1-Copy Equivalence: Atomic Commit Versus Atomic Broadcast in Termination Protocols

M. I. Ruiz-Fuertes, J. M. Bernabé-Gisbert, R. de Juan-Marín, F. D. Muñoz-Escoí

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

{miruifue,jbgisber,rjuan,fmunyoz}@iti.upv.es

Technical Report TR-ITI-SIDI-2010/002

1-Copy Equivalence: Atomic Commit Versus Atomic Broadcast in Termination Protocols

M. I. Ruiz-Fuertes, J. M. Bernabé-Gisbert, R. de Juan-Marín, F. D. Muñoz-Escoí

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

Technical Report TR-ITI-SIDI-2010/002

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

April, 2010

Abstract

Early replicated distributed database systems were strongly concerned about consistency, commonly defined as one-copy serializability. A database spread over multiple sites must behave as a unique node, and the result of concurrent transactions must be equivalent to that of a serial execution of the same set of transactions. In order to ensure such consistency guarantees, concurrency was managed by distributed locking and atomic commit protocols controlled transaction termination. Such systems, however, suffered from low performance and poor scalability due to the high cost of the protocols in use. Research focused then on improving performance and scalability while trying to maintain the correctness criterion of one-copy serializability. Atomic broadcast was proposed as a better alternative to atomic commit protocols, and transactions were locally executed before being broadcast to the rest of replicas. Outstanding performance improvements were achieved. But, against what was initially asserted, correctness guarantees were subtly but significantly modified.This paper carefully analyzes both solutions and states the different guarantees they provide by establishing a correspondence with memory consistency models as defined in the scope of distributed shared memory.

1 Introduction

Distributed and replicated database systems appeared in order to provide a higher level of availability and fault tolerance than existing centralized databases, while ensuring a strong consistency level between replicas. Bernstein et al. defined one-copy serializability (1SR) [6] as a correctness criterion. According to it, the interleaved execution of clients' transactions must be equivalent to a serial execution of those transactions on a stand-alone (non replicated) database. 1SR turned immediately to be the most common correctness criterion for database replication protocols. In order to ensure such guarantees, a conservative approach inherited from distributed database systems was followed. Thus, write operations had to acquire write locks in all data item copies prior to update the local copy. This concurrency control based on distributed locking strongly affected performance, as each write operation must be preceded by a round of messages requesting the corresponding lock in every replica. Moreover, in order to guarantee transaction atomicity, an atomic commit protocol was run for transaction termination. In such protocols, several rounds of messages were required in order to reach a consensus among all participating sites for each transaction commitment, which further penalized performance and scalability.

Later database replication systems introduced several optimizations, possible thanks to full replication, trying to provide correct, i.e. 1SR, replication at a reasonable cost. According to the deferred update replication model (as used in the distributed certification scheme of [22] or in the distributed optimistic two-phase locking, O2PL, of [7]), transactions were processed locally at one server and, at commit time, were

forwarded to the rest of system nodes for validation. This optimization allowed to save communication costs as synchronization with other nodes was only done at transaction termination during the atomic commit protocol. Another important optimization consisted in using atomic broadcast as a substitute for atomic commit protocols [19]. Atomic broadcast is a communication primitive that enables to send a message to a group of nodes, with the guarantee that all nodes agree on the set of messages delivered and on the order according to which those messages are delivered. The order of atomic broadcast was then used as a serialization order for achieving 1SR.

According to the theoretical definition of 1SR [6], replication protocols based on atomic commit and those based on atomic broadcast provided the same correctness criterion. However, carefully analyzing the behavior of the implemented systems, some differences can be observed. In this paper, we demonstrate that the provided guarantees significantly changed when atomic commit protocols were replaced by termination protocols based on atomic broadcast. To this end, two representative database replication systems are analyzed through a case study to reveal the differences between them. We further establish correspondences between memory consistency models used in distributed shared memory (DSM), and replica consistency enforced by database replication protocols.

