Armendáriz-Iñigo, J.; Juárez-Rodríguez, J.; González De Mendívil, J.; Garitagoitia, J.; Irún Briz, L.; Muñoz Escoí, FD. (2011). A formal characterization of SI-based ROWA replication protocols. Data and Knowledge Engineering. 70(1):21-34. doi:10.1016/j.datak.2010.07.012.

☆

A Formal Characterization of SI-Based ROWA Replication Protocols

J.E. Armendáriz-Iñigo[a], J.R. Juárez-Rodríguez[a], J.R. González de Mendívil[a], J.R. Garitagoitia[a], F.D. Muñoz-Escoí[b], L. Irún-Briz[b],

[a] *Departamento de Ingeniería Matemática e Informática. Universidad Pública de Navarra. Campus Arrosadía s/n. 31006 Pamplona (Spain)*
[b] *Instituto Tecnológico de Informática. Universidad Politécnica de Valencia. Camino de Vera s/n. 46022 Valencia (Spain)*

Abstract

Snapshot Isolation (SI) is commonly used in some commercial DBMSs with a multiversion concurrency control mechanism since it never blocks read-only transactions. Recent database replication protocols have been designed using SI replicas where transactions are firstly executed in a delegate replica and their updates (if any) are propagated to the rest of replicas at commit time; i.e. they follow the Read One Write All (ROWA) approach. This paper provides a formalization that shows correctness of abstract protocols which cover these replication proposals. These abstract protocols differ in the properties demanded for achieving a global SI level and those needed for its Generalized SI (GSI) variant –allowing reads from old snapshots–. Additionally, we propose two more relaxed properties that also ensure a global GSI level. Thus, some applications can further optimize their performance in a replicated system while obtaining GSI.

Key words:
Database Replication, Distributed Databases, Snapshot Isolation, Read One Write All, Replica Control, Theory, Verification.

1. Introduction

Snapshot Isolation (SI) is a transaction isolation level introduced in [7] and implemented (using multiversion concurrency control) in several commercial database systems. A transaction executed under SI reads data from the committed state of the database (*snapshot*) as of the time it began and keeps the results of its own writes in local memory and, thus, sees its own updates. Read-only transactions executed under SI level are neither delayed, blocked nor aborted, and they never cause update transactions to block or abort. This behavior is important for workloads with predominant read-only transactions such as those resulting from dynamic content Web servers.

The use of a replicated database system (RDBS) is the best way to improve performance and offer availability in the presence of failures to distributed applications. However, this brings

☆This work has been supported by the Spanish Government under research grant TIN2006-14738-C02.

up the question about how to achieve data consistency; i.e. how to coordinate the execution of updates at different replicas. Recently, several system designs [20, 41, 27, 34, 12, 18, 5, 29] have proposed different replication solutions based on SI databases. An important question is what isolation level is seen by users who imagine their transactions are executed in a single logical database. From this point of view, several systems offer Generalized SI (GSI) [14]; this means that a transaction can read from an older snapshot, rather than seeing the latest one before the transaction's start, as required by SI.

These previous systems run replication protocols that follow the Read-One-Write-All (ROWA) approach [16]. This kind of replication protocol initially executes all of the transaction's operations at a single delegate replica, collecting its writeset, propagating such updates to all replicas (*update propagation*) and taking the same decision in all replicas about such transaction's fate (*atomicity*). These two characteristics will be stated as Property *P1* in this paper.

Certain additional conditions shall be imposed over the RDBS with SI replicas, already satisfying Property *P1*, in order to determine the final isolation level obtained by user transactions derived as a 1-copy-schedule in the single logical database. The two sufficient conditions to derive a 1-copy-SI-schedule (from now on, 1SI-schedule) are the following: (*P2*) *total order of conflicting transactions*; and, (*P3*) *transactions read the latest items' version*. The first condition states that two conflicting transactions must be sequentially executed in the same order at all replicas; while the second one compels every single transaction to read data items belonging to the latest system snapshot. This is a severe restriction because it requires either blocking or delaying the start of transactions [14]; this fact voids one of the main advantages of the SI level.

Note that P1, P2 and P3 are easily satisfied by *primary copy* protocols [34, 12, 29] as all transactions are always served directly by a single primary replica. On the other hand, it is difficult to satisfy condition P3 with *update-everywhere* replication protocols [14, 27]. In the latter, each transaction may select a different delegate server that directly executes the transaction and propagates its updates in total order before such transaction is validated (according to the *certification-based* model as specified in [40]).

If the system wants to provide consistent snapshots while preserving the ordering of conflicting transactions then it is needed to define a new condition (*P4*) *Total order of committed update transactions*. P4 along with P1 are sufficient to obtain a 1-copy-GSI-schedule (1GSI-schedule). P4 is sufficient and imposes no restriction for starting transactions (such as P3 does). These properties have already been used in order to prove the correctness of some existing snapshot-based replication protocols. However, in most cases these properties were not all explicitly stated (i.e., some of them were implicitly assumed) and those correctness proofs were only applied to the replication protocol presented in such papers, as expected from any proof of this kind. So, the first contribution of this paper is the proposal and justification of the general set of properties that should be used for proving the correctness of database replication protocols that guarantee either SI or GSI isolation. This does not imply that previous correctness proofs are incorrect or incomplete, since in some cases the algorithm being followed in such protocols is sufficient for implicitly proving some of these properties.

However, once it is known that this is the set of required conditions, some of their constraints could be broken. Thus, it seems reasonable to relax condition P4 to the limit imposed by P2. As a result, instead of requiring P4, we have found a pair of more relaxed conditions: (*P5*) *Non-conflicting transactions interleaving*; and, (*P6*) *Compatible snapshot read* that, combined together with P1, also ensure 1GSI-schedules. They are inspired in the protocol presented in [27], although our condition P6 is more relaxed than the one used in such paper. So, as a second contribution, this allows the design and development of new protocols (particularly for applica-

tions where the readset of transactions can be forecast, as those based on stored procedures) able to increase the parallelism of the writeset application actions, further improving the performance of these replication solutions.

Therefore, this paper offers a formalized treatment of some replication protocols using the ROWA approach that involve replicas using the SI mechanism. It proves some sufficient conditions based on the previous properties for deriving both 1SI and 1GSI-schedules; this amounts to proving the correctness of two abstract replication protocols that enclose the concrete realizations of already existing protocols [41, 27, 34, 12, 5]. These protocols cover all the protocols' families classified in [16] according to the replica where update transactions can be executed (primary copy, update everywhere) and when updates are executed (eager or lazy). Moreover, these conditions can be used to check the correctness of any new replication protocol proposal: it will only need to show that it satisfies conditions P1, P2 and P3 for 1SI-schedules, whilst 1GSI-schedules need either P1 and P4, or, P1, P5 and P6, respectively.

The rest of the paper is organized as follows. Section 2 introduces the concept of multiversion schedules. Section 3 gives the concepts of SI and GSI. In Section 4, the structure of an RDBS is presented. The details of an abstract ROWA protocol with SI replicas are presented in Section 5. Conditions for 1SI and 1GSI-schedules are introduced in Sections 6 and 7 respectively. We discuss how to relax conditions for 1GSI in Section 8. An overview about recent replication proposals with SI replicas is given in Section 9. Finally, conclusions end the paper.

2. Multiversion Schedules

The concept of *multiversion schedule* is introduced in this section to model the executions generated by a multiversion concurrency control providing snapshot-based isolation levels. In general, a schedule of a set of operations S is a pair (S, t), briefly denoted as S_t, where $t\colon S \rightarrow \mathbb{R}^+$ is an injective function assigning a different time value to each operation of S. In the rest of this paper, we will use the identifiers S_t and H_t in order to refer to schedules.

A database (DB) is a collection of data items, denoted by upper case letters as X. A transaction T_i is a sequence of read and write operations on database items ended by a commit or abort operation. Each T_i's write (read) operation on item X is denoted $W_i(X)$ ($R_i(X)$). The T_i's commit and abort operation are denoted C_i and A_i respectively. The operations of a transaction T_i are totally ordered by the order $\prec_{T_i}$; any operation of T_i is denoted op_i. The *readset* and *writeset* (denoted by RS_i and WS_i respectively) express the sets of items read and, respectively, written by a transaction T_i. Thus, T_i is a *read-only* transaction if $WS_i = \emptyset$ and otherwise it is an *update* transaction. For the sake of simplicity, we assume: (a) each item is read and written at most once in each transaction T_i and, (b) a transaction does not read an item X after it has written it. These assumptions are not a severe restriction of the model, and they were already suggested in [33]. Thus, regarding multiple read accesses, note that the client program can use local variables in order to maintain the value that has been read or written, so all the subsequent read accesses could be made on such local variables. A similar solution could be used for write accesses: if multiple write operations need to be applied to a given item, all but the last one could be applied to local variables.

Let L be a set of transactions. In order to process operations from a transaction $T_i \in L$, a multiversion concurrency control must translate T_i's operations on data items into operations on specific versions of those data items [8]. That is, there is a function $h()$ that maps each $W_i(X)$ into $W_i(X_i)$, each $R_i(X)$ into $R_i(X_j)$ where X_j was generated by some $T_j \in L$, each C_i into C_i, and each A_i into A_i. In the same way, there exists an injective function $t()$ such that it assigns to each

operation the time at which it happens to obtain an ordered set of multiversion operations. We assume that there exists an initial transaction $T_0 \in L$ that is executed and committed before any other transaction and initializes all data items $X \in DB$ to X_0.

Definition 1 (**Complete MV-schedule**). *A complete multiversion schedule (MV-schedule) over a set of transactions L is a schedule H_t such that:*
1) $H = h(\bigcup_{T_i \in L} T_i)$ for some translation function h.
2) if $p_i, q_i \in T_i$ and $p_i \prec_{T_i} q_i$ then $t(h(p_i)) < t(h(q_i))$.
3) if $R_i(X_j) \in H$ then $W_j(X_j), C_j \in H$ and $t(C_j) < t(R_i(X_j))$.

In the previous definition, condition (1) indicates that each operation of a transaction is translated into an appropriate multiversion operation. Condition (2) states that the schedule preserves the order of operations of a transaction. Condition (3) establishes that if a transaction reads a concrete version of a data item, it was written by a transaction that committed before the item was read.

