# Non-blocking ROWA Protocols Implement GSI Using SI Replicas

J.R. González de Mendívil, J.E. Armendáriz, F.D. Muñoz, L. Irún, J.R. Garitagoitia, J.R. Juárez

Instituto Tecnológico de Informática
Univ. Politécnica de Valencia
Camino de Vera, s/n
46022 Valencia (Spain)

{mendivil,joserra,jr.juarez}@unavarra.es, {armendariz,fmunyoz,lirun}@iti.upv.es

Technical Report TR-ITI-ITE-07/10

# Non-blocking ROWA Protocols Implement GSI Using SI Replicas

J.R. González de Mendívil, J.E. Armendáriz, F.D. Muñoz, L. Irún, J.R. Garitagoitia, J.R. Juárez

Instituto Tecnológico de Informática
Univ. Politécnica de Valencia
Camino de Vera, s/n
46022 Valencia (Spain)

Technical Report TR-ITI-ITE-07/10

e-mail: {mendivil,joserra,jr.juarez}@unavarra.es,
{armendariz,fmunyoz,lirun}@iti.upv.es

June 28, 2007

**Abstract**

The concept of Generalized Snapshot Isolation (GSI) has been recently proposed as a suitable extension of conventional Snapshot Isolation (SI) for replicated databases. In GSI, transactions may use older snapshots instead of the latest snapshot required in SI, being able to provide better performance without significantly increasing the abortion rate when write/write conflicts among transactions are low. Its authors also state that GSI is needed because there is no non-blocking implementation of SI in asynchronous systems, even when databases never fail, but they do not prove such statement. We justify such property for ROWA (Read One, Write All) protocols in this paper by using the equivalence between SI-schedules. Additionally, we show and prove that if a replication protocol using SI replicas provides global atomicity and commits update transactions in the same order at all sites then it provides GSI. This sufficient condition prevents the usage of some mechanisms that exclusively order write/write conflicting transactions because they do not guarantee GSI.

## 1 Introduction

Snapshot Isolation (SI) is a transaction isolation level introduced in [4] and implemented (using multiversion concurrency control) in several commercial database systems as Oracle, PostgreSQL, Microsoft SQL Server or InterBase. SI provides a weaker form of consistency than serializability [6]. Indeed, SI allows some read-only anomalies analyzed in [16]. Several researchers [15, 14] have recently demonstrated that, under certain conditions on the workload, transactions executing on a database with SI produce serializable histories. Nevertheless, in practice most applications run serially under SI, including the most widely-used database benchmarks TPC-B, TPC-C, and TPC-W [34]. These characteristics turn SI into an attractive isolation level for a database programmer because it provides sufficient data consistency for non-critical applications while it maintains a good performance, since read-only activity introduces lower overheads in the application. Read-only transactions executed under SI level are neither delayed, blocked nor aborted, since they do not use read-locks, and they never cause update transactions to block or abort. This behavior is important for workloads dominated by read-only transactions, such as those resulting from dynamic content Web servers.

Many enterprise applications demand high availability since they have to provide continuous service to their users. For achieving such availability, the common solution consists in deploying multiple replicas of such application. This leads also to the replication of the information being used; i.e., to managing replicated databases. The concept of Generalized Snapshot Isolation (GSI) has been recently proposed [13] in order to provide a suitable extension of conventional SI for replicated databases based on multiversion concurrency control. In GSI, transactions may use older snapshots instead of the latest snapshot required in SI. Authors of [13] outline an impossibility result which justifies the use of GSI in database replication: "*there is no non-blocking implementation of SI in an asynchronous system, even if databases never fail*". In a *non-blocking* replication protocol, transactions can start at any time without restriction or delay.

Concurrently with that paper, [26] proposes a definition of One Copy Snapshot Isolation (1CSI) for ROWA (Read One Write All) protocols. From the previous impossibility result, it is not possible to obtain the given isolation level with the replication protocols stated in [26] by using SI replicas. Thus, their protocols should be classified as GSI, instead of a strict SI for a replicated system. Note, however, that this is not a critique on the proposal given in [26], since the initial definition of the SI level [4] was ample enough to accept both interpretations of what should be 1CSI (Elniketi's conventional SI, and Lin's 1CSI).

In this paper, we prove that a non-blocking ROWA protocol can not implement SI (as stated above). The proof, given in Section 6, is very simple and is based on a condition that restricts the transaction start time in order to provide SI. The proof forces the resulting protocol to be a blocking one. As SI is not implemented by non-blocking ROWA protocols, we also study the basic requirements that such kind of protocols must verify in order to provide GSI. So, the second contribution of our paper consists in formalizing the requirements for achieving GSI over SI replicas using non-blocking protocols. Thus, the criteria for implementing GSI are: (i) Each submitted transaction to the system either commits or aborts at all sites (*atomicity*); (ii) All update transactions are committed in the same total order at every site (*total order of committed transactions*). Total order ensures that all replicas see the same sequence of transactions, being thus able to provide the same snapshots to transactions, independently of their starting replica. Without such order, those transactions without write/write conflicts might be applied in different orders in different replicas. So, transactions would be able to read different versions in different replicas. Atomicity guarantees that all replicas take the same actions regarding each transaction, so their states should be consistent, once each transaction has been terminated.

In the replication protocols discussed above, ROWA certification-based replication protocols are provided. These protocols are based on the use of atomic broadcast [18] to deliver in total order the update operations of transactions for executing the certification at every database replica. The distributed protocol in [13] applies the update operations of transactions in the same total order in all replicas while the protocol in [26] allows more concurrency in the execution of the update operations of non-conflicting transactions at each replica, although it needs to *block* the execution of the first operation of any starting transaction until the end of the current execution of those concurrent transactions. So, these protocols comply with the requirements of GSI.

The contributions of this paper are twofold. First, a detailed formalization of the GSI definition and its requirements is presented. Second, and as a result of the first one, we provide a complete characterization of the GSI protocols that allows us to state that several kinds of optimizations are not possible if GSI must be ensured. For instance, some replication protocols providing One Copy Serializability (1CS) assume database replicas that follow a strict serializable isolation level. In these protocols, non-conflicting operations can be executed concurrently and different replicas may even commit transactions in different orders as long as they do not conflict. This optimization is important since processing messages serially as supposed for replication protocols deployed over a group communication system [8] would result in significantly lower throughput rates. While this optimization may be partially maintained for obtaining GSI as in the protocol presented in [26], it is not possible for replication protocols that use simpler kinds of broadcast (those that do not enforce a total order) such as some implementations of O2PL [1].

The rest of the work is organized as follows. Section 2 introduces the concept of multiversion histories based on [6]. Sections 3 and 4 give the concepts of Snapshot Isolation and Generalized Snapshot Isolation. In Section 5, the structure of ROWA protocols is introduced. Conditions for One Copy Snapshot Isolation and One Copy Generalized Snapshot Isolation are introduced in Sections 6 and 7 respectively. Finally, a discussion about recent database replication protocol proposals is presented in Section 8 and the paper conclusions in Section 9.

## 2 Multiversion Histories

In the following, we define the concept of multiversion history for committed transactions using the theory provided in [6]. The properties studied in our paper only require to deal with committed transactions. To this end, we first define the basic building blocks for our formalizations, and then the different definitions and properties will be shown.

A database ($DB$) is a collection of data items, which may be concurrently accessed by transactions. A history represents an *overall partial ordering* of the different operations concurrently executed within the *context* of their corresponding transactions. Thus, a multiversion history generalizes a history where the database items are versioned.

To formalize this definition, each transaction submitted to the system is denoted by $T_i$. A transaction is a sequence of read and write operations on database items ended by a commit or abort operation. Each $T_i$'s write operation on item $X$ is denoted $W_i(X_i)$. A read operation on item $X$ is denoted $R_i(X_j)$ stating that $T_i$ reads the version of $X$ installed by $T_j$. Finally, $C_i$ and $A_i$ denote the $T_i$'s commit and abort operation respectively. We assume that a transaction does not read an item $X$ after it has written it, and each item is read and written at most once. Avoiding redundant operations simplifies the presentation. The results for this kind of transactions are seamlessly extensible to more general models. In any case, redundant operations can be removed using local variables in the program of the transaction [29].

Each version of a data item $X$ contained in the database is denoted by $X_i$, where the subscript stands for the transaction identifier that installed that version in the $DB$. The *readset* and *writeset* (denoted by $RS_i$ and $WS_i$ respectively) express the sets of items read (written) by a transaction $T_i$. Thus, $T_i$ is a *read-only* transaction if $WS_i = \emptyset$ and it is an *update* one, otherwise.

Let $T = \{T_1, ..., T_n\}$ be a set of *committed* transactions, where the operations of $T_i$ are ordered by $\prec_{T_i}$. The last operation of a transaction is the commit operation. To process operations from a transaction $T_i \in T$, a multiversion scheduler must translate $T_i$'s operations on data items into operations on specific versions of those data items. That is, there is a function $h$ that maps each $W_i(X)$ into $W_i(X_i)$, and each $R_i(X)$ into $R_i(X_j)$ for some $T_j \in T$.

**Definition 1.** *A Complete Committed Multiversion (CCMV) history $H$ over $T$ is a partial order with order relation $\prec$ such that:*

1. $H = h(\bigcup_{T_i \in T} T_i)$ *for some translation function $h$.*
2. $\prec \supseteq \bigcup_{T_i \in T} \prec_{T_i}$.
3. *If $R_i(X_j) \in H$, $i \neq j$, then $W_j(X_j) \in H$ and $C_j \prec R_i(X_j)$.*

In the previous Definition 1 condition (1) indicates that each operation submitted by a transaction is mapped into an appropriate multiversion operation. Condition (2) states that the CCMV history preserves all orderings stipulated by transactions. 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.

