Bernabe Gisbert, JM.; Muñoz Escoí, FD. (2012). Supporting multiple isolation levels in replicated environments. Data and Knowledge Engineering. 79-80:1-16. doi:10.1016/j.datak.2012.05.001.

Supporting Multiple Isolation Levels in Replicated Environments

José María Bernabé Gisbert[a], Francesc D. Muñoz-Escoí[a]

[a] *Instituto Universitario Mixto Tecnológico de Informática; Universitat Politècnica de València; 46022 Valencia (Spain)*

Abstract

Replication is used by databases to implement reliability and provide scalability. However, achieving transparent replication is not an easy task. A replicated database is *transparent* if it can seamlessly replace a standard stand-alone database without requiring any changes to the components of the system. Database replication transparency can be achieved if: (a) replication protocols remain hidden for all other components of the system; and (b) the functionality of a stand-alone database is provided.

The ability to simultaneously execute transactions under different isolation levels is a functionality offered by all stand-alone databases but not by their replicated counterparts. Allowing different isolation levels may improve overall system performance. For example, the TPC-C benchmark specification [33] tolerates execution of some transactions at weaker isolation levels in order to increase throughput of committed transactions. In this paper, we show how replication protocols can be extended to enable transactions to be executed under different isolation levels.

Key words: replication protocol, isolation level, consistency

1. Introduction

Replication is considered *transparent* if it can be implemented and deployed in existing systems without requiring changes to system components or external applications.

Existing replication protocols hide replication management by implementing a standard database access interface such as *Java database connectivity* (JDBC). Unfortunately, these solutions do not really implement all the functionality available in stand-alone databases. A typical *database management system* (DBMS) simultaneously executes transactions under different isolation levels, but most replication protocols support a single level, which is usually either serialisable [7] or snapshot isolation [4]. The authors of [8] identified this problem as one of the challenges in database replication. Both TPC-C and TPC-W benchmarks also demonstrate that typical applications need to execute transactions at different isolation levels, mainly for performance reasons.

1.1. Benefits of supporting multiple isolation levels

In a database, transactions should ideally be executed in isolation without interfering with each other. However, DBMSs allow some interference in order to increase concurrency and so,

Email addresses: jbgisber@iti.upv.es (José María Bernabé Gisbert), fmunyoz@iti.upv.es (Francesc D. Muñoz-Escoí)

improve system response time and throughput. Unfortunately, concurrent execution of transactions may generate anomalies (or phenomena) that must be resolved at the application tier. For example, transaction T_1 reads x data item while T_2, another concurrent transaction, is updating the same data item. Such a situation is defined as *non-repeatable* or *fuzzy read* phenomenon in [4]. If transaction T_1 reads data item x again, it may read a different value. Isolation levels are defined by the anomalies that are forbidden in the execution of transactions [4].

The strongest isolation level is a serial execution of transactions (i.e., transactions can not be executed concurrently). Due to performance issues, serialisability[1] is usually the strongest isolation level provided by a commercial DBMS. Not all transactions require such strong guarantees. Therefore, commercial DBMSs also support weaker isolation levels. Many DBMSs (e.g., PostgreSQL [29], Oracle [25], and Microsoft SQL Server [24]) use by default the *read committed* [4] (RC) isolation level. This level is much weaker than the serialisable level since it allows non-repeatable reads and phantom reads [4] (a transaction obtains differing results if it executes the same read operation twice). These DBMSs delegate strong isolation levels for sensitive transactions with strong isolation restrictions. For example, a transaction T_1 that withdraws cash from a bank account requires stronger isolation than a transaction T_2 that only retrieves a list of account balances. Notice that transaction T_1 probably first reads the balance to check if there is enough money to withdraw. Meanwhile if another transaction T_3 modifies the same account balance, the application may take incorrect decisions since the account balance changed when T_1 updated the account. However, this anomaly would never be produced by T_2. Therefore, transaction T_2 may be executed under a weaker isolation level.

Although multiple commercial DBMSs agree on using RC by default they do not agree on how the *American National Standards Institute* (ANSI) *serialisable* [14] isolation level should be implemented. Several DBMSs provide a slightly weaker isolation level known as *snapshot isolation* (SI) [4] that is faster but may produce non-serialisable executions. Some commercial DBMSs, such as Microsoft SQL Server [24], allow applications to decide if they want to use serialisable or SI. It is uncommon for applications to mix serialisable and SI transactions; yet this may happen when multiple applications accessing the same database select different isolation levels for transactions with strong isolation requirements. As a result, these DBMSs may face situations in which serialisable, SI and RC transactions need to be executed concurrently.

Weak isolation levels reduce the complexity of concurrency control management and so may improve the performance of DBMSs. Note that these levels can allow higher degrees of concurrency than the strictest isolation levels. Performance improvements are especially appealing for ROWAA (*read-once, write-all-available*) [7] replication protocols since in these protocols isolation management usually involves all database replicas. In several ROWAA replication solutions, transactions are executed optimistically at a single replica (local replica); and updates are propagated to all replicas before the transaction commits. Once updates are delivered, a validation step is executed to guarantee that no forbidden phenomena occur due to conflicts with transactions executed in other replicas. The stronger the isolation level, the larger the set of phenomena that must be checked. Therefore, the complexity of the validation step depends on the isolation level. Furthermore, the stronger the isolation level of a transaction, the greater is the probability that the transaction will abort. For example, as we explain later in Section 5, serialisable transactions require either the inclusion of the accessed values when updates are broadcast; or the propagation of the local replica validation result. However, this step is unnecessary when weak

[1] Serialisability allows concurrency as long as the final result can be considered equivalent to a serial execution.

isolation levels are used. Under high workloads replicas spend resources validating transactions that cannot be mitigated by adding more replicas, since all replicas have to perform the same validation step. If many transactions can be executed under weaker isolation levels such as RC, then network and CPU loads, as well as transaction abort rates, can be reduced significantly. For instance, the performance results presented in [32] show that with the *SIRC replication protocol* and for a given type of update transactions, only 60% of the completion time needed with SI was required when RC isolation was used. Additionally, the abortion rate with RC isolation was 26 times lower than with SI.

In other ROWAA protocols (for example, primary copy replication protocols) validation is entirely performed by one replica (usually known as the primary replica or the central validator). In these cases, the validation gains obtained by weak isolation levels are negligible since no data propagation is demanded to complete the validation step. Note, however, that in a system based on a central validator the performance may degrade under medium and high workloads because the central validator does most of the work and so easily becomes a bottleneck [35]. It also might become a single point of failure. As a result, there is no optimal deployment for the validation tasks, and relaxed isolation levels can always increase concurrency.

In this article we identify the conditions under which replication protocols transparently manage isolation levels and prove that protocols that satisfy these conditions are correct. We then show how some popular ROWAA-based replication schemes can be extended to support different isolation levels.

1.2. Related work

One of the main goals of this paper is to specify the correctness criteria needed to decide whether a replication protocol is correct when it supports multiple isolation levels. Traditionally, serialisability theory has been used as a correctness criterion for centralised systems [7]. The execution of a set of transactions is considered correct if the result is equivalent to one possible serial execution of the same set of transactions. If this set is executed in a replicated system, it is considered correct if it is equivalent to one possible execution of the same set of transactions in a serialisable centralised DBMS. This criterion was introduced by *Bernstein et al.* [7] as one-copy serialisability or 1SR. However, serialisability can be expensive and to improve performance existing DBMSs allow transactions to be executed with weaker isolation guarantees. Most DBMSs support the set of isolation levels defined by ANSI [14]. Unfortunately, ANSI definitions have proven to be ambiguous and incomplete [4] and some systems that apparently provided serialisability were actually providing a slightly weaker isolation level defined as *snapshot isolation* (SI) by *Berenson et al.* [4]. SI is supported today by many commercial DBMSs (Oracle, PostgreSQL, MS SQL Server, etc.) and is widely used by applications. Some works extended Bernstein equivalence theory to one-copy-SI [21, 20] to define under which circumstances a replicated system behaves as a centralised DBMS providing SI.

However, sometimes even SI is too expensive and regular applications use weaker isolation levels for some transactions [33]. This may be reasonable in some cases. Note that some isolation guarantees may be ensured at the application tier. In other cases, the appearance of some phenomena (such as the reading of stale data) in the execution of some transactions is considered a minor problem and can be accepted if there are some performance improvements (e.g., a higher degree of concurrency that enables a faster transaction completion time). Unfortunately, as far as we know, existing one-copy equivalence definitions are not general enough and cannot be used when multiple isolation levels are supported concurrently in a replicated system. Our paper tries to fill this void.

