A Consistency-based Specification for the One-Copy Serializability Variants

M. I. Ruiz-Fuertes, F. D. Muñoz-Escoí

Instituto Tecnológico de Informática
Universitat Politècnica de València
46022 Valencia (SPAIN)

{miruifue,fmunyoz}@iti.upv.es

Technical Report ITI-SIDI-2012/005

A Consistency-based Specification for the One-Copy Serializability Variants

M. I. Ruiz-Fuertes, F. D. Muñoz-Escoí

Instituto Tecnológico de Informática
Universitat Politècnica de València
46022 Valencia (SPAIN)

Technical Report ITI-SIDI-2012/005

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

Abstract

One-copy serializability (1SR) is the accepted correctness criterion of replicated database systems. A distributed database must behave as a single system, and the result of concurrently executing transactions should match a serial execution. In order to ensure 1SR, concurrency was initially managed by distributed locking, and atomic commit protocols controlled transaction termination. Those systems suffered from low throughput. Research focused then on improving performance and scalability, maintaining 1SR. Total order broadcast provided an alternative to atomic commitment, and transactions were locally executed before broadcasting their effects to other replicas. Outstanding performance improvements were achieved. However, the enforced guarantees were subtly but significantly modified. This paper proposes a common specification for the 1SR variants that exist nowadays. Using it, the different replica consistency conditions set by these criteria are precisely stated.

1 Introduction

Distributed and replicated database systems appeared for providing a higher availability than existing stand-alone databases. Bernstein et al. [2] defined one-copy serializability (1SR) as their correctness criterion. In 1SR the interleaved execution of transactions must be equivalent to their serial execution on a stand-alone database. In order to ensure 1SR, an approach inherited from stand-alone database systems was followed [2]. Thus, write operations over any item had to acquire write locks in all its copies before updating it. This concurrency control based on distributed locking strongly affected performance. Moreover, an atomic commit protocol (usually 2PC) was run for transaction termination. In such protocols, several rounds of messages are exchanged among all participating sites, which further penalized performance and scalability.

Several optimizations were added later. According to the deferred update replication model [10], transactions were processed locally at one server and, at commit time, were forwarded to the other replicas for validation. This saved communication costs as distributed synchronization was only done during 2PC. Another optimization consisted in substituting 2PC for a *total order broadcast* (TOB) [1], whose delivery order defines the serialization order for achieving 1SR.

2PC-based replication protocols and TOB-based ones ensure 1SR. However, carefully analyzing these systems, some differences exist. This paper shows that some replication consistency features significantly changed when 2PC was replaced by termination protocols based on TOB.

The remaining sections are structured as follows. Section 2 details the system model. Section 3 describes memory consistency models. Inspired in the differences between those models, Section 4 presents a complete formalization of some 1SR variants. Section 5 gives the conclusions.

2 System Model and Definitions

We assume a partially synchronous distributed system composed of database servers, also called sites or replicas, $R_1, R_2, \ldots, R_n$, which store data, and clients that contact these servers to access the data. Communication between sites is based on message passing. They do not have a global clock.

A database is a collection of logical data items. Each data item is physically stored at the servers. A local relational Database Management System (DBMS) runs at each site. 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. The writeset is the set of items written by a transaction. A transaction is called a query 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. Sessions [11] logically group a set of transactions from the same user. Transactions from different users belong to different sessions. The replica to which a transaction T is addressed is called the delegate replica for T and is responsible for its execution.

3 DSM Consistency Models

Memory consistency models represent how memories from different sites are synchronized to conform a *distributed shared memory* (DSM). The higher the level of consistency, the higher the synchronization and the fewer the divergences on values. According to Mosberger [8], the strictest level of consistency, *atomic consistency* [7], 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 slot-begin time while write operations take effect at slot-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 easiness of design for applications that use atomic memory, this consistency model is often discarded due to its high costs. Consequently, a more relaxed model is used as the regular level: *sequential consistency*. In that model [6] the result of any execution is the same as if the operations of all 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 models is the *old-new inversion*. 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_1 writes a value v_n onto a variable x whose previous value was v_o, another process p_2 reads v_n while later a second reader p_3 reads v_o. This is sequentially consistent since p_3 might read afterwards value v_n, according to the total order associated with sequential consistency. However, such scenario violates atomic consistency since p_3 reads x once the slot given to value v_n was already started and all its reads should return v_n (instead of v_o).