The properties studied in this paper are only required to deal with committed transactions; thus, L is actually a set of committed transactions. We assume that an underlying recovery mechanism of the database will rollback any operation of an aborted transaction. Finally, we assume that an MV-schedule is complete for the sake of simplifying definitions and proofs.

3. Snapshot-Based Isolation Levels

In SI, "reading from a snapshot" means that a transaction T_i sees all the updates done by transactions that committed before the transaction started its first operation. The results of its writes are installed when the transaction commits. However, a transaction T_i will successfully commit if and only if there is not a concurrent transaction T_k that has already committed and some of the written items by T_k are also written by T_i. This condition is necessary to exclude the *lost-update* anomaly [7]. A straightforward generalization of SI, called Generalized SI (GSI), can be obtained by allowing a transaction to observe an older snapshot than the snapshot at its start operation. Similarly to the SI case, GSI needs to avoid the lost-update anomaly.

In the following, we formalize the concept of snapshot of the database. Let us define for each committed transaction T_i in an MV-schedule H_t its associated *commit time* as $c_i = t(C_i)$ and, respectively, its *begin time* as $b_i = t(\textit{first operation of } T_i)$. Let $Ver(X, H_t)$ be the set of versions of a data item X in an MV-schedule H_t; i.e., $Ver(X, H_t) = \{X_j \colon W_j(X_j) \in H\}$. The latest version of the data item X in H_t at time $\tau \in \mathbb{R}^+$ is defined as $latestVer(X, H_t, \tau) = \{X_p \in Ver(X, H_t) \colon \nexists\, X_k \in Ver(X, H_t) \colon c_p < c_k \leq \tau\}$.

Definition 2 (**Snapshot**). *Let H_t be an MV-schedule over L. The snapshot of the database DB at time $\tau \in \mathbb{R}^+$ for H_t is the set $Snapshot(DB, H_t, \tau) = \bigcup_{X \in DB} latestVer(X, H_t, \tau)$*

In order to formalize the concept of GSI-schedule, we utilize the next shorthand predicate for an MV-schedule H_t over L, two transactions T_j and $T_i \in L$, and time $\tau \in \mathbb{R}^+$:
$wconflict(T_j, T_i, H_t, \tau) \equiv WS_j \bigcap WS_i \neq \emptyset \ \wedge \ \tau < c_j < c_i$.

Note that this $wconflict()$ predicate uses τ as the starting point of transaction T_i. Such parameter is needed in order to use different starting points depending on the isolation level being assumed. Thus, a strict SI level needs $\tau = b_i$, but the GSI level may use $\tau \leq b_i$ as stated in the following definitions.

Definition 3 (GSI-schedule). *Let H_t be an MV-schedule over L. H_t is a GSI-schedule if and only if for each $T_i \in L$ there exists a value $s_i \in \mathbb{R}^+$ such that $s_i \leq b_i$ and:*

1. *if $R_i(X_j) \in H$ then $X_j \in Snapshot(DB, H_t, s_i)$; and*
2. *for each $T_j \in L$: $\neg wconflict(T_j, T_i, H_t, s_i)$*

Condition (1) states that every item read by a transaction belongs to the same (possible past) snapshot. Condition (2) establishes that the time intervals $[s_i, c_i]$ and $[s_j, c_j]$ do not overlap for any pair of conflicting transactions T_i and T_j, i.e., if such two transactions have updated a shared item, their running intervals should not overlap in order to admit them. This is the regular condition being evaluated by the database replication protocols when transactions are certified once they have requested commit [14, 27]. If for all $T_i \in L$, conditions (1) and (2) hold for $s_i = b_i$ then H_t is an SI-schedule. Thus, Definition 3 includes as a particular case the next definition of SI-schedules.

Definition 4 (SI-schedule). *Let H_t be an MV-schedule over L. H_t is an SI-schedule if and only if for each $T_i \in L$:*

1. *if $R_i(X_j) \in H$ then $X_j \in Snapshot(DB, H_t, b_i)$; and*
2. *for each $T_j \in L$: $\neg wconflict(T_j, T_i, H_t, b_i)$*

4. Replicated Database System

A Replicated Database System (RDBS) is a distributed system such that each site contains a database and runs an instance of a replication protocol. The distributed system comprises m sites, where I_m is the set of site identifiers. Sites do not share a common clock nor a common memory and they communicate by reliable message passing. We make no assumptions about the time it takes for sites to execute and for messages to be transmitted; they are arbitrary but finite. We consider a system free of failures. The RDBS is fully replicated, that is, there is a logical database DB such that each site $k \in I_m$ contains a physical copy of DB, denoted DB^k. In what follows, L is the set of logical transactions (user programmed) committed over DB, while L^k is the set of physical transactions executed over DB^k at each site $k \in I_m$. We consider that the database DB^k generates MV-schedules over L^k, denoted as H_t^k, and all operations in the system are totally ordered; i.e., there is an injective function t: $\bigcup_{k \in I_m}(H^k) \rightarrow \mathbb{R}^+$ assigning a different time value to each operation. The replication protocol coordinates the execution of physical transactions at every replica in the system so that the resulting interleaving execution seems as though there is a unique set of logical transactions L accessing a single database DB. Hence, the protocol performs a transformation $\mathcal{T}$ such that $\mathcal{T}(L) = \bigcup_{k \in I_m} L^k$. The transformation considered in this paper is the ROWA Transformation.

Definition 5 (**ROWA Transformation**). *Let L be a set of logical transactions over DB. The ROWA Transformation, $\mathcal{T}(L)$, defines for each $T_i \in L$ a set of physical transactions for all $k \in I_m$ such that there is a single transaction satisfying $T_i^{site(i)}$ with $RS_i^{site(i)} = RS_i$ and $WS_i^{site(i)} = WS_i$ where $site(i) \in I_m$ is the delegate replica of T_i. The rest of physical transactions, T_i^k with $k \neq site(i)$, have $RS_i^k = \emptyset$ and $WS_i^k = WS_i$. Thus, $L^k = \{T_i^k : T_i \in L\}$ is the set of physical transactions over DB^k for each site $k \in I_m$ and $\mathcal{T}(L) = \bigcup_{k \in I_m} L^k$.*

A replication protocol generates, physical, replicated schedules over the ROWA Transformation $\mathcal{T}(L)$.

Definition 6 (**R-schedule**). *Let L be a set of logical transactions and $\mathcal{T}(L)$ be the ROWA Transformation. Let H_t^k be the MV-schedule over L^k for each site $k \in I_m$ where $t\colon \bigcup_{k \in I_m}(H^k) \rightarrow \mathbb{R}^+$. A schedule S_t is an R-schedule over $\mathcal{T}(L)$ if and only if $S = \bigcup_{k \in I_m} H^k$.*

Since these operations occur at some real time instant, we consider that there is a mapping t such that it assigns to each operation the real time at which it happens; and it verifies that no two different operations have the same time value.

From the previous definition, H_t^k can be derived from S_t by adequately choosing all operations performed at replica $k \in I_m$. Note also that new data item versions are available as soon as a transaction T_i^k is firstly committed at any replica $k \in I_m$. At this point, we consider that the logical transaction T_i has been globally committed and, as said before, new data item versions are available in RDBS to be read by other transactions. In the following, we present a notation in order to consider the "*first commit*" notion: for each logical transaction $T_i \in L$ and an R-schedule S_t over $\mathcal{T}(L)$, $C_i^{min(i)}$ denotes the commit operation of the transaction $T_i^{min(i)}$ at site $min(i) \in I_m$ such that $c_i^{min(i)} = \min_{k \in I_m}\{c_i^k\}$.

R-schedules are defined over the set of physical transactions $\mathcal{T}(L)$ (actually, it is simple to prove that an R-schedule S_t is an MV-schedule over $\mathcal{T}(L)$); however, in the context of this paper we will focus on a resulting logical schedule over the logical set L with particular properties. Hence, we will define the correspondence between an R-schedule and a 1-copy-MV-schedule (1MV-schedule) and its existence.

Definition 7 (**1MV-schedule**). *Given a set of logical transactions L and the ROWA Transformation $\mathcal{T}(L)$, then an R-schedule S_t over $\mathcal{T}(L)$ is a 1MV-schedule provided that the particular logical schedule $H_{t'}$ is an MV-schedule over L, where $H_{t'} = \mathcal{F}(S_t)$ being $\mathcal{F}$ a transformation defined explicitly as follows:*

For each $T_i \in L$ and $k \in I_m$:

1) Remove from S the operations such that: $W_i(X_i)^k$, with $k \neq site(i)$, and C_i^k, with $k \neq min(i)$.

2) H is obtained with the rest of operations in S after step 1 applying the renaming: $W_i(X_i) = W_i(X_i)^{site(i)}$, $R_i(X_j) = R_i(X_j)^{site(i)}$, and $C_i = C_i^{min(i)}$.

3) Finally, $t'(W_i(X_i)) = t(W_i(X_i)^{site(i)})$, $t'(R_i(X_j)) = t(R_i(X_j)^{site(i)})$, and $t'(C_i) = t(C_i^{min(i)})$.

The previous definition is a constructive one for an R-schedule S_t to be a 1MV-schedule. It states that a logical MV-schedule $H_{t'}$ can be built by properly choosing operations of each

(physical) transaction. More precisely, it selects all operations from transaction $T_i^{site(i)}$ and the first commit operation of all T_i^k with $k \in I_m$ while changing each data item from physical to logical and matching up the versions properly. Regarding the commit operation, only two options seem to make sense: to take the first existing commit action or to take the last one. Whilst the first one does not cause any problem, the latter prevents valid read accesses (e.g., those made by other transactions in such $I_{min(i)}$ replica) from being accepted in these schedules. Due to this, such first alternative is used. Note that all operations of the deferred update copier transactions have been removed; and, the t and t' functions share the same value. In the context of this paper, an RDBS will be correct if it generates 1SI- or 1GSI-schedules. This is formally stated in the next definition.