Papers [6] and [32] presented specific protocols that support several isolation levels simultaneously ([32] includes some empirical results). However, these papers do not present a general solution, nor do they formally demonstrate the correctness of their proposals.

In [30], the authors present a meta-protocol that can execute several replication protocols at the same time. Before the execution of a transaction, one of the supported replication protocols is selected based on the transaction requirements. This approach is more modular and general than [6, 32] since isolation is only one possible criterion to match a protocol to a given transaction. However, sometimes modifying an existing and implemented protocol is a more straightforward solution than deploying a meta-protocol and developing at least one protocol per supported isolation level. Furthermore, the work in [30] does not prove the correctness of the protocols, since it only discusses the architecture of the meta-protocol. The specifications presented here can be applied to prove the correctness of a given combination of protocols managed by the meta-protocol.

In [16] the authors present a replication protocol supporting *serialisable*, *snapshot isolation* (SI) and *generalized SI* (GSI) [11], a variation of SI more suitable to distributed databases (in GSI, each transaction may start immediately in its local replica by reading from its currently available database snapshot, whilst a strict interpretation of SI semantics requires the latest system-wide database snapshot to be obtained and this might demand an additional synchronisation step at each transaction start). Our work can be used to prove the correctness of these protocols and all their variations.

The correctness theory presented here to prove the correctness of replication protocols mixes *serialisation graphs* and one-copy-equivalence concepts introduced by *Bernstein et al.* [7] with the *mixing theorem* introduced by *Adya* [1] for centralised systems.

Serialisation graphs are used to represent dependencies among transactions during an execution in a DBMS. Since Bernstein's work focuses on the serialisable isolation level, we use Adya's *mixed serialisation graphs* (MSG) and *mixing-correct* definition [1] to support other isolation levels as well. Unfortunately, Adya's mixing theory does not include the SI level and, most importantly, his specifications are not directly applicable to replicated systems, as illustrated by *Lin et al.*'s [20] extensions. In Section 3.3 we extend MSG and *mixing-correct* definitions to examine the SI level. In Section 4.1 we make a further extension to support replicated systems.

One-copy-equivalence is needed to decide, from the user point of view, whether the execution of a set of transactions in a replicated system can be considered equivalent to an execution of the same set in a correct stand-alone system. We use our *serialisation graph* extensions to make that comparison. Some other works use similar concepts and methodologies but none support multiple isolation levels. For example, *Lin et al.*'s 1-copy-SI [20] focuses on SI replication protocols. Our paper extends this approach since our correctness criteria can be applied to any replication protocol supporting one, all, or a subset of the main isolation levels.

The protocol SER CBR used in Section 7 is a *serialisable* version of existing SI CBR [31] that supports *snapshot isolation*. Both can be considered variations or interpretations of well known *serialisable* and *snapshot isolation* protocols [17, 11, 21, 9]. MUL CBR combines SER CBR and SI CBR to support the four main isolation levels considered in this work (see Section 3).

As it has been commented above, when multiple isolation levels are allowed in a database replication system, its throughput is improved. There are some other means to optimise performance in this kind of systems. A careful design of the replication mechanisms or the replication protocol [10, 22] is the other common way. Both approaches are complementary and can be integrated without problems.

Another related research line is that of managing data elements as abstract data types, considering their semantics in their accessing operations [23]. Thus, those operations are not restricted to only read or write accesses and higher levels of concurrency may be allowed with specific concurrency control mechanisms.

1.3. Roadmap

In Section 2 we provide a basis for this work by introducing the assumed system model. Section 3 discusses the existing definitions and why they can not be used in our case. Section 4 identifies the conditions a replication protocol must satisfy to behave like a stand-alone commercial DBMS. Section 5 shows a general scheme to upgrade existing replication protocols to simultaneously support multiple isolation levels. Section 6 shows how the suggested scheme can be applied to some existing ROWAA update-everywhere replication protocols; and an example protocol is presented in Section 7. Finally, Section 8 concludes the article.

2. Background

2.1. Model

We assume an asynchronous system composed of a set of nodes $\mathcal{N}$. Each node has a complete copy of the database managed by a typical stand-alone DBMS locally supporting several isolation levels. Replication is implemented by middleware deployed on top of the DBMS. This middleware has access to a group communication system with an atomic broadcast primitive [13]. A transaction can be initially submitted to any node in the system and which then becomes its *local node*.

Nodes may fail by crashing. However, note that node failures and their recoveries are not the focus of this paper. Database replication protocols should deal with failures, and many papers have provided solutions to this problem, including [15, 19]. The recovery subprotocols that are needed to manage this problem do not have any effect on our specifications nor on the architectural protocol details needed for adequately managing multiple isolation levels. Therefore, no further discussion on failures is given.

2.2. Databases and transactions

A database is composed of data items that can be read and modified by clients executing transactions. A transaction T_i is a set of read and write operations executed atomically [1], that is, either all operations execute or none execute. We represent the set of items read and written by T_i as RS_i and WS_i, respectively. We call a transaction *read-only* if it does not contain any write operations ($WS_i = \emptyset$), and *update* otherwise. Every transaction terminates with a special operation which can be a commit c_i (*committed transaction*) or an abort a_i (*aborted transaction*). The commit persists all of T_i's writes and the abort invalidates them. With $r_i(x)$ and $w_i(x)$ we denote T_i's read and write operations for a data item x. $r_i(x_j)$ represents T_i's read on the latest modification of x performed by transaction T_j. $w_j(x_j)$ represents T_j's last update on x. If the data item value read or written is the $l-$th update of T_j on data item x, it is denoted by $r_i(x_{j.l})$ and $w_j(x_{j.l})$. By x_0 we represent an initial state of the item x. Finally, o_i denotes any operation of T_i. Hereafter, we present a formal definition of a transaction:

Definition 1 (Transaction)**.** *A transaction T_i for a set of operations o_i is a total order $<$ for which the following holds:*

- $c_i \in T_i$ *iff* $a_i \notin T_i$
- *If* $c_i \in T_i, \forall o_i \neq c_i \in T_i, o_i < c_i$
- *If* $a_i \in T_i, \forall o_i \neq a_i \in T_i, o_i < a_i$
- *Given* $o_1, o_2 \in T_i, o_1 < o_2 \vee o_2 < o_1$

If it is necessary to explicitly refer to a replicated system, a copy of data item x at node N_a is represented by x^a; while T_i^a denotes the subset of transaction T_i operations executed at N_a. The notation of read, write, commit, and abort operations is also extended in the same way. For example, $r_i^a(x_j)$ represents transaction T_i's read operation executed at node N_a over the last update on x_j performed by transaction T_j on node N_a.

2.3. Execution of transactions

Transaction executions are represented by their histories [7]. Relation $o_1 < o_2$ belongs to a history H if both operations o_1 and o_2 execute in this order and either belong to the same transaction or are conflicting (i.e., both operations access the same data item and at least one of them is a write). Therefore, two read operations from distinct transactions are not directly ordered in a history. More formally:

Definition 2 (History)**.** *A history H over a set of transactions $\mathcal{T} = \{T_1, ..., T_n\}$ is a partial order over a relation $<_H$ where:*

- *For every $T_i \in \mathcal{T}$ and every $o_k \in T_i$: $o_k \in H$.*
- *For every $T_i \in \mathcal{T}$ and every $o_1, o_2 \in T_i$: if $o_1 < o_2 \in T_i$, then $o_1 <_H o_2 \in H$.*
- *If $r_i(x_j) \in H$, then $w_j(x_j) \in H \wedge w_j(x_j) <_H r_i(x_j)$.*
- *For any two conflicting operations $o_i(x), o_j(x) \in H$: $o_i(x) <_H o_j(x) \vee o_j(x) <_H o_i(x)$.*

For the sake of simplicity and readability we use $<$ instead of $<_H$ except in cases of ambiguity. Hence, $o_i < o_j \in H$ is equivalent to $o_i <_H o_j \in H$.