4 1SR Variants

Although memory consistency models only consider individual operations, a correspondence to transactional environments can be established. Indeed, the concept of transaction matches the concept of operation interval [5]: 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), and all write operations are logically moved to the termination point of T (when the transaction commits and its updates are visible for other transactions). Thus, the interval of time where all transaction operations are executed is a *transaction interval*.

Inspired in the idea of a transaction as a macro-operation, and adapting Mosberger's concept of operation interval [8] to its use with transactions, a formalization is presented refining 1SR by distinguishing among its admitted replica consistency models. In the same manner as old-new inversions differentiate atomic and sequential DSM models, we will distinguish among the resulting correctness criteria by identifying divergences in their allowed executions.

Starting from Bernstein et al. [2], a system guarantees 1SR when a complete replicated data history H_{RD} (RD history) is view equivalent to a serial one-copy history H_{1C} (1C history); i.e.: (a) H_{RD} and H_{1C} have the same reads-from relationships, i.e., T_k reads from T_j in H_{RD} iff 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 operation of the other. From the point of view of users, transactions in a serial execution seem to be processed atomically, in intervals.

The following definitions present the foundations of our specification. Next, by requiring some extra conditions over the sequence of intervals of the equivalent 1C history, we will qualify two 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}. The interval of T_j in H_{1C}, $I(T_j)$, is 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.

Aborted transactions do not present intervals.

Definition 2 (Interval order)**.** Let H_{1C} be a serial 1C history. The interval order of H_{1C}, $<_i$, is 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, access the same data item and at least one of them 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 all pairs of transactions in that set are independent.

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

Definition 5 (First start)**.** 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 is considered 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.

In a locking-based concurrency control all operations over database items are preceded by the acquisition of locks. A transaction whose initial operation is held in the database while waiting for the required locks is not first-started until those locks are granted and the first data item is effectively accessed.

After executing all its operations, a transaction asks for commitment. In the ROWAA approach, each committed transaction T_j must apply its updates at every available replica. This can be considered as multiple commit operations for T_j. One of the replicas, R_i, will be the first to complete the commit operation and make the updates of T_j visible to other transactions starting at R_i. This instant is when T_j first-commits.

Definition 6 (First commit)**.** 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)**.** A transaction T_j precedes transaction T_k in real time, $T_j <_{rt} T_k$, iff T_j first-commits before T_k first-starts.

The $<_{rt}$ order is partial.

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.

Definition 9 (Alteration)**.** Let T_j and T_k be two committed transactions in a 1C history H_{1C}. Let $T_j <_{rt} T_k$. An alteration occurs in H_{1C} if $T_k <_i T_j$, i.e., the interval order of H_{1C} is inconsistent with the RT precedence.

Definition 10 (RTC precedence)**.** A partial order called *real-time and conflict precedence*, $<_{rtc}$, is 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 in a 1C history H_{1C}. Let $T_j <_{rtc} T_k$. An inversion occurs in H_{1C} if $T_k <_i T_j$, i.e., the interval order of H_{1C} is inconsistent with the RTC precedence.

An inversion is an alteration between conflicting transactions. Users may only perceive inversions. 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 a single write operation over x, and let T_b and T_c be two queries: T_b reads x and T_c reads y. Thus, the RTC precedence over these transactions is only: $T_a <_{rtc} T_b$. 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 in any order.

The old-new inversion presented in Section 3 corresponds to an inversion where a query overtakes the update transaction from which it was supposed to read a value.

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

For users, this model ensures atomic consistency since the results of an execution match those generated when each transaction is committed at every replica before starting the next one. However, this criterion is less strict than the atomic DSM model since it only precludes inversions.

Strong 1SR restricts the serial orders that would comply with 1SR, so that orders violating RTC are not considered. Examples of Strong 1SR are the algorithms described by Berstein et al. [2] and the strongly serializable DBMSs of Breitbart et al. [3].

This variant is suitable for applications that require a strict consistency. However, the imposed conditions may be relaxed by only requiring that users do not see inversions. Thus, even if some inversions are allowed, individual users perceive an atomic image as in a Strong 1SR system.

