transactions in nodes because when higher it is the replicated system saturates sooner, hiding therefore the overhead introduced by the persisting engines.

It must be remarked, that all these results have been obtained with a 0% aborts, so all the incoming update requests to the replicated system must be broadcast among the replicas. If the abort rate in the serving replica is higher the number of transactions to broadcast decreases, diminishing then the demand to the storing process, reducing therefore the probabilities of reaching the saturation point for the persisting process.

7 Related Work

There are many replicated database works in the literature that give some numeric results about how they behave. The problem is that most of them have not considered the amnesia phenomenon and its possible associated consistency problems that arise when working under specific conditions. So, it is really difficult to compare the results obtained with this simulation with these other works.

Among all these works it is interesting to point out the following two ones [13] and [3]. The first one because it gives the results associated for the same type of replication protocol. While the second one because the way it implements the total order broadcast can avoid in an straightforward way the problems associated to the amnesia phenomenon.

In [13] authors show the results of a replicated system based on a certification replication protocol. It uses also an update everywhere approach broadcasting only update transactions in a constant interaction way –single message per transaction– through a total order communication primitive provided by a GCS. The problem is that these results can not be compared with the ones obtained here, because in their case they provided Snapshot Isolation level instead of the Serializable one used in the simulation. Moreover, they do not specify the GCS they have used and the guarantees it provides from an amnesia point of view.

Authors of Sprint Middleware [3] provide some results of their solution which main characteristic is to take advantage of being an in memory database –IMDB. This solution uses as atomic broadcast protocol the Paxos [12] protocol. This Paxos protocol forces the system to persist messages in order to establish the total order, therefore it can avoid in a straightforward way the basic amnesia phenomenon. But, these results can not be compared with the simulation results because their system is a partial replicated system.

Other interesting works are [20] and [15] papers that also try to provide a more accurate approach for replicated systems adopting the crash-recovery with partial amnesia failure-model.

Authors of [20] make mandatory in their proposal that for any consensus round each replica has to persist its proposed messages –the step before agreeing the order for this round. Later, the basic approach allowed in the recovery process of a crashed node to replay all the consensus rounds avoiding rebuilding therefore the agreed messages queue. This approach allows to overcome the basic amnesia problem presented in paper [8], and also affords the amnesia problem presented in paper [6] because the whole queue of delivered messages is rebuilt in the recovering node. But this also introduces an overhead, since messages should be persisted.

In [15] authors considered that the existing specifications of atomic broadcast were not completely satisfactory in the crash-recovery model. To overcome this problem they build their atomic broadcast specification using three primitives: *abcast*, *adeliver* and *commit* –new one–. This specification also divided the process state into two different states: the application state and the state at the atomic broadcast protocol. They proposed two variants: an uniform –more-consistent– and a non-uniform –more efficient– one as they show in their obtained results. The first one was forced to access to stable storage at the beginning of each consensus round, while the other only at commit time. Therefore, the first one –uniform– could replay the consensus rounds not persisted in the checkpointed state avoiding the amnesia phenomenon, while the second one only can be used for these applications that can afford losing uncommitted parts –in this case the amnesia is not completely avoided. Anyway, it must be pointed out that neither of these two proposals could manage the problem of combining amnesia with the majority progress condition. This is due to the fact that the replay phase –in the uniform solution– needs that all originally proposed values in the consensus round to be replayed must be available –condition that is not fulfilled.

It must be noticed that [14] demonstrated that consensus can be solved without accessing stable storage, implying that replicated systems based on total order broadcast can be built without persisting messages. Obviously, this approach does not introduce any overhead in the replicated system. The problem is that this way of working requires that the always–up processes must be larger than bad processes. Therefore, it can not manage accurately the problem that appears when the amnesia phenomenon is combined with the majority progress condition [6].

In regard to commercial databases that provide replication or clustered solutions, usually they provide different solutions for managing node crashes and node recoveries. But usually these solutions consist in a trade off between consistency and the overhead introduced. This is the case of MySQL [17], in order to avoid consistency problems in the recovery process, the [17] proposes to set the master binary log synchronization parameter to the value which implies the highest rate of binary log flush, decreasing then the probabilities of not having flushed the binary log at crash time but without ensuring it at all. So, the basic idea is that the flush frequency must be increased in order to be more consistent, but this also increases the overhead.