Definition 8. *Given a set of logical transactions L and the ROWA Transformation $\mathcal{T}(L)$, then an R-schedule S_t over $\mathcal{T}(L)$ is a* **1SI-schedule***, respectively a* **1GSI-schedule***, if and only if S_t is a 1MV-schedule such that the particular MV-schedule $\mathcal{F}(S_t)$ over L is an SI-schedule, respectively a GSI-schedule.*

Note that this definition does not restrict the kind of MV-schedule at each replica $k \in I_m$. In other words, Definition 8 is intended to be pretty general; an R-schedule S_t can be an 1SI-schedule (respectively 1GSI-schedule) provided that the MV-schedule $\mathcal{F}(S_t)$ over L is SI (GSI) even though the MV-schedules at each replica are not SI-schedules and the resulting transformation $\mathcal{F}(S_t)$ is also an MV-schedule. Let us see this with an example.

Example 1. *Given two transactions $\{T_1, T_2\}$ where $T_1 = W_1(X)W_1(Y)C_1$ and $T_2 = W_2(X)W_2(Y)C_2$; and, two sites $\{I_1, I_2\}$ where we assume that T_1's site is I_1 and T_2's site is I_2. At I_1 the MV-schedule H_t^1 is as follows $W_1(X_1)W_1(Y_1)C_1W_2(X_2)W_2(Y_2)$ $C2$ where $t()$ assigns for the first three operations the time values 1-3 and for the last three operations values 7-9. Respectively, at I_2 the MV-schedule H_t^2 is as follows $W_2(X_2)W_2(Y_2)C_2W_1(X_1)W_1(Y_1)C_1$ where $t()$ assigns for the first three operations the time values 4-6 and for the last three operations values 10-12.*

Example 1 holds the following properties. Both H_t^1 and H_t^2 are SI-schedules (they are actually serial schedules). The function $t()$ assigns a different value to each operation of both schedules. The union over both schedules is an R-schedule S_t according to Definition 6. We can now build $\mathcal{F}(S_t)$ as $W_1(X_1)W_1(Y_1)C_1W_2(X_2)$ $W_2(Y_2)C_2$ with the time values 1-6. It is easy to show that $\mathcal{F}(S_t)$ is an SI-schedule. Thus, according to Definition 8, S_t is a 1SI-schedule; even though, at the end of execution, I_1 and I_2 have different values for X and Y, as I_1 has the values written by T_2 and I_2 has the values written by T_1. As long as no transaction has read X and Y, S_t is a 1SI-schedule and this holds since we have assumed that our MV-schedules are complete. In this example, this means that no other transaction ever reads X or Y and this sets S_t as a valid 1SI-schedule. Obviously, having any subsequent transaction reading X and Y will invalidate S_t. So, our definitions –requiring completeness– correctly manage 1SI- and 1GSI-schedules. In what follows, we will restrict the whole set of possible valid 1SI- and 1GSI-schedules to the ones derived by ROWA protocols with SI replicas that satisfy certain properties. These protocols' properties are sufficient to generate these schedules and their advantage relies on the fact that SI and GSI benefits are kept while protocols are easy to deploy in a real system. Moreover, it can be also inferred new protocols that will be also easy to develop.

In order to summarize the specifications given up to this point, and to help the reader in the definitions, theorems and proofs of the next sections, Table 1 presents (in alphabetical order) the symbols used in this paper, with a brief description of each of them.

Symbol	Description
A_i	Abort operation for transaction T_i.
b_i	Begin time of transaction T_i.
C_i	Commit operation for transaction T_i.
c_i	Commit time of transaction T_i.
DB	Database.
DB^k	Database copy maintained at site k.
$\mathcal{F}()$	ROWA multi-version transformation (see Definition 7).
H_t	Multi-version schedule.
$h()$	Multi-version mapping function.
I_m	Set of the m system sites where a database is replicated.
k	Site identifier. It may appear as a superindex, meaning that such element is projected to site k. Examples: C_i^k, c_i^k, DB^k, H_t^k, L^k, T_i^k, ...
L	Set of all system transactions.
$min(i)$	Identifier of the replica that first committed T_i.
RS_i	Read-set of transaction T_i.
S	Set of operations to be used in transactions.
S_t	Schedule or R-schedule (i.e. replicated schedule).
s_i	Snapshot time of transaction T_i.
$\mathcal{T}()$	ROWA transformation (see Definition 5).
$site(i)$	Identifier of the replica where T_i was initially executed.
T_i	Transaction with identifier "i".
$t()$	Time mapping function. It sets a total order on the executed operations.
WS_i	Write-set of transaction T_i.
X	Database item.
X_i	Version of database item X generated by transaction T_i.

Table 1: Symbols used in the specification.

5. ROWA protocols with SI replicas

Several replication protocols have been designed and deployed for RDBSs with SI databases [41, 27, 34, 12, 5, 29].

All these protocols share some common characteristics; in particular, they satisfy the ROWA Transformation stated in Definition 5, and guarantee the property of *atomicity* in the RDBS for each transaction. Each transaction T_i has a delegate site ($site(i)$) where it is initially processed. Read-only transactions are executed without further interaction with other replicas. On the other hand, in eager protocols an update transaction T_i is initially executed at a given replica (that is always the same in *primary copy* replication protocols or could be any one –e.g., the one local or closest to the client process– in *update-everywhere* protocols) and it gets propagated with its writeset WS_i to the rest of replicas (possibly including itself too) by the replication protocol. Upon this message delivery, it is decided whether T_i is committed or not. To this end some kind of evaluation should be made, checking for conflicts against the writesets of concurrent transactions, as it has been stated in condition (2) of Definitions 3 and 4. If no conflict arises, WS_i is applied and committed as a remote transaction at each non delegate site (let us name k such other replicas), and directly committed in the delegate site. Hence, update operations of a

remote transaction are always executed after all operations of T_i at $site(i)$ –except its commit, in these eager protocols– have been executed (*update propagation*). Finally, lazy propagation protocols are usually implemented with a primary copy server architecture. This implies that conflict evaluation needs to be done only at the primary site. So, both the conflict evaluation and the commit action are executed in such primary replica ($site(i)$) before the writeset is propagated to the remaining replicas (k). All these characteristics are summarized in the next property.

Definition 9 (**P1** – ROWA protocol)**.** *Given a set of logical transactions L, then an R-schedule S_t over a ROWA Transformation $\mathcal{T}(L)$ satisfies Property P1 when for every update transaction $T_i \in L$ these two conditions are held:*

1. *(*Atomicity*) For every site $k \in I_m$: $C_i^k \in S$.*
2. *(*Update Propagation*) If $op_i^{site(i)} \in S$ is the last operation (before commit) at $site(i) \in I_m$, then for all $k \in I_m \setminus \{site(i)\}$: $t(op_i^{site(i)}) \leq b_i^k$.* □

It has been said how to build a 1MV-schedule from an R-schedule (see Definition 7); in the next theorem it is shown that an R-schedule satisfying P1 is indeed a 1MV-schedule.

Theorem 1. *Given a set of logical transactions L and an R-schedule S_t over the ROWA Transformation $\mathcal{T}(L)$. If S_t satisfies Property P1, then S_t is a 1MV-schedule.*

Proof. By using the transformation $\mathcal{F}(S_t)$ in Definition 7, we build the schedule $\mathcal{F}(S_t) = H_{t'}$, and prove that $H_{t'}$ is an MV-schedule over L.
(1). $H = h(\cup_{T_i \in L} T_i)$, for each $T_i \in L$ we have that $op_i \in H$ if $op_i^{site(i)} \in S$; besides, $C_i \in H$ if there exists $k \in I_m$ such that $C_i^k \in S$. Thus, there exists a function h that maps each operation of $T_i \in L$.
(2). If $p_i, q_i \in T_i$ and $p_i \prec_{T_i} q_i$ then $t'(h(p_i)) < t'(h(q_i))$. Let us consider if $h(q_i) = C_i$ or not. In the latter, by Property P1.1 (*Atomicity*), as H_t^k is an MV-schedule over L^k with $k \in I_m$; if $k = site(i)$ then $t'(h(p_i)) < t'(h(q_i))$ as it is the same as $t(h(p_i^{site(i)})) < t(h(q_i^{site(i)}))$ in S_t. Although in the former ($h(q_i) = C_i$), if $min(i) = site(i)$ then a similar reasoning as before leads to $t'(h(p_i)) < t'(C_i)$. Finally, $t(h(p_i^{site(i)})) < t(h(p_i^{min(i)}))$ by Property P1.2 and $H_t^{min(i)}$ is an MV-schedule by Property P1.1; hence, $t(h(p_i^{min(i)})) < t(C_i^{min(i)})$ and $t'(h(p_i)) < t'(C_i)$ is satisfied.
(3). If $R_i(X_j) \in H$ then $W_j(X_j), C_j \in H$ and $t'(C_j) < t'(R_i(X_j))$. Furthermore, $R_i(X_j)^{site(i)} \in S$ and, as $H_t^{site(i)}$ is an MV-schedule and by Definition 1, $W_j(X_j)^{site(i)} \in S$ too; thus, $C_j^{site(i)} \in S$ and $t(C_j^{site(i)}) < t(R_i(X_j)^{site(i)})$. On the other hand, $W_j(X_j), C_j \in H$ is satisfied by construction of H; by Property P1.1, there exists $C_j^{min(j)} \in S$ and $t'(C_j) = t(C_j^{min(j)}) \leq t(C_j^{site(i)}) < t(R_i(X_j)^{site(i)}) = t'(R_i(X_j))$ is ensured by construction of H. □

In Theorem 1 and Definition 7 as $H_{t'}$ receives the time values from S_t, we write H_t instead of $H_{t'}$. It has not yet been considered the fact that the RDBS can be composed of replicas generating SI-schedules:

Assumption 1 (**SI Replicas**)**.** *For each site $k \in I_m$, every MV-schedule H_t^k over L^k is an SI-schedule.*

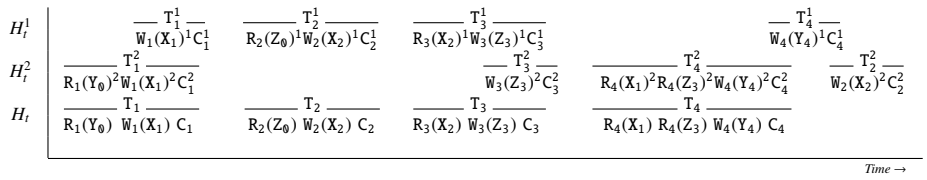

Figure 1: An example of a 1MV-schedule that does not provide a 1SI- nor 1GSI-schedule.

