Beyond 1-Safety and 2-Safety for replicated databases: Group-Safety

Matthias Wiesmann and André Schiper
École Polytechnique Fédérale de Lausanne (EPFL)
CH-1015 Lausanne
e-mail: Matthias.Wiesmann@epfl.ch & Andre.Schiper@epfl.ch

Abstract

In this paper, we study the safety guarantees of group communication-based database replication techniques. We show that there is a model mismatch between group communication and database, and because of this, classical group communication systems cannot be used to build 2-safe database replication. We propose a new group communication primitive called *end-to-end atomic broadcast* that solves the problem, i.e., can be used to implement 2-safe database replication. We also introduce a new safety criterion, called *group-safety*, that has advantages both over 1-safety and 2-safety. Experimental results show the gain of efficiency of group-safety over lazy replication, which ensures only 1-safety.

Keywords: database replication, group communication, database safety, safety criterion, total order broadcast, performance analysis, lazy replication

1 Introduction

Database systems represent an important aspect of any IT infrastructure and as such require high availability. Software-based database replication is an interesting option because it promises increased availability at low cost. Traditional database replication is usually presented as a trade-off between performance and consistency [12], i.e., between eager and lazy replication. Eager replication, based on an atomic commitment protocol, is slow and deadlock prone. Lazy replication, which foregoes the atomic commitment protocol, can introduce inconsistencies, even in the absence of failures.

However, eager replication does not need to be based on atomic commitment. A different approach, which relies on group communication primitives to abstract the network functionality, has been proposed in [30, 1], These techniques typically use an atomic broadcast primitive (also called total order broadcast) to deliver and order transactions in the same serial order on all replicas, and offer an answer to many problems of eager replication without the drawbacks of lazy replication: they offer good performance [21], use the network more efficiently [33] and also reduce the number of deadlocks [17].

Conceptually, group communication-based data replication systems are built by combining two modules: (1) a database module, which handles transactions and (2) a group communication module, which handles communication. When combined, these two module result in a replicated database. However, the two modules assume different failure models, which means that the failure semantics of the resulting system are unclear.

In this paper, we examine the fault tolerance guarantees offered by database replication techniques based on group communication. The model mismatch between group communication and database systems comes from the fact that they originate from two different communities. We explore this mismatch from two point of views: from the database point of view, and from the distributed system point of view. Database replication is usually specified with the *1-safety* and *2-safety* criteria. The first offers good performance, the second strong safety. However, group communication as currently specified, cannot be used to implement 2-safe database replication. The paper shows how this can be corrected. Moreover, we show that the 1-safety and 2-safety criteria can advantageously be replaced by a new safety criterion, which we call *group-safety*. Simulation result show that group-safe database replication leads to improved performance over 1-safety, while at the same time offering stronger guarantees.

The rest of the paper is structured as follows. Section 2 presents the model for the database system and for group communication, and explains the use of group communication (more specifically atomic broadcast) for database replication. Section 3 shows that this solution, based on current specification of atomic broadcast, cannot be 2-safe. Section 4 proposes a new specification for atomic broadcast, in order to achieve 2-safety. Section 5 defines the new safety criterion called *group-safety*. Section 6 compares the efficiency of group-safe replication and 1-safe replication by simulation. Section 7 discusses the relationship between group-safe replication and lazy replication. Finally Section 8 presents related work and Section 9 concludes the paper.

2 Model and Definitions

We assume that the overall system is built from three components (Figure 1): the database component, the group communication component and the replicated database component. The first two components offer the infrastructure needed to build the application – in our case a replicated database. These two infrastructure components are accessed by the application, but they have no direct interaction with each other.

The replicated database component implements the actual replicated database and is described in Section 2.1. The database component contains all the facilities to store the data and execute transactions locally, and is described in Section 2.2. The group communication component offers broadcast primitives, in particular atomic broadcast, and is described in Section 2.3).

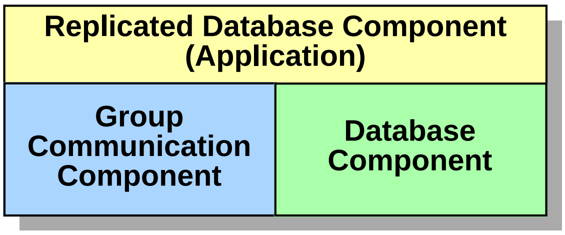


Figure 1: Architecture

2.1 Database Replication Component

The database replication component is modelled as follows. We assume a set of servers $S_1, \ldots, S_n$, and a fully replicated database $D = \{D_1 \ldots D_n\}$, where each server S_i holds a copy D_i of the database. We assume update-everywhere replication [12]: clients can submit transactions to any server S_i. Clients wanting to execute transaction t send it to one server S_d that will act as the *delegate* for this transaction: S_d is responsible for executing the transaction and sending back the results to the client.[1] The correctness criterion for the replicated database is one-copy serialisability: the system appears to the outside world as one single non-replicated database.

2.1.1 Replication Scheme

A detailed discussion of the different database replication techniques appears in [34]. Among these techniques, we consider those that use group communication, e.g., atomic broadcast (see Section 2.3.4). As a representative, we consider the technique called *update-everywhere, non-voting, single network interaction*. Figure 2 illustrates this technique.[2] The technique is called *non-voting* because there is no voting phase in the protocol to ensure that all servers commit or abort the transaction: this property is ensured by the atomic broadcast group communication primitive.

The processing of transaction t is done in the following way. The client C sends the transaction to the delegate server S_d. The delegate processes the transaction, and, if it contains some write operations, broadcasts the transaction to all servers using an atomic broadcast. All servers apply the writes according to delivery order of the atomic broadcast. Conflicts are detected deterministically and so, if a transaction needs to be aborted, it is aborted on all servers. Techniques that fit in this category are described in [19, 31, 27, 21, 3].

[1] The role of the delegate is conceptually the same than the primary server, simply any server can acts as a “primary”.