In replicated systems, a transaction execution history over the replicated nodes is composed of all the local executions. Given a set of transactions $\mathcal{T}$, $\mathcal{T}^a$ is the subset of $\mathcal{T}$ executed in node N_a. Thus, if $T_i \in \mathcal{T}$, $T_i^a \in \mathcal{T}^a$. Hence, the *replicated history* is defined as follows:

Definition 3 (Replicated history)**.** *A replicated history H_r over a set of transactions $\mathcal{T}$ and a set of nodes $\mathcal{N}$ is a partial order with a relation $<_r$ where, for all $N_a \in \mathcal{N}$:*

- *For every $T_i^a \in \mathcal{T}^a$ and every $o_k^a \in T_i^a$: $o_k^a \in H_r$.*
- *For every $T_i^a \in \mathcal{T}$ and every $o_1^a, o_2^a \in T_i^a$: if $o_1^a < o_2^a \in T_i^a$, then $o_1^a <_r o_2^a \in H_r$.*
- *If $r_i^a(x_j) \in H_r$ then there exists $w_j^a(x_j) \in H_r$ such that $w_j^a(x_j) <_r r_i^a(x_j)$.*
- *For any two conflicting $o_i^a(x), o_j^a(x) \in H_r$: $o_i^a(x) <_r o_j^a(x) \in H_r \vee o_j^a(x) <_r o_i^a(x) \in H_r$.*

For the sake of readability we use $o_i^a < o_j^a \in H_r$ instead of $o_i^a <_r o_j^a \in H_r$. We use $o_i \in H_r$ to indicate that at least one node has executed operation o_i. $o_i < o_j \in H_r$ states that the operations are executed in a given order in at least one node and there is no other node where operations execute in different order (i.e., $\exists N_a \in \mathcal{N}$ such that $o_i^a < o_j^a \in H_r$ and $\nexists N_b \in \mathcal{N}$ such that $o_j^b < o_i^b \in H_r$). Finally, to show that any two conflicting operations $o_i \in T_i$ and $o_j \in T_j$, and $o_i < o_j \in H_r$, we use $T_i < T_j \in H_r$.

In this article we focus on *update everywhere* replication solutions that implement a ROWAA scheme. In those protocols, the read operations of a transaction only execute at the local node and write operations must be applied in all the database replicas. More formally:

Definition 4 (ROWA transaction)**.** *Given a transaction T_i and a node N_a, we define T_i^a as:*

- *T_i^a is a subset of T_i .*
- *If $T_i^a \neq \emptyset$ and $c_i \in T_i$, then $c_i^a \in T_i^a$.*
- *If $T_i^a \neq \emptyset$ and $a_i \in T_i$, then $a_i^a \in T_i^a$.*
- *If $r_i(x) \in T_i$, then $r_i^a(x) \in T_i^a$ iff T_i is local to N_a.*
- *If $o_1 < o_2 \in T_i$, and o_1^a, $o_2^a \in T_i^a$, then $o_1^a < o_2^a \in T_i^a$.*
- *If $c_i \in T_i$, then $\forall w_i(x) \in T_i$: $w_i^a(x) \in T_i^a$.*

3. Stand-alone systems

3.1. Concurrency control mechanisms

In stand-alone systems, isolation is managed by concurrency control protocols that usually rely on locks, versions, or both to manage concurrency. With a lock-based concurrency control [4], transaction operations acquire locks to access data items that block the operations of other transactions until the lock is released. A blocked operation can execute once the lock is released. There are two kinds of locks: read and write; and they can be used for different durations: long and short. Read locks only block write operations while write locks block both reads and writes. Long locks are released when the transaction finishes, short locks are released when the operation finishes. The isolation level provided depends on the locks used during transactions execution.

Version-based concurrency control mechanisms store multiple versions per item [7, 34]. A new version of a data item is created in every write operation, but it becomes definitive (i.e., visible to new transactions) only when the transaction commits. In order to commit, a validation test must be applied to abort transactions that violate isolation constraints. The isolation level is determined by phenomena forbidden by this test.

3.2. Isolation levels

Several isolation level classifications have been proposed in the literature [14, 4, 1]. The majority describe and classify different types of transaction phenomena and define isolation levels depending on the forbidden phenomena. ANSI [14] and Berenson et al. [4] are probably two of the most referenced classifications. However, the ANSI classification has been proven to be ambiguous and the classification in [4] is oriented to lock-based concurrency control mechanisms and cannot be directly applied to version-based protocols [1]. To avoid ambiguity and implementation dependency we rely on the specifications given in [2].

Adya et al. [2] analyse each of the isolation levels assuming that the same isolation level is granted to all transactions. The study is based on directed graphs that represent dependencies between transactions — *direct serialisation graphs* (DSG). For completeness, we discuss DSGs in the following section.

3.2.1. Direct serialisation graphs (DSG)

DSGs represent dependencies between transactions. Those dependencies are based on conflicts.

Definition 5 (DSG)**.** *Given a history H, DSG(H) is a directed graph containing one vertex per committed transaction in H and an edge from T_i to T_j if one of the following dependencies occurs*[2]*:*

- **Direct read-dependency***: T_j directly read-depends on T_i, denoted by $T_i \xrightarrow{wr} T_j$, if $r_j(x_i) \in H$.*
- **Direct write-dependency***: T_j directly write-depends on T_i, denoted by $T_i \xrightarrow{ww} T_j$, if $w_i(x_i), w_j(x_j) \in H$, $w_i(x_i) <_H w_j(x_j)$ and there is no $w_k(x_k) \in H$: $w_i(x_i) <_H w_k(x_k) <_H w_j(x_j)$.*
- **Direct anti-dependency***: T_j directly anti-depends on T_i, denoted by $T_i \xrightarrow{rw} T_j$, if $r_i(x_m), w_j(x_j) \in H \wedge r_i(x_m) <_H w_j(x_j)$ and there is no $w_k(x_k) \in H$: $r_i(x_m) <_H w_k(x_k) <_H w_j(x_j)$.*

T_j depends on T_i if T_j either directly read-depends or directly write-depends on T_i. We also say that T_j anti-depends on T_i when T_j directly anti-depends on T_i.

As an example, consider the following history:

$H_1 = r_i(x_0)w_i(x_i)r_i(y_0)c_iw_j(y_j)w_j(x_j)c_j$.

The example assumes that an initial transaction T_0 creates the original versions of all data items. The associated DSG is depicted in Figure 1. The execution history results in four dependencies and one anti-dependency.

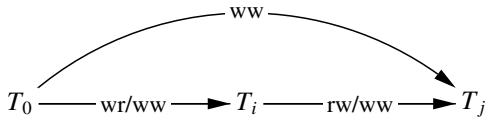

Figure 1: DSG of H_1

Using DSGs, the following isolation levels were defined: PL-1, PL-2 and PL-3 [1] which replace the conventional isolation levels *read uncommitted* (RU), *read committed* (RC) and *serialisable* (S), although they are not exactly the same. For example, PL-2 is weaker than RC [1]. Adya also defined the isolation level PL-SI, equivalent to the original SI [4], using *start-ordered serialisation graphs* (SSGs)[3] instead of DSG. We suggest a different definition that is based on

[2]We refer to the definitions given in [2] instead of those presented in [1].

[3]SSG contemplates start-dependencies, a new type of dependency edges.

DSG and can be easily included in the Adya mixing theory explained later in this section. To this end, the conditions assumed by start dependencies are explicitly stated in our SI condition (b), so avoiding the *G-SIa (interference)* phenomenon introduced by *Adya* [2], whilst our condition (c) complements (b) in order to avoid *G-SIb* phenomenon *(missed effects)*. As a result, our SI definition is equivalent to that of PL-SI in [2], since Adya's definition requires a PL-2 basis complemented with the avoidance of G-SIa and G-SIb phenomena in order to specify the PL-SI level.

The conditions demanded by these basic isolation levels are summarised below:

Definition 6 (Basic isolation levels)**.** *Given a history H:*

- **PL-1***: A history is PL-1 if $DSG(H)$ does not contain any cycles composed only of direct write-dependencies.*
- **PL-2***: A history is PL-2 if:*
  - *(a) A committed transaction does not read a value written by an aborted transaction (aborted value in the sequel)*
  - *(b) For every $r_i(x_{j.l}) \in H$, if T_j commits in H there is no $w_j(x_{j.m})$ with $l < m$ (intermediate value), and*
  - *(c) $DSG(H)$ does not have any cycle composed by dependency edges.*
- **SI***: A history is SI if:*
  - *(a) H is PL-2.*
  - *(b) If $T_j \xrightarrow{ww} T_i \in DSG(H) \vee T_j \xrightarrow{wr} T_i \in DSG(H)$, then $c_j < s_i$.*
  - *(c) If $T_j \xrightarrow{rw} T_i \in DSG(H)$, then $s_j < c_i$.*

  Note that, given an index k, the logical operation s_k is only an alias for the first operation of transaction T_k in H. It can be understood as the starting point of T_k.
- **PL-3***: A history is PL-3 if:*
  - *(a) H is PL-2.*
  - *(b) H does not have any cycle containing anti-dependency edges.*

3.3. Extended mixed serialisation graph (EMSG)

To analyse transactions that can be executed under different isolation levels we use a variation of DSGs: *extended mixed serialisation graphs* (EMSG) [5]. The following example explains why DSGs can not be used in such cases:

The DSG in Figure 2 depicts the dependencies between transactions T_i and T_j. The execution history is PL-1 if both T_i and T_j requested PL-1; if T_i and T_j requested PL-2 the given execution is not PL-2. If transaction T_i requested PL-1 and transaction T_j requested PL-2, the given DSG is insufficient to detect if the given execution history is correct. EMSGs are instead used to define correct execution histories if different isolation levels are used for different transactions.