One could intuitively infer from the previous assumption that if each replica $k \in I_m$ of the RDBS generates an SI-schedule and runs an instance of the ROWA protocol (Property P1) then the whole system will generate a 1SI-schedule. However, the next example proves that the usage of individual SI-schedules at each $k \in I_m$ does not ensure that the global isolation level obtained in RDBS is a 1SI- or a 1GSI-schedule.

Example 2. *In this example two sites $\{I_1, I_2\}$ and the next set of transactions $\{T_1, T_2, T_3, T_4\}$ are considered: $T_1 = R_1(Y)\, W_1(X)\, C_1$, $T_2 = R_2(Z)\, W_2(X)\, C_2$, $T_3 = R_3(X)\, W_3(Z)\, C_3$, $T_4 = R_4(X)\, R_4(Z)\, W_4(Y)\, C_4$.*

Figure 1 illustrates the mapping described in Theorem 1 for building a 1MV-schedule from the SI-schedules seen in the different sites I_m. T_2 and T_3 are locally executed at site I_1 ($RS_2 \neq \emptyset$ and $RS_3 \neq \emptyset$, see H_t^1 in Figure 1) whilst T_1 and T_4 are executed at site I_2 respectively (H_t^2 in Figure 1). The writesets are afterwards applied at their respective remote sites.

In the 1MV-schedule H_t of Figure 1, T_4 reads X_1 and Z_3 but the X_2 version exists between both (since X_2 was installed at site I_1). T_1 and T_2, satisfying that $WS_1 \bigcap WS_2 \neq \emptyset$, are executed at both sites in the same order. As T_1 and T_2 are not executed in the same order with regard to T_3, the obtained 1MV-schedule is not a 1GSI-schedule.

This example has indeed shown that the usage of individual SI-schedules at each $k \in I_m$ does not ensure that the global isolation level obtained in RDBS is 1SI or 1GSI. Schedules H_t^1 and H_t^2 are valid SI-schedules, but despite building the MV-schedule H_t using our proposed approach, its transaction T_4 reads an X version (X_1) that should not belong to its intended snapshot. In what follows we will see further properties that these ROWA protocols do satisfy to generate one-copy schedules that are either 1SI-Schedules (Section 6) or 1GSI-Schedules (Section 7).

6. One Copy Snapshot Isolation Schedules

If a ROWA replication protocol wants to generate 1SI-schedules, it seems natural to assume the requirements set up in Definition 4. Hence, a nice property to be considered is that the commit operations of conflicting transactions must be totally ordered at site $k \in I_m$, i.e., it serializes the execution of conflicting transactions in the 1MV-schedule as the second part of Definition 4 states. This reasoning is formalized in the next property.

Definition 10 (**P2** – Total Order of Conflicting Transactions)**.** *Given a set of logical transactions L, then an R-schedule S_t over the ROWA Transformation $\mathcal{T}(L)$ satisfies Property P2 provided*

that for every T_i, $T_j \in L$ such that $WS_i \bigcap WS_j \neq \emptyset$, and for every k, $k' \in I_m$, if $c_i^k < c_j^k$ in S_t, then $c_i^{k'} < c_j^{k'}$ in S_t.

Property P2 serializes the execution of conflicting transactions in the 1MV-schedule (as Definition 4 about SI-schedules requires). However, the execution of conflicting transactions in the same order at all sites does not ensure SI nor GSI. This has been previously seen in Example 2 where only T_1 and T_2 are conflicting transactions and they have been executed in the same order at every site. It is necessary that a transaction reads the latest installed version in the system of any data item; this corresponds to the first part of Definition 4. The following property formally states that.

Definition 11 (**P3** – Latest-Version Read)**.** *Given a set of logical transactions L, then an R-schedule S_t over the ROWA Transformation $\mathcal{T}(L)$ satisfies Property P3 provided that for every T_i, $T_j \in L$ such that $WS_j \cap RS_i \neq \emptyset$, if $R_i(X_j)^{site(i)} \in S_t$ then no transaction $T_r \in L$ with $X \in WS_r$ satisfies $c_j^{site(i)} < c_r^k < b_i^{site(i)}$ in S_t for every $k \in I_m$.*

If we define a ROWA protocol that ensures Properties P1, P2 and P3 with SI replicas (Assumption 1), it is expected that its associated 1MV-schedule satisfies Definition 8 with regard to a 1SI-schedule. This is formally shown in the following theorem.

Theorem 2. *Given a set of logical transactions L and an R-schedule S_t over the ROWA Transformation $\mathcal{T}(L)$. If S_t satisfies Assumption 1 (SI replicas) and also Properties P1, P2 and P3, then S_t is a 1SI-schedule.*

Proof. By the conditions in Theorem 1, S_t is a 1MV-schedule. Thus, let H_t be the MV-schedule over L such that $H_t = \mathcal{F}(S_t)$ (Definition 7). By Definition 8 of 1SI-schedule, we will prove that H_t is an SI-schedule (Definition 4). For each $T_i \in L$:
1) if $R_i(X_j) \in H$ then $X_j \in Snapshot(DB, H_t, b_i)$.

Thus, $\{X_j\} = latestVer(X, H_t, b_i)$ by Definition 2. Suppose $R_i(X_j) \in H$ and $X_j \notin latestVer(X, H_t, b_i)$. There is a version X_r installed by some transaction T_r such that $c_j < c_r < b_i$.
If $R_i(X_j) \in H$, by construction of H_t in Definition 7, $R_i(X_j)^{site(i)} \in H^{site(i)}$. From Assumption 1 (SI replicas), it is satisfied that $\{X_j^{site(i)}\} = latestVer(X^{site(i)}, H_t^{site(i)}, b_i^{site(i)})$ and $c_j^{site(i)} < b_i^{site(i)}$. Also, Property P1 (Atomicity) ensures that $C_r^{site(i)} \in H^{site(i)}$ and $X_r^{site(i)} \in Ver(X^{site(i)}, H_t^{site(i)})$. As $X \in WS_j \cap WS_r$, by Property P2, if $c_r^{site(i)} < c_j^{site(i)}$ then $c_r < c_j$. Since this contradicts the initial supposition, we have that $c_j^{site(i)} < c_r^{site(i)}$. Then by Property P3 as $X \in WS_r$, $b_i^{site(i)} < c_r^k$ holds for every $k \in I_m$. Thus, $b_i^{site(i)} < c_r^{min(r)}$ holds. By construction of H_t (Definition 7): $b_i < c_r$. A contradiction with the initial supposition.
2) for each $T_j \in L$: $\neg wconflict(T_j, T_i, H_t, b_i)$.

By Assumption 1 (SI replicas), at any site $k \in I_m$ we have that H_t^k is an SI-schedule. By Definition 2; for each pair $T_j^k, T_i^k \in L^k$: $\neg wconflict(T_j^k, T_i^k, H_t^k, b_i^k)$. That is, $WS_j^k \cap WS_i^k = \emptyset \vee \neg(b_i^k < c_j^k < c_i^k)$.
(2.a) If $WS_j^k \cap WS_i^k = \emptyset$ then by Property P1 and Definition 5 (ROWA Transformation) we have that $WS_j \cap WS_i = \emptyset$, for $T_i, T_j \in L$. Thus, $\neg wconflict(T_j, T_i, H_t, b_i)$ holds.

(2.b) If $WS_j^k \cap WS_i^k \neq \emptyset$ for $T_i, T_j \in L$, then $\neg(b_i^k < c_j^k < c_i^k)$ holds. Again (by Property P1 and Definition 5) for T_j and T_i we have that $WS_j \cap WS_i \neq \emptyset$. Let us suppose (by contradiction) that $wconflict(T_j, T_i, H_t, b_i)$ holds. Thus, $b_i < c_j < c_i$ holds. By construction of H_t in Definition 7, $b_i^k = b_i$ while $c_j = c_j^{min(j)}$ and $c_i = c_i^{min(i)}$ respectively; more precisely, $c_j^{min(j)} \leq c_j^k$, $c_i^{min(i)} \leq c_i^k$ for all $k \in I_m$. By Property P2, either (a) $c_j^k < c_i^k$ or (b) $c_i^k < c_j^k$. In the former, $c_j^{min(j)} < c_i^{min(i)}$ while in (b) $c_i^{min(i)} < c_j^{min(j)}$, i.e. $c_i < c_j$ that leads to a contradiction. Hence, only (a) is possible. As $\neg(b_i^k < c_j^k < c_i^k)$, then either $c_j^{min(j)} \leq c_j^k < b_i^k$ or $c_i^k < c_j^k$. That is, either $c_j < b_i$ or $c_i < c_j$. A contradiction with $b_i < c_j < c_i$ is obtained and 2) holds. □

Property P3 forces compliant ROWA protocols to ensure that a transaction sees the latest installed version in the system. As the assumed ROWA protocols only send the writeset of a transaction to remote sites, it can not be checked whether another site has installed a more recent version than that currently received by a given replica. As a direct consequence of this, it is not possible to abort the transaction violating the SI level. In [14, 5], when a database site receives a transaction, it has to provide that transaction with the latest snapshot or a desired snapshot. This forces the site either to delay the start of the transaction, or to stop accepting new transactions during some time intervals. Thus, a transaction would be blocked from starting until Property P3 becomes true; however, this issue goes against the main benefit of SI.

In the next section, we are going to introduce additional properties that relax the need to block or delay the start of transactions. ROWA protocols verifying these properties will ensure a GSI level to transactions; as already mentioned, GSI maintains all nice properties of SI but allows transactions to read from an older snapshot.