[2] However, the results in this paper apply as well to the other techniques in [34] based on group communication.

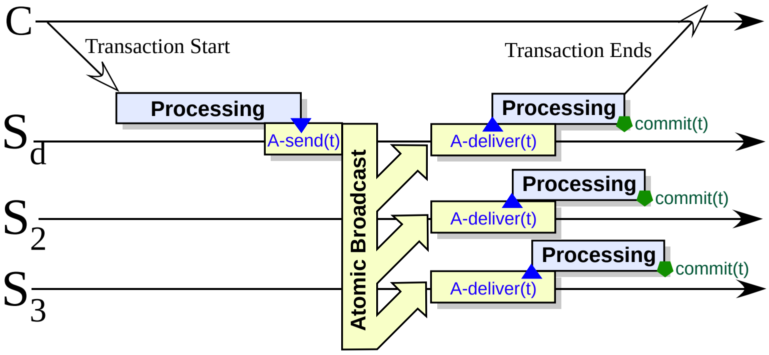


Figure 2: Non-Voting replication

2.1.2 Safety Criteria for Replicated Databases

There are three safety criteria for replicated database, called *1-safe*, *2-safe* and *very safe* [13]. When a client receives a message indicating that his transaction committed, it means different things depending on the safety criterion.

1-safe: If the technique is *1-safe*, when the client receives the notification of t's commit, then t has been logged and will eventually commit on the delegate server of t.

2-safe: If the technique is *2-safe*, when the client receives the notification of t's commit, then t is guaranteed to have been logged on *all available* servers, and thus will eventually commit on all available servers.

Very safe: If the technique is *very safe*, when the client receives the notification of t's commit, then t is guaranteed to have been logged on *all* servers, and thus will eventually commit on all servers.

Each safety criterion shows a different tradeoff between safety and availability: the more safe a system, the less available it is. *1-safe* replication ensures that transactions can be accepted and committed even if only one server is available: synchronisation between copies is done outside of the scope of the transaction's execution. So a transaction can commit on the delegate server even if all other servers are unavailable. On the other hand, *1-safe* replication schemes can lose transactions in case of a crash. A *very safe* system ensures that a transaction is committed on all servers, but this means that a single crash renders the system unavailable. This last criterion is not very practical and most systems are therefore 1-safe or 2-safe.

The distinction between *1-safe* and *2-safe* replication is important. If the technique is *1-safe*, transactions might get lost if one server crashes and another takes over, i.e., the durability part of the ACID properties is not ensured. If the technique is *2-safe*, no transaction can get lost, even if all servers crash.

2.2 Database Component

We assume a database component on each node of the system, and each database component with a full copy of the database. The database component executes local transactions and enforces the ACID properties (in particular serialisability) locally.

We also assume that the local database component offers all the facilities and guarantees needed by the database replication technique (see [34]), and has a mechanism to detect and handle transactions that are submitted multiple times, e.g., *testable transactions* [8].

2.3 Group Communication Component

Each server S_i hosts one *process* p_i, which implements the group communication component. While the database model is quite well established and agreed upon, there is a large variety of group communication models [11]. Considering the context of the paper, we mention two of them. The first model is the *dynamic crash no-recovery* model, which is assumed by most group communication implementations. The other model is the *static crash-recovery* model, which has been described in the literature, but has seen little use in actual group communication infrastructure.

2.3.1 Dynamic crash no-recovery model

The dynamic crash no-recovery model has been introduced in the Isis system [5], and is also sometimes called the *view based model*. In this model, the group is *dynamic*: processes can join and leave after the beginning of the computation. This is handled by a list, which contains the processes that are member of the group. The list is called the *view* of the group. The history of the group is represented as a sequence of views $v_0, \ldots v_m$, a new view being installed each time a process leaves or joins the group.

In this model, processes that crash do not recover. This does not prevent crashed processes from recovering. However, a process that recovers after a crash has to take a new identity before being able the rejoin the group. When a crashed process recovers in a new incarnation, it requests a view change to join the group again. During this view change, a *state transfer* occurs: the group communication system requests that one of the current members of the view makes a checkpoint, and this checkpoint is transferred to the joining process. Most current group-communication toolkits [5, 25, 32, 14, 4, 26] are based on this model or models that are similar.

Dynamic crash no-recovery group communication systems cannot tolerate the crash of *all* the members of a view. Depending on synchrony assumptions, if a view contains n processes, then at best $n - 1$ crashes can be tolerated.

2.3.2 Static crash recovery model

In the static crash recovery model, the group is *static*, i.e., no process can join the group after system initialisation. In this model, processes have access to stable storage, which allows them to save (part of) their state. So, crashed processes can recover, keep the same identity, and continue their computation. Most database system implement their atomic commitment protocol in this model.

While this model might seem natural, handling of recovery complicates the implementation. For this reason, in the context of group communication, this model has mostly been considered in papers [2, 10]. Practical issues, like recovery, are not well defined in this model (in [28] the recovery is log based). Because of the access to stable storage, static crash recovery group communication systems can tolerate the simultaneous crash of *all* the processes [2].

2.3.3 Process classes

In one system model, processes do not recover after a crash. In the other model, processes may recover after a crash, and possibly crash again, etc. Altogether this leads us to consider three classes of processes: (1) *green* processes, which never crash, (2) *yellow* processes, which might crash one or many times, but eventually stay forever up, and (3) *red* processes, which either crash forever, or are unstable (they crash and recover indefinitely). Figure 3 illustrates those three classes, along with the corresponding classes described by Aguilera *et al.* [2]. Our terminology, with the distinction between *green* and *yellow* processes, fits better the needs of this paper. In the dynamic crash no-recovery model processes are either green or red. In the static crash recovery model, processes may also be yellow.