Definition 1 is more specific than the one stated in [6], since the former only includes committed transactions and explicitly indicates that a new version may not be read until the transaction that installed the new version has committed.

In general, two histories over the same set of transactions are *view equivalent* [6] if they contain the same operations, have the same *reads-from* relations, and produce the same final writes. The notion of equivalence of CCMV histories reduces to a simple condition if the following *reads-from* relation is used: $T_i$ *reads* $X$ *from* $T_j$ in a CCMV history $H$, if $R_i(X_j) \in H$.

Let $H$ and $H'$ be two CCMV histories over the same set of committed transactions $T$. These histories are (view-) equivalent, denoted as $H \equiv H'$, if and only if they have the same operations. Both of them have the same writes, moreover all write operations are final as all versions they produce are different. As $R_i(X_j) \in H$ and $R_i(X_j) \in H'$ then they have the same *reads-from* relations.

In the rest of the paper, we use the following conventions:

$(i)$ $T = \{T_1, ..., T_n\}$ is the set of committed transactions for every defined history;

$(ii)$ any history $H$ is a CCMV history over $T$; and

$(iii)$ for each $X \in DB$: $Ver(X,H) = \{X_j : W_j(X_j) \in H\} \bigcup \{X_0\}$ is the set of installed versions of the data item $X$ in $H$, being $X_0$ its initial version.

Note that these conventions will be also applicable when a superscript is used to denote the site of the database where the history is generated.

## 3 Snapshot Isolation

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 $T_i$. From our point of view, histories generated by a given concurrency control providing SI may be interpreted as multiversion histories with time restrictions.

Let $H$ be a history and $t\colon H \to \mathbb{R}^+$ a mapping such that it assigns to each operation $op \in H$ its real time occurrence $t(op) \in \mathbb{R}^+$. The mapping also verifies: $op \prec op'$ in $H \Rightarrow t(op) < t(op')$. The mapping $t()$ totally orders all operations of $H$, and the total order $<$ is compatible with the partial order $\prec$. For simplicity, we assume different times for different operations; that is, $t(op) = t(op') \Leftrightarrow op = op'$. The pair $(H,t)$ defines a schedule of $H$, and it is denoted $H_t$. It is clear that each compatible mapping with the partial order of the history determines the obtained schedule. In the following, we additionally use the next convention:

$(iv)$ A schedule $H_t$ is a schedule of a history $H$.

We define the "commit time" ($c_i$) and "begin time" ($b_i$) for each transaction $T_i \in T$ in a schedule $H_t$ as $c_i = t(C_i)$ and $b_i = t($*first operation of* $T_i)$, holding $b_i < c_i$ by definition of $t()$ and $\prec_{T_i}$.

In the following, we formalize the concept of snapshot of the database. Intuitively it comprises the latest version of each data item. A sample of a SI-schedule might be:
$b_1 r_1(x_0)\, w_1(x_1)\, c_1\, b_2\, r_2(z_0)\, b_3\, r_3(y_0)\, w_3(x_3)\, c_3\, r_2(x_1)\, w_2(y_2)\, c_2$.

As this example shows, each transaction is able to include in its snapshot (and read from it) the latest committed version of each existing item at the time such transaction was started. Thus $T_2$ has read version $1$ of item $X$ since $T_1$ has generated such version and it has already committed when $T_2$ started. But it only reads version $0$ of item $Z$ since no update of such item is seen by $T_2$. This is true despite transactions $T_2$ and $T_3$ are concurrent and $T_3$ updates $X$ before $T_2$ reads such item, because the snapshot taken for $T_2$ is previous to the commit of $T_3$. This provides the basis for defining what a snapshot is.

**Definition 2.** *The snapshot of the database $DB$ at time $\tau \in \mathbb{R}^+$ for a schedule $H_t$, is $Snapshot(DB, H_t, \tau) = \bigcup_{X \in DB} latestVer(X, H_t, \tau)$, where the latest version of each item $X \in DB$ at time $\tau$ is the set $latestVer(X, H_t, \tau) = \{X_p \in Ver(X,H) \colon (\nexists X_k \in Ver(X,H) \colon c_p < c_k \leq \tau)\}$*

From the previous definition, it is easy to show that a snapshot is modified each time an update transaction commits. If $\tau = c_m$ and $X_m \in Ver(X,H)$, then $latestVer(X,\, H_t,\, c_m) = \{X_m\}$. In order to formalize the concept of SI-schedule, we utilize a slight variation of the predicate *impacts* for update transactions presented in [13]. Two transactions $T_j$, $T_i \in T$ impact at time $\tau \in \mathbb{R}^+$ in a schedule $H_t$, denoted $T_j$ *impacts* $T_i$ *at* $\tau$, if the following predicate holds: $WS_j \bigcap WS_i \neq \emptyset \wedge \tau < c_j < c_i$.

**Definition 3.** *A schedule $H_t$ is a SI-schedule if and only if for each $T_i \in T$:*

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

Condition (1) states that all the versions read by a transaction $T_i$ are obtained from $Snapshot(DB, H_t, b_i)$; that is, versions are obtained from the snapshot of the database $DB$ at the time the transaction starts its first operation. Condition (2) states that any pair of transactions $T_j$ and $T_i$, writing over some common data items, can not overlap their time intervals $[b_i, c_i]$ and $[b_j, c_j]$. In other words, they have to be executed in a serial way.

Other definitions of SI have been provided in the literature. It was firstly introduced as an isolation level generated by multiversion concurrency control in [4]. It states that a transaction reads data that were already committed when the transaction started. Read operations never blocked as long as the snapshot can be maintained. On the other hand, transaction's writes are reflected in its snapshot. Concurrent updates are invisible to the transaction. In order to prevent lost updates [4], it applies the *First-Committer-Wins* rule. A transaction will successfully commit only if no other transaction has already committed writes to items that the transaction intends to write. A similar definition to [4] has also been used in [23, 36, 26, 15]. In [13], where we have taken part of the notations of our work, the GSI concept is introduced (see Section 4) and by means of the notion of impacting transactions with the read and commit rules, it can be inferred SI.

In the previous Section, the concept of view equivalence between two histories has been introduced. In the following, we explore a notion of equivalence between SI-schedules.

**Definition 4.** *Let $H_t$ and $H'_{t'}$ be two SI-schedules. $H_t$ is SI-equivalent to $H'_{t'}$, denoted $H_t \equiv_{SI} H'_{t'}$, if and only if for any $T_j, T_i \in T$ the following conditions hold:*

1. *If $WS_i \bigcap WS_j \neq \emptyset$: $c_i < c_j$ in $H_t \Leftrightarrow c'_i < c'_j$ in $H'_{t'}$.*
2. *If $WS_i \bigcap RS_j \neq \emptyset$: $c_i < b_j$ in $H_t \Leftrightarrow c'_i < b'_j$ in $H'_{t'}$.*

Condition (1) indicates that transactions with write/write conflicts must be committed in the same order in both schedules; and condition (2) states that the order of commit time and begin time between two transactions having a reads-from relation has to be the same in both SI-schedules. Definition 4 of SI-equivalence is equivalent to the one provided in [26]. It is based on the fact that both schedules are SI-schedules and does not use the concrete operations in the histories; it only uses the definitions of items to be read or written by the transactions. This motivates the next property, stating that if two SI-schedules are SI-equivalent, all transactions contained in them have read and written the same item versions in both schedules. As a result, both schedules are also equivalent (in a general sense, similarly to the one defined at the end of Section 2 for history equivalence, i.e., the operations being used in each schedule produce the same results in each one of them).

**Property 1.** *Let $H_t$ and $H'_{t'}$ be two SI-schedules. If $H_t \equiv_{SI} H'_{t'}$ then $H \equiv H'$.*

*Proof.* Let $R_j(X_i) \in H$, $H_t$ is a SI-schedule, thus $\{X_i\} = latestVer(X, H_t, b_j)$. By Definition 2, $c_i < b_j$ and $\nexists\, X_k \in Ver(X, H)$: $c_i < c_k < b_j$. Assume $c'_i < c'_k < b'_j$ in $H'_{t'}$ and $X_k \in Ver(X, H')$. In that case, $X_i$ is not the latest version of $X$ for the transaction $T_j$ in $H'_{t'}$; and, as the $RS_j$ is the same for the transaction $T_j$ in $H$ and $H'$, it reads a different version of $X$ in $H$ and $H'$.
By Condition (1) in Definition 4, $c'_i < c'_k \Rightarrow c_i < c_k$ in $H_t$, and by Condition (2) in Definition 4, $c'_k < b'_j \Rightarrow c_k < b_j$ in $H_t$. As the $WS_k$ is the same for the transaction $T_k$ in $H_t$ and $H'_{t'}$, $X_k \in Ver(X, H)$. Therefore, $\exists\, X_k \in Ver(X, H)$: $c_i < c_k < b_j$. By contradiction, $\{X_i\} \neq latestVer(X, H_t, b_j)$. In conclusion, if $R_j(X_i) \in H$, then $R_j(X_i) \in H'$ holds.
$T_j$ reads the same versions in $H$ and $H'$, thus it produces the same writes in both histories. Let $W_j(X_j) \in H$, as the $WS_j$ is the same for the transaction $T_j$ in $H'$ and $H$, then $W_j(X_j) \in H'$.
$H$ and $H'$ have the same operations and, as we have previously seen, two histories are equivalent if they have the same set of operations. Therefore, we conclude that $H \equiv H'$. □

SI-equivalence allows SI-schedules to differ in the time occurrence of operations, but it has to maintain a relative order among certain key operations of transactions: commit and first operations. However, conditions in Definition 4 can not be used to show if an arbitrary schedule $H'_{t'}$ is equivalent to a given SI-schedule $H_t$. This is because in a multiversion history a transaction may read any available version of a data item, and conditions in Definition 4 do not restrict that fact.