EMSG [5] is an extension of the *mixed serialisation graph* (MSG) introduced by Adya [1]. MSGs represent dependencies among PL-1, PL-2 and PL-3 transactions in histories. Given a

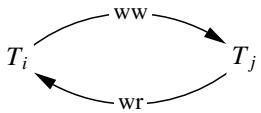

Figure 2: Example graph

history H, $MSG(H)$ is a subset of $DSG(H)$ including only the *obligatory* edges that depend on the isolation levels of the different transactions. EMSG extends MSG to also consider histories with SI transactions:

Definition 7 (EMSG[4])**.** *Given a history H, an $EMSG(H)$ has the same vertices as $DSG(H)$ and a subset of its edges known as obligatory edges. The edge $e \in DSG(H)$ from T_i to T_j is an obligatory edge in $EMSG(H)$ if either:*

- *e is a direct write-dependency edge, or*
- *e is a direct read-dependency edge and T_j is executed under PL-2, PL-3 or SI, or*
- *e is a direct anti-dependency edge and T_i is executed under PL-3.*

Therefore, EMSGs only include those edges that may violate at least one transaction isolation restriction. Adya's mixing-correct [1] definition can also be extended to identify valid execution histories with PL-1, PL-2, SI and PL-3 transactions:

Definition 8 (Valid history)**.** *A history H is valid if:*

(a) PL-2, PL-3 or SI transactions do not read aborted or intermediate values.

(b) $EMSG(H)$ does not contain any cycle.

(c) If $r_i(x_k) < w_j(x_j) \in H$ and T_i is SI, then $s_i < c_j \in H$.

(d) If $T_j \xrightarrow{ww} T_i \in EMSG(H)$ or $T_j \xrightarrow{wr} T_i \in EMSG(H)$ and T_i is SI, then $c_j < s_i \in H$.

(e) If $T_i \xrightarrow{ww} T_j \in EMSG(H_r)$ and T_i is SI, then $c_i < c_j \in H$.

Note that EMSGs should not contain any cycle. Since cycles with anti-dependency edges are not explicitly considered in our SI definition, these edges are not *obligatory* for SI transactions in Definition 7.

Thus, the execution history presented in Figure 2 is valid if transaction T_i is executed under PL-1 and transaction T_j is executed under PL-2. The history is also invalid if T_i requested PL-2 and T_j requested PL-1.

Definition 8 does not necessarily mean that concurrency control protocols should search for cycles in EMSGs. It only points out *what* should be considered to detect valid execution histories and not *how* that should be done.

[4]This definition is taken from [5]. Def. 8, Sect. 4 and the appendices extend, complement, and prove the preliminary results presented in [5].

Unfortunately, EMSGs and Definition 8 cannot be used in replicated environments. In replicated systems there is one transaction execution history per node and so Definition 8 cannot be applied directly because it does not account for dependencies between nodes. For example, two conflicting transactions can execute at two distinct nodes in different order but the local graphs at each node could still satisfy the conditions of Definition 8. In the following section, we extend the EMSGs and define an equivalent set of conditions suitable for replicated environments.

4. Extending EMSG to replicated environments

Firstly, we extend EMSGs to model replicated executions; then, in Section 4.2 we define when a replicated execution is equivalent to a given stand-alone execution; independently of the isolation level used by transactions. Finally, in Section 4.3 we identify the conditions under which the transaction history produced by a replication protocol is correct.

4.1. Extending EMSG to replicated systems

Although EMSGs can be used to separately model and evaluate the execution of each node, they cannot represent a transaction execution history of the whole replicated system. For example, assume two transactions T_i and T_j update the same data item x at nodes N_a and N_b but in a different order. The EMSGs representing the two executions are depicted in Figure 3.

Figure 3: Example: N_a and N_b EMSGs

Taken separately each EMSG presents a valid execution history, but unfortunately, the global execution is invalid. The cycle is depicted in Figure 4.

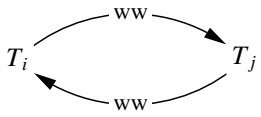

Figure 4: Example: global execution

We extended EMSG by combining local EMSGs into a single graph — a *replicated mixed serialisation graph* or RMSG. Some authors have proposed similar solutions but oriented to single isolation-level executions. Recently, *Lin et al.* [20] defined *union serialisation graphs* (USG) to present 1-copy-SI, a set of conditions to ensure valid SI-only executions. Previously, *Bernstein et al.* [7] used a similar approach to define 1-copy-serialisability.

Definition 9 (Replicated mixed serialisation graph (RMSG))**.** *Given a replicated history H_r over a set of transactions $\mathcal{T}$ and a set of local histories $\mathcal{H}$, $RMSG(H_r)$ is constructed as follows:*

- *Every committed transaction in $\mathcal{T}$ is a vertex in $RMSG(H_r)$.*
- *For every local history $H^a \in \mathcal{H}$, if $T_i^a \xrightarrow{ww} T_j^a \in EMSG(H^a)$, then $T_i \xrightarrow{ww} T_j \in RMSG(H_r)$. Similarly, if $T_i^a \xrightarrow{wr} T_j^a \in EMSG(H^a)$ (resp. $T_i^a \xrightarrow{rw} T_j^a \in EMSG(H^a)$), then $T_i \xrightarrow{wr} T_j \in RMSG(H_r)$ (resp. $T_i \xrightarrow{rw} T_j \in RMSG(H_r)$).*

4.2. Equivalence between replicated and stand-alone histories

To the best of our knowledge, all research on equivalence between transaction histories is based on a single isolation level. Conflict and view equivalences were defined for protocols that guarantee serialisability [7]; *SI-equivalence* was proposed in [21]. Hereafter we propose a new definition that is suitable for transaction executions where multiple isolation levels are supported.

Informally, a replicated history is equivalent to a stand-alone history if all the following conditions are held:

- *Uniform writes*: all replicas and the stand-alone system see the same sequence of updates on each database item (however, note that this does not imply that every replica sees exactly the same sequence of database states since updates on different items may be served in different orders in different replicas),
- *Uniform reads*: a read operation obtains the same value in the replica histories and in the stand-alone history,
- *Uniform isolation management*: all replicas and the stand-alone system implement isolation levels in the same way.

The above concepts are formalised as follows:

Definition 10 (Equivalence definition)**.** *Given a replicated history H_r and a stand-alone history H, H_r is equivalent to H if:*

C1: H and H_r execute the same set of transactions $\mathcal{T}$ and commit the same subset $\mathcal{T}_c \in \mathcal{T}$.

C2: For any committed transaction $T_i \in \mathcal{T}_c$, $r_i(x_j) \in H$ iff $r_i(x_j) \in H_r$.

C3: For every two transactions $T_i, T_j \in \mathcal{T}_c$, $w_i(x) < w_j(x) \in H$ iff $w_i(x) < w_j(x) \in H_r$.

C4: For every transaction $T_i \in \mathcal{T}_c$ executed under SI and any other transaction $T_j \in \mathcal{T}_c$:

(a) If $WS_j \cap WS_i \neq \emptyset$ then $c_j < s_i \in H$ iff $c_j < s_i \in H_r$
(b) If $WS_j \cap WS_i \neq \emptyset$ then $c_i < c_j \in H$ iff $c_i < c_j \in H_r$
(c) If $WS_j \cap RS_i \neq \emptyset$ then $c_j < s_i \in H$ iff $c_j < s_i \in H_r$.

An index k, in a replicated environment s_k, represents the starting point of T_k. Thus, s_k is an alias of the first T_k operation executed in its local node. For example, if N_a is the local node of T_k and o_1 is its first operation (i.e., $\nexists o_i \in T_k$ for which $o_i < o_1 \in T_k$), then $s_k = o_1^a$. Thus, $c_i < s_k \in H_r$ means that $c_i^a < o_1^a \in H_r$.

C1, C2 and C3 are taken from *Bernstein et al.* [7] equivalence definition and adapted to our context. C1 ensures that H and H_r are over the same sets of transactions and committed transactions. C2 guarantees that every read sees the same value in H_r and in H. C3 ensures that both H and H_r see the same sequence of states on every database item. Finally, C4 shows that H and H_r resolve snapshot-read and snapshot-write SI restrictions [1] in the same way.