The rest of the paper is structured as follows. Section 2 details the assumed system model and provides basic definitions. Section 3 states the ambiguity of the term *one-copy equivalence*, by showing important differences in consistency between two systems providing 1SR. Section 4 presents and compares two memory consistency models used in the scope of DSM, for later establishing, in Sect. 5, a correspondence between these models and replica consistency levels ensured in database replication systems. Section 6 suggests that an atomic consistency level is bearable in a transactional context, while it is generally discarded for DSM due to its cost. Finally Sect. 7 concludes the paper.

2 System Model and Definitions

The replication systems analyzed in this paper consider an asynchronous distributed system composed of database servers, called replicas, $R_1, R_2, \ldots, R_n$, and clients. Communication between components is based on message passing. Servers can also communicate through atomic broadcast, described below.

Replicas fail independently by crashing (Byzantine failures are not considered). Sites may eventually recover after a crash. A site is correct if it never crashes, otherwise it is faulty.

Each server stores a full copy of the database[1]. The database workload is composed of transactions, $T_1, T_2, \ldots$ Transactions are sequences of read and write operations followed by a commit or an abort operation, and maintain the ACID properties as defined in [12]: atomicity[2], consistency, isolation and durability. Transactions of the same client session are submitted sequentially, but may be addressed to different replicas. Transactions are locally executed under the serializable isolation level [4], according to strict two-phase locking (2PL) [6].

To ensure consistent termination of distributed transactions, database systems have traditionally resorted to an atomic commit protocol, where each transaction participant starts by voting yes or no and each site reaches the same decision about the outcome of the current transaction: commit or abort. To support failures, a non-blocking atomic commit protocol (NB-AC) [6, 23] must be used. In these protocols, each participant reaches a decision despite the failure of other participants. A NB-AC protocol fulfills the following properties. (a) Agreement: no two participants decide different outcomes. (b) Termination: every correct participant eventually decides. (c) Validity: if a participant decides commit, then all participants have voted yes. (d) Non-triviality: if all participants vote yes, and no participant fails, then every correct participant eventually decides commit.

Atomic broadcast is a group communication abstraction used by replicas to communicate. Atomic broadcast is defined by the primitives $broadcast(m)$ and $deliver(m)$, and satisfies the following properties [11]. (a) Validity: if a correct site broadcasts a message m, then it eventually delivers m. (b) Agreement: if a correct site delivers a message m, then every correct site eventually delivers m. (c) Integrity: for every

[1] Bernstein and Goodman [5, 6] did not require full replication, as their system was aimed at distributed databases with some degree of replication.

[2] In this context, atomicity is a synonym of indivisibility.

message m, every site delivers m at most once, and only if m was previously broadcast. (d) Total Order: if two correct sites deliver two messages m and m', then they do so in the same order.

3 1-Copy Serializability: an Ambiguous Term

In many cases, implemented systems are quite more restrictive than the correctness models they follow. Situations that would be allowed by the model are rejected in the real system, or they are simply not possible due to implementation issues. This mismatching is usually due to pragmatic considerations, as an exact model reflection would increment the system complexity. Although correctness is never impaired, other aspects such as performance, concurrency or scalability are negatively affected.

When Bernstein and Goodman proposed their protocol [5] for concurrency control in replicated distributed databases, they were following the one-copy serializability criterion. The atomic commit protocol used for transaction termination, two-phase commit (2PC), ensured that all replicas agreed on the set of committed transactions and a serializable history was guaranteed. However, due to distributed locking and atomic commit, replicas did not only commit the same sequence of transactions but did it in such a way that any new issued transaction was able to see all the changes performed by previous transactions, with independence of the replica where this new transaction was executed. That is, the system nodes were behaving as a one-copy database in that any copy of a written data item was updated before any subsequent access to that copy was allowed: a transaction T always got the most recent vision of the database, as created by the last transaction executed in the system immediately before T. Let's consider an simple example to illustrate this feature.

