Refinement of the One-Copy Serializable Correctness Criterion

M. I. Ruiz-Fuertes, F. D. Muñoz-Escoí

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

{miruifue,fmunyoz}@iti.upv.es

Technical Report ITI-SIDI-2011/004

Refinement of the One-Copy Serializable Correctness Criterion

M. I. Ruiz-Fuertes, F. D. Muñoz-Escoí

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

Technical Report ITI-SIDI-2011/004

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

November, 2011

Abstract

Correctness in early replicated distributed database systems was usually defined as one-copy serializability. A database spread over multiple sites must behave as a unique node, and the result of concurrently executing transactions must be equivalent to that of a serial execution of the same set of transactions. In order to ensure such 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 maintaining 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. However, enforced guarantees were subtly but significantly modified. This paper proposes a refinement of the correctness criterion of one-copy serializability by defining two new correctness criteria enclosed on it. With these refined criteria, the different guarantees provided by the analyzed systems can be precisely stated.

1 Introduction

Distributed and replicated database systems appeared in order to provide a higher level of availability and fault tolerance than existing stand-alone (centralized, i.e. non-replicated) databases, while still ensuring a correct management of data. Traiger et al. [1] suggested the concepts of one-copy equivalence and location, replica, concurrency, and failure transparencies, as desirable features for distributed systems. Bernstein, Hadzilacos and Goodman later defined one-copy serializability (1SR) [2] as a correctness criterion. According to it, the interleaved execution of users' transactions must be equivalent to a serial execution of those transactions on a stand-alone database. 1SR turned immediately to be the accepted correctness criterion for database replication protocols. In order to ensure such guarantees, a conservative approach inherited from stand-alone database systems was followed [3]. Thus, write operations over any data item had to acquire write locks in all its copies prior to updating 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 are required in order to reach a consensus among all participating sites for each transaction commitment, which further penalized performance and scalability.

Several optimizations were added later to database replication systems, trying to provide correct, i.e. 1SR, replication at a reasonable cost. According to the deferred update replication model [1,4], transactions

were processed locally at one server and, at commit time, were forwarded to the rest of system nodes for validation. This optimization saved communication costs as synchronization with other nodes was only done at transaction termination during the atomic commit protocol. Another important optimization consisted in substituting the atomic commit protocol for an atomic (total order) broadcast [5, 6], whose delivery order was then used as a serialization order for achieving 1SR. Wiesmann and Schiper [7] provide a performance comparison of database replication techniques and conclude, among other insights, that techniques based on total order broadcast significantly outperform traditional database replication protocols like distributed locking, and that this performance difference is larger if the network is slow and subject to contention. This is due to the fact that distributed locking requires many messages, while techniques based on total order broadcast demand less networking resources, typically by restricting interactions to a single broadcast, which makes these techniques very efficient with a slow network.

According to the definition of 1SR [2], replication protocols based on atomic commit and those based on atomic broadcast ensure the same correctness criterion. However, carefully analyzing the behavior of both systems, some differences can be observed. In a traditional stand-alone system, a user could transfer certain amount of money from one account A to another B, commit the transaction, and read the increased balance in B in the following transaction. The same is ensured in replicated systems based on distributed locking and atomic commit, precisely due to these inherited techniques. As both systems behave identically, the user will then not be able to distinguish between the stand-alone and the replicated one. Moreover, the user will consider this behavior correct, as it perfectly matches their expectations. However, the deferred update propagation and the atomic broadcast of an optimized replicated system may cause that the same operation ends in a worried user when, after committing the bank transfer, they temporarily read the old balance in B, as if the money had disappeared (we show later in the paper why this may happen). Clearly, the user might consider this behavior incorrect until, after some time, the increased balance can be finally read. This situation would never have occurred in a traditional stand-alone database. Does this mean that the system based on atomic broadcast is not behaving as one copy as it claims to do? Are then one-copy serializable database replication systems actually behaving as one copy?

In this paper, we demonstrate that some consistency-related features significantly changed when atomic commit protocols were replaced by termination protocols based on atomic broadcast. These visible differences in consistency, admitted in 1SR, might originate some confusion among users. To show this, we made a historical review through different 1SR systems and chose two representative database replication system models, which are analyzed through a case study to reveal the differences between them. As a solution, and inspired in the memory consistency models used in distributed shared memory (DSM), we define new correctness criteria for distinguishing among different types of 1SR database replication.

The rest of the paper is structured as follows. Section 2 presents other works that relate to this one. Section 3 details the assumed system model and provides basic definitions. Section 4 states the ampleness of the term *one-copy serializability*, by showing important differences in consistency between two systems providing 1SR. Section 5 presents and compares the two strictest memory consistency models used in the scope of DSM. Inspired in the fundamental difference between those models, Section 6 presents a complete formalization of our proposed criteria. Section 7 gives a final discussion.

2 Related Work

Several previous papers [8, 9, 10, 11, 12] noted that 1SR admits multiple consistency degrees. Due to this, different authors have qualified 1SR with additional requirements and different names in order to capture database users' expectations. For instance, Breitbart, Garcia-Molina and Silberschatz [8] defined *strong serializability*; Daudjee and Salem [9] proposed *strong 1SR* and *strong session 1SR*; Zuikevičiūtė and Pedone [10] adopted the use of *strong serializability* and *session consistency*; and Kemme et al. [11, 12] explained the meaning of the terms *strong consistency* and *weak consistency*. However, as far as we know, none of the previous works meets the objectives of this paper, namely: (a) provide a complete and general overview of this matter that clarifies the underlying reason why 1SR needed to be qualified or refined; (b) provide a careful formalization of the different approaches with precise definitions and theorems; and (c) survey existing database replication systems in order to classify them regarding their interpretation of 1SR.

Similar refinements to the one proposed here for 1SR can be done with other isolation levels, such as

snapshot isolation [13, 14]. In this line, our group surveyed existing works and proposed uniform names for the correctness criteria of replicated databases that work with snapshot isolation [15].

The concept of *serializability with external consistency*, used in general distributed systems (not specifically oriented to database replication systems) and treated in different papers [16, 17], would correspond with the strictest interpretation of the database correctness criterion of 1SR. Such works, however, neither provide a careful formalization for that concept.