4.3. Replication protocol correctness

Definition 11 extends Definition 8 and defines when a replicated history can be considered valid.

Definition 11 (Valid replicated history)**.** *A replicated history H_r is considered valid if the following conditions hold:*

(a) Transactions executing under PL-2, PL-3 or SI isolation levels never read aborted or intermediate data at any replica.

(b) $RMSG(H_r)$ does not contain any cycle.

(c) If $r_i(x_k) < w_j(x_j) \in H_r$ and T_i executes under SI, then $s_i < c_j \in H_r$.

(d) If $T_j \xrightarrow{ww} T_i \in RMSG(H_r)$ or $T_j \xrightarrow{wr} T_i \in RMSG(H_r)$ and T_i executes under SI, then $c_j < s_i \in H_r$.

(e) If $T_i \xrightarrow{ww} T_j \in RMSG(H_r)$ and T_i executes under SI, then $c_i < c_j \in H_r$.

Definition 12 states when a valid replicated history H_r should be considered correct.

Definition 12 (Correct replicated history)**.** *A valid replicated history H_r is correct if there is an equivalent valid history H.*

Theorem 1 says that all valid replicated histories are correct.

Theorem 1 (Correctness)**.** *Every valid replicated history H_r has an equivalent valid history H. Therefore, every valid H_r is correct.*

A correctness proof and a set of interesting properties can be found in the appendices.

5. Supporting multiple isolation levels in replication protocols

Theorem 1 meets one of the main goals of this work by showing when a replication protocol that allows transaction executions under several isolation levels, produces a correct execution history. We show below how some popular replication solutions can be extended to support different isolation levels.

5.1. Protocol classification

There have been several attempts in the past to classify replication protocols [12, 36, 35, 9]. In this paper we focus on the classification presented in [36]. Different protocol types are identified based on three parameters (server architecture, server interaction and transaction termination), with each parameter having two possible alternatives. Replication protocols belonging to any of the eight possible categories should be able to provide multiple isolation levels. Unfortunately, there are subtle differences among the protocols in different classes, and providing a single schema to extend them to support different isolation levels is not a trivial task.

Based on the server architecture two protocol classes are distinguished: *primary copy* and *update everywhere*. Primary copy protocols use a primary database replica where all update transactions are executed and only read-only transactions may be executed in secondary replicas. Update everywhere protocols allow the execution of update transactions at any node. It is fairly

straightforward to provide multiple isolation levels in primary copy protocols, since the execution of update transactions is fully handled by local concurrency control mechanisms in a primary replica. Protocols that fall into linear interaction category (one of the alternatives for server interaction) result in high communication overheads among replicas, and are not considered in this work.

Therefore, only two of the original eight classes identified in [36] are further studied in this section: those based on update everywhere replication that require a constant number of messages to complete the transaction.

5.2. Protocol implementation choices

There are many update everywhere replication protocols with constant server interaction proposed in the literature [3, 27, 18, 28, 21]; many of which share the same characteristics:

- Transactions can be submitted and executed in any replica. There is no dedicated replica that centralises transaction management.
- For any transaction a constant number of messages is exchanged among replicas. These messages usually propagate transaction updates. Although other solutions are possible, we limit our discussion to protocols that broadcast transaction data at the end of each transaction, i.e., when the transaction requests a commit locally.
- Write-set (and, in some cases, read-set [26]) dissemination is achieved using atomic broadcast primitives. This ensures that all replicas see the same sequence of updates, i.e., the same sequence of transactions.
- Local transactions execute under the requested isolation level provided by the underlying DBMS. The replication middleware must ensure that the global execution history, that combines both local and remote transactions, is correct.

Depending on transaction termination, two conflict resolution schemes are possible [36]: voting and certification. Protocols based on voting can be further divided into two categories: those that are symmetric and require a vote from every replica, and those that rely on the decision of a single replica (*weak voting replication* [35]). In the following, we will focus only on the weak voting approach. The communication costs imposed by symmetric voting are equivalent to 2PC [36]. Certification-based replication is quite similar to weak voting and only differs on how transaction reads are propagated for PL-3 protocols. Certification-based PL-3 solutions require both the read-set and the write-set for the validation step at every replica. Weak voting PL-3 protocols validate reads only at the local replica which later broadcasts the decision to the rest of the nodes. For SI protocols no read-set or extra message propagation is necessary [17] and so there are no substantial differences between both replication schemes.

5.3. A generic scheme to provide multiple isolation levels in replication

Our generic scheme is based on the principles presented in [6]. These principles are general enough to be applied to any transaction termination approach (i.e., voting and certification).

A database replication protocol based on weak voting or certification consists of the following steps [35]:

1. *Start*: when a local node N_i receives a transaction T_i from a client C_i and executes the transaction locally.

2. *Broadcast*: when client C_i requests the transaction commit, the transaction write-set (and maybe the read-set) is propagated to all replicas using atomic broadcast. Note that if a transaction has an empty write-set (i.e., it is a read-only transaction) no broadcast is needed and it can commit immediately.
3. *Validation*: on delivery of such a broadcast message, each replica checks if any conflicting transaction has been executed concurrently and is already committed.
4. *Termination*: if a conflict is detected, transaction T_i is aborted. Otherwise, it is committed. Depending on the protocol, the local replica N_i may use a reliable broadcast to propagate the commit or abort decision to other replicas.

To support multiple isolation levels in such a sample protocol, we apply our generic scheme and extend the protocol steps as follows:

a) The *start* step is extended to request the appropriate isolation level from the DBMS.
b) The *broadcast* step is extended to include the isolation level of the transaction in the broadcast message.
c) The *validation* step is updated to use the appropriate conflict detection rules for each isolation level.

This last topic deserves further explanation and is discussed in the following section.

6. Conflict resolution

In this section we modify the *start*, *broadcast* and *validation* steps of a protocol that manages PL-3 to also support PL-1, PL-2 and SI isolation levels. For the sake of simplicity, we focus on certification-based techniques. We later explain adjustments that can be made if weak voting is used. We also show how Theorem 1 can be used to prove the correctness of the new scheme.

Multiple isolation level scheme or MLS:

1. *Start*: when a replica N_i receives a transaction T_i from a client C_i:
   - The number of committed update transactions is taken as T_i's start-timestamp st_i.
   - T_i is started at the underlying DBMS with the requested isolation level.
   - T_i's operations are executed until commit is requested[5]. In PL-2, PL-3 and SI transactions, reads are not allowed to obtain values written by non-committed transactions. In lock-based concurrency control protocols this is achieved by using long write locks for writes (released only at transaction commit or abort) and at least short read locks for reads (released at the end of operation). In version-based protocols, the same is ensured by obtaining the latest version confirmed by committed transactions. For PL-1 transactions, read dependencies are not obligatory and so there are no restrictions on reads.
2. *Broadcast*: when C_i requests to commit T_i:
   - T_i's writes are collected into WS_i. For PL-3 transactions, reads are also gathered into RS_i.

[5] Some protocols [17] delay writes until the termination step.

- If $WS_i = \emptyset$, T_i is committed.
- If $WS_i \neq \emptyset$, WS_i, RS_i (only for PL-3 transactions), T_i's isolation level and st_i are atomically broadcast to all replicas.

3. *Validation*: when WS_i is delivered at a replica N_a:
   - The number of committed update transactions is taken as T_i's commit-timestamp ct_i.
   - If T_i is PL-3, WS_i is discarded if there is a T_j, that has already been validated, $RS_i \cap WS_j \neq \emptyset$ and $st_i < ct_j$.
   - If T_i is SI, WS_i is discarded if there is a T_j, that has already been validated, $WS_i \cap WS_j \neq \emptyset$ and $st_i < ct_j$.
   - If there are any other PL-3 T_j transaction that is local to node N_a, and has not reached the validation yet and WS_i contains an x item read by T_j, then T_j is aborted and the respective client C_j is informed.
4. *Termination*: if T_i terminates the validation step in N_a:
   - If $N_i = N_a$, T_i is aborted if WS_i is discarded during the validation step. Otherwise, it is committed and the updates are persisted. Notice that local transaction writes are persisted in committing order, not in execution order. Fortunately, all known concurrency control mechanisms ensure such a restriction. The result is sent to C_i.
   - If T_i passes the validation step and $N_i \neq N_a$, WS_i is applied. The implementation of the protocol must ensure that WS_i is not aborted by the replica's DBMS.