It is easy to show that if a concurrency control algorithm returns to a transaction the current snapshot at the time of its first operation and the algorithm updates the DB at commit time if no transaction impacts with it (or in the contrary case, aborts it), then the algorithm produces SI-schedules.

## 4 Generalized Snapshot Isolation

The concept of Generalized Snapshot Isolation (or GSI, for short) was firstly applied to database replication in [13]. A hypothetical concurrency control algorithm could have stored some past snapshots. A transaction may receive a snapshot that happened in the system before the time of its first operation (instead of its current snapshot as in a SI concurrency control algorithm). The algorithm may commit the transaction if no other transaction impacts with it from that past snapshot. Thus, a transaction can observe an older snapshot of the DB but the write operations of the transaction are still valid update operations for the DB at commit time. These previous ideas define the concept of GSI.

**Definition 5.** *A schedule $H_t$ is a GSI-schedule if and only if for each $T_i \in T$ 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 $T_j \in T$: $\neg$($T_j$ impacts $T_i$ at $s_i$).*

Condition (1) states that every item read by a transaction belongs to the same (possible past) snapshot. Condition (2) also establishes that the time intervals $[s_i, c_i]$ and $[s_j, c_j]$ do not overlap for any pair of write/write conflicting transactions $T_i$ and $T_j$. If for all $T_i \in T$, conditions (1) and (2) hold for $s_i = b_i$ then $H_t$ is a SI-schedule. Thus, Definition 5 includes as a particular case the Definition 3. Another observation of the definition concludes that if there exists a transaction $T_i \in T$ such that conditions (1) and (2) are only verified for a value $s_i < b_i$ then there is an item $X \in RS_i$ for which $latestVer(X, H_t, s_i) \neq latestVer(X, H_t, b_i)$. That is, the transaction $T_i$ has not seen the latest version of $X$ at the begin time $b_i$. There was a transaction $T_k$ with $W_k(X_k) \in H$ such that $s_i < c_k < b_i$.

The following is an example of a GSI-schedule:
$b_1\, r_1(x_0)\, w_1(x_1)\, c_1\, b_2\, r_2(x_0)\, r_2(z_0)\, b_3\, r_3(y_0)\;\; w_3(x_3)\, c_3\, w_2(y_2)\;\; c_2$.

In this schedule, transaction $T_2$ reads $x_0$ after the commit of $T_1$ appears. This would not be correct for a SI-schedule (since the read version of $X$ is not the latest one), but it is perfectly valid for a GSI-schedule, taken the time point of the snapshot provided to $T_2$ (i.e. $s_2$) previous to the commit of $T_1$, as it is shown:
$b_1\, r_1(x_0)\, \mathbf{s_2}\, w_1(x_1)\, c_1\, b_2\, r_2(x_0)\, r_2(z_0)\;\; b_3\, r_3(y_0)\, w_3(x_3) c_3\;\; w_2(y_2)\, c_2$.

The intuition under this schedule in a distributed system is that the message containing the modifications of $T_1$ (the write operation on $X$) would have not yet arrived to the site at the time transaction $T_2$ began. This may be the reason for $T_2$ to see this past version of item $X$. The fact that GSI captures these delays into schedules makes attractive its usage on distributed environments.

The value $s_i$ in Definition 5 plays the same role as $b_i$ in Definition 3. Thus, it is possible to think that if the operations in the GSI-schedule obtained from the history $H$ had been 'on time' then the schedule would have been a SI-schedule.

Let us use the previous example to show how a GSI-schedule can be transformed into a SI-schedule. Thus, to turn that GSI-schedule into a SI-schedule, it is just needed to move the beginning of $T_2$ back to $s_2$, and consequently, the resulting schedule will be a SI-schedule:
$b_1\, r_1(x_0)\, \mathbf{b_2}\, w_1(x_1)\, c_1\, r_2(x_0)\, r_2(z_0)\, b_3\, r_3(y_0)\, w_3(x_3)\, c_3\;\; w_2(y_2)\, c_2$.

However, this schedule does not fit the definition of $b_i$, which was described as the time of the first operation a transaction performs. Thus, such first operation of transaction $T_2$ must be also moved in the SI-schedule, resulting in the following:
$b_1\, r_1(x_0)\, \mathbf{b_2}\, \mathbf{r_2(x_0)}\, w_1(x_1)\, c_1\, r_2(z_0)\, b_3\, r_3(y_0)\, w_3(x_3)\, c_3\;\; w_2(y_2)\, c_2$.

The following property describes the previous transformation in a formal way:

**Property 2.** *Let $H_t$ be a GSI-schedule. There is a mapping $t'\colon H \rightarrow \mathbb{R}^+$ such that $H_{t'}$ is a SI-schedule.*

*Proof.* Let $T_i$ be a transaction such that $RS_i \neq \emptyset$ and $s_i < b_i$ in $H_t$. In order to make the proof simple we consider transactions in which all read operations are done before any write operation. Let $R_i(X_{j_1})...R_i(P_{j_p})...R_i(Z_{j_z})$ be the sequence of read operations in the same order imposed by $\prec_{T_i}$. Let $T_{j_s}$ be the first transaction with $W_{j_s}(S_{j_s}) \in H$ such that $s_i < c_{j_s} < b_i$ and $S \in RS_i$. We move the time occurrence of the read operations of the transaction $T_i$ in the same order $\prec_{T_i}$ as follows: $t'(R_i(P_{j_p})) = s_i + \epsilon_{j_p}$ where $\epsilon_{j_1} = 0$ for the first read, and $\sum_{j_p} \epsilon_{j_p} < c_{j_s} - s_i$. For the rest of operations $op \in H$, $t'(op) = t(op)$. It is easy to show that this new mapping is compatible with $\prec$ of $H$.

For this transaction $T_i$, $\neg$($T_j$ *impacts* $T_i$ *at* $b'_i$) in $H_{t'}$ because $\neg$($T_j$ *impacts* $T_i$ *at* $s_i$) for every $T_j \in T$ and $b'_i = s_i$. Only the read operations of $T_i$ have been moved; the commit operations, that may modify the predicate *impacts*, have not changed their time occurrences. In $H_{t'}$, for each $R_i(P_{j_p})$ of $T_i$, $\{P_{j_p}\} = latestVer(P, H_{t'}, b'_i)$. Again, $b'_i = s_i$, and $\{P_{j_p}\} = latestVer(P, H_t, s_i)$.

The previous process is done for any transaction $T_i$ such that in $H_{t'}$, $s'_i < b'_i$. This process is finite and the resulting schedule is a SI-schedule. □

The next definition is the generalization of SI-equivalence (Definition 4) between GSI-schedules.

**Definition 6.** *Let $H_t$ and $H'_{t'}$ be two GSI-schedules. $H_t$ is GSI-equivalent to $H'_{t'}$, denoted $H_t \equiv_{GSI} H'_{t'}$, if and only if for any $T_j, T_i \in T$ the following conditions hold:*

1. *If $WS_i \bigcap WS_j \neq \emptyset$: $c_i < c_j$ in $H_t \Leftrightarrow c'_i < c'_j$ in $H'_{t'}$.*
2. *If $WS_i \bigcap RS_j \neq \emptyset$: $c_i < s_j$ in $H_t \Leftrightarrow c'_i < s'_j$ in $H'_{t'}$.*

If two GSI-schedules are GSI-equivalent then their histories are also view equivalent.

**Property 3.** *Let $H_t$ and $H'_{t'}$ be two GSI-schedules. If $H_t \equiv_{GSI} H'_{t'}$ then $H \equiv H'$.*

*Proof.* This proof is the same as the proof of Property 1 if we substitute $b_j$ by $s_j$, $b'_j$ by $s'_j$ and Conditions (1) and (2) of Definition 4 by Conditions (1) and (2) of Definition 6 respectively. □

We remark that the definition of GSI-equivalence allows a GSI-schedule to be equivalent to a SI-schedule, but the reverse case is, in general, not true. In particular the SI-schedule $H_{t'}$ obtained in Property 2 for $H_t$ satisfies $H_t \equiv_{GSI} H_{t'}$.

Finally, it is also important to note that GSI-schedules allow read-only transactions to obtain versions of their accessed items arbitrarily old. This could be an inconvenience for many applications, since the freshness of the accessed data is unknown.

## 5 The ROWA Strategy

The GSI concept is particularly interesting in replicated databases, since many replication protocols execute each transaction initially in a delegate replica, propagating later its updates to the rest of replicas [26, 13, 3]. This means that transaction writesets cannot be immediately applied in all replicas at a time and, due to this, the snapshot being used in a transaction might be "previous" to the one that (regarding physical time in a hypothetical centralized system) would have been assigned to it. In this Section we consider a distributed system that consists of $m$ sites, being $I_m = \{1..m\}$ the set of site identifiers. Sites communicate among them by reliable message passing. We make no assumptions about the time it takes for sites to execute and for messages to be transmitted. We assume a system free of failures. Each site $k$ runs an instance of the database management system and maintains a copy of the database $DB$. We will assume that each database copy, denoted $DB^k$ with $k \in I_m$, provides the Snapshot Isolation level.