3 System Model and Definitions

The replication systems analyzed in this paper consider a partially synchronous distributed system composed of database servers, also called sites, nodes or replicas, $R_1, R_2, \ldots, R_n$, which store data, and clients or users[1] that contact these servers to access the data. Communication between components is based on message passing, as they do not have shared memory. Consequently, they neither have a global clock. Servers can also communicate through atomic broadcast, described below.

From the point of view of the user, a database is a collection of logical data items. Each logical data item is physically stored at the servers. A local Database Management System, or DBMS for short, runs at each site and is responsible for the control of the data. 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 [18]. The set of logical items a transaction reads is called the readset. Similarly, the writeset is the set of logical items written by a transaction, and it also includes the updated or inserted values. A transaction is called a query (or a read-only transaction) if it does not contain any write operation; otherwise it is called an update transaction. In the Read One Write All Available (ROWAA) [2] approach, queries only need to access one replica, while update transactions are required to modify all available replicas.[2] Database users are provided with the concept of session [19], in order to logically group a set of transactions from the same user. Transactions from different users belong to different sessions. However, it can be left to the user the decision of using one or multiple sessions to group their transactions. Transactions of the same user session are submitted sequentially, but may be addressed to different replicas. The replica to which a transaction is addressed is called the delegate replica for that transaction and it is responsible for its execution. The rest of the nodes are remote replicas for that transaction. Each client request is only processed by its delegate, which later transfers the updates to the remote nodes. In other words, a passive, update-everywhere replication is performed.

Transactions may be executed concurrently in a DBMS, raising conflicts. Two transactions conflict if they have conflicting operations. Two operations conflict if they are issued by different transactions, access the same data item and at least one of the operations is a write. The local concurrency control of the DBMS establishes which executions of concurrent transactions are correct or legal regarding certain isolation level. The strictest isolation level that a DBMS can provide is the serializable isolation level [20]. This level guarantees a completely isolated execution of transactions, as if they were serially performed, one after the other. Changes made by transaction T are only visible to other transactions after the commitment of T. On the other hand, if T aborts, then its changes are never seen by any other transaction. The serializable isolation level can be achieved, e.g., by the strict two-phase locking (2PL) [2].

To ensure consistent termination of 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. A widely used atomic commit protocol is the two-phase commit protocol (2PC) [21, 22], which involves two rounds of messages for reaching a consensus on the termination of each transaction. 2PC can be centralized or decentralized. In the centralized approach, the coordinator first sends a message to the rest of nodes, with information about the ending transaction. Each server must then reply to the coordinator whether it agrees or not to commit the transaction. If all replies are positive, the coordinator sends a commit message and waits for acknowledgments from all the nodes. If any of the replies was negative, an abort message is sent in the second phase. The decentralized version is similar but with any server starting the process and with responses to every other server.

[1] We employ *user* to specifically refer to a human agent.

[2] Unavailable replicas are updated once they recover.

Atomic broadcast (abcast for short) 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 [23]. (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 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. Due to this last property, atomic broadcast is also known as total order broadcast.

Either through atomic commit protocols or with the aid of atomic broadcast, applying the writesets and correctly managing conflicting operations are necessary actions for maintaining the correctness criterion and required level of replica consistency. The systems considered in this paper guarantee the correctness criterion of 1SR. Replica consistency measures the synchronization among the copies of the same data item. Different levels of replica consistency may be enforced.

4 1SR: an Ample 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.

This is the case of the distributed concurrency control protocol described by Bernstein and Goodman [3]. It is based on a serializable isolation in local nodes (implemented with 2PL), distributed locking and two-phase commit (2PC), and follows the ROWAA approach and the one-copy serializability criterion. We will refer to this general model as a *DLAC*[3] system. The 2PC protocol used for transaction termination ensures that all replicas agree on the set of committed transactions and a serializable history is guaranteed. However, due to distributed locking and atomic commit, replicas do not only commit the same set of transactions but do it in such a way that any new issued transaction is able to see all the changes performed by previous transactions, with independence of the replica where this new transaction is executed. That is, the system nodes behave as a traditional one-copy database in that any copy of a written data item is updated before any subsequent access to that copy is allowed: a transaction T always gets the most recent vision of the database, as created by the last transaction executed in the system immediately before T. Let us consider a simple database execution to illustrate this feature.

Execution 1 (DLAC systems, Figure 1(a)) *A user 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 the commitment of T_1 in R_1, the same user, who wants to be sure about the success of their updates, starts a query, T_2, for reading x. T_2 is also started in replica R_1, where value x_1 is already available. T_2 reads x_1, as expected, and finishes. Suppose that replica R_2 is slower and it did not yet apply the new value to data item x, so it still has value x_0. Now the user starts a second query T_3 for reading x again and, due to some load balancing, T_3 is run at replica R_2. When T_3 tries to read x, a read lock is requested. As the item is not yet updated, the write lock granted to T_1 still holds, and T_3 must wait. Once R_2 applies the update of T_1, the write lock is released and T_3 is able to read value x_1, as expected, and commit. Note that R_2 was not updated when T_3 was launched, but distributed locks prevented T_3 from accessing an outdated value. Both T_2 and T_3 were able to access the value updated by T_1, which is the expected behavior (traditional stand-alone databases provide this same effect). The equivalent serial order of this execution is T_1, T_2, T_3 or T_1, T_3, T_2.*

The observed behavior, i.e. transactions getting the last value of all database items, is not required by the 1SR correctness criterion. Indeed, serializability (and thus 1SR) only states that the effect of transactions must be equivalent to that of *any* serial order. No restriction is imposed over this serial order. In

[3] Distributed Locking and Atomic Commit.