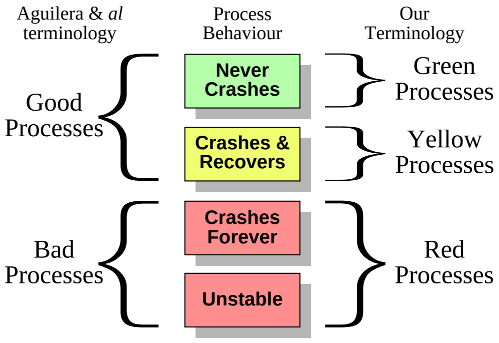


Figure 3: Process Classes

2.3.4 Atomic Broadcast

We consider that the group communication component offers an atomic broadcast primitive. Informally, atomic broadcast ensures that messages are delivered in the same order by all destination processes. Formally, atomic broadcast is defined by two primitives A-broadcast and A-deliver that satisfy the following properties:

Validity: If a process A-delivers m, then m was A-broadcast by some process.

Uniform Agreement: If a process A-delivers a message m, then all non-red processes eventually A-deliver m.

Uniform Integrity: For every message m, every process A-delivers m at most once.

Uniform Total Order: If two process p and q A-deliver two messages m and m', then p delivers m before m' if and only if q delivers m before m'.

In the following, we assume a system model where the atomic broadcast problem can be solved, e.g., the asynchronous system model with failure detectors [6, 2], or the synchronous system model [24].

2.4 Inter-Component Communications

Inter-component communication, and more specifically communication between the group communication component and the application component, is usually done using function calls. This leads to problems in case of a crash, since a message might have been delivered by the group communication component, but the application might not have processed it. To address this issue, we express the communication between the group communication layer and the application layer as *messages* (Figure 4). When the application executes A-send(m) (A stands for Atomic Broadcast), it sends the message $\langle m, \text{A-send}\rangle$ to the group communication layer. To deliver message m to the application (i.e execute A-deliver(m)), the group communication component sends the message $\langle m, \text{A-deliver}\rangle$ to the application.

So, we model the inter-component (intra-process) communication in the same way as inter-process communication. The main difference is that all components reside in the same process, and therefore fail together. This inter-layer communication is reliable (no message loss), except in case of a crash.

3 Group communication-based database replication is not 2-safe

In this section, we show that traditional group communication systems cannot be used to implement 2-safe replication. There are two reasons for this. The first problem that arises when trying to build a

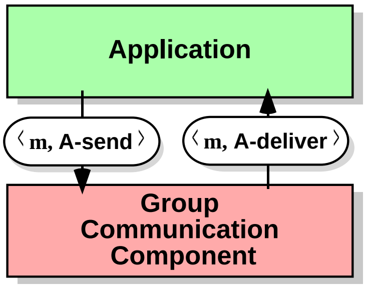


Figure 4: Message exchange for atomic broadcast

2-safe system is the number of crashes the system can tolerate. The 2-safety criterion imposes no bounds on the number of servers that may crash, but the dynamic crash no-recovery model does not tolerate the crash of all servers. This issue can be addressed by relying on the static crash recovery model.

The second problem is not linked to the model, but related to message delivery and recovery procedures. The core problem lies in the fact that the delivery of a message does not ensure the processing of that message [29]. Ignoring this fact can lead to incorrect recovery protocols [15]. Note that this second problem exists in all group communication toolkits (implemented in different models), which rely on the state transfer mechanism for recovery, including partionnable group communication systems.

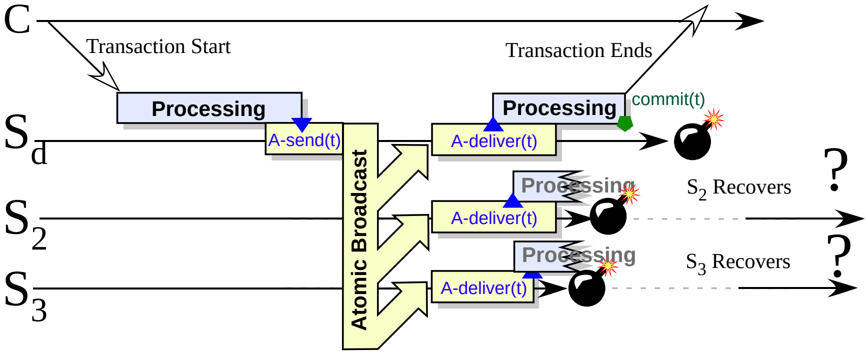


Figure 5: Unrecoverable failure scenario

To illustrate this problem, consider the scenario illustrated in Figure 5. Transaction t is submitted on the delegate server S_d. When t terminates, S_d sends a message m containing t to all replicas. The message m is sent using an atomic broadcast. The delegate S_d delivers m, the local database component locally logs and commits t and confirms the commit to the client: transaction t is committed in the database component of S_d. Then S_d crashes. All other replicas (S_2 and S_3) deliver m, i.e., the group communication components of S_1, S_2 and S_3 have done their job. Finally S_2 and S_3 crash (before committing t), and later recover (before S_d).

The system cannot rebuild a consistent state that includes t's changes. Servers S_2 and S_3 recover to the state of the database D that does not include the execution of t. Message m that contained t is not kept in any group communication component (it was delivered everywhere) and t was neither committed nor logged on servers S_2 and S_3: the technique is not 2-safe.

In this replication scheme, when a client is notified of the commit of transaction t, the only guarantee is that t was committed by the delegate S_d. The use of group communication does not ensure that t *will* commit on the other servers, but merely that the message m containing t will be *delivered* on all servers in the view. If those servers crash after the time of m's delivery and before t is actually committed or logged to disk, then transaction t is lost. In the scenario of Figure 5, if the recovery is based on the *state transfer mechanism* (Sect. 2.3.1), there is no available server that has a state containing t's changes. If recovery is log-based (Sect. 2.3.2), the group communication system cannot deliver again message m without violating the uniform integrity property (m cannot be delivered twice).