7. One Copy Generalized Snapshot Isolation Schedules

Properties P1, P2 and P3 allow a ROWA protocol to obtain 1SI-schedules. However, Property P2 is not enough for obtaining such schedule –without Property P3–, and Property P3 is too restrictive, since it needs to block the start of transactions in order to be held. Most of existing replication protocols actually guarantee a commit total order in all replicas. Indeed, primary copy solutions [34, 12, 29] rely on the concurrency control of the primary whilst secondaries apply update transactions in that order. Update everywhere solutions [14, 27, 5] obtain the commit ordering thanks to the total order broadcast primitive [9] (more details about this will be given in Section 9). As a result, since all replicas generate SI-schedules (by Assumption 1) and their local snapshots have received the same sequence of updates, transactions starting at any site are able to read a snapshot that perhaps is not the latest one, but it is consistent with those of other replicas. These two characteristics are equivalent to those expressed in Definition 3. This total order of update transactions is formally introduced in the next property.

Definition 12 (**P4** – Total Order of Update Transactions)**.** *Given a set of logical transactions L, then an R-schedule S_t over the ROWA Transformation $\mathcal{T}(L)$ satisfies Property P4 provided that for every pair of update transactions $T_i, T_j \in L$ and for every $k, k' \in I_m$, if $c_i^k < c_j^k$ in S_t, then $c_i^{k'} < c_j^{k'}$ in S_t.*

The aim of the next theorem is to prove that the 1MV-schedules generated by a ROWA protocol that satisfies Properties P1 and P4 are actually 1GSI-schedules; i.e., they comply with all conditions stated in Definition 3.

Theorem 3. *Given a set of logical transactions L, and an R-schedule S_t over the ROWA Transformation $\mathcal{T}(L)$. If S_t satisfies Assumption 1 (SI replicas) and also Properties P1 and P4, then S_t is a 1GSI-schedule.*

Proof. By conditions of Theorem 1, S_t is a 1MV-schedule. Thus, let H_t be the MV-schedule over L such that $H_t = \mathcal{F}(S_t)$ (Definition 7). By Definition 8 of a 1GSI-schedule, we will prove that H_t is a GSI-schedule (Definition 3). Under Assumption 1, Property P1 and Property P4, the SI-schedules H_t^k have the same total order of committed transactions. Without loss of generalization, we consider the following total order in the rest of the proof: $c_0^k < c_1^k < c_2^k < \cdots < c_n^k$ for every $k \in I_m$. H_t also satisfies such an order $c_0 < c_1 < c_2 < \cdots < c_n$ by construction of H_t (Definition 7). The proof is done by induction over such a total order. Firstly, we define for each $T_i \in L$ the subsets of transactions $L(i) = \{T_0, T_1, T_2, \ldots, T_i\} \subseteq L$ and $L^k(i) = \{T_0^k, T_1^k, T_2^k, \ldots, T_i^k\} \subseteq L^k$ for each $k \in I_m$. Using these subsets we define $H_t^k(i)$ and $H_t(i)$. They are exactly equal to H_t^k and H_t respectively, except that they only include the operations in $L^k(i)$ or $L(i)$. Thus, it is clear that $H_t^k(n) = H_t^k$ and $H_t(n) = H_t$.

Induction Base. $H_t(0)$ is a GSI-schedule. $L(0) = \{T_0\}$ so $H_t(0)$ trivially satisfies Definition 3.

Induction Hypothesis. $H_t(j)$ is a GSI-schedule $1 \leq j \leq i-1$.

Induction Step. We will prove that $H_t(i)$ is a GSI-schedule, with $T_i \in L$. Note that $L(i) = L(i-1) \cup \{T_i\}$. As $H_t(i-1)$ is a GSI-schedule, by Hypothesis, for any pair $T_j, T'_j \in L(i-1)$ it is held that $\neg wconflict(T_j, T'_j, H_t(i-1), s'_j)$. As $c_j < c_i$ for $0 \leq j \leq i-1$, by the considered total order, $\neg wconflict(T_i, T_j, H_t(i), s_j)$. If $R_j(X_r) \in H(i-1)$ and $X_r \in Snapshot(DB, H_t(i-1), s_j)$ for $0 \leq j \leq i-1$ then $R_j(X_r) \in H(i)$. $X_r \neq X_i$ because $c_j^{site(j)} < c_i^{site(j)}$ and $H_t^{site(j)}$ is an SI-schedule and, hence, $X_r \in Snapshot(DB, H_t(i), s_j)$.

Therefore, in order to prove that $H_t(i)$ is a GSI-schedule, we only need to prove for $T_i \in L$ that there exists a value $s_i \leq b_i$ such that:

(1) if $R_i(X_r) \in H(i), X_r \in Snapshot(DB, H_t(i), s_i)$; and,

(2) for each $T_j \in L(i)$: $\neg wconflict(T_j, T_i, H_t(i), s_i)$.

Proof of (1) Consider $RS_i \neq \emptyset$, if $RS_i = \emptyset$ then it holds. Let T_{i_0} be the last transaction in the schedule from which T_i does not obtain a newer version from transactions committed after T_{i_0} when T_i performs a read operation. By Assumption 1, for all $Y \in WS_{i_0} \cap RS_i$, $\{Y_{i_0}^{site(i)}\} = latestVer(Y^{site(i)}, H_t(i)^{site(i)}, b_i^{site(i)})$. By construction of H_t (Definition 7), $c_{i_0} \leq c_{i_0}^{site(i)} < b_i^{site(i)}$ and $c_{i_0} < b_i$ in H_t respectively.

Let $X \in RS_i$ be an item read by T_i such that $X \notin WS_{i_0} \cap RS_i$ and $X_r^{site(i)} = latestVer(X^{site(i)}, H_t(i)^{site(i)}, b_i^{site(i)})$. We prove that $\nexists T_k \in L\colon X_k \in Ver(X, H_t) \wedge c_r < c_k < c_{i_0}$. By contradiction, if there exists T_k and $c_r < c_k < c_{i_0}$ then by Properties P4 and P1: $c_r^{site(i)} < c_k^{site(i)} < c_{i_0}^{site(i)}$. Thus, $X_r^{site(i)}$ is not the latest version in $H_t(i)^{site(i)}$ at $b_i^{site(i)}$. Therefore, for all $X \in RS_i$, if $R_i(X_r) \in H(i)$ then $X_r \in Snapshot(DB, H_t(i), c_{i_0})$. As $c_{i_0} < b_i$, by construction of H_t (Definition 7), then there exists $s_i \in \mathbb{R}^+$ such that $c_{i_0} \leq s_i \leq b_i$ from which (1) holds.

Proof of (2). By previous proof of (1), $T_{i_0} \in L(i)$ and $c_{i_0} \leq s_i \leq b_i$ in $H_t(i)$. The begin time $b_i^{site(i)}$ and the commit time $c_{i_0}^{site(i)}$ of transaction T_{i_0} at $site(i)$ allow us to define the next two sets:

$L_1(i) = \{T_j \in L\colon b_i^{site(i)} < c_j^{site(i)} < c_i^{site(i)}\}$

$L_2(i) = \{T_j \in L\colon c_{i_0}^{site(i)} < c_j^{site(i)} < b_i^{site(i)}\}$
By Property P1 and Assumption 1: $WS_j \cap WS_i = \emptyset$ for any $T_j \in L_1(i)$. By definition of $T_{i_0} \in L(i)$, Property P1, Assumption 1 and the proof of (1): $WS_j \cap RS_i = \emptyset$ for any $T_j \in L_2(i)$. Let $T_{i_2} \in L_2(i)$ be the last transaction such that in the total order it satisfies $WS_{i_2} \cap WS_i \neq \emptyset$ and let $T_{i_1} \in L_1(i)$ be the first transaction such that $c_{i_1} < b_i$ in H_t and $WS_{i_1} \cap RS_i \neq \emptyset$. Note that in H_t, obtained from Definition 7, a commit time $c_j^{site(i)}$ for a transaction in $L_1(i)$ may change its relation with respect to b_i, but maintains its order relation with respect to the other commit times; in particular $c_{i_2} < c_{i_1}$. Thus, $c_{i_0} < c_{i_2} < c_{i_1} < b_i$ holds in H_t.
For any value $s_i \in (c_{i_2}, c_{i_1})$ (2) holds by the way $T_{i_2} \in L$ has been defined. More precisely, for each T_j such that $c_{i_2} < c_j < b_i$, if $T_j \in L_2(i)$ then $WS_j \cap WS_i = \emptyset$. Otherwise, T_{i_2} is not the last transaction satisfying such a condition; and if $T_j \in L_1(i)$ then $WS_j \cap WS_i = \emptyset$. Thus, these transactions do not conflict with T_i. The rest of transactions do not either conflict with T_i because their commit times are sooner than s_i.
To conclude, if there does not exist T_{i_1}, then $s_i = b_i$ and, therefore (2) holds. In case that T_{i_2} does not exist then $s_i \in (c_{i_0}, c_{i_1})$ and again (2) holds. □

To the best of our knowledge, this proof has not been given before in any ROWA-based SI replication protocol ensuring total order for the commit operations of all updated transactions in the system replicas. This theorem formally justifies protocols correctness and establishes that their resulting isolation level is 1GSI. However, P4 does impose a very restrictive order of update transactions: all of them have to be serialized. In the following section we are going to relax this property in order to apply certain update transactions concurrently and even in different order.

8. Relaxing Properties

There is a middle point from what is set up in Property P4, enforcing that every single update transaction is committed in the same order at all replicas, and Property P2, requiring that only conflicting update transactions must be committed in the same order at all replicas respectively. We need to guarantee that conflicting transactions are executed in the same order while the rest of non-conflicting transactions can be committed in any other. However, though this can be easily achieved in a centralized setting, it is not trivial in a replicated setting since, e.g., a non-conflicting transaction can be applied between two conflicting transactions at one site while at another one just after the second conflicting transaction has been applied. Hence, snapshots generated at these two sites may differ. We have to set up a consistent partial order for applying all transactions, i.e., for every pair of conflicting transactions the same set of non-conflicting transactions are applied. The next property formalizes the previous reasoning and constitutes a weakened version of Property P4 while it is still stronger than Property P2.