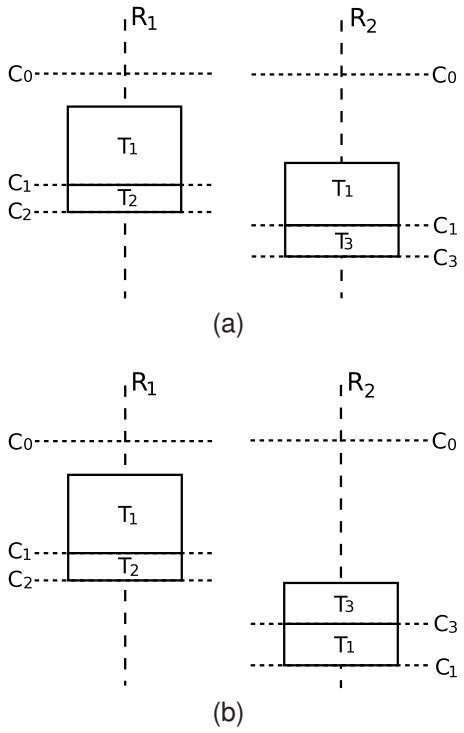

Figure 1: Diagrams for Executions 1 and 2. Two replicas are visualized: R_1 and R_2. Vertical broken lines represent real time at each replica, increasing downwards. Horizontal dotted lines represent commit operations in each replica. A transaction T_0 is assumed to create the initial database state. In DLAC systems (a), T_3 does not start in R_2 until T_1 finishes in this node. In DUABLOQ systems (b), however, transactions may swap. (Note that T_2 is not executed at R_2, and neither does T_3 at R_1, as they are read-only transactions in a ROWAA system.)

particular, no real-time consideration is taken into account, such as respecting the order on which transactions were started by the user. In a traditional stand-alone database providing serializability, however, if a transaction T_2 started *after* the termination of conflicting transaction T_1, the equivalent serial order necessarily respected this precedence (there is only *one* node executing all transactions). Just like traditional stand-alone systems, DLAC systems are more restrictive than the correctness model they follow. The stricter observed behavior of stand-alone systems was due to the physical laws of time, while that of DLAC systems is a consequence of the techniques used in their implementation. However, these same techniques made those earlier systems slow and barely scalable.

More modern systems [5, 6] proposed the substitution of the atomic commit protocol by a termination protocol based on atomic broadcast that performs the deferred update propagation.[4] In these systems, which also follow the ROWAA approach, transactions are locally executed at their delegate replica. When a transaction ends, its readset and writeset are sent to the rest of replicas in total order, using the atomic broadcast. After delivery, if no conflict is found in the certification, the transaction commits at every replica. Read-only transactions, or queries, do not need to be broadcast: they are committed locally and are inserted in an appropriate position inside the serialization order. These *DUABLOQ*[5] systems performed significantly better than previous DLAC ones [7]. These improvements were possible thanks to the adoption of the full replication model,[6] where all replicas store a complete copy of the database, instead of the more general model of distribution with partial replication, which forced to use costly distributed transactions. This new

[4]Deferred update propagation was already used in the distributed certification scheme proposed by Sinha, Nanadikar and Mehndiratta [4] or in the distributed optimistic two-phase locking, O2PL, of Carey and Livny [24].

[5]Deferred Updates, Atomic Broadcast and Locally Ordered Queries.

[6]Partial replication is also possible in DUABLOQ systems.

approach allowed replication systems to boost performance and scalability while the correctness criterion was claimed to be one-copy serializability as traditionally defined. However, a significant difference with regard to DLAC systems was introduced. Let us analyze the new behavior with the same client requests.

Execution 2 (DUABLOQ systems, Figure 1(b)) *A user 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 same user immediately starts a query T_2 for reading the updated item. T_2 is directed to replica R_1 and so it reads the updated value and finishes. But again, the user starts a second query T_3 that is addressed to replica R_2. Similarly to the DLAC case, R_2 agrees on committing T_1 but has not yet started to apply the writeset of T_1. The key difference is that now, and assuming that there are no other transactions running, data item x is not locked (supposing that the concurrency control is based on locks) or otherwise protected and T_3 is able to access x without waiting. 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_3 gets an unexpected value and the user, who was previously able to read the updated item with T_2, is now not able to see it with the following transaction T_3. Nevertheless, 1SR is guaranteed: the effect of transactions is equivalent to that of the serial order T_3, T_1, T_2. The fact that T_1 precedes T_3 in real time is not considered in 1SR. However, the user reads an* outdated *value, which clearly is not what they expected.*

There are indeed two possibilities for this situation in locking-based systems: either the certification involves getting the necessary locks and R_2 has not yet run the certification for T_1, or the certification is based on history logs, required locks are only set on writeset application and such writeset application has not yet started. The sample execution depicts the second case, but both possibilities have the same result: T_3 will read the old value of x and T_1 will finally commit in R_2. In any case, the origin of the inversion is two-fold: first, locks are locally and independently managed (there is neither a distributed locking mechanism nor replicas must confirm each other the acquisition of locks for a certification consensus); and second, replicas perform at different paces and some nodes may be behind in their writeset application, allowing new transactions started at such nodes to *overtake* already broadcast writesets. The first cause is inherent to the mechanisms of these protocols. The second one, however, depends on the workload, the differences of performance between servers, etc. This way, inversions will not always appear and they are not the common behavior of DUABLOQ protocols. Nevertheless, they are admitted when the above conditions occur: the stricter behavior of both traditional stand-alone and DLAC systems is no longer enforced. In DUABLOQ systems, 1SR is guaranteed to the letter, but not further. The difference between DLAC and DUABLOQ systems lies in the distinct replica consistency maintained.[7] While in DLAC systems distributed locks prevent transactions from accessing outdated values, in DUABLOQ systems different values for the same data item are accessible at the same time at different replicas. Both system models ensure 1SR because replica consistency was not considered in correctness criteria for database replication. Indeed, the definition of 1SR does not deal with the level of synchronization between replicas, leaving this aspect open to the system designer. The fact of not enforcing any replica consistency level renders the definition ample enough to allow several interpretations. Although this can now lead to ambiguity, that was not the case back when 1SR was defined, as existing techniques only allowed one possibility. However, although different techniques appeared later, no specification regarding replica consistency was added to the definition. This fact is interesting, as a similar concept was already defined and studied in the scope of distributed shared memory, surveyed in multiple papers [25, 26, 27].

5 DSM Consistency Models

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

[7] Note that these executions show a unique user performing all accesses. This is only for the sake of a clearer perception of the problem, represented as a user getting different values from an item that they updated in an immediately previous transaction. But the consistency problem is the same if transactions are from different users.

synchronization and the fewer the divergences on values. According to Mosberger [25], the strictest level of consistency, *atomic consistency* [28] (a.k.a. *linearizability* [29]) 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 design for applications that use atomic memory, this consistency level is often discarded due to its high associated cost [27, 30]. Consequently, a more relaxed model is used in practice as the common level: *sequential consistency* [31]. In a sequentially consistent system, all processors agree on the order of observed effects [25]. According to Lamport [31], 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.

The key difference between these two consistency models is the *old-new inversion* between read operations [32]. Such issue is precluded in the atomic model but it may arise in the sequential one. 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 reads x 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 Execution 2.