The problem lies in the lack of end-to-end guarantees of group communication systems described by Cheriton and Skeen [7]. Group communication systems enforce guarantees on the delivery of messages to the application, but offer no guarantees with respect to the application level: 2-safety is an application level guarantee.

4 Group Communication with end-to-end guarantees for 2-safe replication

We have shown in the previous section that it is impossible to implement a 2-safe database replication technique using a group communication toolkit that offers a traditional atomic broadcast. In order to build a 2-safe replication technique, we need to address the end-to-end issue.

4.1 Ad-hoc solution

One way to solve the problem would be to add more messages to the protocol: for instance each server could send a message signalling that t was effectively logged and will eventually commit. The delegate S_d would confirm the commit to the client after receiving those messages. While such an approach could work, it has two drawbacks. First the technique would have a higher latency because of the additional waiting: synchronisation between replicas is expensive [33]. But most importantly, this approach ruins the modularity of the architecture. The point of using a group communication system is to have all complex network protocols implemented by the group communication component and not to clutter the application with communication issues. If additional distributed protocols are implemented in an ad-hoc fashion inside the application, they risk being less efficient (some functionality of the group communication will be duplicated inside the application, in this case, acknowledgement messages), and less reliable (distributed system protocols tend to be complex; when implemented in an ad-hoc fashion, they might be incorrect).

4.2 End-To-End Atomic Broadcast

The problem of lost transactions appears when a crash occurs between the time a message is delivered and the time it is processed by the application. When a message is delivered to the application and the application is able to process the message, we say that the delivery of the message is *successful*. However, we cannot realistically prevent servers from crashing during the time interval between delivery and successful delivery. In the event of a crash, messages that were not successfully delivered must be delivered again: we have to make sure that all messages are eventually delivered successfully.

With current group communication primitives, there is no provision for specifying successful delivery. For this reason, we introduce a new inter-component message that acknowledges the end of processing of m (i.e., successful delivery of m). We denote this message $ack(m)$. The mechanism is similar to acknowledgement messages used in inter-process communications. Figure 6 shows the exchange of messages for an atomic broadcast. First, the application sends message m, represented by the inter-component message $\langle m, \textsf{A-send}\rangle$ to the group communication system. When the group communication components is about to deliver m, it sends the inter-component message $\langle m, \textsf{A-deliver}\rangle$. Once the application has processed message m, it sends the inter-component message $\langle m, \textsf{ack}\rangle$ to signal that m is *successfully delivered*.

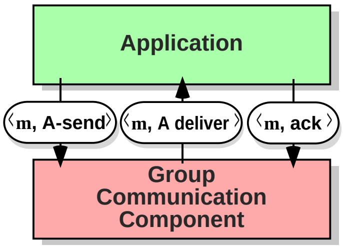


Figure 6: Messages exchange for with successful delivery

If a crash occurs, and the group communication component did not receive the message $\langle m, \textsf{ack}\rangle$, then $\langle m, \textsf{deliver}\rangle$ should be sent again to the application upon recovery. This requires the group communication component to log messages and to use log-based recovery (instead of checkpoint-based recovery). So after each crash, the group communication component "replays" all messages m such that

$\langle m, \textsf{ack}\rangle$ was not received from the application. By replaying messages, the group communication component ensures that, if the process is eventually forever up, i.e., non-red, then all messages will eventually be successfully delivered.

We call the new primitive *end-to-end atomic broadcast.* The specification of end-to-end atomic broadcast is similar to the specification of atomic broadcast in Section 2.3.4, except for (1) a new end-to-end property, and (2) a refined uniform integrity property: a message m might be delivered multiple times, but can only be delivered *successfully once*. A message m is said to be *successful delivered* when $ack(m)$ is received. The new properties are the following:

End-to-End: If a non-red process A-delivers a message m, then it eventually *successfully* A-delivers m.

Uniform Integrity For every message m, every process *successfully* A-delivers m at most once.

We assume a well-behaved application, that is, when the application receives message $\langle m, \textsf{A-deliver}\rangle$ from the group communication component, it sends $\langle m, \textsf{ack}\rangle$ as soon as possible.

4.3 2-Safe Database Replication using end-to-end atomic broadcast

2-safe database replication can be built using end-to-end atomic broadcast. The replication technique uses the end-to-end atomic broadcast instead of the ”classical” atomic broadcast. The only difference is that the replication technique must signal successful delivery, i.e., generate $ack(m)$. This happens when the transaction t contained in m is logged and is therefore guaranteed to commit. According to the specification of the end-to-end atomic broadcast primitive, every non-red process eventually successfully delivers m. The *testable transaction* abstraction described in Section 2.2 ensures that a transaction is commited at most once. So every process that is not permanently crashed or unstable eventually commits t exactly once: the technique is 2-safe.

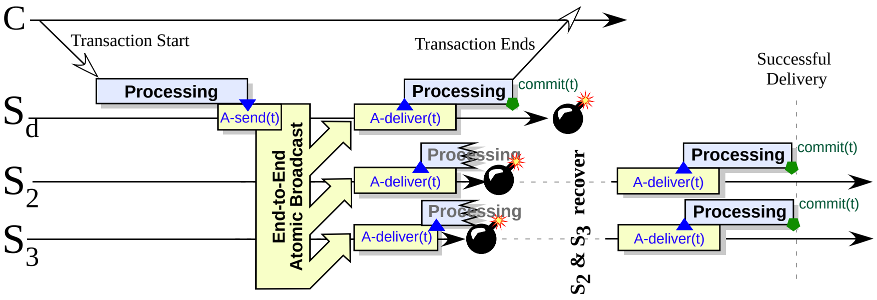


Figure 7: Recovery with end-to-end atomic broadcast

Figure 7 shows the scenario of Figure 5 using end-to-end atomic broadcast. After the recovery of servers S_2 and S_3, message m is delivered again. This time, S_2 and S_3 do not crash, the delivery of m is successful and t is committed on all available servers.