Definition 13 (**P5** – Non-Conflicting Transactions Interleaving)**.** *Given a set of logical transactions L, then an R-schedule S_t over the ROWA Transformation $\mathcal{T}(L)$ satisfies Property P5 provided that for every $T_i, T_j \in L$ and for every $k, k' \in I_m$ such that $WS_i \cap WS_j \neq \emptyset$: if there exists $T_p \in L$ with $WS_p \neq \emptyset$ such that $c_i^k < c_p^k < c_j^k$ in S_t then $c_i^{k'} < c_p^{k'} < c_j^{k'}$ in S_t.*

Next, we are going to see in an example how it is possible to build an R-schedule where Property P5 holds.

Example 3. *Let us suppose that there are two replicas* $\{I_1, I_2\}$ *and the set of update transactions:* $\{T_0, T_1, T_2, T_3, T_4, T_5, T_6\}$*, with* $WS_1 \cap WS_3 \neq \emptyset$*,* $WS_2 \cap WS_6 \neq \emptyset$ *and the rest do not conflict among each other. At the first site you can find this commit order of the SI-schedule:* $c_0^1 < c_1^1 < c_2^1 < c_3^1 < c_4^1 < c_5^1 < c_6^1$ *whilst at the second site it can be:* $c_0^2 < c_1^2 < c_2^2 < c_3^2 < c_5^2 < c_4^2 < c_6^2$*. In the latter, the commit ordering of* T_4 *and* T_5 *is different from the scheduling of the former.*

Property P5 becomes Property P4 whenever the pattern of transactions does not allow transactions commit reordering. In Example 3, $c_3^2 < c_2^2$ cannot happen without violating Property P5. On the other hand, if all transactions do not conflict, then the derived schedule H_t^k at each replica will have nothing in common with the one of any other replica. This issue is of great importance since data items read by these transactions would not belong to any valid global snapshot (recall Definitions 3 and 4 of an SI- and GSI-schedule, respectively). This can be overcome by constraining that each transaction reads from the same global snapshot like in Property P3 but relaxed. Thus, the snapshot gotten by a transaction at its delegate corresponds to a valid point in the R-schedule S_t such that is equivalent to a valid global snapshot; recall Definition 7.

Definition 14 (**P6** – Compatible Snapshot Read)**.** *Given a set of logical transactions L, then an R-schedule* S_t *over the ROWA Transformation* $\mathcal{T}(L)$ *satisfies Property P6 provided that for each* $T_i \in L$*,* $\exists\, s_i \in \mathbb{R}^+$ *with* $s_i \leq b_i^{site(i)}$ *such that if* $R_i^{site(i)}(X_j) \in S$ *then* $X_j^{site(i)} \in Snapshot(\cup_{k \in I_m} DB^k, S_t, s_i)$*.*

From the R-schedule S_t definition given in Definition 6, it is easy to infer that S_t is an MV-schedule of $\mathcal{T}(L)$. This transformation $\mathcal{T}(L)$ refers to $\cup_{k \in I_m} DB^k$ and, hence, $Snapshot(\cup_{k \in I_m} DB^k, S_t, s_i)$ is well defined in this last definition. From Property P6 one can infer that replicas are somehow imposed to install certain snapshots; i.e. those belonging to a valid global snapshot from the 1MV-schedule derived by the ROWA Protocol (satisfying Property P1). It is easy to see that these forced snapshots are those imposed by Property P5. In the next theorem we prove that R-schedules complying with Properties P5 and P6 satisfy Definition 3; i.e., they generate 1GSI schedules.

Theorem 4. *Given a set of logical transactions L, and an R-schedule* S_t *over the ROWA Transformation* $\mathcal{T}(L)$*. If* S_t *satisfies Assumption 1 (SI replicas) and also Properties P1, P5 and P6 then* S_t *is a 1GSI-schedule.*

Proof. By conditions of Theorem 1, S_t is a 1MV-schedule. Thus, let H_t be the MV-schedule over L such that $H_t = \mathcal{F}(S_t)$ (Definition 7). By Definition 8 of a 1GSI-schedule, we will prove that H_t is a GSI-schedule (Definition 3). Additionally, by Property P6, we have that for each $T_i \in L$ if $R_i^{site(i)}(X_j) \in S_t$ then $X_j^{site(i)} \in Snapshot(\cup_{k \in I_m} DB^k, S_t, s_i)$. By Theorem 1 and Definition 7, it is derived that $X_j \in Snapshot(DB, H_t, s_i)$. Finally, by Assumption 1 and construction of Theorem 1, we obtain that $X_j^{site(i)} \in Snapshot(DB^{site(i)}, H_t^{site(i)}, b_i^{site(i)})$ with $s_i \leq b_i$; this fact makes true Condition (1) in Definition 3. In particular, if $s_i = b_i$ for every $T_i \in L$ then S_t is trivially a 1GSI-schedule. Otherwise ($s_i < b_i$), there must be a transaction $T_p \in L$ such that $s_i < c_p < b_i$ and $WS_p \cap RS_i \neq \emptyset$; moreover, $WS_j \cap WS_p = \emptyset$ and $c_j^{site(i)} < c_p^{site(i)}$. Let T_p be the first transaction in S_t satisfying such condition. On the other hand, by Theorem 1, Definition 7 and Property P5, we have that either $c_j < c_p$, this is not possible by Property P6 as in such a

case s_i must have been shifted to match c_p; or, $c_p < c_j$ that leads to a contradiction. Therefore, Condition (1) of Definition 3 is satisfied.

If we focus on Condition (2) of Definition 3 we have to verify if for each $T_j \in L$: $\neg wconflict(T_j, T_i, H_t, s_i)$. Again, two intervals must be considered (s_i, b_i) and (b_i, c_i). In the first case, let T_j be the last transaction that satisfies $\neg wconflict(T_j, T_i, H_t, s_i)$ $(c_j < s_i)$ and $WS_j \cap WS_i \neq \emptyset$. Assume that there exists a transaction T_p such that $WS_p \cap WS_i \neq \emptyset$ and $WS_j \cap WS_p \neq \emptyset$ and $s_i < c_p < b_i$; it is also worth noting that, $c_j^{site(i)} < c_i^{site(i)}$. Hence, we have that $c_j^{site(i)} < c_p^{site(i)} < c_i^{site(i)}$ which is not possible since, by Property P1.1 the rest of replicas $k \in I_m \setminus \{site(i)\}$ will have $c_j^k < c_i^k$ that go against Property P5, conflicting transactions must be applied in the same order at all replicas. Whilst in the second range, (b_i, c_i), we use the same transaction T_j and assume there is a transaction T_p such that $WS_p \cap WS_i \neq \emptyset$ and $WS_j \cap WS_p \neq \emptyset$ that leads to the same conclusion, it violates Property P5. Hence, the derived 1MV-schedule is a 1GSI-schedule. □

As pointed out before, the ideal global snapshot candidate could be the snapshot obtained after executing a consecutive pair of conflicting transactions, as Property P5 indicates. However, special care has to be taken to set up the start of local transactions at their respective delegate replicas: either immediately started up provided that they will access data items that belong to a global valid snapshot (e.g. by means of stored procedures); or, delayed until the last conflicting writeset has been applied. These ideas have been implemented in [27] and will be discussed in the next section.

9. Related Work

Let us start with a brief presentation of previous works that have provided correctness proofs for database replication protocols using SI replicas, to be followed by another subsection where such protocols are described according to how they comply with the conditions identified in our paper.

9.1. Previous Correctness Justifications

SI is a relatively recent isolation level, since it was first identified in [7]. At that time, all database replication protocols were focused in guaranteeing 1-copy-serializability (1CS) and their correctness proofs were inspired in those given in [8]. As SI was based on multi-versioned concurrency control and such latter work also described that mechanism in one of its chapters and replication strategies in another, it still provided a good guide for proving the correctness of SI protocols. In such work, two complementary aspects were considered. Firstly, one-copy equivalence should be proved. To this end, execution histories are used and one needs to prove that the *Replicated Data History* (RD history [8]) is equivalent to a logical one-copy serial history. We have followed a similar approach in our work, and some of our first definitions are inspired in [8]. Secondly, such resulting history should comply with the intended isolation level. In [8] such level was the serializable one, and on that purpose a *replicated data serialization graph* (RDSG) was used. This second aspect has not been followed in our paper. Thus, instead of a graphical representation of transactions dependencies –that can only be checked over all possible schedules generated by a protocol– we have preferred to identify a set of conditions that can be easily checked either against the schedules (as in the previous approach) or against the steps followed by each particular algorithm/protocol being analyzed. We believe that this second alternative

is easier to use and that it generates more intuitive correctness proofs than a graph-based one. Moreover, our correctness conditions are general. Additionally, they can be directly applied to all types of replication protocols [16]: both primary-copy and update-everywhere (considering server architecture), and both eager and lazy (regarding update propagation).

Note that RDSG did not provide any support for non-serializable isolation levels. This issue was solved in [1], where serialization graphs were extended to all currently specified isolation levels and a set of validation rules were given for all of them. Inspired in such work, Lin et al. [26] have provided a graph-inspired specification of the correctness conditions of SI replication protocols, including also support for integrity constraints. So, that work perfectly complements ours, using a widely accepted approach.

Another important parameter that has conditioned recent works related to database replication protocol correctness is the mechanism being assumed for update propagation. Until 1997, database replication protocols were based on the same principles used in distributed databases: usage of subtransactions for managing their execution and usage of two-phase (2PC) [24, 39] or three-phase commit (3PC) [38] for managing their termination. But the advances in *group communication systems* [9] allowed the elimination of both mechanisms [2], replacing them by writeset propagation (instead of subtransactions) using reliable total order broadcast (instead of 2PC). Due to this, correctness proofs, although still inspired in the examples given in [8] were now relying on the sequential application guaranteed by total-order broadcasts and on the validation rules being used in such new protocols. Valid examples of such new approach can be found in [21].

Note that update propagation in total order does not guarantee that all replicas will be able to apply all writesets at a similar pace, but only in the same sequence. This implies that there can be a quite long interval from the moment a first site applies a given writeset to the instant that the slowest site also applies it. In the meantime, the transaction associated to such writeset would have been able to commit at some sites. Due to this, a given client is able to perceive a transaction inversion (i.e., new read requests are retrieving older values than past read requests) if some of its read accesses are forwarded to different server replicas [31]. This explains why in our Definition 7, the commit/write time assigned to the commit or write operations in a 1MV-schedule is that of the first replica that has executed such operations. Note that if for a given update action we had selected the slowest replica as that to be mapped into the 1MV-schedule, some valid read operations from subsequent transactions (executed in the delegate or other fast replicas, and accepted in common replication protocols) would have been rejected in the resulting 1MV-schedules.