Example 1. Distributed locking and two-phase commit A client starts transaction T_1 at replica R_1. The database contains data item x with an initial value x_0. T_1 requests write access to x, which requires that a write lock is acquired at each replica, as stated by the distributed locking concurrency control. After getting all the locks, T_1 modifies the value of x and finishes. During the atomic commit protocol all replicas agree on the commitment, so T_1 can commit, the new value x_1 can be assigned to x, and the write lock on x can be released. After T_1's commitment, the same client starts a second transaction T_2 that reads x. If T_2 is also started in replica R_1, where value x_1 is already available, T_2 will read it and finish. But suppose that T_2 is run at replica R_2. Suppose also that this node is slower and it did not yet apply the new value to data item x, so it still has value x_0. T_2 tries to read x, so a read lock is requested. As the item is not yet updated, the write lock granted to T_1 still holds, and T_2 must wait. Once R_2 applies T_1's update, the write lock is released and T_2 is able to read value x_1, as expected, and commit. Note that R_2 was not updated when T_2 started, but distributed locks prevented T_2 from accessing an outdated value.

The observed behavior (transactions getting the last value of all database items) is not required by the 1SR correctness criterion (which only states that the effect of transactions must be equivalent to that of a serial order), but it is a consequence of the techniques used in the system implementation. However, these same techniques made those earlier systems slow and barely scalable.

Pedone et al. [19] proposed the substitution of the atomic commit protocol by a termination protocol based on atomic broadcast that performs the deferred update propagation [7, 22]. These improvements allowed to boost performance and scalability while correctness criterion was claimed to be one-copy serializability as traditionally defined. However, a significant difference with regard to Bernstein's implementation was introduced. Let's analyze the new behavior.

Example 2. Deferred update propagation and atomic broadcast The client starts transaction T_1 in replica R_1, where it updates the local copy of data item x, after acquiring the local write lock. At transaction termination, the deferred update propagation is done and thus the writeset, i.e. data item x and the updated value x_1, is broadcast to all replicas. A deterministic certification process is run independently at each replica before writeset application. If certification is successful, replica R_1 commits T_1 and the client immediately starts a new transaction T_2 for reading the updated item. But this new transaction is addressed to replica R_2. If R_2 has not yet started to apply T_1's writeset, data item x is not locked (assume there are no other transactions running) and T_2 is able to access x without waiting. But the accessed version x_0 corresponds to the last *locally* committed transaction that updated that item, and not to the last globally

certified transaction in the system. So T_2 gets an unexpected value and the client is not able to see the writes made by her previous transaction, which would not have been so if T_2 had executed at replica R_1. Nevertheless, 1SR is guaranteed: the effect of transactions is equivalent to that of the serial order T_2, T_1. The fact that T_1 precedes T_2 in real-time is not considered at 1SR. However, the client reads an outdated value, which clearly is not what she expected.

The difference between both systems lies in the distinct replica consistency maintained. While in the first system distributed locks prevent transactions from accessing outdated values, in the second one different values for the same data item are accessible at the same time at different replicas in the system. If these two systems ensure 1SR, it is because replica consistency was not considered in correctness criteria for database replication. This fact is interesting, as a similar concept was already defined and studied in the scope of distributed shared memory, surveyed in multiple papers [1, 16, 24].

4 Consistency Models in Distributed Shared Memory

Memory consistency models represent the way on which memories from different sites are synchronized in order to conform a distributed shared memory. The higher the level of consistency, the higher the synchronization and the fewer the divergences on values. According to Mosberger [16], the strictest level of consistency, *atomic consistency* [14] (a.k.a. *linearizability* [13]) considers that operations take effect inside an *operation interval*. Operation intervals are non overlapping, consecutive time slots. Several operations can be executed in the same slot; read operations take effect at read-begin time while write operations take effect at write-end time. Thus, read operations see the effects of all write operations of the previous slot but not those of the same slot.

Despite the simplicity of the idea and the easiness of application design, atomic consistency is often discarded due to the high associated cost that renders it impractical [24, 25]. Consequently, a more relaxed model is used in practice as the common correctness criterion: *sequential consistency* [15]. In a sequentially consistent system, all processes agree on the order of observed effects. According to Lamport [15], the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program.