6 Refinement of 1SR

According to Guerraoui, Garbinato and Mazouni [33], in general distributed executions (i.e. not necessarily related to DBMSs), 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 ones. Following such trend, Mosberger [25] states that the one-copy serializability concept as defined by Bernstein et al. [2] is equivalent to sequential consistency. On the other hand, Fekete and Ramamritham [34] distinguish between strong consistency (which is explicitly associated with 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 (write or read a variable in memory), a correspondence to a transactional environment can be established. Indeed, the concept of transaction matches very well the concept of operation interval. As explained by Fekete and Ramamritham [34], a transaction T can be considered as a macro-operation from the DSM point of view, where all read operations can be logically moved to the starting point of T (as read versions correspond to those available when the transaction started),[8] and all write operations logically moved to the termination point of T (when the transaction commits and its updates are visible for other transactions). Thus, we can consider the interval of time where all transaction operations are done as a *transaction interval*.

The fact that a transaction is a macro-operation also reduces the high costs that led to dismiss atomic consistency in DSM. 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 amortize the cost of the operations, 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 is thus regarded as a macro write operation, as opposite to multiple individual write operations.

Although transactions have been widely used in distributed systems, very few works have considered the DSM consistency model resulting from database replication protocols, being that of Fekete and Ramamritham [34] one of such few exceptions. We consider that this is an interesting line of action, as

[8] We can safely assume that a transaction will never read a data item after writing it or that the application is able to cache written versions and serve those read operations without accessing the database.

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.

Inspired in the idea of a transaction as a macro-operation, and adapting Mosberger's concept of operation interval [25] to its use with transactions, we now present a complete formalization in order to refine 1SR with two new correctness criteria, by distinguishing among the possible replica consistency levels. In the same manner as old-new inversions allow us to distinguish between the atomic and the sequential memory consistency levels, we will distinguish among the resulting correctness criteria by means of the presence or absence of similar phenomena in the executions of the system. For this we consider database replication systems that guarantee 1SR.

Starting from the Chapter 8 of the book by Bernstein et al. [2], if a system guarantees 1SR, then a complete replicated data history H_{RD} (RD history) over a set of transactions executed in that system is 1SR, i.e., it is view equivalent to a serial one-copy history H_{1C} (1C history): (a) H_{RD} and H_{1C} have the same reads-from relationships, i.e., T_k reads from T_j in H_{RD} if and only if the same holds in H_{1C}; and (b) for each final write in H_{1C}, there is a final write in H_{RD} for some copy of the same data item. The equivalent 1C history is serial: for every pair of transactions, all the operations of one transaction are executed before any of the operations of the other. From the point of view of users, transactions in a serial execution seem to be processed atomically, in what we identify as an interval. We now provide the foundations of our reasoning model, based on the concept of intervals. Next, by requiring some extra conditions over the sequence of intervals of the equivalent 1C history, we will qualify two new criteria that are enclosed in 1SR.

Definition 1 (Interval) *Let H_{1C} be a serial 1C history, and T_j a transaction committed in H_{1C}. We define the interval of T_j in H_{1C}, $I(T_j)$, as the block composed by the operations of T_j in H_{1C}. As H_{1C} is serial, it constitutes a sequence of the intervals of all committed transactions.*

Note that aborted transactions do not present intervals.

Definition 2 (Interval order) *Let H_{1C} be a serial 1C history. We define the interval order of H_{1C}, $<_i$, as the total order in which the intervals appear in the sequence defined by H_{1C}.*

Definition 3 (Conflicting transactions) *Two transactions conflict if they have conflicting operations. Two operations conflict if they belong to different transactions, they access the same data item and at least one of the operations is a write.*

Definition 4 (Independent transactions) *Two transactions that do not conflict are called independent transactions. Multiple transactions compose a set of independent transactions if they are independent two by two.*

Independent transactions may be executed at any order: as they do not conflict, the results of the transactions and the final state of the database will be the same at any possible execution order. However, when transactions conflict, the order in which they are executed is important, as the results of the transactions and the final state of the database depend on this execution order. One of all possible equivalent serial orders of a set of conflicting transactions must be chosen. 1SR admits any of them, but from the point of view of the user, the natural and intuitive order is the order in which transactions have been started in real time.

Definition 5 (First start) *We say that a transaction T_j first-starts when it accesses any item of the replicated database for the first time through any system replica R_i (the delegate replica).*

Real time must be considered in order to capture the point of view of users. However, this does not impose any real-time constraints on the system, nor it requires any notion of global clock among replicas.

Note also that in a locking-based concurrency control all operations over database items must be preceded by the acquisition of the necessary locks. A transaction whose initial operation is held in the database while waiting for the required locks is not considered as first-started until those locks are granted and the first data item is effectively accessed. This way, the versions of the data that a transaction reads depend on the time when such transaction first-started.

After executing all its operations, if no abortion occurs, a transaction asks for commitment. In the Read One Write All Available (ROWAA) [2] approach, each committed transaction T_j must apply its updates at every available replica, which can be considered as multiple commit operations for the same transaction. In a quorum approach, only a subset of the replicas must be updated. In any case, one of the replicas, R_i, will be the first to complete the commit operation and make the updates of T_j visible to local transactions starting at R_i. This instant is when T_j first-commits.

Definition 6 (First commit) *We say that a transaction T_j first-commits when it commits in the replicated database for the first time, through any system replica R_i.*

Definition 7 (RT precedence) *We define a partial, transitive order of real-time precedence, $<_{rt}$, in the following way: a transaction T_j precedes transaction T_k in real time, $T_j <_{rt} T_k$, if and only if T_j first-commits before T_k first-starts.*

Definition 8 (Concurrent transactions) *Two transactions T_j and T_k are concurrent if both $T_j <_{rt} T_k$ and $T_k <_{rt} T_j$ are false, i.e., there is no real-time precedence between them.*

Definition 9 (Alteration) *Let T_j and T_k be two committed transactions in the serial 1C history H_{1C}. Let T_j precede T_k in real time, $T_j <_{rt} T_k$. We say that an alteration occurs in H_{1C} if T_k precedes T_j in the interval order of H_{1C}, $T_k <_i T_j$, i.e., the interval order of H_{1C} is inconsistent with the real-time precedence of the transactions. T_k is called the altered transaction, while T_j is the overtaken transaction.*