Notice that validation and termination steps are executed as a single atomic step. To increase the performance of the system, some existing protocols [17, 21] execute validation and termination separately, allowing validation even if the previous write-sets are not yet applied. However, the authors of [31] revealed that this improvement can produce some undesirable effects (for example, the abort of correct transactions).

Below we first prove that all replicas apply write-sets in the same order, and then we show that the conditions of Definition 11 are satisfied.

Lemma 1 (MLS applies write-sets in the same order)**.** *Given a replicated history H_r over a set of transactions $\mathcal{T}$ and a set of nodes $\mathcal{N}$.*

- *for any two replicas $N_a, N_b \in \mathcal{N}$ that apply WS_i and WS_j, they apply WS_i and WS_j in the same order.*
- *if a replica $N_a \in \mathcal{N}$ applies WS_i then every other replica $N_b \in \mathcal{N}$ applies WS_i.*

Proof. Write-sets are propagated using an atomic broadcast and, hence, are delivered to all active replicas in the same order. Validation and termination are executed in a single atomic step and so all the write-sets are applied in order of their delivery, and this fact proves the first part of Lemma 1.

We prove the second part of the Lemma by induction. Assume WS_i to be validated. If no update transaction has been previously delivered and committed, T_i validates and commits at N_a and N_b. Assume now that the same set $\mathcal{U}$ of update transactions has committed previously at N_a and N_b. If T_i requests a PL-1 or PL-2 isolation level it is directly validated. If T_i is SI, then the validation result will be different in both replicas if there is a committed transaction T_j at N_a for

which $WS_i \cap WS_j \neq \emptyset$ and $st_i < ct_j$; but this does not happen at N_b. Since T_j commits before T_i at N_a and it is an update transaction, $T_j \in \mathcal{U}$. Since $\mathcal{U}$ has been applied also at N_b, from the first part of the lemma, T_j commits at N_b before the T_i validation and so the only possible difference can be in how ct_j and st_i are ordered.

st_i is calculated at the local replica when the transaction starts and is propagated in the broadcast message. Hence, all replicas share the same value. ct_j is calculated at every replica as the number of previously committed transactions when T_j is validated. Since we assume all nodes have applied $\mathcal{U}$ and, from the first part of the lemma, in the same order, ct_i must take the same value at N_a and N_b. Thus, if st_i and ct_j have the same value at N_a and N_b, and WS_j is applied in both before T_i is validated, then the validation must produce the same result, i.e., T_i is accepted or aborted at both replicas. Notice that the same proof can be applied for PL-3 transactions since the only difference is that RS_i is used instead of WS_i. However, in both cases, it is compared against previously applied write-sets. Therefore, T_i never obtains a different validation result at N_a and N_b and this proves the Lemma's second assertion.

□

Theorem 2 (MLS protocol correctness)**.** *Any possible history H_r that can be produced by the MLS protocol is correct.*

Proof. To begin with, we prove each condition of Definition 11 separately:

- **Condition (a)**. MLS is based on the assumption that every DBMS locally ensures the isolation levels requested globally by replication protocols. Thus, the local DBMS never allows PL-2, SI or PL-3 transactions to read any aborted or intermediate value.

- **Condition (b)**. This condition forbids cycles in $RMSG(H_r)$. We can prove this by defining a function f over H_r in such a way that for every two committed transactions $T_i, T_j \in H_r$ and an edge $e \in RMSG(H_r)$ from T_i to T_j, $f(T_i) < f(T_j)$. If a function such as this can be defined in MLS, a cycle $T_1 \rightarrow T_2 \rightarrow ... \rightarrow T_n \rightarrow T_1$ is impossible because, otherwise, $f(T_1) < f(T_2) < ... < f(T_n) < f(T_1)$, which is a contradiction.

  To define f we use transaction event ordering. Thus, given a transaction T_i, $f(T_i) = c_i$ (i.e., the position in H_r of the transaction commit event). We should prove that for any edge $e \in RMSG(H_r)$ from T_i to T_j, $c_i <_r c_j \in H_r$ ($c_i < c_j \in H_r$ in the sequel).

  As shown in Definition 9, the dependency edges included in RMSG depend on the isolation levels requested by their vertices. Given a committed transaction T_i and an edge e involving T_i and another transaction T_j, e is in $RMSG(H_r)$ if:

  - e is $T_j \xrightarrow{ww} T_i$.
  - e is $T_j \xrightarrow{wr} T_i$ and T_i is PL-2, PL-3 or SI.
  - e is $T_i \xrightarrow{rw} T_j$ and T_i is PL-3.

  Lemma 1 shows that write-sets are applied in the same order at every node and, hence, if $e = T_i \xrightarrow{ww} T_j$ then $c_i < c_j$ at all replicas.

  If $e = T_i \xrightarrow{wr} T_j$ then T_j has read a value written by T_i at its local node N_a. Thus, the read has been executed after WS_i is applied at N_a and hence, $c_i^a < c_j^a$. If T_j is read-only then it does not propagate any write-set and $c_i < c_j$ is the global order. Otherwise, Lemma 1 ensures that WS_j is delivered after WS_i at all nodes and so $c_i < c_j$ at all replicas.

If $e = T_i \xrightarrow{rw} T_j$ then T_i is PL-3. Hence, $r_i(x_k) < w_j(x_j)$ (otherwise x_j value would be read) and $s_i < c_j$ at the local replica of T_i. If T_i is validated at this replica after T_j then PL-3 validation step aborts T_i because $st_i < ct_j$ and $RS_i \cap WS_j \neq \emptyset$. Hence, if $e = T_i \xrightarrow{rw} T_j$ then T_i must be validated and applied before T_j, which means $ct_i \leq ct_j$ (if T_i is read-only ct_i may be equal to ct_j). Since the commit-timestamp is the number of committed update transactions when the transaction commits and T_j is an update transaction, if $ct_i \leq ct_j$ then $c_i < c_j$. If T_i is a read-only transaction that order can be taken as the global order. Otherwise, atomic broadcast ensures that $c_i < c_j$ at all system nodes.

To sum up, for any edge $e \in RMSG(H_r)$, if e connects T_i to T_j then $f_c(T_i) = c_i < c_j = f_c(T_j)$ and, hence, cycles are impossible.

- **Condition (c)**. This condition is held if for every SI transaction T_i, if $r_i(x_k) < w_j(x_j)$ then $s_i < c_j$. $s_i < c_j$ in a replicated history H_r if $\nexists N_b$ for which $c_j < s_i \in H^b$ (Section 2.3). Assume N_a as T_i local replica. Since T_i reads are executed only at N_a and local DBMSs support SI locally, $s_i^a < c_j^a \in N_a$. st_i^a is calculated only at the local replica and, from Lemma 1, ct_j obtains the same value at all replicas because the same sequence of write-sets is observed in them. Hence, if $st_i^a \leq ct_j^a \in N_a$ (T_i may be a read-only transaction) then $st_i \leq ct_j$ in the entire system, and this also implies that $s_i < c_j$.

- **Condition (d)**. In this condition, if $T_j \xrightarrow{ww} T_i \in RMSG(H_r)$ or $T_j \xrightarrow{wr} T_i \in RMSG(H_r)$ and T_i is SI then $c_j < s_i$. Assume N_a as T_i's local replica. Since N_a's local DBMS ensures SI for T_i, $c_j^a < s_i^a$ and $ct_j^a < st_i^a$. From Lemma 1, ct_j is the same at all system replicas. Hence, if $c_j^a < s_i^a \in N_i$, then $c_j < s_i$ for the entire system.

- **Condition (e)**. In this condition, if $T_i \xrightarrow{ww} T_j$ and T_i is SI then $c_i < c_j$. By Lemma 1, this property is trivially ensured in MLS protocols.

As a result, every replicated history H_r generated by the MLS protocol is valid. Hence, Theorem 1 is applicable to H_r and the valid replicated history is correct. □

6.1. MLS and weak-voting

Below we explain how MLS can be modified if weak-voting is used instead of certification. In weak voting, read-sets (RSs) are not included in the update (or write-set, WS) propagation message. Thus, if reads are necessary during the validation step, only the local replica can validate a transaction. In this case, the result must be propagated and updates will not be applied until that validation message is delivered. Luckily, reads are only needed to validate PL-3 transactions while the others can be deterministically validated in the same way as with certification. We detail below the necessary changes to the original scheme:

1. *Broadcast*: when C_i requests to commit T_i:
   - T_i's writes are gathered into WS_i.
   - If $WS_i \neq \emptyset$, WS_i, T_i isolation level and st_i are atomically broadcast to all replicas.
2. *Validation*: when WS_i is delivered at a replica N_a:
   - If T_i is PL-3 and $N_a = N_i$, WS_i is discarded if T_j exists and has been previously validated, $RS_i \cap WS_j \neq \emptyset$ and $st_i < ct_j$. The result is propagated.
   - If T_i is PL-3 and $N_a \neq N_i$, wait for the N_a validation result.