The existence of several variants of SI in replicated environments was first presented in [14], where GSI was specified and regular SI was named *Conventional SI* (CSI). Such work was later refined in [12] where *session consistency* was introduced, GSI was renamed as *weak SI* and regular SI as *strong SI*, although such variants were only supported in that paper using a lazy primary copy protocol.

Finally, another general correctness justification for SI protocols has been given in [15]. Such correctness proof is applicable to any *certification-based* database replication protocol that uses SI replicas –so, it is not so general as the one presented in this paper–, and it is inspired in the traditional distributed algorithm proofs; i.e., besides one-copy equivalence it justifies the safety and liveness of the protocol assuming also that crash failures may occur. Most of the published correctness proofs, ours included, do not deal with failures.

9.2. Discussion of Concrete Protocols

This paper has presented a set of properties that ROWA protocols must satisfy in order to generate 1SI- or 1GSI-schedules. This obviously implies that already developed replication protocols do satisfy the proper subset of properties to generate the desired schedule and new protocols can be derived by considering the properties while designing them. In this section we are going to consider already existing replication protocols [20, 41, 27, 30, 34, 12, 5, 29] that follow the ROWA approach, assume that the underlying DBMS provides SI (Assumption 1) and ensure, each one in a different manner, the atomicity of transactions as Property P1.1 (*Atomicity*) states. To that purpose, we will consider the classification given in [16] according to who can perform update transactions (*primary-copy*/*update-everywhere*) and when updates are propagated (*eager*/*lazy*).

Let us start with the latter, eager protocols are those where updates are propagated to the rest of replicas as part of the transaction being executed at the delegate. On the contrary, lazy ones are those that propagate updates asynchronously, once the main transaction has been committed, typically as a separate transaction for each site. In both cases, all protocols do satisfy Property P1.2 (*Deferred-Updates*) as transactions are firstly executed at their delegate replica and at commit time changes are sent to the rest of replicas (recall that read-only transactions do not interact with the rest of replicas). Moreover, these replication protocols follow a *mix* approach where replicas are not necessarily updated inside transaction boundaries. However, the system always makes sure that temporary inconsistencies are hidden from the clients and, therefore, it acts like an eager replication system. For example certification-based [20, 41, 27, 14, 30, 5] or primary copy [34, 12, 29] protocols will apply updates in the context of a remote transaction. However, in most of these proposals, clients will not be aware of that, since we can forward the next transaction of a client to a site where it has been applied by tagging [34, 5] or defining the context of a session [12]. Recently, it has been shown in [29] that it is possible to obtain the latest snapshot for any transaction by regulating the snapshot acquisition through mutual exclusion; or, by delaying the start of transactions [23] until all previous updates have been applied. In any case, our formalization supports these approaches since all of them satisfy Property P1.

With regard to who can perform updates, primary-copy protocols [34, 12, 14, 22, 29] perform updates at a single site (the primary or master) and updates are later propagated to the rest of replicas; on the other hand, read-only transactions can be executed in any replica (primary or secondaries). This permits us to state that this family of protocols, as noted in the previous paragraph, satisfies Property P1. Besides, these protocols trivially hold Property P2 (*Total Order of Conflicting Transactions*) as update transactions are exclusively executed at the primary. Hence, these transactions are ordered by the primary's associated DBMS; and, afterwards, are multicast in that order (using FIFO primitive [9]) to the rest of sites where they are applied observing the order of delivery. A similar mechanism sets up a procedure to execute write operations in the same order following the first updater wins rule in [29] by a leader replica while the followers simply replicate it. Property P3 (*Latest-Version Read*) can be satisfied in different ways; the most straightforward one is to execute all read-only transactions at the primary [34, 12, 14]. Nevertheless, there are other ways to attempt this without causing a bottleneck at the primary or compromising scalability by exchanging additional messages. This can be done either by acquiring a snapshot through a critical region as in [29]; or, by delaying the start of a given transaction until all previous committed updates have been applied [23]. Hence, in this case primary-copy satisfies all requirements to generate lSI-schedules. In order to provide 1GSI-schedules, Property P4 (*Total Order of Update Transactions*) is held because update transactions are executed at the

primary and applied in the same order at the rest of secondaries while, in these protocols, read-only transactions can be executed at any secondary replica where they probably obtain an older snapshot. Therefore, they generate lGSI-schedules. Finally, to the best of our knowledge, neither Property P5 (*Non-Conflicting Transactions Interleaving*) nor Property P6 (*Compatible Snapshot Read*) have been studied for primary-copy systems to date. On the contrary, update-everywhere protocols [20, 41, 27, 14, 30, 5] execute update transactions at their respective delegate sites (which can be any site) and some coordination is needed to commit update transactions. SI-based update-everywhere protocols perform a *certification* test [14] in order to commit an update transaction. Certification consists in porting the *first-committer-wins* rule [7] to the replicated environment. Upon the commit request of a transaction at its delegate replica, the writeset is collected from the database along with the snapshot version gotten at the database and must pass a validation test (*certification*). It checks if the writeset intersects with any other writeset of previously validated transactions that are concurrent to the transaction (specified by their snapshot version). Again, this way of committing transactions ensures that Property P1 holds. The most straightforward way to implement this technique is to choose a central certifier replica [14] that stores all successfully validated transactions and performs all the validations; thus, all replicas reach the same decision. This central certifier is in charge of propagating writesets in the order they are validated to the rest of sites that apply them in this order. Hence, Property P4 is satisfied and generates a lGSI-schedule; this replication technique has been implemented in Tashkent [13] or in [23]. It is possible to replicate the role of the master certifier (in order to increase the availability and avoid the bottleneck of the central certifier) to all replicas if it is ensured that writesets are delivered in the very same order at all replicas so the validation test is replicated at all sites. Thus, all of them will reach the same decision solely based on the delivered transaction (i.e., the writeset and version tuple) and previously validated update transactions [20, 41, 27, 30, 5]. Again, we have that this kind of protocol applies transactions in the order they have been validated (Property P4) that generates lGSI-schedules. If this certification protocol wants to satisfy Property P3, the obtained R-schedule must ensure that a transaction obtains the latest snapshot at its delegate replica. Nevertheless, this may lead to delaying the start of the transaction or stop accepting new transactions (preventing transactions from starting at any time and thus losing one of the advantages of SI: its non-blocking behavior for read accesses). This leads to a blocking implementation of this kind of schedules as it has been already indicated in [14]: "*There is no non-blocking implementation of SI in an asynchronous system, even if database sites never fail*". However, data freshness will depend on the needs of applications whilst some applications demand up-to-date data others will prefer a fast response time with outdated data. These ideas have been ported to database replication in two directions either requiring a certain snapshot [35] or ensuring that the retrieved snapshot is not older, in a certain amount, than the latest snapshot version [5]. Note that the ideas presented in [23, 29] can perfectly fit here too for update everywhere protocols to provide strong consistent replication at the price of blocking. Finally, a certification based protocol variation is presented in [27] that permits the concurrent execution of non-conflicting validated transactions so that Property P5 is satisfied. However, this relaxation triggers the appearance of holes in the validated sequence of transactions (i.e. those ones committed before their validation order) which in turn generates inconsistent global snapshots. Therefore, the start of new transactions must be blocked until no holes are present to satisfy Property P6 and generate a lGSI-schedule. Nevertheless, this is an interesting approach in terms of performance as transaction reordering has been shown as a valid approach to increase the throughput of the system [28, 36].

These properties have been presented for full database replication protocols, but they also

encompass partial replication. This is an attractive solution to scale databases by horizontal partitioning [37, 6, 10, 25] or to cope with mobile transactions [19]. To this end, we should consider that those properties stated for all replicas do not refer to all server nodes but only to the database nodes that are configured to hold a copy of the item (or items) being accessed in each transaction. Most of partial replication protocols [37, 6] are certification-based and hold Properties P1 and P4 at the sites where there exists a copy of the database partition and, thus, generate 1GSI-schedules. On the other hand, there exists a family of protocols that uses some heuristics, statistics and graph partitioning algorithms, to minimize the number of transactions to be executed in different replicas with the analysis of the workloads [10, 25]. Hence, if we obtain a perfect partitioning (i.e., each transaction only accesses one partition) for a given application, it is naturally satisfied that Properties P5 and P6 hold. This increases the application performance, as shown in [10, 37], and generates 1GSI-schedules in each partition, independently from the rest. Other partitionings (i.e., non-perfect ones) are seldom assumed in partial replication. In that case, the protocol needs additional steps to guarantee the intended isolation level and its performance is hardly acceptable.

Finally, considering scalability –specially important in cloud computing environments where snapshot isolation could be used–, it is worth mentioning that Property P3 could be supported in structured overlay networks, as suggested in [4]. This could be taken as a basis for developing SI-related replication protocols in those systems. Recently, a couple of systems have been developed to support transactions in cloud-based environments. All of them assume a data storage rather than a DBMS in each replica and data partitioning [11, 3, 32, 17]. They take advantage of the fact that transactions only access a small portion of data (i.e., a partition); hence, they define a primary replica for each partition where update transactions are executed, while read-only transactions can be executed on any replica storing a copy of that partition. This kind of protocol trivially holds Properties P1 and P4 for replicas that store a copy of a given partition; further, they also trivially hold Properties P5 and P6. All this reasoning leads us to assure that they generate 1GSI-schedules. Nevertheless, these systems stem from the same problem of non-perfect partitioning as in partial database replication. This issue is specially true for a transaction that updates two different partitions. We have two different approaches to solve this situation: to implement a variant of the *2 Phase Commit* protocol as in [11, 3] or to decouple data contention from transaction management as in [32, 17]. This multipartition update access in the context of a single transaction satisfies Properties P1 and P2. In any case, nothing can be said about transactions reading from different partitions but that they read from different, though valid, snapshots (each partition might have a different snapshot). Thus, these systems supporting the access to different partitions will more likely generate 1MV-schedules and they do not regularly ensure Property P3 nor P4.