Definition 10 (RTC precedence) *We define a partial order called real-time & conflict precedence, $<_{rtc}$, as the transitive closure of the relationship that includes all pairs $T_j <_{rtc} T_k$ such that (a) $T_j <_{rt} T_k$, and (b) T_j and T_k are conflicting transactions.*

Definition 11 (Inversion) *Let T_j and T_k be two conflicting transactions, committed at the serial 1C history H_{1C}. Let T_j precede T_k in RTC order, $T_j <_{rtc} T_k$. We say that an inversion occurs in H_{1C} if T_k precedes T_j in the interval order of H_{1C}, $T_k <_i T_j$, i.e., the interval order of H_{1C} is inconsistent with the RTC precedence of the transactions. T_k is called the inverted transaction, while T_j is the overtaken transaction. More simply, an inversion is an alteration between two conflicting transactions.*

From the point of view of the users, only the alterations of conflicting transactions, i.e., only the inversions, are detectable. Imagine transactions T_a, T_b and T_c, started by the same user one after the commitment of the other, i.e., $T_a <_{rt} T_b <_{rt} T_c$. Let T_a contain only a write operation over item x, and let T_b and T_c be two read-only transactions. Suppose T_b reads x and T_c reads y. This way, the RTC precedence over this set of transactions is composed by only one relationship: $T_a <_{rtc} T_b$. Indeed, the user expects T_b to run after T_a and thus get the updated value of x, while T_c is completely independent of the other transactions and could be executed at any place: the result would be exactly the same.

The phenomenon of old-new inversion presented in Section 5 corresponds to an inversion of a transaction that overtakes the update transaction from which it was supposed to read a value.

Definition 12 (1ASR correctness criterion) *A complete RD history H_{RD} over a set of transactions is 1ASR if it is view equivalent to a serial 1C history H_{1C} which respects the RTC precedence, i.e., which contains no inversions.*

The 'A' in the name stands for "atomic" as, from the point of view of the users, the results of the execution of a set of transactions are equivalent to those that one could obtain by ensuring that a transaction is committed at every replica before starting the next transaction. This way, users perceive an *atomic* nature in the replica consistency of the system.[9] Informally, 1ASR restricts the set of all possible equivalent serial orders that would comply with 1SR, so that orders violating RTC are not considered. This is the case of the DLAC systems described in Section 4. We can also say that a database replication system provides the correctness criterion of 1ASR if the user cannot differentiate it from a traditional stand-alone system even when spreading their accesses all over the set of replicas.

[9]However note that 1ASR is less strict than the atomic DSM level as it only maintains the property of the absence of inversions.

To implement a 1ASR system, no global clock is necessary. Even if the system cannot decide whether two operations of independent transactions happen in one or the other order when the client and the replica processes are all different, there are mechanisms that allow the system to respect the real-time ordering of transactions. Distributed locks are an example. Moreover, several implementation approaches can be followed in order to construct a 1ASR system. As a first approach, all alterations can be avoided by totally ordering transactions before starting execution, and executing them sequentially following that order, in such a way that a transaction must have committed at all available and necessary replicas before the next transaction is started. Such a total order can be determined by a sequencer, by requiring transactions to acquire locks at each replica they must access, or by assigning starting timestamps to transactions. However, avoiding all alterations reduces concurrency and performance. When two transactions are independent, the database state after committing both transactions will be the same either with a serial order T_1, T_2 or T_2, T_1. The system could execute both transactions concurrently to increase performance without implications on perceived consistency. This is the approach followed by DLAC systems, which allow alterations[10] but avoid inversions. While there are additional approaches for implementation, the common objective of all of them is to provide users with an atomic image, either pessimistically (avoiding alterations or inversions) or optimistically (aborting a transaction whose commit would create an inversion).

Examples of systems that provide 1ASR are the algorithm by Bernstein and Goodman [3], the distributed two-phase locking, wound-wait, basic timestamp ordering and distributed certification algorithms surveyed by Carey and Livny, along with the distributed optimistic two-phase locking algorithm they propose [24], the strongly serializable DBMSs of Breitbart, Garcia-Molina and Silberschatz [8], the first three algorithms proposed by Agrawal et al. [6], Pronto [35], the refined OTP algorithm [36], DBSM-RO-opt and DBSM-RO-cons [37], the WCRQ algorithm [38], and BaseCON for strong 1SR [10].

1ASR is suitable for applications that require a strict level of consistency and cannot tolerate any inversion. However, avoiding all inversions may affect performance. We can relax the imposed conditions by only requiring that individual users do not suffer inversions. This way, even if some inversions are allowed, individual users perceive an atomic image, as if the system were 1ASR.

Definition 13 (Projection of a history) *We define the projection of a serial 1C history for a set of transactions $\mathbb{T}$ as the result of deleting from the history all intervals corresponding to transactions not belonging to $\mathbb{T}$.*

Definition 14 (Session) *Database users are provided with the concept of session, in order to logically group a set of transactions from the same user. Transactions from different users belong to different sessions. However, it can be left to the user the decision of using one or multiple sessions to group their transactions.*

Definition 15 (Session projection) *For a given session S_i, composed by a set of transactions $\mathbb{T}_{S_i}$, the projection of a serial 1C history for the set of session transactions is called the session projection of that history for S_i.*

Definition 16 (1SR+ correctness criterion) *A complete RD history H_{RD} over a set of transactions is 1SR+ if it is view equivalent to a serial 1C history H_{1C} such that, for each existing session S_i, the session projection of H_{1C} for S_i respects the RTC precedence.*

The extra requirement of 1SR+ (inversion preclusion in sessions) further prevents the occurrence of undesirable conditions over the ensured 1SR foundation: the equivalent serial order must respect RTC precedence over the transactions of the same user session. From the point of view of an individual user grouping all their transactions within the same session, the system cannot be distinguished from a system guaranteeing 1ASR.

Examples of systems that provide 1SR+ (but do not provide 1ASR) are the Block and Forward algorithms by Daudjee and Salem [9] and the BaseCON algorithm for SC [10].

Finally, it is obvious that not all applications have the same requirements. Those not sensitive to real-time precedence do not need to care about such issue: 1SR is enough. This is the case of the DUABLOQ

[10] Messages from 2PC are not ordered and different independent transactions may execute the atomic commit protocol at the same time, possibly committing in different orders at different replicas.