We use the transaction model of Section 2. Let $T = \{T_i : i \in I_n\}$ be the set of transactions submitted to the system; where $I_n = \{1..n\}$ is the set of transaction identifiers.

The ROWA [17] strategy is quite general and replication protocols implementing this strategy will vary in their concrete implementation [1, 7, 9, 10, 19, 20, 24, 25, 26, 28, 30, 31, 32, 36]. The ROWA approach may range according to the next two parameters: *when* updates take place, either before committing the transaction (*eager*) or after (*lazy*); and, *where* update transactions can be initiated, either each database has a primary replica where all updates are initially applied, propagating them to the secondary replicas (*primary copy*) or each replica may accept updates (*update anywhere*).

Note also that we use a ROWA strategy since we have assumed a failure-free system. Otherwise, a ROWAA approach is needed, i.e., writes will only be applied on the available replicas, but all our discussion is orthogonal to failures and can be seamlessly extended to a system where failures might arise.

An archetypal example of a replication protocol following the ROWA strategy is as follows: all operations are firstly executed at a delegate replica; afterwards, the writeset and the state of each data item contained in the writeset are collected and multicast in total order to the rest of replicas for completing the transaction. Each transaction is committed or aborted by all replicas once its writeset has been delivered and locally *certified* (i.e., checked for conflicts with the rest of concurrent transactions) in each replica.

Since all replicas see the same sequence of writesets, all of them decide the same action (either commit or abort) on each transaction. This kind of approach is named *certification-based replication* in [35].

Atomic broadcast [18] (or total order broadcast) based replication protocols [24, 25, 30, 36, 26] ensure that all replicas receive writesets in the same order. When a writeset is delivered to a replica, it is firstly checked against local conflicting transactions. As a result of this, the writeset may be applied (aborting all local conflicting transactions) or not. In other works, either priorities are used to avoid the latency introduced by the atomic broadcast [1] or by epidemic propagation using vector clocks [19]. However, both approaches [1, 19] need a Two Phase Commit protocol or a quorum [19] for the commitment of a transaction. In all cases, a submitted transaction may become blocked as a consequence of the DBMS activity. Hence, a "*non-blocking*" ROWA strategy is defined as the one where the first operation of a transaction in its delegate replica never becomes blocked by the replication protocol during its execution at such replica. Finally, if we assume that every transaction is going to be committed at every site, we consider that is committed as soon as it has been firstly committed at any replica.

The ROWA strategy defines for each transaction $T_i \in T$, the set of transactions $\{T_i^k \colon k \in I_m\}$ in which there is only one, denoted $T_i^{site(i)}$, verifying $RS_i^{site(i)} = RS_i$ and $WS_i^{site(i)} = WS_i$; for the rest of the transactions, $T_i^k$, $k \neq site(i)$, $RS_i^k = \emptyset$ and $WS_i^k = WS_i$. $T_i^{site(i)}$ determines the local transaction of $T_i$, i.e., the transaction executed at its delegate replica or site, whilst $T_i^k, k \neq site(i)$, is a remote transaction of $T_i$, i.e., the updates of the transaction executed at a remote site. An update transaction reads at one site and writes at every site, while a read-only transaction only exists at its local site. In the rest of the paper, we consider the general case of update transactions with non-empty sets.

Let $T^k = \{T_i^k \colon i \in I_n\}$ be the set of transactions submitted at each site $k \in I_m$ for the set $T$. Some of these transactions are local at $k$ while others are remote ones. Assumption 1 implies that each transaction submitted to the system either commits at all replicas or in none of them. Thus, the updates applied in a delegate replica by a given transaction are also applied in the rest of replicas. Obviously, we consider a fully-replicated system. Since only committed transactions are relevant, the histories being generated at each site should be histories over $T^k$, as defined above.

**Assumption 1** (Atomicity). *$H^k$ is a CCMV history over $T^k$ for all sites $k \in I_m$.*

In the considered distributed system there is not a common clock or a similar synchronization mechanism. However, we can use a real time mapping $t \colon \bigcup_{k \in I_m}(H^k) \to \mathbb{R}^+$ that totally orders all operations of the system. This mapping is compatible with each partial order $\prec^k$ defined for $H^k$ for each site $k \in I_m$. In the following, we consider that each $DB^k$ provides SI-schedules under the previous time mapping.

**Assumption 2** (SI Replicas). *$H_t^k$ is a SI-schedule of the history $H^k$ for all sites $k \in I_m$.*

In order to study the level of consistency implemented by a non-blocking ROWA protocol is necessary to define the one copy schedule (1C-schedule) obtained from the schedules at each site. In the next definitions, properties and theorems we use the following notation: for each transaction $T_i$, $i \in I_n$, $C_i^{min(i)}$ denotes the commit operation of the transaction $T_i$ at site $min(i) \in I_m$ such that $c_i^{min(i)} = \min_{k \in I_m}\{c_i^k\}$ under the considered mapping $t()$.

**Definition 7.** *Let $T = \{T_i \colon i \in I_n\}$ be the set of submitted transactions to a replicated database system with a non-blocking ROWA strategy that verifies Assumption 1 and Assumption 2. Let $S = \bigcup_{k \in I_m}(H^k)$ be the set formed by the union of the histories $H^k$ over $T^k = \{T_i^k \colon i \in I_n\}$. And let $t \colon S \to \mathbb{R}^+$ be the mapping that totally orders the operations in $S$.*

*The 1C-schedule, $H_{t'} = (H, t' \colon H \to \mathbb{R}^+)$, is built from $S$ and $t()$ as follows:*

*For each $i \in I_n$ and $k \in I_m$*

1. *Remove from $S$ operations such that:*
   $W_i(X_i)^k$ *, with $k \neq site(i)$, or*
   $C_i^k$ *, with $k \neq min(i)$*
2. *$H$ is obtained with the rest of operations in $S$ after step 1, applying the renaming:*
   $W_i(X_i) \quad = W_i(X_i)^{site(i)}$
   $R_i(X_j) \quad = R_i(X_j)^{site(i)}$*, and*
   $C_i \quad = C_i^{min(i)}$
3. *Finally, $t'()$ is obtained from $t()$ as follows:*
   $t'(W_i(X_i)) \quad = t(W_i(X_i)^{site(i)})$
   $t'(R_i(X_j)) \quad = t(R_i(X_j)^{site(i)})$*, and*
   $t'(C_i) \quad = t(C_i^{min(i)})$

As $t'()$ receives its values from $t()$, we write, $H_t$ instead of $H_{t'}$. In the 1C-schedule $H_t$, for each transaction $T_i$, is trivially verified $b_i < c_i$ because the ROWA strategy guarantees that for all $k \neq site(i)$, $b_i^{site(i)} < b_i^k$. The 1C history $H$, that is formed by the operations over the logical $DB$, is also a history over $T$. We prove this fact informally. By the renaming (2) in Definition 7, each transaction $T_i$, has its operations over the data items in $RS_i$ and $WS_i$, and $\prec_{T_i}$ is trivially maintained in a partial order $\prec$ for $H$, because $H_t$ contains the local operations of $T_i^{site(i)}$. $H$ is also formed by committed transactions, under Assumption 1; for each $T_i$, $C_i \in H$. Finally, if $R_i(X_j) \in H$, then $R_i(X_j)^{site(i)} \in H^{site(i)}$. As $H^{site(i)}$ is a history over $T^{site(i)}$ then $C_j^{site(i)} \prec R_i(X_j)^{site(i)}$. By defining $C_j^{min(j)} \prec C_j^{site(i)}$ in $S$ then $C_j^{min(j)} \prec R_i(X_j)^{site(i)}$ and so $C_j \prec R_i(X_j)$. Thus $H$ can be defined as a history over $T$.

Condition (2) on Definition 7 ensures that a transaction is committed as soon as it has been committed at the first replica. Finally, no restriction about the beginning of a transaction is imposed in this definition. Hence, this definition is valid for the most general case of non-blocking protocols.

Although Assumptions 1 and 2 are included in Definition 7, they do not guarantee that the obtained 1C-schedule is a SI-schedule. This is best illustrated in the following example, where it is also shown how the 1C-schedule may be built from each site SI-schedules. In this example two sites and the next set of transactions are considered:

$$T_1 = \{R_1(Y), W_1(X)\}, \quad T_2 = \{R_2(Z), W_2(X)\},$$
$$T_3 = \{R_3(X), W_3(Z)\}, \quad T_4 = \{R_4(X), R_4(Z), W_4(Y)\}$$

Figure 1 illustrates the mapping described in Definition 7 for building a 1C-Schedule from the SI-schedules seen in the different nodes $I_m$. $T_2$ and $T_3$ are locally executed at site 1 ($RS_2 \neq \emptyset$ and $RS_3 \neq \emptyset$) whilst $T_1$ and $T_4$ are executed at site 2 respectively. The writesets are afterwards applied at the remote sites.

Schedules obtained at both sites are SI-schedules, i.e. transactions read the latest version of the committed data at each site. The 1C-schedule is obtained from Definition 7. For example, the commit of $T_1$ occurs for the 1C-schedule in the minimum of the interval between $C_1^1$ and $C_1^2$ and so on for the remaining transactions.