10. Conclusions

This paper starts with the introduction of complete MV-schedule and its most outstanding types, SI- and GSI-schedules, which are well suited for a centralized environment. We have focused on porting SI- and GSI-schedules to a database replication scenario. First, we have stated the notion of one-copy-schedules, more precisely 1SI- and 1GSI-schedules, as a transformation function based on the replicated data schedule that obtains an SI- or GSI-schedule respectively, considering that the possible set of valid 1SI- and 1GSI-schedules is restricted to the family of ROWA protocols that use SI replicas. Moreover, we have proposed a set of properties based on the definitions of SI- and GSI-schedules that have been shown as sufficient conditions to derive

1SI- and 1GSI-schedules. Furthermore, we have discussed that the set of properties that are formally stated in this work are indeed used by most of the already existing replication protocols in the literature, regardless of the protocol features (i.e. they are valid for partial or full replication, eager or lazy protocols and primary-copy or update-everywhere protocols); however, none of such protocols carefully specified all of these properties. With our contribution, it will be straightforward to prove which kind of snapshot-related isolation level is being provided by any new ROWA database replication protocol developed for SI DBMSs.

Acknowledgments

The authors wish to thank the reviewers for their valuable comments that helped us to greatly improve the quality and readability of this paper.

References

[1] A. Adya, B. Liskov, P. O'Neil, Generalized Isolation Level Definitions, in: Intnl. Conf. on Data Eng. (ICDE), IEEE-CS, 2000, pp. 67–78.

[2] D. Agrawal, G. Alonso, A. El Abbadi, I. Stanoi, Exploiting atomic broadcast in replicated databases, in: Euro-Par, Springer, 1997, pp. 496–503.

[3] M. K. Aguilera, A. Merchant, M. Shah, A. Veitch, C. Karamanolis, Sinfonia: a new paradigm for building scalable distributed systems, ACM Trans. Comput. Syst. 27 (3) (2009) 11:1–11:49.

[4] R. Akbarinia, E. Pacitti, P. Valduriez, Data currency in replicated DHTs, in: SIGMOD Conf., ACM, 2007, pp. 211–222.

[5] J. E. Armendáriz, J. R. Juárez, J. R. González de Mendívil, H. Decker, F. D. Muñoz, *K*-bound GSI: a flexible database replication protocol, in: Symp. on Applied Comput. (SAC), ACM, 2007, pp. 556–560.

[6] J. E. Armendáriz-Iñigo, A. Mauch-Goya, J. R. González de Mendívil, F. D. Muñoz-Escoí, SIPRe: A partial database replication protocol with SI replicas, in: Symp. on Applied Comput. (SAC), ACM, 2008, pp. 556–560.

[7] H. Berenson, P. A. Bernstein, J. Gray, J. Melton, E. J. O'Neil, P. E. O'Neil, A critique of ANSI SQL isolation levels., in: SIGMOD Conf., ACM Press, 1995, pp. 1–10.

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

[9] G. Chockler, I. Keidar, R. Vitenberg, Group communication specifications: a comprehensive study., ACM Comput. Surv. 33 (4) (2001) 427–469.

[10] C. Curino, E. Jones, Y. Zhang, S. Madden, Schism: a workload-driven approach to database replication and partitioning., in: 36th Intnl. Conf. on Very Large Data Bases (VLDB), Singapore, 2010. *Accepted for publication*

[11] S. Das, D. Agrawal, A. El Abbadi, ElasTraS: an elastic transactional data store in the cloud, in USENIX Workshop on Hot Topics in Cloud Computing, San Diego, CA, USA, 2009.

[12] K. Daudjee, K. Salem, Lazy database replication with snapshot isolation., in: 32nd Intnl. Conf. on Very Large Data Bases (VLDB), Seoul, Korea, 2006, pp. 715–726.

[13] S. Elnikety, S. G. Dropsho, F. Pedone, Tashkent: uniting durability with transaction ordering for high-performance scalable database replication, in: Y. Berbers, W. Zwaenepoel (Eds.), EuroSys, ACM, 2006, pp. 117–130.

[14] S. Elnikety, F. Pedone, W. Zwaenepoel, Database replication using generalized snapshot isolation., in: Symp. on Reliable Distrib. Syst. (SRDS), Orlando, FL, USA, IEEE-CS, 2005, pp. 73–84.

[15] J. R. González de Mendívil, J. E. Armendáriz-Íñigo, J. R. Garitagoitia, F. D. Muñoz-Escoí, A formal analysis of database replication protocols with SI replicas and crash failures, J. of Supercomputing 50 (2) (2009) 121–161.

[16] J. Gray, P. Helland, P. E. O'Neil, D. Shasha, The dangers of replication and a solution., in: H. V. Jagadish, I. S. Mumick (Eds.), SIGMOD Conf., ACM Press, 1996, pp. 173–182.

[17] E. P. C. Jones, D. J. Abadi, S. Madden, Low overhead concurrency control for partitioned main memory databases, in: SIGMOD Conf., ACM Press, 2010, pp. 603–614.

[18] J. R. Juárez, J. E. Armendáriz, J. R. González de Mendívil, F. D. Muñoz, J. R. Garitagoitia, A weak voting database replication protocol providing different isolation levels, in: NOTERE'07, Marrakesh, Morocco, 2007, pp. 261–268.

[19] S. Jung, K. Choi, A concurrency control scheme for mobile transactions in broadcast disk environments, Data Knowl. Eng. 68 (10) (2009) 926–945.

[20] B. Kemme, Database replication for clusters of workstations (ETH Nr. 13864), Ph.D. thesis, Swiss Federal Institute of Technology, Zurich, Switzerland (2000).

[21] B. Kemme, G. Alonso, A new approach to developing and implementing eager database replication protocols, ACM Trans. Database Syst. 25 (3) (2000) 333–379.
[22] S. U. Khan, I. Ahmad, A pure Nash equilibrium-based game theoretical method for data replication across multiple servers, IEEE Trans. Knowl. Data Eng. 21 (4) (2009) 537–553.
[23] K. Krikellas, S. Elnikety, Z. Vagena and O. Hodson, Strongly consistent replication for a bargain., in: Intnl. Conf. on Data Eng. (ICDE), Long Beach, USA, IEEE-CS, 2010, pp. 52–63.
[24] B. W. Lampson, Atomic Transactions, in: Advanced Course: Distributed Systems, 1980, pp. 246–265.
[25] M. Lin, An optimal workload-based data allocation approach for multidisk databases, Data Knowl. Eng. 68 (5) (2009) 499–508.
[26] Y. Lin, B. Kemme, R. Jiménez-Peris, M. Patiño-Martínez, J. E. Armendáriz-Íñigo, Snapshot isolation and integrity constraints in replicated databases, ACM Trans. Database Syst. 34 (2) (2009) 11:1–11:49.
[27] Y. Lin, B. Kemme, M. Patiño-Martínez, R. Jiménez-Peris, Middleware based data replication providing snapshot isolation., in: SIGMOD Conf., ACM, Baltimore, MD, USA, 2005, pp. 419–430.
[28] G. Luo, J. F. Naughton, C. J. Ellmann, M. Watzke, Transaction reordering, Data Knowl. Eng. 69 (1) (2010) 29–49.
[29] T. Mishima, H. Nakamura, Pangea: an eager database replication middleware guaranteeing snapshot isolation without modification of database servers., in: 35th Intnl. Conf. on Very Large Data Bases (VLDB), Lyon, France, 2 (1) (2009) 1066–1077.
[30] F. D. Muñoz-Escoí, J. Pla-Civera, M. I. Ruiz-Fuertes, L. Irún-Briz, H. Decker, J. E. Armendáriz-Iñigo, J. R. González de Mendívil, Managing transaction conflicts in middleware-based database replication architectures, in: Symp. on Reliable Distrib. Syst. (SRDS), IEEE-CS, Leeds, UK, 2006, pp. 401–410.
[31] R. C. Oliveira, J. Pereira, A. Correia Jr., Revisiting 1-copy equivalence in clustered databases, in: Symp. on Applied Comput. (SAC), ACM, 2006, pp. 728–732.
[32] I. Pandis, R. Johnson, N. Hardavellas, A. Ailamaki, Data-oriented transaction execution, in: Intnl. Conf. on Very Large Data Bases (VLDB), Singapore, Singapore, 2010. *Accepted for publication.*
[33] C. Papadimitriou, The Theory of Database Concurrency Control, Computer Science Press (1986).
[34] C. Plattner, G. Alonso, M. T. Özsu, Extending DBMSs with satellite databases, VLDB J. 17 (4) (2008) 657–682.
[35] C. Plattner, A. Wapf, G. Alonso, Searching in time, in: S. Chaudhuri, V. Hristidis, N. Polyzotis (Eds.), SIGMOD Conf., ACM, 2006, pp. 754–756.
[36] T. Ragunathan, P. K. Reddy, Improving the performance of read-only transactions through asynchronous speculation, in: H. Rajaei, G. A. Wainer, M. J. Chinni (Eds.), SpringSim, SCS/ACM, 2008, pp. 467–474.
[37] D. Serrano, M. Patiño-Martínez, R. Jiménez-Peris, B. Kemme, Boosting database replication scalability through partial replication and 1-copy-snapshot-isolation, in: 13th IEEE Pacific Rim Dependable Computing Conf. (PRDC), IEEE-CS, 2007, pp. 290–297.
[38] D. Skeen, Nonblocking Commit Protocols, in: SIGMOD Conf., ACM, 1981, pp. 133–142.
[39] I. L. Traiger, J. Gray, C. A. Galtieri, B. G. Lindsay, Transactions and Consistency in Distributed Database Systems, ACM Trans. Database Syst. 7 (3) (1982) 323–342.
[40] M. Wiesmann, A. Schiper, Comparison of database replication techniques based on total order broadcast, IEEE Trans. Knowl. Data Eng. 17 (4) (2005) 551–566.
[41] S. Wu, B. Kemme, Postgres-R(SI): Combining replica control with concurrency control based on snapshot isolation., in: Intnl. Conf. on Data Eng.(ICDE), Tokyo, Japan, IEEE-CS, 2005, pp. 422–433.