5 A new safety criterion: group-safety

We have shown in Section 3 that the techniques of Section 2.1.1 based on traditional group communication are not 2-safe. They are only 1-safe: when the client is notified of t's commit, t did commit on the delegate server. As shown in Section 4, 2-safety can be obtained by extending group communication with end-to-end guarantees. However, group communication without end-to-end guarantees, even though it does not ensure 2-safety, provides an additional guarantee that is orthogonal to 1-safety and 2-safety. We call this guarantee *group-safety*.

5.1 Group Safety

A replication technique is *group-safe* if, when a client receives confirmation of a transaction's commit, the message that contains the transaction is guaranteed to be *delivered* (but not necessarily processed) on all available servers. In contrast, 2-safety guarantees that the transaction will be *processed* (i.e., committed) on all available servers. Group-safety relies on the group of servers to ensure durability, whereas 2-safety relies on stable storage. With group-safety, if the group does not fail, i.e., enough servers remain up,[3] then durability is ensured. Notice that group safety does not guarantee that the transaction was logged or committed on any replica. A client might be notified of the termination of some transaction t before t was actually logged on any replica.

		Transaction Logged		
		No Replica	1 Replica	All Replica
Transaction Delivered	1 Replica	**No Safety** **(0-Safe)**	**1-Safe**	▨
	All Replica	**Group-Safe**	**Group-Safe** **& 1-Safe** **(Group-1-Safe)**	**2-Safe**

Table 1: Summary of different safety levels

The relationship between group-safety, 1-safety and 2-safety is summarised in Table 1. We use two

[3] The number depends on the system model and on the algorithm used.

criteria: (1) the number of servers that are guaranteed to *deliver* the (message containing the) transaction (vertical axis), and (2) the number of servers that are guaranteed to eventually *commit* the transaction (horizontal axis), that is the number of servers that have logged the transaction. We distinguish a transaction *delivered* on (*one*, *all*) replicas, and a transaction *logged* on (*none*, *one*, *all*) replicas. A transaction cannot be logged on a site before being delivered, so the corresponding entry in the table is grayed out. For each remaining entry in the table the corresponding safety level is indicated:

No Safety: The client is notified as soon as the transaction t is delivered on one server S_d (t did not yet commit). No safety is enforced. If S_d crashes before t's writes are flushed to stable storage, then t is lost. We call this *0-safe* replication.

1-Safe: With *1-safety,* the client is notified when transaction t is delivered and logged on one server only, the delegate server S_d. If S_d crashes, then t might get lost. Indeed, while S_d is down, the system might commit new transactions that conflict with t: t must be discarded when S_d recovers. The only alternative would be to block all new transactions while S_d is down.

Group-Safe: The client is notified when a transaction is guaranteed to be delivered on all available servers (but might not be logged on any servers). If the group fails because too many servers crash,[4] then t might be lost. Group-safe replication basically allows all disk writes to be done asynchronously (outside of the scope of the transaction) thus enabling optimisations like write caching. Typically, disk writes would not be done immediately, but periodically. Writes of adjacent pages would also be scheduled together to maximise disk throughput.

Group-Safe & 1-Safe: The client is notified when transaction t is guaranteed to be delivered on all servers *and* was logged on one server, the delegate S_d and thus will eventually commit on S_d. Since the system is both group-safe and 1-safe, we call this safety level *group-1-safety*. With group-1-safety, the transaction might be lost if too many servers servers, **including** S_d, crash. A transaction loss occurs either if S_d never recovers, or the system accepts conflicting transactions while S_d is crashed. Most proposed database replication strategies based on group communication fall in this category [9, 27, 20, 23, 16].

2-Safe: The client is notified when a transaction is logged on all available servers. Even if all servers crash, the transaction will eventually commit and therefore cannot get lost.

If we consider the number of crashes that can be tolerated, we have basically three safety levels (Table 2):

[4] The number depends on the system model, typically a majority of servers.

- 0-safe and 1-safe replication cannot tolerate any crash, i.e., one single crash can lead to loose a transaction.
- Group-safe replication cannot tolerate the crash of all n servers.
- 2-safe replication can tolerate the crash of all n servers.

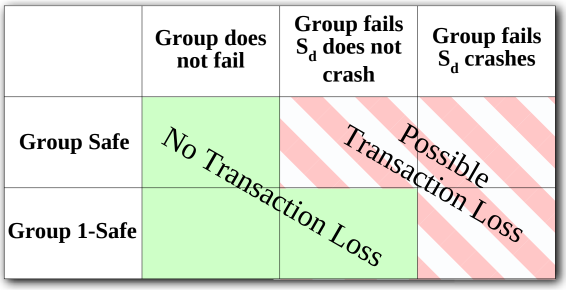
Tolerated Number of Crashes	Safety Property
0 crashes	0-safe, 1-safe
less than n crashes	group-safe, group-1-safe
n crashes	2-safe

Table 2: Safety property and number of crashes

5.2 Group Safety is preferable to Group-1-Safety

Group-safe as well as group-1-safe replication techniques cannot tolerate the crash of all servers. So, what is the real difference between both criteria? Table 3 summaries the conditions that lead to the loss of the transaction, using two criteria: (1) failure of the group (typically failure of a majority of servers) and (2) failure of the delegate server S_d. The difference appears in the middle column (failure of the group, but not of S_d).

Group communication-based replication scheme are specially interesting in update-everywhere settings, where the strong properties of atomic broadcast are used to handle concurrent transactions. If the replication is update-everywhere, then all servers $S_1 \ldots S_n$ might be the delegate server for some transaction.[5] If the group fails, at least one server crashed, and this server might be the delegate server S_d for some transaction t so the middle column conceptually does not exist. In such settings it makes little sense to deploy a group-1-safe replication technique.