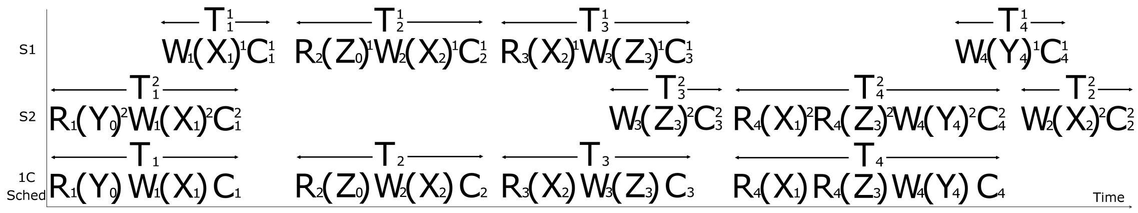

Figure 1: Execution not providing SI nor GSI.

In the 1C-schedule 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 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 1C-schedule is neither SI nor GSI.

## 6 One Copy Snapshot Isolation Schedules

The 1C-schedule $H_t$ obtained in Definition 7 will be a SI-schedule if it verifies the conditions given in Definition 3. The question is what conditions local SI-schedules, $H_t^k$, have to verify in order to guarantee that $H_t$ is a SI-schedule. Definition 4 of SI-equivalence provides a starting point because if $H_t$ is a SI-schedule, it would be SI-equivalent to each SI-schedule $H_t^k$. Taking into account the first condition of SI-equivalence in Definition 4, we consider the kind of ROWA protocols that guarantee the same total order of the commit operations for the transactions with write/write conflicts at every site. This is stated in Assumption 3.

**Assumption 3** (Total order of conflicting transactions)**.** *For each pair $T_i, T_j \in T$ with $WS_i \bigcap WS_j \neq \emptyset$, a unique order relation $c_i^k < c_j^k$ holds for all SI-schedules $H_t^k$ with $k \in I_m$.*

Under this Assumption, it seems clear that a 1C-schedule serializes the execution of conflicting transactions as Definition 3 about SI-schedules requires.

**Property 4.** *Under Assumption 3, the 1C-schedule $H_t$ verifies that for each pair $T_i, T_j \in T$: $\neg(T_j\ impacts\ T_i\ \ at\ b_i)$.*

*Proof.* By Assumption 2, at any site $k \in I_m$, for each pair $T_j^k, T_i^k \in T^k$: $\neg(T_j^k\ impacts\ \ T_i^k\ at\ b_i^k)$. That is, $WS_j^k \cap WS_i^k = \emptyset \vee \neg(b_i^k < c_j^k < c_i^k)$.

1. If $WS_j^k \cap WS_i^k = \emptyset$, by definition of $T_j$ and $T_i$, $WS_j \cap WS_i = \emptyset$. Then, $\neg(T_j\ impacts\ T_i\ at\ b_i)$.

2. Let $WS_j^k \cap WS_i^k \neq \emptyset$. Again, by definition of $T_j$ and $T_i$, $WS_j \cap WS_i \neq \emptyset$. Hence, either $\neg(T_j^k\ impacts\ T_i^k\ at\ b_i^k)$ or $\neg(T_i^k\ impacts\ T_j^k\ at\ b_j^k)$. Thus, $c_i^k < b_j^k$ or $c_j^k < b_i^k$ holds. By Assumption 3, $c_i^k < c_j^k$ for all sites $k \in I_m$. Thus, $c_i^k < b_j^k$ for all $k \in I_m$. In particular, $c_i^{site(j)} < b_j^{site(j)}$. By definition of $H_t$: $c_i < c_j$ and $c_i \leq c_i^{site(j)} < b_j$ holds in $H_t$.

   Suppose that $T_j$ *impacts* $T_i$ *at* $b_i$ in $H_t$. That is, $WS_j \cap WS_i \neq \emptyset$ and $b_i < c_j < c_i$. A contradiction with $c_i < c_j$ is obtained. Therefore, $\neg(T_j\ impacts\ T_i\ at\ b_i)$.

   Analogously, if $T_i$ *impacts* $T_j$ *at* $b_j$ in $H_t$. That is, $WS_j \cap WS_i \neq \emptyset$ and $b_j < c_i < c_j$. A contradiction with $c_i < b_j$ is obtained again, and therefore, $\neg(T_i\ impacts\ T_j\ at\ b_j)$.

□

However, the execution of write/write conflicting transactions in the same order at all sites does not offer SI nor GSI, as it has been shown in the example of Figure 1. In that example only $T_1$ and $T_2$ had write/write conflicts and they have been executed in the same order at every site. Therefore, taking into account the second condition in Definition 4, it is necessary that a transaction reads the latest installed version in the system of a data item.

**Assumption 4** (Latest-version read)**.** *For each pair of transactions $T_i, T_j \in T$ with $WS_j \cap RS_i \neq \emptyset$: they verify that if $c_j < b_i$ in $H_t$ then $c_j^{site(i)} < b_i^{site(i)}$ in $H_t^{site(i)}$.*

Under Assumptions 3 and 4 it is easy to proof the next Theorem. It states that the 1C-schedule is a SI-schedule.

**Theorem 1.** *Under Assumption 3 and Assumption 4, the 1C-Schedule $H_t$ is a SI-schedule.*

*Proof.* By Property 4 the condition (2) in Definition 3 of SI-schedule is verified for $H_t$. We only need to prove the statement:
"$if\ R_i(X_j) \in H\ then\ \{X_j\} = latestVer(X, H_t, b_i)$".
Suppose $R_i(X_j) \in H$ and $\{X_j\} \neq latestVer(X, H_t, b_i)$. 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 Definition 7 of $H_t$, $R_i(X_j)^{site(i)} \in H^{site(i)}$. From Assumption 2, it is always satisfied that $\{X_j^{site(i)}\} = latestVer(X^{site(i)}, H_t^{site(i)}, b_i^{site(i)})$.

Also, Assumption 1 ensures that $C_r^{site(i)} \in H^{site(i)}$ and $X_r^{site(i)} \in Ver(X^{site(i)}, H^{site(i)})$. As $X \in WS_j \cap WS_r$, by Assumption 3, if $c_r^{site(i)} < c_j^{site(i)}$ then $c_r < c_j$. Since this contradicts the initial supposition, we have that $c_j^{site(i)} < c_r^{site(i)}$.
Finally, $X \in WS_r \cap RS_i$ and $c_r < b_i$ in $H_t$ by the supposition. Considering Assumption 4, if $c_r < b_i$ in $H_t$ then $c_r^{site(i)} < b_i^{site(i)}$.
In conclusion, $X_r^{site(i)} \in Ver(X^{site(i)}, H^{site(i)})$ and $c_j^{site(i)} < c_r^{site(i)} < b_i^{site(i)}$. Therefore, $\{X_j^{site(i)}\} \neq latestVer(X^{site(i)}, H_t^{site(i)}, b_i^{site(i)})$ and $H_t^{site(i)}$ is not a SI-schedule against Assumption 2. The Theorem holds. □

It is easy to show that every 1C-schedule that is a SI-schedule satisfies Assumption 4. If $R_i(X_j) \in H_t$ then $\{X_j\} = latestVer(X, H_t, b_i)$ and $c_j < b_i$. As $\{X_j^{site(i)}\} = latestVer(X^{site(i)}, H_t^{site(i)}, b_i^{site(i)})$ holds, $H_t^{site(i)}$ is a SI-schedule by Assumption 2, then $c_j^{site(i)} < b_i^{site(i)}$. Hence, both assumptions 3 and 4 compose a single sufficient condition for SI-schedules. Assumption 4 is also a necessary condition, but Assumption 3 is not. To show that, consider two sites and two transactions, $T_1$ and $T_2$, such that $WS_1 \cap WS_2 \neq \emptyset$. In the site 1, $T_1^1$ is executed before $T_2^1$; and in the site 2, $T_1^2$ is executed after $T_2^2$. Besides, consider that $c_1^1 < b_2^2$ holds. If other transaction $T_3$ starts its execution in the site 1 after the commit times of $T_2^1$ and $T_1^2$ and it writes again the same updates of $T_2^1$ in both sites, then the obtained 1C-schedule will be a SI-schedule but Assumption 3 will not hold.

In the ROWA protocols considered in Section 5, Assumption 4 emphasizes that a transaction must see the latest installed version in the system. In other words, a transaction must remain blocked until Assumption 4 becomes true. As the considered ROWA protocol only sends the writeset to remote sites, it can not be checked whether another site has installed a new version. As a straight consequence of this, it is not possible to abort the transaction violating the SI level. Thus, only a blocking ROWA protocol may obtain the SI level.

To circumvent such inconvenience, one approach to obtain SI is sending the readsets and writesets of transactions to all sites in order to know if some transaction has missed a more recent version. Sending the readset to all sites is prohibitive (leading to lower performance, poor scalability and higher abortion rates) but, on the other hand, stronger isolation levels than SI may be obtained, more precisely serializable as it has been pointed out in [13].

Another alternative approach consists in broadcasting a start message containing the transaction identifier. This message is sent when the transaction is submitted at its delegate local replica. Thus, the execution of the transaction is delayed until the message has been delivered at its local replica. In the other replicas, this message is used to mark the starting point of the remote transaction. This approach delimits the time a transaction waits for the latest version installed in the system. However, any start message must be delivered at every site in the same order with regard to any other message containing either a writeset or another start indication; and the processing of any start message must follow the same order as that obtained upon the delivery of other messages. Hence, a start indication is not processed before finishing the application of writesets received in previous messages.