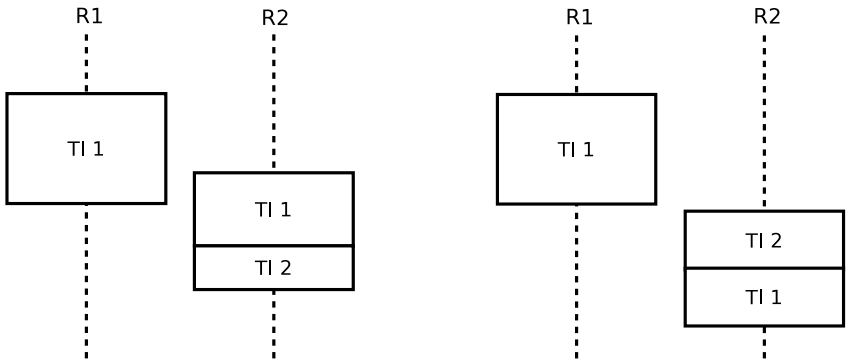
One of the key differences between these two consistency models is the *old-new inversion* between read operations [2]. Such issue may only arise in the sequential model. It consists in the following: once a process p writes a value v_n onto a variable x whose previous value was v_o, another process r_1 is able to read v_n while later a second reader r_2 reads v_o. Note that this is sequentially consistent since r_2 is still able to read afterwards value v_n, according to the total order associated with sequential consistency. However, such scenario violates atomic consistency since r_2's read has been made once the slot given to value v_n was already started and all reads in such slot should return value v_n (instead of v_o). This old-new inversion is exactly the situation arisen in previous Ex. 2.

5 Correspondence Between Memory Consistency Models and Database Replica Consistency

According to [10], one-copy equivalence can be guaranteed when all processes see all memory-related events in the same order. This implies that the DSM consistency models able to achieve one-copy equivalence are both the sequential and the atomic/linearizable ones. Following such trend, Mosberger [16] states that the one-copy serializability concept as defined by Bernstein et al. [6] is equivalent to sequential consistency (although, as previously demonstrated, the *implementation* of 1SR proposed in [5] actually provides atomic consistency). On the other hand, authors of [9] distinguish between strong consistency (which is explicitly associated to the atomic DSM model and mentioned as a synonym of one-copy equivalence) and weak consistency (where sequential consistency is included). As it can be seen, no agreement on the exact level of replica consistency needed to achieve one-copy equivalence exists nowadays.

Although memory consistency models only consider individual operations (writes and reads over variables in memory), a correspondence to a transactional environment can be easily established. Indeed, the concept of transaction matches very well the concept of operation interval. As explained in [9], a transaction T can be considered as a macro-operation from the DSM point of view, where all read operations can be logically moved at T's starting point (as read versions correspond to those available when the transaction started), and all write operations logically moved to T's termination point (when the transaction commits and its updates are visible for other transactions). Although transactions have been widely used in distributed systems, very few works have considered the DSM consistency model resulting from database replication protocols, being [9] one of such few exceptions.

From this new point of view, a database system featuring atomic consistency between replicas commits every transaction in such a way that its updates are applied in all sites at the same *transaction interval*, and all transactions issued in the next interval are able to see those updates. Transaction intervals start and end at the same logical time in all replicas, but not necessarily at the same real time. On the other hand, with sequential consistency, replicas go through the same sequence of database states but their transaction intervals are not guaranteed to start and end at the same logical time and so old-new inversion between read operations may arise. According to this, the replication system proposed by Bernstein and Goodman [5] provides atomic consistency, while in that presented by Pedone et al. [19] replica consistency is weakened from the atomic to the sequential level.



a. Atomic consistency b. Sequential consistency

Figure 1: Transaction intervals for Ex. 1 and 2. In the atomic consistency example, T_2's transaction interval does not start until that of T_1 finishes. With sequential consistency, transaction intervals may swap.