	Group does not fail	Group fails S_d does not crash	Group fails S_d crashes
Group Safe	🟩	🟥	🟥
Group 1-Safe	🟩	🟩	🟥

No Transaction Loss
Possible Transaction Loss

Table 3: Safety of comparison between group safety and group-1-safety

[5] This is not the case with the *primary-copy* technique.

The replication technique illustrated in Figure 2 ensures group-1-safety. It can be transformed into group-safe-only quite easily. Figure 8 illustrates the group-safe version of the same technique. Read operations are typically done only on the delegate server S_d before the broadcasting, writes are executed once the transactions is delivered by the atomic broadcast. The main difference with Figure 2 is the response to the client, which is sent back as soon as the transactions is delivered by the atomic broadcast and the decision to commit/abort the transaction is known. The observed response time by the client is shortened by the time needed to write the decision to disk. The performance gain is shown by the simulation results presented in Section 6.

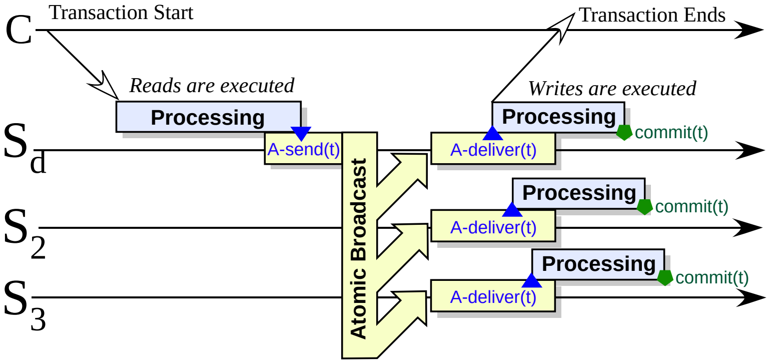


Figure 8: Group Safe Replication

6 Performance Evaluation

In this section we compare the performance of group-safety, group-1-safety and 1-safety (i.e., lazy replication). The evaluation is done using a replicated database simulator [33]. The group communication-based technique is the database state machine technique [27], which is an instance of the replication technique illustrated on Figure 2 (for group-1-safety) and Figure 8 (for group-safety). The setting of the simulator are described in Table 4. The load of the system is between 20 and 40 transactions per second; the network settings correspond to a 100 Mb/s LAN.

Figure 9 shows the results of this experiment. The X axis represents the load of the system in transactions per second, the Y axis the response time, in milliseconds. Each replication technique is represented by one curve. The results show that group-safe replication has very good performance: it even outperforms lazy replication when the load is below 38 transactions per second. The abort rate of the group-safe technique was constant, slightly below 7%. As the lazy technique does no conflict handling, abort rate is unknown. The very good performance of the group safe technique is due to the

Parameter	Value
Number of items in the database	10’000
Number of Servers	9
Number of Clients per Server	4
Disks per Server	2
CPUs per Server	2
Transaction Length	10 – 20 Operations
Probability that an operation is a write	50%
Probability that an operation is a query	50%
Buffer hit ratio	20%
Time for a read	4 - 12 *ms*
Time for a write	4 - 12 *ms*
CPU Time used for an I/O operation	0.4 *ms*
Time for a message on the Network	0.07 *ms*
CPU time to send/receive a message	0.07 *ms*
Time for a broadcast on the Network	0.07 *ms*
CPU time to send/receive a broadcast	0.07 *ms*

Table 4: Simulator parameters

asynchrony of the writes (the writes to disk are done outside the scope of the transaction). In high-load situations, group-safe replication becomes less efficient than lazy replication. The results show also that group-1-safe replication behaves significantly worse than group-safe replication: the technique scales poorly when the load increases.

To summarise, the results show that transferring the responsibility of durability from stable storage to the group is a good idea in a LAN: in our setting, writing to disk takes around 8 *ms*, while performing an atomic broadcast takes approximately 1 *ms*.

7 Group-safe replication *vs.* lazy replication

On a conceptual level, group-safe replication can be seen as a complement to lazy replication. Both approaches try to get better performance by weakening the link between some parts of the system. Figure 10 illustrates this relationship. Group-safe replication relaxes the link between server and stable storage: when a transaction t commits, the state in memory and in stable storage might be different (t’s writes are not committed to disk, they are done asynchronously). Lazy replication relaxes the link between replicas: when a transaction commits, the state in the different replicas might be different (some replicas have not seen transaction t; t’s writes are sent asynchronously). The two approaches relax the synchrony that is deemed too expensive.

The main difference is the condition that leads to a violation of the ACID properties. In an update-

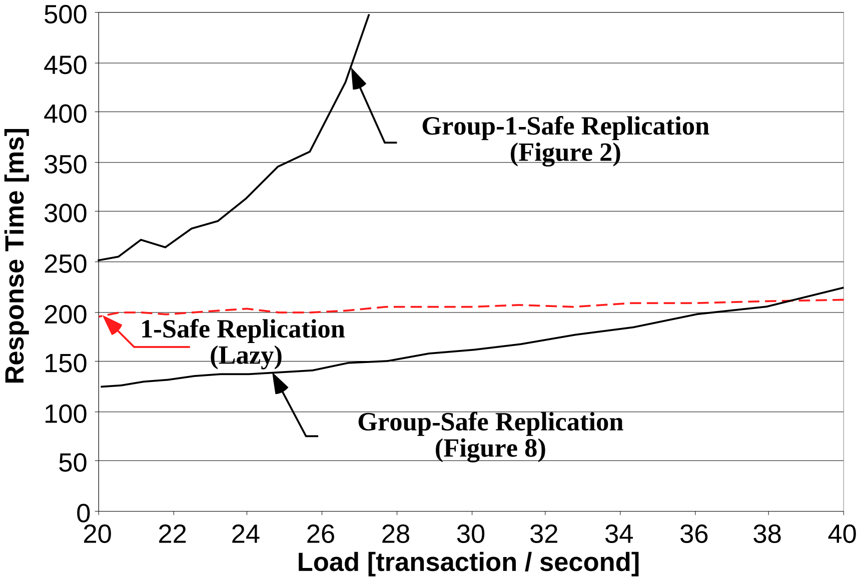


Figure 9: Simulation results

everywhere setting, a lazy technique can violate the ACID properties even if no failure occurs. On the other hand, a group-safe replication will only violate this ACID properties if the group fails (too many servers crash). Group-safe replication has another advantage over lazy replication. With lazy replication in an update-everywhere setting, if the number of servers grow, the chances that two transaction originating from two different sites conflict grows. So the chances that the ACID properties are violated grows with the number of servers. With group-safe replication the ACID properties might get violated if too many servers crash. If we assume that the probability of the crash of a server is independent of the number of servers, the chance of violating the ACID properties decreases when the number of servers in-