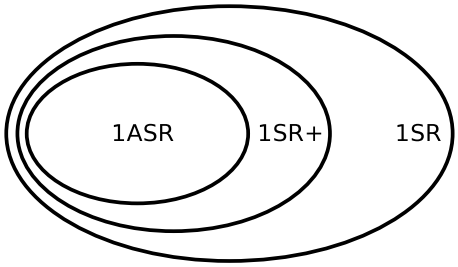

Figure 2: Refinement of the 1SR correctness criterion

systems described in Section 4. If the commit order followed by each replica matches the delivery order of writesets of the total order broadcast, then the replica consistency of such DUABLOQ system is equivalent to the sequential DSM model.[11] Nevertheless, this is not necessary (specifically, for independent writesets [39]) and, thus, more relaxed consistency levels are admitted in 1SR.

Examples of systems providing 1SR (but not 1SR+ or 1ASR) are the last algorithm proposed by Agrawal et al. [6], the lazy transaction reordering protocol by Pedone, Guerraoui, and Schiper [40], the initial version of OTP by Kemme et al. [41], the fast refresh algorithms by Pacitti, Minet and Simon [42], DBSM [39], the SER and the hybrid algorithms by Kemme and Alonso [43], NODO and REORDERING [44], DBSM-RAC [45], the epidemic algorithms by Holliday, Agrawal and El Abbadi [46], the OTP-Q, OTP-DQ and OTP-SQ protocols [36], DBSM* [47], the one-at-a-time and many-at-a-time algorithms by Schiper, Schmidt and Pedone [48], *k*-bound GSI [49], Mid-Rep [50], the MPF and MCF algorithms [51], and the BaseCON for 1SR [10].

Both 1ASR and 1SR+ histories are included in the set of 1SR histories. Moreover, 1ASR histories are a subset of 1SR+ histories. Additionally, all three sets are different. This corresponds to Figure 2 and is formalized in the following theorems.

Theorem 1 *The set of 1ASR histories is a subset of the set of 1SR histories.*

Proof 1 *By definition 12, a 1ASR history is view-equivalent to a serial 1C history, thus also satisfying the 1SR correctness criterion.*

Theorem 2 *The set of 1SR+ histories is a subset of the set of 1SR histories.*

Proof 2 *By definition 16, a 1SR+ history is view-equivalent to a serial 1C history, thus also satisfying the 1SR correctness criterion.*

Theorem 3 *The set of 1ASR histories is a subset of the set of 1SR+ histories.*

Proof 3 *By definition 12, a 1ASR history is view-equivalent to a serial 1C history H_{1C} that respects the RTC precedence. Any projection of H_{1C} (consisting in removing intervals from the history) will also respect the RTC precedence. In particular, any session projection of H_{1C} will respect the RTC precedence, thus also satisfying the 1SR+ correctness criterion.*

Theorem 4 *The set of 1ASR histories is not equal to the set of 1SR+ histories.*

Proof 4 *The conditions satisfied by 1SR+ histories are less strong than the conditions required by 1ASR: even if every session projection of the equivalent serial 1C history H_{1C} of a 1SR+ system respects RTC, H_{1C} (which includes all the transaction intervals) is not guaranteed to also respect RTC. Thus it is possible to find systems that produce histories that guarantee 1SR+ but do not guarantee 1ASR, as listed before.*

Theorem 5 *The set of 1ASR histories is not equal to the set of 1SR histories.*

[11] Note that in the general case total-order broadcasts also respect the FIFO order needed by sequential consistency.

Proof 5 *No restriction is imposed over the serial 1C history equivalent to a 1SR history. Particularly, RTC is not guaranteed to be respected. Thus it is possible to find systems that produce histories that guarantee 1SR but do not guarantee 1ASR, as listed before.*

Theorem 6 *The set of 1SR+ histories is not equal to the set of 1SR histories.*

Proof 6 *No restriction is imposed over the serial 1C history equivalent to a 1SR history. Particularly, RTC is not guaranteed to be respected in every session projection. Thus it is possible to find systems that produce histories that guarantee 1SR but do not guarantee 1SR+, as listed before.*

We have defined two correctness criteria that refine 1SR. The resulting three criteria cover the different possible interpretations regarding replica consistency as perceived by users: preclusion of inversions (1ASR), preclusion of inversions within sessions (1SR+), and possibility of inversions (1SR). As noted before, there are different names in the literature that would correspond to the correctness criteria formalized here: e.g., 1ASR would be equivalent to *strong 1SR* and 1SR+ would match *strong session 1SR* [9, 10].

7 Discussion and Conclusions

Serializability is the highest level of transaction isolation. Under it, the effect of concurrently executing transactions must be the same as that of *any* serial execution of those transactions. Traditional stand-alone databases providing serializability were actually more restrictive than required by the model: the equivalent serial execution never presented alterations with regard to real-time precedence. Database users became accustomed to this stricter behavior, which ensures that a transaction executed after another is able to see the effects of that previous transaction. This became a fundamental principle of stand-alone systems from the users' point of view, while never required by any correctness criterion.

When databases were distributed and replicated for increased performance, one-copy equivalence was added to serializability to define one-copy serializability (1SR) [2], under which the interleaved execution of transactions must be equivalent to a serial execution of the same set of transactions on a one-copy (non-replicated) database. Early database replication systems based on distributed locking and atomic commit (DLAC systems), designed to provide 1SR, were again more restrictive than required, due to the employed techniques. Users did not complain: distributed systems behaved as traditional stand-alone ones.

Later, performance-improved systems appeared, where old techniques were substituted with deferred update propagation and atomic broadcast, and where queries were committed and serialized locally (DUABLOQ systems). One-copy serializability was guaranteed but nothing enforced the more restrictive behavior to which users were accustomed: a transaction executed after another might not be able to see the effects of that previous transaction, due to an inversion.

The difference between DLAC and DUABLOQ systems is a different consistency level among replicas. This aspect was never considered in database correctness criteria, thus allowing replication systems to freely relax replica consistency, depriving users of the stricter behavior they were familiarized with, while still being as correct as other systems maintaining that behavior. Both system models were equally acceptable under one-copy serializability. This correctness criterion became then an ample term but, up to our knowledge, such fact has not been stated nor justified yet. However, 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 1SR systems.

