A Closer Look at Optimistic Replica Control

Fernando Pedone*

Département d'Informatique
Ecole Polytechnique Fédérale de Lausanne
1015 Lausanne, Switzerland
e-mail: pedone@lse.epfl.ch

1 Overview

Replication is a cheap software based way to increase the availability of databases when compared to hardware based solutions. Replication increases the liveness of the transactions accessing the database, as these transactions are more likely to commit than in non-replicated contexts. However, replication introduces an inherent complexity in the traditional database scheme since concurrent accesses to different replicas may lead to inconsistencies, violating the safety of the system.

A lot of work on replication techniques ensuring some safety (consistency) properties to replicated databases has been done in the past years. Recently, these techniques have gained even more interest due to the increasing demand on highly available distributed architectures. Despite this fact, it is still difficult to find one's way in the entangled world of replication protocols. The work done in classifying these techniques was mostly based on performance comparison. We recognize that this is an important criteria, but it is not the only one, as the user of a distributed architecture should also know which levels of liveness and safety are guaranteed by the system.

In this paper we focus on a set of optimistic replication techniques found in the literature and compare them with regard to the levels of safety and liveness they guarantee. We show that although these techniques can be considered as equivalent from a performance point of view, they differ in the guarantees provided. We present a general framework that enables us to fairly compare these techniques. In this framework, we use an atomic broadcast primitive that alleviates the need of a costly atomic commitment protocol. Finally we point out some improvements to the liveness of some existing replica control protocols.

The levels of safety are inspired in the degrees of isolation introduced by Gray *et al* [4] and in the work presented in [1]. Roughly speaking, a level of safety characterizes the consistency of the replicated data. Serial equivalence is a typical example of a safety level (it is in fact the highest one) that ensures replication transparency. Some interesting replication protocols relax these techniques accepting non-serializable behavior. They guarantee a lower level of consistency. Usually, just two levels of liveness are informally distinguished in the literature. The transactions may either block waiting for a site to recover (level 0) or they are guaranteed to terminate (level 1). This is an incomplete scheme, as termination may just mean aborting transactions which is actually not very useful in practice. We introduce additional levels of liveness that help to better understand the states a transaction may pass.

The rest of the paper is organized as follows. Section 2 presents our system model, assumptions, replicated architecture and the safety/liveness characterization. Section 3 describes our general framework for optimistic replication protocol. Section 4 compares three techniques found in the literature. In Section 5 we show how to increase the level of liveness and we point out the price of this gain. Section 6 summarizes our classification.

*On leave from Colégio Técnico Industrial, University of Rio Grande, Brazil

2 Replicated Database Model

2.1 Transaction Model

A transaction T_i interacts with the database submitting read and write requests to a process called *Transaction Manager* (TM). The transaction manager is responsible for distributing the operations to the correct sites, mapping data items onto their replicas and coordinating transactions commits. Each site has a process called *Data Manager* (DM) that schedules and executes the operations received from the TM. The DM also takes part in the transaction termination (e.g., executing the validation test to check whether the execution is valid).

We introduce the concepts of readset (RS_i) and writeset (WS_i) to represent the set of data accessed by a transaction. The RS_i contains an identification for each data item read by T_i, and the WS_i contains the data items written by T_i. Each data item x in the database has a version number associated to it indicating when it was last updated. In the commit, written data items have their version numbers incremented.

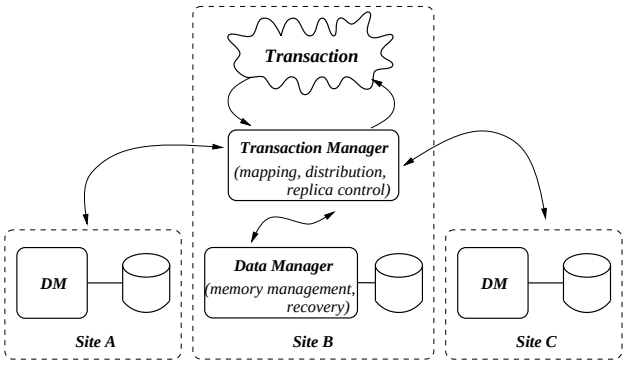

Figure 1: Database Architecture

2.2 System Model

We consider a peer-to-peer, fully replicated system with no centralized control over the replicas [3]. All sites have the same functionality (i.e., transactions may run at any site) and there is a copy of each data item at every site (see Figure 1). Processes can only fail by crashing and we do not consider Byzantine failures.