In this way, by an appropriate modification of the definition 7, in order to define $b_i = min_k\{b_i^k\}$, it is obtained that, if $c_j < b_i$ in the 1C-schedule $H_t$, then $c_j^k < b_i^k$ for each SI-schedule $H_t^k$ at every site, and in particular $c_j^{site(i)} < b_i^{site(i)}$ in $H_t^{site(i)}$. If it is not true for a site, that is, $b_i^r < c_j^r$, it means that the same total order for the start message has been violated.

## 7 One Copy Generalized Snapshot Isolation Schedules

In the previous section, Assumptions 3 and 4 have been used to prove how a 1C-schedule that verifies the conditions of a SI-schedule can be obtained using ROWA protocols. However, Assumption 3 is not enough for obtaining such schedule –without Assumption 4–, and Assumption 4 is too restrictive for a regular replicated system, since it needs to block the start of transactions in order to be held. As explained in the next section, most SI replication protocols are based on multicasting transaction writesets in total order, and on guaranteeing a commit total order in all replicas. Actually, the main issue in these protocols is to maintain such total order of transaction commits. As a result, since all replicas generate SI-schedules and their local snapshots have received the same sequence of updates, transactions starting at any site are able to read a particular snapshot, that perhaps is not the latest one, but that is consistent with those of other replicas.

**Assumption 5** (Total order of committing transactions)**.** *For each pair $T_i, T_j \in T$, a unique order relation $c_i^k < c_j^k$ holds for all* SI*-schedules $H_t^k$ with $k \in I_m$.*

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 this section: $c_1^k < c_2^k < ... < c_n^k$ for every $k \in I_m$.

In the next property we are going to verify that, thanks to the total order, versions of items read by a transaction belong to the same snapshot in a given time interval. This interval is determined for each transaction $T_i$ by two commit times, denoted $c_{i_0}$ and $c_{i_1}$. The former corresponds to the commit time of a transaction $T_{i_0}$ such that $T_i$ *reads from* $T_{i_0}$ for the last time and from then it performs no other read operation. The latter corresponds to the commit time of a transaction $T_{i_1}$, so that it is the first transaction, after $T_{i_0}$, that verifies $WS_{i_1} \cap RS_i \neq \emptyset$ and hence modifying the snapshot of the transaction $T_i$. In case that $T_{i_1}$ does not exist, the correctness interval for $T_i$ will extend from $c_{i_0}$ to $b_i$.

**Property 5.** *Let $H_t$ be a 1C-schedule verifying Assumption 5. For each $T_i \in T$ if $R_i(X_j) \in H$ then $X_j \in Snapshot(DB, H_t, \tau)$ and $\tau \in \mathbb{R}^+$ satisfies $c_{i_0} \leq \tau < c_{i_1} \leq b_i$.*

*Proof.* Let $T_{i_0}^{site(i)}$ be a transaction such that $WS_{i_0} \cap RS_i \neq \emptyset$ and $c_{i_0}^{site(i)}$ defines the last time in $H_t^{site(i)}$ from which transaction $T_i^{site(i)}$ no longer *reads from* $T_{i_0}^{site(i)}$ a version of a data item. By Assumption 2: $\forall Y \in WS_{i_0} \cap RS_i\colon \{Y_{i_0}^{site(i)}\} = latestVer(Y^{site(i)}, H_t^{site(i)}, b_i^{site(i)})$. By Assumption 1 and Definition 7: $T_{i_0} \in T$ and $c_{i_0} < b_i$.
Let $X \in RS_i$ be an item read by $T_i$ such that $X \notin WS_{i_0} \cap RS_i$ and $\{X_j^{site(i)}\} = latestVer(\, X^{site(i)}, H_t^{site(i)}, b_i^{site(i)})$. We prove that $\nexists T_r \in T\colon X_r \in Ver(X, H) \wedge c_j < c_r < c_{i_0}$. Note that if this property is false, then the version $X_r$ will be more up-to-date than $X_j$ in $H_t$ when $T_i$ *reads from* $T_{i_0}$. The 1C-schedule $H_t$ will not be a GSI-schedule. By contradiction, if there exists $T_r$ and $c_j < c_r < c_{i_0}$ then by Assumption 5 and 1: $c_j^{site(i)} < c_r^{site(i)} < c_{i_0}^{site(i)}$. Thus, $X_j^{site(i)}$ is not the latest version in $H_t^{site(i)}$ at $b_i^{site(i)}$.
It is important to note that $c_{i_0}^{site(i)}$ defines the moment where $T_i^{site(i)}$ reads the latest version for $H_t^{site(i)}$. Hence, $c_{i_0}$ will define for $H_t$ the time instant of $T_i$'s snapshot. If there exists a transaction $T_{i_1}$ with $WS_{i_1} \cap RS_i \neq \emptyset$, then $T_i$ will not see the versions installed by $T_{i_1}$. Thus, $b_i^{site(i)} < c_{i_1}^{site(i)}$. However, it may happen in $H_t$ that $c_{i_1} < b_i$. In fact, this is the main reason to be $H_t$ a GSI-schedule.
In conclusion, for all $X \in RS_i$, $X_j \in Snapshot(DB, H_t, c_{i_0})$ holds. This is valid for every $\tau, c_{i_0} \leq \tau$, until the first transaction $T_{i_1} \in T$ such that $WS_{i_1} \cap RS_i \neq \emptyset$ or until $b_i (b_i = b_i^{site(i)})$ if there not exists such a transaction. Therefore, $c_{i_0} \leq \tau < c_{i_1} \leq b_i$ holds.

□

The aim of the next theorem is to prove that the 1C-schedules generated by any ROWA protocol that verifies Assumption 5 are actually GSI-schedules; i.e., they comply with all conditions stated in Definition 5. Whilst proving that a transaction always reads from the same snapshot in a particular time interval is easy, it is not trivial to prove that for a given transaction $T_i$ there has not been any other transaction $T_j$ that has impacted $T_i$ and that has been committed whilst $T_i$ was being executed. However, due to the total commit order an induction proof is possible, showing that the obtained 1C-schedule verifies all conditions in order to be a GSI-schedule.

**Theorem 2.** *Under Assumption 5, the 1C-schedule $H_t$ is a GSI-schedule.*

*Proof.* $H_t$ is a GSI-schedule if verifies Definition 5. Under Assumption 5 and 2, the SI-schedules $H_t^k$ have the same total order of committed transactions: $c_1^k < c_2^k < ... < c_n^k$ for every $k \in I_m$. $H_t$ also verifies such an order $c_1 < c_2 < ... < c_n$ because by Definition 7 $min_{k \in I_m}\{c_i^k\} < min_{k \in I_m}\{c_j^k\}$ with $1 \leq i < j \leq n$.
The rest of the proof is made by induction over such a total order. First, we define the subsets of transactions for each $i \in I_n\colon T(i) = \{T_1, T_2, ..., T_i\} \subseteq T$ and for each $k \in I_m\colon T^k(i) = \{T_1^k, T_2^k, ..., T_i^k\} \subseteq T^k$. Using these subsets we define $H^k(i)$, $H_t^k(i)$, $H(i)$ and $H_t(i)$. They are exactly equal to $H^k$, $H_t^k$, $H$ and $H_t$ respectively, except that they only include the operations in $T^k(i)$ or $T(i)$. Thus, it is clear that $H^k(n) = H^k$, $H_t^k(n) = H_t^k$, $H(n) = H$ and $H_t(n) = H_t$.

***Induction Base.*** $H_t(1)$ is a GSI-schedule. There is only one committed transaction in $T(1)$. Therefore, Definition 5 is trivially verified for $H_t(1)$.
***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, $i \in I_m$. Note that $T(i) = T(i-1) \cup \{T_i\}$. As $H_t(i-1)$ is a GSI-schedule, by Hypothesis, for any pair $T_j, T'_j \in T(i-1)$: $\neg(T_j$ impacts $T'_j$ at $s'_j)$. As $c_j < c_i$ for $1 \leq j \leq i-1$, by the considered total order, $\neg(T_i$ impacts $T_j$ at $s_j)$ in $H_t(i)$. If $R_j(X_r) \in H(i-1)$ and $X_r \in Snapshot(DB, H_t(i-1), s_j)$ for $1 \leq j \leq i-1$ then $R_j(X_r) \in H(i)$. $X_r \neq X_i$ because $c_j^{site(j)} < c_i^{site(j)}$ and $H_t^{site(i)}$ is a 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 $T_i \in T$ that there exists a value $s_i \leq b_i$ such that:

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

(b) for each $T_j \in T(i) : \neg(T_j$ impacts $T_i$ at $s_i)$.