7. Example

t.start ← count	1	t.start ← count
Execute t.	2	Execute t.
On t commit request:	3	On t commit request:
ws.data ← wset(t)	4	ws.data ← wset(t)
ws.start ← t.start	5	ws.start ← t.start
ws.local ← N_a	6	ws.local ← N_a
	7	ws.level ← t.level
	8	if (t.level = PL-3)
	9	rs.data ← rset(t)
	10	else
rs.data ← rset(t)	11	rs.data ← 0
TO-bcast(N, ⟨rs, ws⟩)	12	TO-bcast(N, ⟨rs, ws⟩)
Upon ⟨rs, ws⟩ reception:	13	Upon ⟨rs, ws⟩ reception:
mutex.lock	14	mutex.lock
status$_t$ ← certify(rs, ws, wslist$_a$)	15	status$_t$ ← certify(rs, ws, wslist$_a$)
if (status$_t$ = commit) then	16	if (status$_t$ = commit) then
count++	17	count++
ws.commit ← count;	18	ws.commit ← count;
append(wslist$_a$, ws)	19	append(wslist$_a$, ws)
if (ws.local ≠ N_a) then	20	if (ws.local ≠ N_a) then
DB.apply(ws)	21	DB.apply(ws)
status$_t$ ← DB.commit(t)	22	status$_t$ ← DB.commit(t)
if (status$_t$ = abort) then	23	if (status$_t$ = abort) then
remove(wslist$_a$, ws)	24	remove(wslist$_a$, ws)
else DB.abort(t)	25	else DB.abort(t)
mutex.unlock	26	mutex.unlock
if (ws.local = N_a) then	27	if (ws.local = N_a) then
send(c, status$_t$)	28	send(c, status$_t$)
certify(rs, ws, wslist$_a$):	29	certify(rs, ws, wslist$_a$):
for (old_ws ∈ wslist$_a$) do	30	for (old_ws ∈ wslist$_a$) do
	31	if (level = PL-3) and
(ws.start ⟨ old_ws.commit) and	32	(ws.start ⟨ old_ws.commit) and
(rs ∩ old_ws ≠ ∅) and	33	(rs ∩ old_ws ≠ ∅) and
(ws.local ≠ old_ws.local) then	34	(ws.local ≠ old_ws.local) then
return abort	35	return abort
	36	else if (level = SI) and
	37	(ws.start ⟨ old_ws.commit) and
	38	(ws ∩ old_ws ≠ ∅) then
	39	return abort
return commit	40	return commit

a) SER CBR protocol. b) MUL CBR protocol.

Figure 5: SER and MUL certification-based protocols.

In this section we show how the changes suggested in Section 6 can be applied to an existing protocol. We have taken SER CBR, a variation of the SI CBR protocol from [31] (also an adaptation from [18, 28]), which supports only PL-3, and extended it to also support SI, PL-2 and PL-1. We call the new protocol *multiple isolation level certification-based replication protocol* or MUL CBR. SER CBR and SI CBR are combinations and variations of other well known protocols [17, 11, 21, 9] and the changes suggested in this section can be easily exported to those protocols. SER CBR and MUL CBR are presented in Figure 5.

SER CBR needs transaction read-sets in order to validate transactions (see its `certify`

method). The first difference revealed by MUL CBR is that it includes transaction isolation level as a part of the broadcast message (line 7). The second difference is that read-set propagation is used only for PL-3 transactions (lines 8-11). In contrast to SER CBR, the `certify` method of MUL CBR validates transactions depending on their isolation level (lines 31-35). PL-2 and PL-1 transactions do not require validation: atomic broadcast together with local DBMS concurrency control are sufficient to forbid the phenomena. SI transactions only require checks for conflicts between writes. Thus, reads are only involved in PL-3 transactions validation and, hence, only in those cases where the local replica decision must be propagated. Notice that only a few changes are required to the original protocol.

8. Conclusions

We have addressed the problem of supporting multiple isolation levels in existing ROWAA replication protocols. The authors of [8] identified this problem as one of the challenges in database replication. Most existing replication solutions support a single isolation level which is one generally of the strictest: serialisability or snapshot isolation.

In this article we have identified the conditions under which replication protocols may manage multiple isolation levels transparently and proven that protocols that satisfy these conditions are correct. We have then modified the popular ROWAA-based replication scheme to support different isolation levels. As an example, we have further demonstrated how these extensions can be applied to a specific protocol. The majority of the replication solutions under consideration require only minor changes to support multiple isolation levels, which may result in an improved degree of concurrency and minor transaction completion times for those transactions that can be executed in a relaxed isolation level.

Our model is general enough to be applied to any database replication protocol. As a possible future work we plan to use this model in order to prove the correctness of several existing metaprotocols (e.g., [30]) that concurrently support several replication protocols, each being able to support a different isolation level.

A. Appendices

The aim of these appendices is to provide the necessary basis for proving the correctness of Theorem 1 (given in Section A.2). To this end, Section A.1 presents several properties derived from Def. 11 that will provide lemmas needed in Section A.2.

A.1. Properties of valid replicated histories

A valid replicated history H_r provides a set of useful properties that are deduced from Def. 11. For example, given any pair of conflicting operations o_i, o_j, if $\exists N_a$ for which $o_i^a < o_j^a \in H^a$ then $o_i < o_j \in H_r$.

Lemma 2 (Global order of conflicting operations)**.** *Given a valid replicated history H_r over a set of transactions $\mathcal{T}$, for any two conflicting operations o_i, o_j of committed transactions $T_i, T_j \in \mathcal{T}$, $o_i < o_j \in H_r \vee o_j < o_i \in H_r$.*

Proof. Recall that $o_i < o_j \in H_r$ if $\exists N_a$ for which $o_i^a < o_j^a \in H^a$ but $\nexists N_b$ for which $o_j^b < o_i^b \in H^b$. Since we follow ROWAA protocols, if one operation is a read then the conflict appears only in that specific read local replica and, hence, it is impossible to have them ordered backwards

in any other node. If both are writes, then they are executed in all replicas in the system and in the same order because otherwise there would be a cycle in the $RMSG$. As an example, assume $w_i^a(x) < w_j^a(x) \in H^a$ but $w_j^b(x) < w_i^b(x) \in H^b$. Write-dependencies are always obligatory edges, so, $T_i \xrightarrow{ww} T_j \in EMSG(H^a)$ and $T_j \xrightarrow{ww} T_i \in EMSG(H^b)$. Since $RMSG(H_r)$ is the union of all local $EMSG$, from H^a we obtain $T_i \xrightarrow{ww} T_j \in RMSG(H_r)$ and from H^b we obtain $T_j \xrightarrow{ww} T_i \in RMSG(H_r)$, which close the cycle, and contradict the absence of cycles in a valid H_r.

□

Another interesting property says that transactions are ordered due to their writes.

Lemma 3 (Write-write order of transactions)**.** *Given a valid replicated history H_r over a set of transactions $\mathcal{T}$, $\forall T_i, T_j \in \mathcal{T}$, if $w_i(x) < w_j(x) \in H_r$ then $\nexists w_i(y), w_j(y) \in H_r$ for which $w_j(y) < w_i(y) \in H_r$.*

We also show that $WS_i < WS_j \in H_r$ (WS_i represents the set of writes performed by T_i).

Proof. The correctness proof is similar to the one in Lemma 2. If $w_i(x) < w_j(x) \in H_r$ then $T_i \xrightarrow{ww} T_j$ in $RMSG(H_r)$. In the same way, if $w_j(y) < w_i(y) \in H_r$ then $T_j \xrightarrow{ww} T_i$ in $RMSG(H_r)$. If both edges appear at the same time, we have a cycle in $RMSG(H_r)$ involving T_i and T_j. However, cycles are forbidden in H_r. □

When SI transactions come into play we can still extract some other interesting properties. For example:

Lemma 4 (SI transaction write-write ordering)**.** *Given a valid replicated history H_r over a set of transactions $\mathcal{T}$, $\forall T_i, T_j \in \mathcal{T}$, if T_i is SI and $WS_i < WS_j \in H_r$ then $c_i < c_j \in H_r$. However, if $WS_j < WS_i \in H_r$ then $c_j < s_i \in H_r$.*

Proof. This lemma is directly proven by applying Lemma 3 to the conditions (d) and (e) of Def. 11. □

Lemma 5 (Read-write order of SI transactions)**.** *Given a valid replicated history H_r over a set of transactions $\mathcal{T}$, $\forall T_i, T_j \in \mathcal{T}$, if T_i is SI then never $r_i(x_k) < w_j(x_j) \in H_r \wedge r_i(y_j) \in H_r$.*

We also show that as $RS_i < WS_j \in H_r$ if $r_i(x_k) < w_j(x_j) \in H_r$ or $WS_j < RS_j \in H_r$ if $r_i(y_j) \in H_r$.