The communication is reliable, that is, messages are neither lost nor corrupted. We assume the existence of an atomic broadcast that totally orders the messages and guarantees the all-or-nothing property among all correct processes [7]. For instance, if two processes broadcast the messages m_1 and m_2, all processes (it does not matter in which site they execute) will deliver them in the same order, e.g., first m_1 and then m_2 or m_2 and m_1.

2.3 Safety *vs.* Liveness

In section 4 we compare a set of optimistic replica control algorithms in according to their levels of safety and liveness. Levels of safety represent the degrees of consistency guaranteed by the algorithm (e.g., serial equivalence) whereas levels of liveness express the different expectations a transaction may have concerning its termination.

The safety characterization, adapted from [1] is not absolute but sufficient to capture the different nuances found in the algorithms considered here (see Figure 2). Greater the level, stricter it is. Safety level i is level $i+1$ relaxed to permit the phenomena presented in its scope. Level 3 for instance guarantees that transactions execute in a serial equivalent fashion.

The liveness characterization reflects the extent to which the system guarantees the transaction termination (see Figure 3). The highest level of liveness (level 3) characterizes situations where transactions are guaranteed to commit. It is important to point out that this is something rather difficult to achieve.

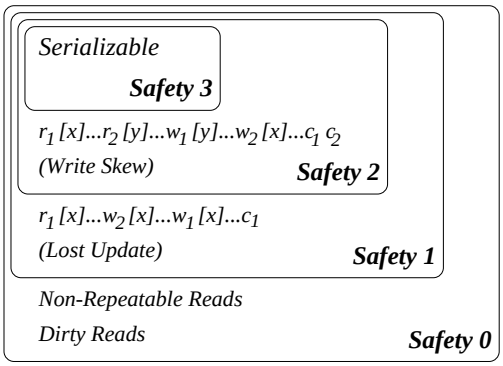

Figure 2: Levels of Safety

Read-only transactions in level 2 are never aborted by the system. Level 1 represents the standard transaction model where aborts may occur at the end of the execution and level 0 may arise if the replica control algorithm depends on non-fault tolerant mechanisms (e.g., two phase commit).

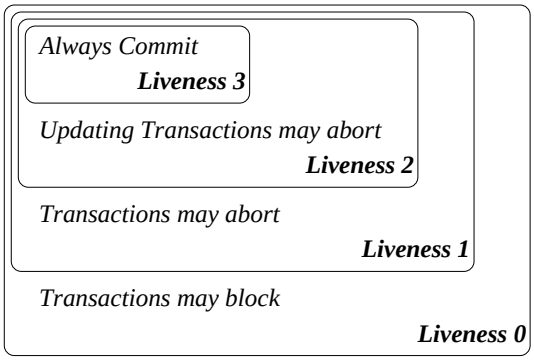

Figure 3: Levels of Liveness

3 Optimistic Replication Control

In a general way, optimistic algorithms usually have three stages [2]. In the first stage, transactions execute without performing their writes to the database. The second stage constitutes the validation, when the execution is checked to see if it is valid according to a replication control policy. In the third stage, the deferred updates are applied by all replicas in case the execution is valid. The validation and updating stages are done atomically. The algorithms we describe here use a replicated multi-version approach to manage the database.

Figure 4 shows a general framework for a replicated database using an optimistic replica control mechanism. During the first stage, each read and write is executed locally, at the site where the transaction initiated. The transaction T_i uses the newest version of the data and its writes are not directly performed but deferred to the commit. If another transaction T_j starts concurrently with T_i, it will see the same data version as T_i. Transactions just use committed version of the database, meaning that tentative writes executed by one transaction are not seen by the others. Transactions are never aborted during their processing (stage 1), as long as the system is able to maintain the same version for all the data they need.

In order to commit, the transaction is forwarded to the other replicas (in fact only the readset and writeset are forwarded). Each replica executes the validation test and independently takes its decision to commit or abort the transaction. When a transaction commits, a new version is generated. This is an

atomic operation so that all sites, including the one where the transaction originated, execute the second and third stages in the same order, i.e., if more than one site starts the commit at the same time.