Definition 13 (History projection)**.** 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)**.** A session logically groups 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 her 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 (Session 1SR correctness criterion)**.** A complete RD history H_{RD} over a set of transactions is Session 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 *Session 1SR* (inversion preclusion in sessions) further prevents undesirable conditions from those avoided by the plain 1SR criterion: its serial order ensures RTC precedence over the transactions of each user session. For a user grouping all her transactions within the same session, the system cannot be distinguished from a Strong 1SR system. Algorithms that provide Session 1SR (but do not provide Strong 1SR) are the Block and Forward ones by Daudjee and Salem [4].

Finally, applications that are not sensitive to real-time precedence are not concerned about such issue: 1SR is enough. If the commit order in each replica matches the writeset delivery order of the FIFO TOB,

then the system follows a sequential consistency [8]. Nevertheless, this is not mandatory (specifically, for independent writesets [9]) and, thus, even more relaxed replica consistencies are admitted in plain 1SR. A system providing 1SR (but not Session nor Strong 1SR) is DBSM [9].

Both Strong 1SR and Session 1SR histories are included in the set of 1SR histories. Moreover, Strong 1SR histories are a subset of Session 1SR histories. Additionally, all three sets are different.

Our specification supports two correctness criteria that refine 1SR. The resulting three criteria cover several replica consistency conditions: preclusion of inversions (strong 1SR), preclusion of inversions within sessions (session 1SR), and possibility of inversions (1SR).

5 Conclusions

Early database replication systems based on *distributed locking* and *atomic commit* (DLAC systems), designed to provide 1SR, were 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 *TOB* (DUTOB systems). Queries were committed and serialized locally. 1SR was still guaranteed but nothing enforced the DLAC behavior: a transaction T_2 executed after another T_1 might not see the effects of T_1 due to an inversion.

The difference between DLAC and DUTOB systems is their replication consistency. The ambiguity stemming from not considering replica consistency in correctness criteria should be removed as it leads to unfairness when comparing replication systems performance. An explicit distinction must be made. Different authors had proposed several 1SR variants: Strong 1SR, Session 1SR and plain 1SR. A consistency-based specification for these 1SR criteria is provided in this paper.

Acknowledgements

This work has been supported by EU FEDER and the Spanish MICINN under grants TIN2009-14460-C03, TIN2010-17193; and by the Spanish MEC under grant BES-2007-17362.

References

[1] D. Agrawal, G. Alonso, A. El Abbadi, and I. Stanoi. Exploiting atomic broadcast in replicated databases. In *Proc. Int. Euro-Par Conf. Parall. Process.*, volume 1300 of *Lect. Notes Comput. Sc.*, pages 496–503. Springer, August 1997.

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

[3] Y. Breitbart, H. Garcia-Molina, and A. Silberschatz. Overview of multidatabase transaction management. *VLDB J.*, 1(2):181–239, October 1992.

[4] K. Daudjee and K. Salem. Lazy database replication with ordering guarantees. In *Proc. Int. Conf. Data Eng.*, pages 424–435. IEEE-CS, March 2004.

[5] 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 *Lect. Notes Comput. Sc.*, pages 1–17. Springer, 2010.

[6] L. Lamport. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE T. Comput.*, 28(9):690–691, September 1979.

[7] L. Lamport. On interprocess communication. *Distrib. Comput.*, 1(2):77–101, 1986.

[8] D. Mosberger. Memory consistency models. *ACM Oper. Syst. Rev.*, 27(1):18–26, January 1993.

[9] F. Pedone. *The Database State Machine and group communication issues*. PhD thesis, EPFL, Switzerland, 1999.

[10] M. K. Sinha, P. D. Nanadikar, and S. L. Mehndiratta. Timestamp based certification schemes for transactions in distributed database systems. In *Proc. ACM SIGMOD Int. Conf. Manage. Data*, pages 402–411. ACM Press, May 1985.

[11] D. B. Terry, A. J. Demers, K. Petersen, M. Spreitzer, M. Theimer, and B. B. Welch. Session guarantees for weakly consistent replicated data. In *Proc. Intnl. Conf. Paral. Distrib. Inform. Syst.*, pages 140–149. IEEE-CS, September 1994.