In Fig. 1, a visual representation of the transaction intervals for the analyzed examples is depicted. Vertical dotted lines represent real time at each replica, increasing downwards. In the atomically consistent system (Fig. 1a), transaction T_1 defines transaction interval $TI1$ in its delegate replica R_1. When it requests a distributed write lock for item x, transaction interval $TI1$ starts at the other replicas. Once atomic commit reaches consensus, replicas may apply the updates, but there is no real time restriction for doing so. This way, transaction intervals may last differently. When the same client starts transaction T_2 after T_1's completion, the correspondent transaction interval $TI2$ is not started at replica R_2 until $TI1$ finishes. This ensures that all changes performed during $TI1$ are visible in $TI2$. Note that if T_2 were executed at R_1, $TI2$ could have started immediately after $TI1$'s completion in that replica, which corresponds to an earlier real time than in R_2 (but to the same logical point). Note also that T_2 has no transaction interval at R_1, as it is a read-only transaction. On the other hand, in the sequentially consistent system (Fig. 1b), transaction interval $TI1$ is not started at R_2 until this replica launches the application of T_1's writeset, which may occur after an arbitrarily long time after T_1's completion at its delegate replica. Due to this, when client starts transaction T_2 at R_2, the corresponding transaction interval may be swapped before T_1's one, which prevents T_2 from reading the updated value.

Given that almost all existing protocols apply the set of update transactions in FIFO total order in all the replicas, they can be immediately tagged as sequentially consistent. Some of them are also able to avoid the old-new inversion for read accesses, and in that case they can be classified as atomically consistent

[3]. It is important to note that a deferred update replication model does not always mean a weak level of consistency, as several kinds of modern replication protocols, such as [18] and the strong 1SR of [27], have provided the same consistency level as in [5] (one-copy serializability with atomic consistency) relying on a sequential consistency model between replicas and adding several restrictions to transaction processing in order to avoid the old-new inversion. As the observable result from the point of view of the client is that of an atomic consistency model, we say that those systems are atomically consistent.

As a consequence of unwittingly but persistently ignoring replica consistency when characterizing database replication systems, what should be considered different correctness criteria are identified as the same criterion: one-copy serializability. This leads to ambiguity and unfairness when comparing different replication systems. The solution to this problem starts from refining the definition of correctness criteria, explicitly stating the two different kinds of consistency encompassed in a database replication system: the one being provided by the concurrency control mechanisms, able to ensure some transaction isolation level, and the other related to replica consistency. Our group already raised this topic when authors of [17], focusing on systems providing snapshot isolation (more popular than those providing serializability), reported the different names used in literature for referring to the same correctness criteria and proposed the replacement of the old ambiguous nomenclature for a new one, where it is explicitly denoted if the system is atomically or sequentially consistent.

6 Atomic Consistency in Transactional Contexts

Although it is true that maintaining strong consistency levels for individual operations is costly, the fact of using transactions that group arbitrarily large sets of operations into one unit of execution allows to reduce costs, thus making atomic consistency bearable in a transactional context. With transactions, synchronization between sites must be done only once for the full set of write operations included in a transaction. This writeset can be thus regarded as a macro write operation, as opposite to multiple individual write operations. The performance gains achieved by this grouping allow the usage of group communication systems that in turn allow to simplify algorithm designs and failure management (not considered in distributed shared memory in order to avoid even heavier models).

Transferring the concept of transaction into traditional memory consistency models makes it possible to provide the strongest level of consistency in database replication systems at a reasonable cost. Examples of database replication systems providing 1SR with atomic consistency can be found in [18, 20, 27].

Other existing systems also provide atomic consistency but increase performance by reducing transaction isolation from serializability to snapshot isolation (SI), which is enough for multiple database applications [8, 21].

7 Conclusions

Early database replication systems based on distributed locking and atomic commit protocols were defined as guaranteeing one-copy serializability [6], where the interleaved execution of transactions is equivalent to a serial execution of the same set of transactions on a one-copy –non replicated– database.