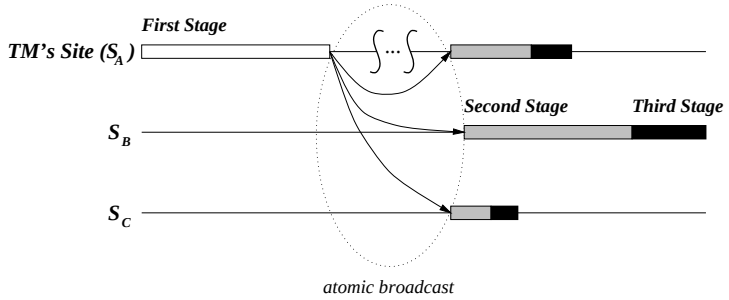

Figure 4: Optimistic Active Replication

The atomic update mentioned above may be implemented by means of an atomic commit protocol. In this case the site where the transaction originates acts as the coordinator for the protocol. However, this choice seems inadequate because atomic commitment protocols have shown to be either blocking in presence of failures (e.g., 2PC) or expensive (e.g., 3PC) [6]. Furthermore, atomic commitment protocols do not guarantee any ordering property among replicas, which is highly desired if multiple transactions start their commit requests at the same time.

If we assume deterministic operation for all replicas and use an ordered atomic broadcast to initiate the second stage, there is no need for an atomic commitment since all sites reach the same conclusion considering the outcome of the transaction. The third stage generates a new version of each data item modified by T_i. This solution has the advantage of being both fault tolerant (e.i., non-blocking) and efficient.

4 Standard Replication Control Algorithms

4.1 Serial Equivalence

The validation test used here guarantees that concurrent transactions execute in complete isolation (which would be true in a serial execution). In this case, no transaction T_j would run concurrently with another transaction T_i and so no data item read or written by T_i would be overwritten by T_j. This leads us to the following test, assuming a group of $\{T_i, T_{i+1}, T_{i+2}, \ldots, T_j\}$ transactions executing concurrently [1], and at the moment T_i is tested all the other $\{T_{i+1}, T_{i+2}, \ldots, T_j\}$ transactions have already committed [2].

$$RS_i \cap (WS_{i+1} \cup WS_{i+2} \cup \ldots \cup WS_j) = \emptyset \quad (1)$$

The algorithm presented here guarantees *safety 3*. In order to guarantee this safety transactions may be aborted by the system, but the multiversion approach prevents read-only transactions from being aborted. This corresponds to degree 2 of *liveness*.

4.2 Snapshot Isolation

Snapshot isolation [1] is a relaxation of the serial equivalence. It does not consider read dependencies among transactions but only write dependencies. The idea is to prevent lost update, a phenomenon that occurs when a transaction T_i reads a data item, another transaction T_j updates this data (possibly based on a previous read) and T_i (based on its earlier read value) updates the data item and commits.

This can be better understood by the following example. Assume that $r_i[x]$ is a read operation issued by T_i, $w_i[x]$ a write operation and c_i the commit. The role of T_i is to increment the value of x by 10 and

[1] Transactions are said to execute concurrently if none of them uses a version of a data item that was created by the other.

[2] This test checks if data items read by T_i were not written by T_j. Interleaving writes are avoided by the multiversion approach with updates applied atomically at the end of the execution.

T_j is to decrement x of 20.

$$r_i[x = 100] \ldots r_j[x = 100] \ldots w_j[x = 80]\ c_j \ldots w_i[x = 110]\ c_i$$

The problem with this history is that T_j loses its update because T_i writes over it. This history is not serial and would not be allowed in the previous test (T_i would be aborted).

The snapshot validation test aborts transactions that overwrite the data written by other concurrent transactions. For the same model as before, the test that evaluates this may be stated as condition (2).

$$WS_i \cap (WS_{i+1} \cup WS_{i+2} \cup \ldots \cup WS_j) = \emptyset \tag{2}$$

Snapshot isolation can be extended to a replicated system by the same general framework used in the previous section. The first phase is done locally to T_i and to commit the writeset is distributed to the replicas. The validation test is applied by each replica that individually commits or aborts T_i. As before, this is just possible because of the total order guaranteed by the atomic broadcast protocol and the assumption that the replicas execution is deterministic.