Finally, this simulation can be seen as a continuation of the work started in [9], measuring the overhead introduced when forcing the system to persist atomically the total ordered broadcast messages in the delivery process. Moreover, this solution fulfills completely the properties defined by [16].

8 Conclusions

As it has been seen in this paper the overhead introduced by the proposed solution varies with the number of nodes, workload, % of local aborts, % of read transactions, average time for processing transactions and last but not least message size. Thus, if this solution must be applied a proper study of the most common values of these variables in the replicated system must be performed in order to determine the minimum write transfer speed in order to avoid the saturation of the storing process.

Anyway, for minimizing the overhead cost the best option is to use the fastest storing engines. As it has been seen in the results the solid memory is a good option for keeping the overhead introduced by the proposed solution in low values in almost all the cases making it acceptable. Moreover, the continuous increase of transfer speeds both for flash memories and HDD will minimize this overhead in the future. Otherwise for systems managing big messages and high workloads there are other solutions as *ioDrive* [11] which provides 600 MB/s write transfer rates.

9 Acknowledgements

This work has been partially supported by EU FEDER and Spanish MEC under grant TIN2006-14738-C02 and by IMPIVA under grant IMIDIC/2007/68.

References

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

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

[3] Lásaro Camargos, Fernando Pedone, and Marcin Wieloch. Sprint: a middleware for high-performance transaction processing. *SIGOPS Oper. Syst. Rev.*, 41(3):385–398, 2007.

[4] Gregory V. Chockler, Idit Keidar, and Roman Vitenberg. Group communication specifications: A comprehensive study. *ACM Computing Surveys*, 33(4):427–469, December 2001.

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

[6] Rubén de Juan-Marín, Luis Irún Briz, and Francesc D. Muñoz-Escoí. Ensuring Progress in Amnesiac Replicated Systems. In *3rd International Conference on Availability, Reliability and Security, March 2008, Barcelona, Spain*, March 2008.

[7] Rubén de Juan-Marín, Luis Héctor García-Muñoz, Jose Enrique Armnedáriz-Íñigo, and Francesc D. Muñoz-Escoí. Reviewing Amnesia Support in Database Recovery Protocols. In *9th International Symposium on Distributed Objects, Middleware and Applications, Vilamoura, Portugal*. Springer, November 2007. Accepted for publication.

[8] 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 2007), Faro, Portugal*, 2007.

[9] Rubén de Juan-Marín, María Idoia Ruiz-Fuertes, Jerónimo Pla-Civera, Luis Héctor García-Muñoz, and Francesc D. Muñoz-Escoí. On Optimizing Certification-Based Database Recovery Supporting Amnesia. In *XV Jornadas de Concurrencia y Sistemas Distribuidos (JCSD 07), Torremolinos, Spain*, pages 145–157, June 2007.

[10] 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.

[11] Fusionio. iodrive, 2007. Accessible in URL: *http://www.fusionio.com*.

[12] Leslie Lamport. The part-time parliament. *ACM Transanctions on Computer Systems*, 16(2):133–169, 1998.

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

[14] Sam Tueg Marcos Kawazoe Aguilera, Wei Chen. Failure detection and consensus in the crash recovery model. In *DISC*, pages 231–245, 1998.

[15] Sergio Mena and André Schiper. A new look at atomic broadcast in the asynchronous crash-recovery model. In *SRDS*, pages 202–214. IEEE-CS Press, 2005.

[16] Francesc D. Muñoz-Escoí, Rubén de Juan-Marín, José Enrique Armendáriz-Iñigo, and José Ramón González de Mendívil. Persistent Logical Synchrony. Technical report, Technical Report ITI-ITE-08/02, Instituto Tecnológico de Informática, January 2008.

[17] MySQL AB. Mysql 5.1 reference manual, 2006. Accessible in URL: *http://dev.mysql.com/doc/*.

[18] 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.

[19] 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.

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

[21] Fred B. Schneider. Byzantine generals in action: implementing fail-stop processors. *ACM Trans. Comput. Syst.*, 2(2):145–154, 1984.

[22] Matthias Wiesmann, Fernando Pedone, André Schiper, Bettina Kemme, and Gustavo Alonso. Understanding replication in databases and distributed systems. In *ICDCS '00: Proceedings of the The 20th International Conference on Distributed Computing Systems (ICDCS 2000)*, page 464, Washington, DC, USA, 2000. IEEE Computer Society.

[23] 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.