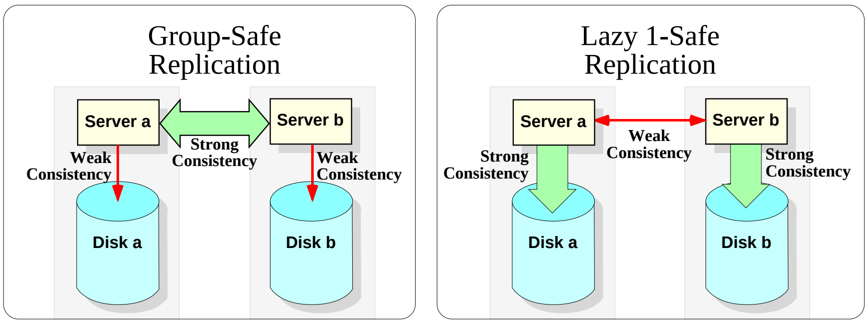


Figure 10: group-safe replication and lazy replication

creases. So, the chances that something *bad* happens increases with n for lazy replication, and decreases with group-safe replication.

8 Related Work

As already mentionned, traditional database replication is usually either (i) 2-safe and built around an atomic commitment protocol, or (ii) does not rely on atomic commitment and is therefore 1-safe [13]. As the the atomic commitment protocol is implemented inside the database system, coupling between database and communication systems is not an issue in these protocols. Note that techniques have been proposed to improve atomic commitment using group communication [18].

The fact that 2-safety does not require atomic commitment has been hinted at in [35]. The paper explores the relationship between safety levels and the semantics of communication protocols. However, the distinction between 2-safety and the safety properties ensured by traditional group communication does not appear explicitely in the paper.

The Disk Paxos[10] algorithm can also be loosely related to 2-safety, even though the paper does not address database replication issues. The paper presents an original way, using stable storage, to couple the application component with a component solving an agreement problem. However, the paper assumes a network attached storage, wich is quite different from the model considered here, where each network node only has direct access to its own database. The issue of connecting the group communication component and the database component can also be related to the *exactly once* property in three-tier applications [8]. Our group communication system and database system can be seen as two tiers co-located on the same machine that communicate using messages.

While the notion of group safety is formally defined here, existing database replication protocols have in the past rely on this property, e.g., [22, 15]. In [3] a mechanism to synchronise the database with the group communication system is presented in the context of partitionable membership systems. The mechanism is used to handle merges between partitions; the case of a total crash is not considered. None of these protocols ensure 2-safety.

9 Conclusion

In this paper, we have shown that traditional group communication primitives are not suited for building 2-safe database replication techniques. This led us to introduce *end-to-end atomic broadcast* to solve the problem. We have also shown that, while traditional group communication (without end-to-end guarantees) are not suited for 2-safe replication, they offer stronger guarantees than 1-safety. To formalise this, we have introduced a new safety criterion, called *group-safety* that captures the safety guarantees of group communication-based replication techniques. Performance evaluation show that group-safe replication compares favourably to lazy replication, while providing better guarantees in terms of the ACID properties.

References

[1] D. Agrawal, G. Alonso, A. El Abbadi, and I. Stanoi. Exploiting atomic broadcast in replicated databases. Technical report, Department of Computer Science, University of California, Santa Barbara, California USA, 1996.

[2] M. K. Aguilera, W. Chen, and S. Toueg. Failure detection and consensus in the crash recovery model. *Distributed Computing*, 13(2):99–125, 2000.

[3] Y. Amir and C. Tutu. From total order to database replication. Technical Report CNDS-2001-6, Departement of Computer Science John Hopkins University, Baltimore, MD 21218 USA, November 2001.

[4] K. Birman, R. Constable, M. Hayden, C. Kreitz, O. Rodeh, R. van Renesse, and W. Vogels. The Horus and Ensemble projects: Accomplishments and limitations. In *Proceedings of the DARPA Information Survivability Conference & Exposition (DISCEX '00)*, Hilton Head, South Carolina USA, 2000.

[5] K. P. Birman and T. A. Joseph. Exploiting virtual synchrony in distributed systems. In *Proceedings of the* 11^{th} *ACM Symposium on OS Principles*, pages 123–138, Austin, TX, USA, 1987. ACM SIGOPS, ACM.

[6] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Communications of the ACM*, 43(2):225–267, 1996.

[7] D. R. Cheriton and D. Skeen. Understanding the limitations of causally and totally ordered communication. In Barbara Liskov, editor, *Proceedings of the* 14^{th} *Symposium on Operating Systems Principles*, volume 27, pages 44–57, Asheville, North Carolina, 1993. ACM Press, New York, NY, USA.

[8] S. Frølund and R. Guerraoui. Implementing e-transactions with asynchronous replication. *IEEE Transactions on Parallel and Distributed Systems*, 12(2), june 2001.

[9] A. W. Fu and D. W. Cheung. A transaction replication scheme for a replicated database with node autonomy. In *Proceedings of the International Conference on Very Large Databases*, Santiago, Chile, 1994.