This approach guarantees *safety 2*. It means that transactions have a better chance to commit (greater *liveness*) but histories may be not serial, as shown by the following example that presents the *write skew* phenomenon (see Figure 2).

$$r_i[x = 1] \ldots r_j[y = 2] \ldots w_i[y = 1] \ldots w_j[x = 2]\ c_i\ c_j$$

5 Increasing Liveness

5.1 Flexible Snapshot

Interestingly, snapshot isolation aborts histories that are in fact serializable. Consider for instance the history below, where T_i and T_j update a common field in the database.

$$r_i[i = 1] \ldots r_j[j = 2] \ldots w_j[x = 2]\ c_j \ldots w_i[x = 1]\ c_i$$

This history would be aborted by condition (2) (both update x) although it is serializable, and allowed by condition (1). This fact suggests a variation in the test where both equations are checked. If the first one fails the second is used. With this we extend the *liveness* of the method without changing its *safety* [3]. The new validation test would look like condition (3).

$$\begin{gathered} RS_i \cap (WS_{i+1} \cup WS_{i+2} \cup \ldots \cup WS_j) = \emptyset \\ or \\ WS_i \cap (WS_{i+1} \cup WS_{i+2} \cup \ldots \cup WS_j) = \emptyset \end{gathered} \tag{3}$$

5.2 True Optimism

Naturally, the previous section suggests a model with no test. This means that transactions are never aborted, something that would give us the highest level of liveness. The question that arises is whether this method guarantees something to the transactions. In fact, it is still possible to offer a useful degree of safety with it. The benefits achievable here are a direct consequence of the atomic broadcast and the snapshot view of the database.

The atomic broadcast guarantees that the database will not diverge among the replicas, causing an unwanted phenomenon named *system delusion* [3]. The snapshot view offers a scenario where reads are repeatable and updates done by one transaction are not overwritten by another transaction (*safety 1*).

Back to our tradeoff perspective, true optimism situates itself in the opposite end of serial equivalence. It offers the best *liveness* (transactions are never aborted) but at the cost of possibly producing inconsistent data (low *safety*). As shown in [4], transactions can coexist using different levels of isolation (*safety*). Transactions executing in lower levels may produce bad data for transaction in upper ones, and this has to be regulated by the user. In such scenarios read-only transactions would execute in a true optimistic fashion without incurring any addition overhead with tests.

[3] Of course there is the additional overhead with keeping the readsets and testing more conditions.

6 Summary

This work compares some optimistic replica control algorithms by means of their safety/liveness characteristics. To present these algorithms, we define a general framework that is detailed for each case. An atomic broadcast protocol is used to implement this general framework, increasing the liveness and performance of the algorithms. Our characterization defines different levels of safety and liveness. Levels of safety are an adaptation of degrees of isolation present in [5] but augmented to consider more specific cases. Levels of liveness define an expectation about the transaction termination.

	Safety	***Liveness***
Serial Equivalence (without Multiversion)	*3*	*1*
Serial Equivalence	*3*	*2*
Snapshot Isolation	*2*	*2*
Flexible Snapshot	*2*	*2**
True Optimism	*1*	*3*

* *Less histories are aborted by the system.*

Figure 5: The Tradeoff Safety *vs.* Liveness

For all algorithms, the cost in number of messages is the same, being this the cost of an atomic broadcast protocol. Since the only difference concerns the way the validation test is executed, this study offers a fair comparison among the replica control mechanisms presented. Figure 5 summarizes our results.

Acknowledgments

We thank Rachid Guerraoui for the valuable discussions and successive readings of previous versions of this paper.

References

[1] H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O'Neil, and P. O'Neil. A critique of ANSI SQL isolation levels. *Proc. ACM SIGMOD*, pages 1–10, June 1995.

[2] P. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.

[3] J. Gray, P. Helland, P. O'Neil, and D. Shasha. The dangers of replication and a solution. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, June 1996.

[4] J. Gray, R. Lorie, G. Putzolu, and I. Traiger. *Readings in Database Systems*, chapter 3, Granularity of Locks and Degrees of Consistency in a Shared Database. Morgan Kaufmann, 1994.

[5] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, 1993.

[6] R. Guerraoui and A. Schiper. The decentralized non-blocking atomic commitment protocol. In *IEEE International Symposium on Parallel and Distributed Processing*, 1995.

[7] V. Hadzilacos and S. Toueg. *Distributed Systems, 2ed*, chapter 3, Fault-Tolerant Broadcasts and Related Problems. Addison Wesley, 1993.