Later, performance-improved systems appeared, where distributed locking and atomic commit were substituted with deferred update propagation and atomic broadcast while maintaining the same correctness criterion of one-copy serializability. Although the theoretical definition of the criterion was the same, the consistency among replicas was actually different from that of the earlier systems. One-copy equivalence became then an ambiguous term but, up to our knowledge, such fact has not been stated nor justified yet, since these variations have been accepted under the one-copy equivalence umbrella[3].

This ambiguity stems from the fact of not considering replica consistency when defining correctness criteria for database replication systems. Memory consistency models can be borrowed from the distributed shared memory scope in order to clearly state the features of each system. Thus, although both system types provide one-copy serializability as defined in [6], systems based on distributed locking and atomic commit

[3] Although the consistency level provided by recent systems has been branded as weak, strong, strict, etc., in an attempt to better characterize it with regard to other so-called one-copy equivalent systems.

feature a stronger consistency level among replicas (atomic consistency) than those based on deferred update propagation and atomic broadcast (which generally provide sequential consistency).

We consider that this ambiguity is not trivial and that an explicit distinction must be made between atomically and sequentially consistent systems, as the actual DSM consistency model implemented by a replication protocol partially explains the complexity and performance of such protocol. Indeed, significant performance boosts can be achieved with minor DSM consistency relaxations. In addition to such performance implications, there are also important differences from the point of view of the user: in sequentially consistent systems, the *Read Your Writes* guarantee [26] is lost, as a transaction may be unable to read a value updated by a previous transaction of the same client.

A further remark can be done about atomic consistency. This model is commonly discarded in DSM due to the high cost of providing strong consistency to individual memory operations. But when bringing DSM consistency models to the transactional context, costs are reduced thanks to the fact of grouping multiple operations into one single transaction and thus the atomic model becomes bearable for those systems requiring a high level of replica consistency.

Acknowledgments

This work has been partially supported by EU FEDER and the Spanish MEC under grant TIN2009-14460, and by the Spanish MEC under grant BES-2007-17362.

References

[1] S. V. Adve and K. Gharachorloo. Shared Memory Consistency Models: A Tutorial. *IEEE Computer*, 29(12):66–76, 1996.

[2] H. Attiya. Robust Simulation of Shared Memory - 20 Years After. *Bulletin of the EATCS*, (100): 100–114, February 2010.

[3] H. Attiya and J. L. Welch. Sequential Consistency versus Linearizability. *ACM Trans. Comput. Syst.*, 12(2):91–122, 1994.

[4] H. Berenson, P. A. Bernstein, J. Gray, J. Melton, E. J. O'Neil, and P. E. O'Neil. A Critique of ANSI SQL Isolation Levels. In *Proceedings of the ACM SIGMOD Intnl. Conf. on Management of Data*, pages 1–10. ACM Press, 1995.

[5] P. A. Bernstein and N. Goodman. An Algorithm for Concurrency Control and Recovery in Replicated Distributed Databases. *ACM Transactions on Database Systems (TODS)*, 9(4):596–615, 1984.

[6] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987. ISBN 0-201-10715-5.

[7] M. J. Carey and M. Livny. Conflict Detection Tradeoffs for Replicated Data. *ACM Transactions on Database Systems (TODS)*, 16(4):703–746, 1991.

[8] K. Daudjee and K. Salem. Lazy Database Replication with Snapshot Isolation. In *Proceedings of the 32nd Intnl. Conf. on Very Large Data Bases (VLDB)*, pages 715–726. ACM, 2006.

[9] A. Fekete and K. Ramamritham. Consistency Models for Replicated Data. In B. Charron-Bost, F. Pedone, and A. Schiper, editors, *Replication. Theory and Practice*, volume 5959 of *Lecture Notes in Computer Science*, pages 1–17. Springer Berlin / Heidelberg, 2010.

[10] R. Guerraoui, B. Garbinato, and K. Mazouni. The GARF Library of DSM Consistency Models. In *ACM SIGOPS European Workshop*, pages 51–56, 1994.