The begin time $b_i^{site(i)}$ and the commit time $c_{i_0}$ of the transaction $T_{i_0} \in T(i)$ from which $T_i$ reads for the last time, allow us to define the sets:

$T_1(i) = \{T_j \in T : b_i^{site(i)} < c_j^{site(i)} < c_i^{site(i)}\}$

$T_2(i) = \{T_j \in T : c_{i_0}^{site(i)} < c_j^{site(i)} < b_i^{site(i)}\}$

By Assumption 2, $\forall T_j \in T_1(i) : WS_j \cap WS_i = \emptyset$, i.e. $H_t^{site(i)}$ is a SI-schedule, and by definition of $T_{i_0} \in T$ and again Assumption 2, $\forall T_j \in T_1(i) : WS_j \cap RS_i = \emptyset$. Let $T_{i_2} \in T_2(i)$ be the last transaction such that in the total order it verifies $WS_{i_2} \cap WS_i \neq \emptyset$. Note that in the 1C-schedule, obtained from Definition 7, a commit time $c_j^{site(i)}$ for a transaction in $T_1(i)$ may change its relation with respect to $b_i$, but maintaining the order relation with respect the other commit times. Let $T_{i_1} \in T_1(i)$ be the first transaction such that $c_{i_1} < b_i$ in $H_t$ and $WS_{i_1} \cap RS_i \neq \emptyset$ Thus, $c_{i_0} < c_{i_2} < c_{i_1} < b_i$ holds in $H_t$.
For any value $s_i \in (c_{i_2}, c_{i_1})$, (a) holds for Property 5 and (b) holds by the way $T_{i_2} \in T$ has been defined. For each $T_j$ such that $c_{i_2} < c_j < b_i$, if $T_j \in T_2(i)$ then $WS_j \cap WS_i = \emptyset$. If not, $T_{i_2}$ is not the last transaction verifying such a condition; and if $T_j \in T_1(i)$ then $WS_j \cap WS_i = \emptyset$. Thus, these transactions do not impact with $T_i$. The rest of transactions do not either impact 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 (a) and (b) holds. In case that it does not exist $T_{i_2}$ then $s_i \in (c_{i_0}, c_{i_1})$ and again (a) and (b) holds.

□

This proof has not been given before in any ROWA-based SI replication protocol ensuring total order for the commit operations of all transactions in the system replicas. This theorem formally justifies such protocols correctness and establishes that their resulting isolation level is GSI. Additionally, it is worth noting that Assumption 5 is a sufficient condition, but not necessary, for obtaining GSI. Despite this, replication protocols that comply with such an assumption are easily implementable. In the next section, we analyze some recent SI replication protocols and other related works.

## 8 Related Work

The principles stated in this paper have been already used in many database replication protocols that provide the SI level [26, 22, 23, 27, 13]. Indeed, all of them use an atomic broadcast protocol for writeset propagation, and thus all transactions are totally ordered. In [26] those delivered writesets that do not intersect are allowed to proceed concurrently, and this may imply that they might be applied in different orders in different replicas. However, this creates *holes* in the writeset list being managed by this protocol and, as a result, local transactions are blocked until these holes disappear. When this happens, the effects of these concurrent writeset applications are the same as those of an application in total order, and the local transactions are then allowed to begin.

The assurance of this total order in the application of transactions in all replicas also leads to the typical certification carried out in this kind of protocols. All delivered writesets are ordered in the same way in all replicas and a conflict checking is made in order to certify each transaction. This implies that local transactions that collide with such writesets must be aborted. Moreover, this kind of certification can be locally performed, without needing an additional voting round among replicas (the needed coordination has been achieved using the total order broadcast). Unfortunately, in some cases a local DBMS may abort the application of some writeset (e.g., due to a deadlock, or due to a local failure), but the replication protocol has to be prepared to manage appropriately these events. To this end, such writeset applications must be retried until they are successful, and the order of their application must be correctly ensured. Some protocols have already described this kind of events (e.g., [26]).

All certification-based protocols need to maintain a historic list of already applied writesets in order to check each incoming writeset for validating it. If no conflict arises, the transaction can commit, otherwise –in case of "impact"– the transaction being certified should be aborted. The problem with such historic list is that the writesets are not small, and they demand a lot of memory (either main memory or secondary storage) for maintaining such list. This might compromise the performance of this kind of replication protocols. Hopefully, such historic list does not need to store its writesets indefinitely. A sequence of writesets can be dropped from this list when there is no currently active transaction in any replica that started before the certification time of such writesets. A short subprotocol should be added to the SI replication protocols in order to ensure such garbage collection of the historic list, and [2] provides a first example of this kind of subprotocols.

The certification process outlined in the previous paragraphs –and detailed in some parts of this paper– can be optimized in a middleware-based implementation (as in our MADIS middleware [21]) using the assistance of the concurrency control mechanisms of the underlying DBMS. To this end, the middleware protocols need to scan periodically a catalog table of the underlying DBMS where the blocking relations between active transactions are maintained. We have thoroughly analyzed this mechanism in [27] comparing three different implementations of a SI replication protocol, and its results show that using such approach the protocol is able to abort sooner those conflicting transactions that otherwise will have been aborted in the certification phase. Due to this, the transaction completion time of the committed transactions can be reduced (from 1% to 10%, depending on the number of replicas and the transaction length), improving thus the performance of a middleware-based replication protocol.

The need of blocking local transactions when the total order cannot be ensured is a little flaw that might generate a serious performance loss for these blocked transactions. To reduce this problem, some complementary techniques must be used to ensure that writesets are quickly applied. One of the best examples is [12] where both commit ordering and transaction durability are merged either into the middleware or into the database engine. Thus, a batch of ordered writesets is applied at once, achieving a throughput increase that is 3 to 5 times better than that of the traditional middleware techniques.

In [11] an isolation level intermediate to the GSI and SI levels discussed in this paper is proposed and justified: *strong session SI*. It is meaningful for systems with *lazy* update propagation, according to the classification given in [17]. This *strong session SI* level is able to avoid transaction inversions, as our defined 1CSI also does, but without the overall blocking behavior of a 1CSI protocol. The problem of *transaction inversions* arises when a particular client that submits two consecutive transactions is not able to see in the second one the updates generated by the first one. This might happen using a GSI level combined with lazy propagation. In order to avoid such problem, a given application should be divided into multiple *sessions* (disjoint subsets of transactions) and the *reads-from* relation between committed transactions should be enforced only inside each session. Although the aim of that paper is not the same as ours, the definitions given in [11] for 1CSI and GSI (named, respectively, *strong* and *weak SI* in that paper) are identical to the ones used in our paper.

In the field of federated databases there have been some proposals for achieving a SI level. One of the first papers of this kind is [33] that presents two different techniques for providing a global SI level (i.e., something similar to 1CSI for federated databases). The first one is based on synchronizing the start of subtransactions; i.e., it requires that all subtransactions that will compose a federated transaction start at the same time. To this end, this requires that all of them start a *begin* operation at once and such operation can not be interleaved with the *atomic commitment protocol* of other federated transactions. This is a stronger requirement than the one described in this paper, but the context is also different since federated transactions are more complex than *update anywhere* [17] replication protocols (many subtransactions can be directly initiated in a federated transaction, whilst in a replicated environment only a single delegate replica directly executes the transaction whilst all other replicas simply apply the updates contained in the propagated writeset). Its authors also discourage this approach due to its transaction throughput loss. The second technique uses an optimistic approach and only requires two conditions on transaction execution;

thus, two federated transactions cannot be (1) concurrent in one database, and (2) have a reads-from relation in another database. That paper formalizes and justifies (although does not prove its correctness in depth) this second technique. It also recommends its use since that technique does not incur in the performance penalties of the first one. However, such two conditions can not arise in a replicated setting. As a result, the formalization and justification given in [33] do not exactly match what has been presented in this paper. Indeed, such paper does not discuss anything similar to GSI nor 1C-GSI.

Coming back to the contents of Section 4, an appropriate generalization of our definition 5 might include a third property of a GSI-schedule. This property bounds the freshness of the snapshot provided to a transaction $T_i$, in terms of a *distance function* $d$:

$$3. \quad d(Snapshot(s_i), Snapshot(b_i)) \in [0, k_i]$$

The above mentioned *distance function* $d$ could be defined in a variety of ways, ranging from a time-based one (e.g. $d(Snapshot(s_i), Snapshot(b_i)) = b_i - s_i$) to more complex value-based specifications, using the number of updated items, or even the relative value change on items read by a transaction. We have taken this research line in [3] providing there a flexible replication protocol that is able to bound the degree of freshness for the transactions' snapshot. As a result, our *k-bound* SI replication protocol [3] is able to range from a relaxed GSI level to a strict 1CSI level as defined in this paper. In order to achieve the 1CSI level it only requires an empty total-order multicast message that is needed to start a transaction. Such message should be multicast by the transaction delegate replica, and once delivered, such transaction is allowed to start. Note that, as proved in the current paper and previously suggested in [13], the strict SI protocol needs to block transactions start. Additionally, this protocol combines this certification-based variant for the GSI and 1CSI levels with a weak-voting [35] one that allows the support of the serializable isolation level. So, in a single protocol we have provided support for three different isolation levels.

Regarding this *k-bound* replication protocol, other recent papers [5] of our group state that for supporting multiple isolation levels the simplest approach is to use a single protocol architecture (any of those distinguished in the classification given in [35], for instance). So, we have also unified the protocol architecture of the original *k-bound* protocol (in [22]), using a weak-voting approach for all its three levels. Up to our knowledge this is one of the first protocols that uses the weak-voting approach for SI replication protocols and can improve the system performance for those applications having a high degree of transaction conflicts, since the validation/certification step is only made in one replica. Additionally, with a weak-voting approach no replica needs to maintain any writeset history and this reduces the amount of memory needed by each site in order to execute the protocol.

## 9 Conclusions

This paper studies ROWA protocols for database replication, where each replica uses a DBMS providing SI isolation level. ROWA replication protocols exclusively based on propagating the writeset of transactions may not achieve the one-copy-SI consistency level unless they do block the beginning of transactions until they get the latest system snapshot. This potential blocking of transactions makes the main attraction of SI, the non-blocking execution of read operations, not feasible. This is the main reason for introducing GSI in database replication scenarios.