[10] E. Gafni and L. Lamport. Disk paxos. Technical Report SRC 163, Compaq Systems Research Center, 130, Lytton Avenue Palo Alto, California 94301 USA, July 2000.

[11] F. C. Gärtner. A gentle introduction to failure detectors and related problems. Technical Report TUD-BS-2001-01, Darmstadt University of Technology, Department of Computer Science, 2001.

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

[13] J. N. Gray and A. Reuter. *Transaction Processing: concepts and techniques*. Data Management Systems. Morgan Kaufmann Publishers, Inc., San Mateo (CA), USA, 1993.

[14] M. A. Hiltunen and R. D. Schlichting. The cactus approach to building configurable middleware services. In *Proceedings of the Workshop on Dependable System Middleware and Group Communication (DSMGC 2000)*, Nürnberg, Germany, 2000.

[15] J. Holliday. Replicated database recovery using multicast communications. In *Proceedings of the Symposium on Network Computing and Applications (NCA'01)*, pages 104–107, Cambridge, MA, USA, 2001. IEEE.

[16] J. Holliday, D. Agrawal, and A. El Abbadi. The performance of database replication with group multicast. In *Proceedings of International Symposium on Fault Tolerant Computing (FTCS29)*, pages 158–165. IEEE Computer Society, 1999.

[17] J. Holliday, D. Agrawal, and A. El Abbadi. Using multicast communication to reduce deadlocks in replicated databases. In *Proceedings of the* 19^{th} *Symposium on Reliable Distributed Systems SRDS'2000*, pages 196–205, Nürnberg, Germany, 2000. IEEE Computer Society, Los Alamitos, California.

[18] R. Jiménez-Paris, M. Patiño-Martínez, G. Alonso, and S. Arévalo. A low latency non-blocking commit server. In J. Welch, editor, *Proceeedings of the* 15^{th} *Internationnal Conference on Distributed Computing (DISC 2001)*, volume 2180 of *lecture notes on computer science*, pages 93–107, Lisbon, Portugal, 2001. Springer Verlag, Berlin Heidelberg.

[19] I. Keidar. A highly available paradigm for consistent object replication. Master's thesis, The Hebrew University of Jerusalem, Jerusalem, Israel, 1994. also technical report CS94.

[20] B. Kemme and G. Alonso. A suite of database replication protocols based on group communication primitives. In *Proceedings of the* 18^{th} *International Conference on Distributed Computing Systems (ICDCS'98)*, Amsterdam, The Netherlands, 1998.

[21] B. Kemme and G. Alonso. Don't be lazy, be consistent: Postgres-R, a new way to implement database replication. In *Proceedings of the* 26^{th} *International Conference on Very Large Databases (VLDB)*, Cairo, Egypt, 2000.

[22] B. Kemme, A. Bartoli, and Ö. Babaoğlu. Online reconfiguration in replicated databases based on group communication. In *Proceedings of the Internationnal Conference on Dependable Systems and Networks (DSN2001)*, Göteborg, Sweden, 2001.

[23] B. Kemme, F. Pedone, G. Alonso, and A. Schiper. Processing transactions over optimistic atomic broadcast protocols. In *Proceedings of the International Conference on Distributed Computing Systems*, Austin, Texas, 1999.

[24] N. Lynch. *Distributed Algorithms*. Morgan Kaufmann, San Francisco, CS, 1996.

[25] C. P. Malloth, P. Felber, A. Schiper, and U. Wilhelm. Phoenix: A toolkit for building fault-tolerant distributed applications in large scale. In *Workshop on Parallel and Distributed Platforms in Industrial Products*, San Antonio, Texas, USA, 1995. IEEE. Workshop held during the 7^{th} Symposium on Parallel and Distributed Processing, (SPDP-7).

[26] H. Miranda, A. Pinto, and L. Rodrigues. Appia: A flexible protocol kernel supporting multiple coordinated channels. In *Proceedings of the* 21^{st} *International Conference on Distributed Computing Systems (ICDCS-01)*, pages 707–710, Phoenix, Arizona, USA, 2001. IEEE Computer Society.

[27] F. Pedone, R. Guerraoui, and A. Schiper. The database state machine approach. Technical Report SSC/1999/008, École Polytechnique Fédérale de Lausanne, Switzerland, 1999.

[28] L. Rodrigues and M. Raynal. Atomic broadcast in asynchronous crash-recovery distributed systems. In *Proceedings of the* 20^{th} *International Conference on Distributed Systems (ICDCS'2000)*, pages 288–295, Taipei, Taiwan (ROC), 2000. IEEE Computer Society, Los Alamitos USA.

[29] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems*, 2(4):277–288, 1984.

[30] A. Schiper and M. Raynal. From group communication to transactions in distributed systems. *Communications of the ACM*, 39(4):84–87, 1996.

[31] I. Stanoi, D. Agrawal, and A. El Abbadi. Using broadcast primitives in replicated databases (abstract). In *Proceedings of the* 16^{th} *Annual Symposium on Principles of Distributed Computing*, page 283, Santa Barbara, California USA, 1997. ACM.

[32] R. van Renesse, K. P. Birman, and S. Maffeis. Horus: A flexible group communication system. *Communications of the ACM*, 39(4):76–83, 1996.

[33] M. Wiesmann. *Group Communications and Database Replication: Techniques, Issues and Performance*. PhD thesis, École Polytechnique Fédérale de Lausanne, Switzerland, may 2002.

[34] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, and G. Alonso. Database replication techniques: a three parameter classification. In *Proceedings of* 19^{th} *Symposium on Reliable Distributed Systems (SRDS'2000)*, pages 206–215, Nürnberg, Germany, 2000. IEEE Computer Society.

[35] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, and G. Alonso. Understanding replication in databases and distributed systems. In *Proceedings of* 20^{th} *International Conference on Distributed Computing Systems (ICDCS'2000)*, Taipei, Taiwan, R.O.C., 2000. IEEE Computer Society Los Alamitos California.