[11] V. Hadzilacos and S. Toueg. A Modular Approach to Fault-Tolerant Broadcasts and Related Problems. Technical Report 94-1425, Department of Computer Science, Cornell University, May 1994.

[12] T. Härder and A. Reuter. Principles of Transaction-Oriented Database Recovery. *ACM Comput. Surv.*, 15(4):287–317, 1983.

[13] M. Herlihy and J. M. Wing. Linearizability: A Correctness Condition for Concurrent Objects. *ACM Trans. Program. Lang. Syst. (TOPLAS)*, 12(3):463–492, 1990.

[14] L. Lamport. On Interprocess Communication. *Distributed Computing*, 1(2):77–101, 1986.

[15] L. Lamport. How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs. *IEEE Trans. Computers*, 28(9):690–691, 1979.

[16] D. Mosberger. Memory Consistency Models. *Operating Systems Review*, 27(1):18–26, 1993.

[17] F. D. Muñoz-Escoí, J. M. Bernabé-Gisbert, R. de Juan-Marín, J. E. Armendáriz-Iñigo, and J. R. González de Mendívil. Revising 1-Copy Equivalence in Replicated Databases with Snapshot Isolation. In *OTM Conferences (1)*, volume 5870 of *Lecture Notes in Computer Science*, pages 467–483. Springer, 2009.

[18] R. C. Oliveira, J. Pereira, A. Correia Jr., and E. Archibald. Revisiting 1-Copy Equivalence in Clustered Databases. In *Proceedings of the ACM Symp. on Applied Computing (SAC)*, pages 728–732. ACM, 2006.

[19] F. Pedone, R. Guerraoui, and A. Schiper. Exploiting Atomic Broadcast in Replicated Databases. In *Proceedings of the 4th Intnl. Euro-Par Conf. on Parallel Processing*, volume 1470 of *Lecture Notes in Computer Science*, pages 513–520. Springer, 1998.

[20] L. Rodrigues, N. Carvalho, and E. Miedes. Supporting Linearizable Semantics in Replicated Databases. In *Proceedings of the 7th IEEE Intnl. Symp. on Networking Computing and Applications (NCA)*, pages 263–266. IEEE Computer Society, 2008.

[21] R. Salinas-Monteagudo, F. D. Muñoz-Escoí, J. E. Armendáriz-Íñigo, and J. R. González de Mendívil. A Performance Evaluation of g-Bound with a Consistency Protocol Supporting Multiple Isolation Levels. In *OTM Workshops*, volume 5333 of *Lecture Notes in Computer Science*, pages 914–923. Springer, 2008.

[22] M. K. Sinha, P. D. Nanadikar, and S. L. Mehndiratta. Timestamp Based Certification Schemes for Transactions in Distributed Database Systems. In *Proceedings of the ACM SIGMOD Intnl. Conf. on Management of Data*, pages 402–411. ACM Press, 1985.

[23] D. Skeen. Nonblocking Commit Protocols. In *Proceedings of the ACM SIGMOD Intnl. Conf. on Management of Data*, pages 133–142. ACM Press, 1981.

[24] R. C. Steinke and G. J. Nutt. A Unified Theory of Shared Memory Consistency. *J. ACM*, 51(5): 800–849, 2004.

[25] A. S. Tanenbaum. *Distributed Operating Systems*. Prentice-Hall, 1995. ISBN 0131439340.

[26] D. B. Terry, A. J. Demers, K. Petersen, M. Spreitzer, M. Theimer, and B. B. Welch. Session Guarantees for Weakly Consistent Replicated Data. In *Proceedings of the 3rd Intnl. Conf. on Parallel and Distributed Information Systems (PDIS)*, pages 140–149. IEEE Computer Society, 1994.

[27] V. Zuikevičiūtė and F. Pedone. Correctness Criteria for Database Replication: Theoretical and Practical Aspects. In *OTM Conferences (1)*, volume 5331 of *Lecture Notes in Computer Science*, pages 639–656. Springer, 2008.