This paper establishes that the sufficient condition for obtaining a GSI consistency level is the same total order of committed transactions. This fact limits the kind of replications protocols to be implemented in a replicated setting in order to obtain GSI.

To sum up, all the properties that have been formalized in our paper seem to be assumed in some previous works, but none of them carefully identified nor formalized such properties. As a result, we have provided a sound theoretical basis for designing and developing future replication protocols with Generalized Snapshot Isolation.

## Acknowledgment

This work has been supported by the EU FEDER and Spanish MEC under grant TIN2006-14738-C02.

## References

[1] José Enrique Armendáriz, José Ramón Garitagoitia, José Ramón González de Mendívil, and Francesc D. Muñoz-Escoí. Design of a MidO2PL database replication protocol in the MADIS middleware architecture. In *Proc. of 20th Intnl. Conf. on Advanced Information Networking and Applications - Vol. 2 (AINA'06)*, pages 861–865. IEEE-CS, 2006.

[2] J. E. Armendáriz-Iñigo, J. R. Garitagoitia, F. D. Muñoz-Escoí, and J. R. González de Mendívil. MADIS-SI: A database replication protocol with easy recovery. Technical Report ITI-ITE-06/05, Instituto Tecnológico de Informática, Valencia, Spain, July 2006.

[3] J. E. Armendáriz-Iñigo, J. R. Juárez, J. R. González de Mendívil, H. Decker, and F. D. Muñoz. k-Bound GSI: A flexible database replication protocol. In *Proc. of ACM Symposium on Applied Computing*, Seoul, Korea, March 2007. ACM Press.

[4] Hal Berenson, Philip A. Bernstein, Jim Gray, Jim Melton, Elizabeth J. O'Neil, and Patrick E. O'Neil. A critique of ANSI SQL isolation levels. In *SIGMOD Conference*, pages 1–10. ACM Press, 1995.

[5] Josep Maria Bernabé-Gisbert, Raúl Salinas-Monteagudo, Luis Irún-Briz, and F. D. Muñoz-Escoí. Managing multiple isolation levels in middleware database replication protocols. *Lecture Notes in Computer Science*, 4330:511–523, December 2006.

[6] Philip A. Bernstein, Vassos Hadzilacos, and Nathan Goodman. *Concurrency Control and Recovery in Database Systems*. Addison Wesley, 1987.

[7] Yuri Breitbart, Raghavan Komondoor, Rajeev Rastogi, S. Seshadri, and Abraham Silberschatz. Update propagation protocols for replicated databases. In Alex Delis, Christos Faloutsos, and Shahram Ghandeharizadeh, editors, *SIGMOD Conference*, pages 97–108. ACM Press, 1999.

[8] Gregory Chockler, Idit Keidar, and Roman Vitenberg. Group communication specifications: a comprehensive study. *ACM Comput. Surv.*, 33(4):427–469, 2001.

[9] Parvathi Chundi, Daniel J. Rosenkrantz, and S. S. Ravi. Deferred updates and data placement in distributed databases. In *Proc. of Intnl. Conf. on Data Engineering, New Orleans, Louisiana, USA*, pages 469–476. IEEE-CS, 1996.

[10] Khuzaima Daudjee and Kenneth Salem. Lazy database replication with ordering guarantees. In *Proc. of Intnl. Conf. on Data Engineering, Boston, MA, USA*, pages 424–435. IEEE-CS, 2004.

[11] Khuzaima Daudjee and Kenneth Salem. Lazy database replication with snapshot isolation. In *Proc. of the 32nd International Conference on Very Large Data Bases, Seoul, Korea*, pages 715–726, 2006.

[12] Sameh Elnikety, Steven Dropsho, and Fernando Pedone. Tashkent: Uniting durability with transaction ordering for high-performance scalable database replication. In *Proc. of EuroSys Conference*, Leuven, Belgium, April 2006.

[13] Sameh Elnikety, Fernando Pedone, and Willy Zwaenopoel. Database replication using generalized snapshot isolation. In *Symposium on Reliable Distributed Systems, Orlando, FL, USA*, pages 73–84. IEEE-CS, 2005.

[14] Alan Fekete. Allocating isolation levels to transactions. In Chen Li, editor, *PODS*, pages 206–215. ACM, 2005.

[15] Alan Fekete, Dimitrios Liarokapis, Elizabeth O'Neil, Patrick O'Neil, and Dennis Shasha. Making snapshot isolation serializable. *ACM Trans. Database Syst.*, 30(2):492–528, 2005.

[16] Alan Fekete, Elizabeth J. O'Neil, and Patrick E. O'Neil. A read-only transaction anomaly under snapshot isolation. *SIGMOD Record*, 33(3):12–14, 2004.

[17] Jim Gray, Pat Helland, Patrick E. O'Neil, and Dennis Shasha. The dangers of replication and a solution. In H. V. Jagadish and Inderpal Singh Mumick, editors, *SIGMOD Conference*, pages 173–182. ACM Press, 1996.

[18] Vassos Hadzilacos and Sam Toueg. A modular approach to fault-tolerant broadcasts and related problems. Technical Report TR94-1425, Dep. of Computer Science, Cornell University, Ithaca, New York (USA), 1994.

[19] JoAnne Holliday, Robert C. Steinke, Divyakant Agrawal, and Amr El Abbadi. Epidemic algorithms for replicated databases. *IEEE Trans. Knowl. Data Eng.*, 15(5):1218–1238, 2003.

[20] L. Irún, F. Muñoz, H. Decker, and J. M. Bernabéu-Aubán. COPLA: A platform for eager and lazy replication in networked databases. In *5th Int. Conf. Enterprise Information Systems (ICEIS'03)*, volume 1, pages 273–278, April 2003.

[21] Luis Irún, Hendrik Decker, Rubén de Juan, Francisco Castro, Jose E. Armendáriz, and Francesc D. Muñoz-Escoí. MADIS: A slim middleware for database replication. In *Euro-Par*, volume 3648 of *LNCS*, pages 349–359. Springer, 2005.

[22] J. R. Juárez, J. R. González de Mendívil, J. R. Garitagoitia, J. E. Armendáriz, and F. D. Muñoz-Escoí. A middleware database replication protocol providing different isolation levels. In *15th Euromicro Intnl. Conf. on Parallel, Distributed, and Network-based Processing (WIP Track)*, Naples, Italy, February 2007. IEEE-CS Press.

[23] B. Kemme. *Database Replication for Clusters of Workstations (ETH Nr. 13864)*. PhD thesis, Swiss Federal Institute of Technology, Zurich, Switzerland, 2000.

[24] Bettina Kemme and Gustavo Alonso. A new approach to developing and implementing eager database replication protocols. *ACM Trans. Database Syst.*, 25(3):333–379, 2000.

[25] Bettina Kemme, Fernando Pedone, Gustavo Alonso, André Schiper, and Matthias Wiesmann. Using optimistic atomic broadcast in transaction processing systems. *IEEE Trans. Knowl. Data Eng.*, 15(4):1018–1032, 2003.

[26] Yi Lin, Bettina Kemme, Marta Patiño-Martínez, and Ricardo Jiménez-Peris. Middleware based data replication providing snapshot isolation. In *SIGMOD Intnl. Conf. on Management of Data, Baltimore, Maryland, USA*, pages 419–430. ACM Press, 2005.

[27] F. D. Muñoz-Escoí, J. Pla-Civera, M. I. Ruiz-Fuertes, L. Irún-Briz, H. Decker, J. E. Armendáriz-Iñigo, and J. R. González de Mendívil. Managing transaction conflicts in middleware-based database replication architectures. In *25th Symposium on Reliable Distributed Systems*, pages 401–410, Leeds, UK, October 2006. IEEE-CS Press.

[28] Esther Pacitti and Eric Simon. Update propagation strategies to improve freshness in lazy master replicated databases. *VLDB J.*, 8(3-4):305–318, 2000.

[29] Christos Papadimitriou. *The Theory of Database Concurrency Control*. Computer Science Press, 1986.

[30] Marta Patiño-Martínez, Ricardo Jiménez-Peris, Bettina Kemme, and Gustavo Alonso. Consistent database replication at the middleware level. *ACM Transactions on Computers*, 23(4):375–423, 2005.

[31] Christian Plattner and Gustavo Alonso. Ganymed: Scalable replication for transactional web applications. In Hans-Arno Jacobsen, editor, *Middleware*, volume 3231 of *Lecture Notes in Computer Science*, pages 155–174. Springer, 2004.

[32] Uwe Röhm, Klemens Böhm, Hans-Jörg Schek, and Heiko Schuldt. FAS - a freshness-sensitive coordination middleware for a cluster of OLAP components. In *Proc. of Intnl. Conf. on Very Large Data Bases, Hong Kong, China*, pages 754–765, 2002.

[33] Ralf Schenkel, Gerhard Weikum, Norbert Weißenberg, and Xuequn Wu. Federated transaction management with snapshot isolation. *Lecture Notes in Computer Science*, 1773:1–25, 2000.

[34] Transaction Processsing Performance Council. Benchmarck specifications. Accessible in URL: `http://www.tpc.org/`, 2006.

[35] Matthias Wiesmann and André Schiper. Comparison of database replication techniques based on total order broadcast. *IEEE Trans. Knowl. Data Eng.*, 17(4):551–566, 2005.

[36] Shuqing Wu and Bettina Kemme. Postgres-R(SI): Combining replica control with concurrency control based on snapshot isolation. In *Proc. of Intnl. Conf. on Data Engineering, Tokyo, Japan*, pages 422–433. IEEE-CS, 2005.