We consider that the ambiguity stemming from the fact of not considering replica consistency in correctness criteria for database replication systems is not trivial, as it may lead to user confusion and unfairness when comparing the performance of different replication systems. We think that an explicit distinction must be made. Memory consistency models can be borrowed from the distributed shared memory scope in order to clearly state the features of each system. Inspired in these models, we propose two correctness criteria that refine the 1SR criterion: 1SR+ and 1ASR. These criteria differ in the inversions that users may perceive and therefore define three levels of user-centric consistency: in 1ASR no inversions are perceived, in 1SR+ no inversions within user sessions are perceived, in 1SR inversions may be perceived. A complete formalization is also provided.

We have identified *DLAC* systems (those using distributed locking and the two-phase commit protocol) as 1ASR systems. On the other hand, general *DUABLOQ* systems (those locally executing transactions and finally broadcasting updates in total order) guarantee the 1SR correctness criterion, but not 1SR+ or 1ASR. We have also surveyed existing systems, classifying them according to their correctness criterion. The provided samples prove that the 1SR criterion clearly admits refined subcriteria.

To conclude, there are two answers for the question that was raised in the introduction: "Are one-copy serializable database replication systems actually behaving as one copy?". According to the theory on isolation levels and consistency models, one-copy serializable systems do behave as one copy. According to the users' subjective experience, probably accustomed to the behavior of traditional stand-alone databases (more restrictive than the correctness model they enforced), only systems ensuring 1ASR would behave as one copy.[12] From the point of view of such users, the transparency required to all distributed systems [30] would not be ensured in replicated databases that provide 1SR but do not guarantee 1ASR.

Funding

This work has been partially supported by EU FEDER and the Spanish MICINN [grant numbers TIN2009-14460-C03, TIN2010-17193]; and by the Spanish MEC [grant number BES-2007-17362].

Acknowledgements

The authors would like to thank J. M. Bernabé-Gisbert and J. M. Bernabéu-Aubán for their comments in draft versions of this paper.

References

[1] Traiger, I. L., Gray, J., Galtieri, C. A., and Lindsay, B. G. (1982) Transactions and Consistency in Distributed Database Systems. *ACM T. Database Syst.*, **7**, 323–342.

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

[3] Bernstein, P. A. and Goodman, N. (1984) An Algorithm for Concurrency Control and Recovery in Replicated Distributed Databases. *ACM T. Database Syst.*, **9**, 596–615.

[4] Sinha, M. K., Nanadikar, P. D., and Mehndiratta, S. L. (1985) Timestamp Based Certification Schemes for Transactions in Distributed Database Systems. *Proc. ACM SIGMOD Int. Conf. Manage. Data*, pp. 402–411.

[5] Stanoi, I., Agrawal, D., and El Abbadi, A. (1997) Using Broadcast Primitives in Replicated Databases. *Proc. ACM Symp. Princ. Distrib. Comput.* 283.

[6] Agrawal, D., Alonso, G., El Abbadi, A., and Stanoi, I. (1997) Exploiting Atomic Broadcast in Replicated Databases. *Proc. Int. Euro-Par Conf. Parall. Process.*, Lect. Notes Comput. Sc., **1300**, pp. 496–503. Springer.

[7] Wiesmann, M. and Schiper, A. (2005) Comparison of Database Replication Techniques Based on Total Order Broadcast. *IEEE T. Knowl. Data En.*, **17**, 551–566.

[8] Breitbart, Y., Garcia-Molina, H., and Silberschatz, A. (1992) Overview of Multidatabase Transaction Management. *VLDB J.*, **1**, 181–239.

[9] Daudjee, K. and Salem, K. (2004) Lazy Database Replication with Ordering Guarantees. *Proc. Int. Conf. Data Eng.*, pp. 424–435. IEEE-CS.

[12] With 1SR+, although each individual user is provided with a one-copy image of the system, multiple users may collectively perceive the relaxation in replica consistency.

[10] Zuikeviciute, V. and Pedone, F. (2008) Correctness Criteria for Database Replication: Theoretical and Practical Aspects. *Proc. OTM Conf.*, Lect. Notes Comput. Sc., **5331**, pp. 639–656. Springer.

[11] Kemme, B., Jiménez-Peris, R., Patiño-Martínez, M., and Alonso, G. (2010) Database Replication: A Tutorial. In Charron-Bost, B., Pedone, F., and Schiper, A. (eds.), *Replication: Theory and Practice*, Lect. Notes Comput. Sc., **5959**, pp. 219–252. Springer.

[12] Kemme, B., Jiménez-Peris, R., and Patiño-Martínez, M. (2010) *Database Replication* Synthesis Lect. Data Manage. Morgan & Claypool Publishers.

[13] Elnikety, S., Pedone, F., and Zwaenepoel, W. (2005) Database Replication Using Generalized Snapshot Isolation. *Proc. IEEE Symp. Rel. Distrib. Syst.*, pp. 73–84. IEEE-CS.

[14] Daudjee, K. and Salem, K. (2006) Lazy Database Replication with Snapshot Isolation. *Proc. Int. Conf. Very Large Data Bases*, pp. 715–726. ACM Press.

[15] Muñoz-Escoí, F. D., Bernabé-Gisbert, J. M., de Juan-Marín, R., Armendáriz-Iñigo, J. E., and González de Mendívil, J. R. (2009) Revising 1-Copy Equivalence in Replicated Databases with Snapshot Isolation. *Proc. OTM Conf.*, Lect. Notes Comput. Sc., **5870**, pp. 467–483. Springer.

[16] Gifford, D. K. (1982) Information Storage in a Decentralized Computer System. PhD thesis Stanford University. Also available as Tech. Rep. CSL-81-8, Xerox Corporation.

[17] Herlihy, M. P. (1984) Replication Methods for Abstract Data Types. PhD thesis MIT. Also available as Tech. Rep. MIT/LCS/TR-319.

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

[19] Terry, D. B., Demers, A. J., Petersen, K., Spreitzer, M., Theimer, M., and Welch, B. B. (1994) Session Guarantees for Weakly Consistent Replicated Data. *Proc. Intnl. Conf. Paral. Distrib. Inform. Syst.*, pp. 140–149. IEEE-CS.

[20] Berenson, H., Bernstein, P. A., Gray, J., Melton, J., O'Neil, E. J., and O'Neil, P. E. (1995) A Critique of ANSI SQL Isolation Levels. *Proc. ACM SIGMOD Int. Conf. Manage. Data*, pp. 1–10.

[21] Gray, J. (1978) Notes on Database Operating Systems. In Bayer, R., Graham, R. M., and Seegmüller, G. (eds.), *Advanced Course: Operating Systems*, Lect. Notes Comput. Sc., **60**, pp. 393–481. Springer.

[22] Lampson, B. W. (1981) Atomic transactions. In Lampson, B. W., Paul, M., and Siegert, H.-J. (eds.), *Advanced Course: Distributed Systems*, Lect. Notes Comput. Sc., **105**, pp. 246–265. Springer.

[23] Hadzilacos, V. and Toueg, S. (1994) A Modular Approach to Fault-Tolerant Broadcasts and Related Problems. Technical Report 94-1425. Dep of Computer Science, Cornell Univ.

[24] Carey, M. J. and Livny, M. (1991) Conflict Detection Tradeoffs for Replicated Data. *ACM T. Database Syst.*, **16**, 703–746.

[25] Mosberger, D. (1993) Memory Consistency Models. *ACM SIGOPS Oper. Syst. Rev.*, **27**, 18–26.

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

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

[28] Lamport, L. (1986) On Interprocess Communication. *Distrib. Comput.*, **1**, 77–101.

[29] Herlihy, M. and Wing, J. M. (1990) Linearizability: A Correctness Condition for Concurrent Objects. *ACM T. Progr. Lang. Sys.*, **12**, 463–492.

[30] Tanenbaum, A. S. and van Steen, M. (2002) *Distributed Systems. Principles and Paradigms*. Prentice-Hall.

[31] Lamport, L. (1979) How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs. *IEEE T. Comput.*, **28**, 690–691.

[32] Attiya, H. (2010) Robust Simulation of Shared Memory - 20 Years After. *B. EATCS* , **100**, 100–114.

[33] Guerraoui, R., Garbinato, B., and Mazouni, K. (1994) The GARF Library of DSM Consistency Models. *Proc. ACM SIGOPS Eur. Worksh.*, pp. 51–56.

[34] Fekete, A. and Ramamritham, K. (2010) Consistency Models for Replicated Data. In Charron-Bost, B., Pedone, F., and Schiper, A. (eds.), *Replication. Theory and Practice*, Lect. Notes Comput. Sc., **5959**, pp. 1–17. Springer.

[35] Pedone, F. and Frølund, S. (2000) Pronto: A Fast Failover Protocol for Off-the-shelf Commercial Databases. *Proc. IEEE Symp. Rel. Distrib. Syst.*, pp. 176–185. IEEE-CS.

[36] Kemme, B., Pedone, F., Alonso, G., Schiper, A., and Wiesmann, M. (2003) Using Optimistic Atomic Broadcast in Transaction Processing Systems. *IEEE T. Knowl. Data En.*, **15**, 1018–1032.

[37] Oliveira, R. C., Pereira, J., Correia Jr., A., and Archibald, E. (2006) Revisiting 1-Copy Equivalence in Clustered Databases. *Proc. ACM Symp. Appl. Comput.*, pp. 728–732.

[38] Rodrigues, L., Carvalho, N., and Miedes, E. (2008) Supporting Linearizable Semantics in Replicated Databases. *Proc. IEEE Int. Symp. Netw. Comput. Applic.*, pp. 263–266. IEEE-CS.

[39] Pedone, F. (1999) The Database State Machine and Group Communication Issues. PhD thesis EPFL, Switzerland.

[40] Pedone, F., Guerraoui, R., and Schiper, A. (1997) Transaction Reordering in Replicated Databases. *Proc. IEEE Symp. Rel. Distrib. Syst.*, pp. 175–182. IEEE-CS.

[41] Kemme, B., Pedone, F., Alonso, G., and Schiper, A. (1999) Processing Transactions over Optimistic Atomic Broadcast Protocols. *Proc. Int. Conf. Distrib. Comput. Syst.*, pp. 424–431. IEEE-CS.

[42] Pacitti, E., Minet, P., and Simon, E. (1999) Fast Algorithms for Maintaining Replica Consistency in Lazy Master Replicated Databases. *Proc. Int. Conf. Very Large Data Bases*, pp. 126–137. Morgan Kaufmann.

[43] Kemme, B. and Alonso, G. (2000) A New Approach to Developing and Implementing Eager Database Replication Protocols. *ACM T. Database Syst.*, **25**, 333–379.

[44] Patiño-Martínez, M., Jiménez-Peris, R., Kemme, B., and Alonso, G. (2000) Scalable Replication in Database Clusters. *Proc. Int. Conf. Distrib. Comput.*, Lect. Notes Comput. Sc., **1914**, pp. 315–329. Springer.

[45] Sousa, A. L., Oliveira, R. C., Moura, F., and Pedone, F. (2001) Partial Replication in the Database State Machine. *Proc. IEEE Int. Symp. Netw. Comput. Applic.*, pp. 298–309. IEEE-CS.

[46] Holliday, J., Agrawal, D., and El Abbadi, A. (2002) Partial Database Replication using Epidemic Communication. *Proc. IEEE Int. Conf. Distrib. Comput. Syst.*, pp. 485–493. IEEE-CS.

[47] Zuikevičiūtė, V. and Pedone, F. (2005) Revisiting the Database State Machine Approach. *Proc. VLDB Worksh. Design, Implementation and Deployment of Database Replication*.

[48] Schiper, N., Schmidt, R., and Pedone, F. (2006) Optimistic Algorithms for Partial Database Replication. *Proc. Int. Conf. Princ. Distrib. Syst.*, Lect. Notes Comput. Sc., **4305**, pp. 81–93. Springer.

[49] Armendáriz-Íñigo, J. E., Juárez-Rodríguez, J. R., González de Mendívil, J. R., Decker, H., and Muñoz-Escoí, F. D. (2007) *k*-bound GSI: A Flexible Database Replication Protocol. *Proc. ACM Symp. Appl. Comput.*, pp. 556–560.

[50] Juárez-Rodríguez, J. R., Armendáriz-Íñigo, J. E., González de Mendívil, J. R., Muñoz-Escoí, F. D., and Garitagoitia, J. R. (2007) A Weak Voting Database Replication Protocol Providing Different Isolation Levels. *Proc. Int. Conf. New Tech. Distrib. Syst.*, pp. 261–268.

[51] Zuikevičiūtė, V. and Pedone, F. (2008) Conflict-Aware Load-Balancing Techniques for Database Replication. *Proc. ACM Symp. Appl. Comput.*, pp. 2169–2173.