Proof. From $r_i(x_k) < w_j(x_j)$ and Definition 11 condition (c) we obtain that $s_i < c_j \in H_r$ but from read-dependency $r_i(y_j)$ and condition (d) we obtain that $c_j < s_i$ and it is impossible to have both at the same time. □

Notice that Lemmas 4 and 5 are equivalent to *snapshot write* and *snapshot read* conditions of Adya's *snapshot isolation* definition [1].

A.2. Correctness proof of Theorem 1

Hereafter, we prove the correctness of Theorem 1; i.e. if H_r is valid (see Definition 11), then there is a valid and equivalent H. To this end, we firstly suggest a methodology to extract a stand-alone history H from the RMSG of a replicated history H_r. We then show that H is equivalent to H_r and fulfils the conditions of Definition 8. Theorem 1 is therefore proven.

Definition 13 (Replicated history projection)**.** *Given a valid replicated history H_r, we can construct a stand-alone history H by applying the following steps:*

1. *If operation $o \in H_r$ then $o \in H$.*
2. *$\forall T_i \in \mathcal{T}$ and $\forall o_1, o_2 \in T_i$, if $o_1 < o_2 \in T_i$ then $o_1 < o_2 \in H$.*
3. *$\forall o_i, o_j \in H_r$, if $o_i < o_j \in H_r$ then $o_i < o_j \in H$.*
4. *If $w_j(x) < w_i(x) \in H_r$, T_i and T_j commit and T_i is SI then $c_j < s_i \in H$.*
5. *If $w_i(x) < w_j(x) \in H_r$, T_i and T_j commit and T_i is SI then $c_i < c_j \in H$.*
6. *If $r_i(x_j) \in H_r$, T_i, T_j commit and T_i is SI then $c_j < s_i \in H$.*
7. *If $r_i(x_k) < w_j(x_j) \in H_r$, T_i, T_j commit and T_i is SI then $s_i < c_j \in H$.*

As previously remarked, this process assumes that H_r is valid. Otherwise, some steps would be impossible to accomplish. For example, if two committed transactions execute conflicting writes over all the replicas but in a different order, it is impossible to decide how they are ordered in H. We also assume a *read one write all available* (ROWAA) scheme (i.e., writes are executed everywhere but reads only in the local replica). If this condition is not accomplished, it is not possible to decide which value has been read by an operation in H unless a new condition is added to ensure that the same value is obtained in all the replicas in which the operation is executed.

With the previous steps we produce a stand-alone history H, but the question is whether it is equivalent to H_r.

Lemma 6 (A replicated history and its projection are equivalent)**.** *Given a valid replicated history H_r and its projection H constructed as shown in Def. 13, H_r and H are equivalent.*

Proof. This can be proven by showing that all equivalence conditions are held.

C1: By construction of H (steps 1 and 2), both H and H_r execute the same operations and, hence, the same transactions commit.

C2: Step 3 ensures that H reads the same values as H_r. Recall that reads are executed in just one replica and, hence, the same operation never obtains different values in different nodes.

C3: From Lemma 2, all replicas execute conflicting writes (of committed transactions) in the same order. This order is preserved in H by step 3.

C4(a): If $WS_i \cap WS_j \neq \emptyset$ then, from Lemma 3, either $WS_i < WS_j \in H_r \vee WS_j < WS_i \in H_r$. From Lemma 4, if T_i is SI then either $c_i < c_j \in H_r \vee c_j < s_i \in H_r$. Since $s_i < c_i$ then either $s_i < c_j \in H_r \vee c_j < s_i \in H_r$. Steps 4 and 5 ensure the same ordering in H.

C4(b): From C4(a) proof, if $WS_i \cap WS_j \neq \emptyset$ then either $c_i < c_j \in H_r \vee c_j < s_i \in H_r$. Since $s_i < c_i$ we can also deduce that either $c_i < c_j \in H_r \vee c_j < c_i \in H_r$. Steps 4 and 5 ensure the same ordering in H.

C4(c): If $WS_j \cap RS_i \neq \emptyset$ then, from Lemma 5, either $WS_j < RS_i \in H_r \vee RS_i < WS_j \in H_r$. If T_i is SI then we can deduce that either $c_j < s_i$ (if $WS_j < RS_i$) or $s_i < c_j$. Steps 6 and 7 ensure the same ordering in H.

□

Therefore, we have proved that given any valid H_r we can construct an equivalent stand-alone history H. However, is H valid?

Lemma 7 (A replicated history projection is valid)**.** *Given a valid replicated history H_r and its projection H constructed as shown in Def. 13, H is valid.*

Proof. It is valid if the conditions (a), (b), (c), (d) and (e) of Def. 8 are held:

- **Condition (a)**: That condition is trivially held since reads obtain the same value in H and H_r which is supposed to be valid and, hence, reads never obtain aborted or intermediate values.
- **Condition (b)**: The form in which H is constructed ensures that it contains all H_r dependencies and hence, $EMSG(H)$ has all $RMSG(H_r)$ edges. However, is it possible for H to show dependencies not present in H_r? Since H and H_r are over the same operations, if o_i, o_j conflict in H then both also conflict in H_r. By Lemma 2, only $o_i < o_j \in H$ but not in H_r if $o_j < o_i \in H_r$. However, by construction of H, if $o_j < o_i \in H_r$ hence $o_j < o_i \in H$ and this contradicts the initial assumption. Thus, H and H_r show the same dependencies and hence $EMSG(H)$ and $RMSG(H_r)$ have the same vertices, edges and cycles. Since there are no cycles in $RMSG(H_r)$ then there are no cycles in $EMSG(H)$.
- **Conditions (c), (d) and (e)**: We have proven for condition (b) that H and H_r show the same dependencies. Because of steps 4, 5, 6 and 7, dependencies involving SI transactions produce the same orderings between commits and starts. Therefore, since H_r is valid then H must also be valid.

□

Therefore, given any valid replicated history H_r, we have proven that it is possible to construct a valid and equivalent stand-alone history H. As a result, H_r is correct and Theorem 1 is proven.

References

[1] A. Adya. *Weak Consistency: A Generalized Theory and Optimistic Implementations for Distributed Transactions*. PhD thesis, Massachusetts Institute of Technology, Mar. 1999.

[2] A. Adya, B. Liskov, and P. O'Neil. Generalized isolation level definitions. In *IEEE Intnl. Conf. on Data Engineering*, pages 67–78, San Diego, CA, USA, Mar. 2000.

[3] D. Agrawal, G. Alonso, A. El Abbadi, and I. Stanoi. Exploiting atomic broadcast in replicated databases (extended abstract). In *3rd Intnl. Euro-Par Conf.*, volume 1300 of *Lect. Notes Comput. Sc.*, pages 496–503, Passau, Germany, 1997. Springer.

[4] H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O'Neil, and P. O'Neil. A critique of ANSI SQL isolation levels. In *Intnl. Conf. on Manag. of Data (SIGMOD)*, pages 1–10, San José, CA, USA, May 1995. ACM Press.

[5] J. M. Bernabé-Gisbert and F. D. Muñoz-Escoí. Extending mixed serialisation graphs to replicated environments. In *3rd Intnl. Conf. on Avail., Reliab. and Security (ARES)*, pages 369–375, Barcelona, Spain, Mar. 2008. IEEE-CS Press.

[6] J. M. Bernabé-Gisbert, R. Salinas-Monteagudo, L. Irún-Briz, and F. D. Muñoz-Escoí. Managing multiple isolation levels in middleware database replication protocols. In *6th Intnl. Symp. on Paral. and Distrib. Proces. and Appl. (ISPA)*, volume 4330 of *Lect. Notes Comput. Sc.*, pages 511–523, Sorrento (Naples), Italy, Dec. 2006. Springer.

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

[8] E. Cecchet, G. Candea, and A. Ailamaki. Middleware-based database replication: The gaps between theory and practice. In *Intnl. Conf. on Manag. of Data (SIGMOD)*, Vancouver, Canada, June 2008. ACM Press.

[9] B. Charron-Bost, F. Pedone, and A. Schiper. *Replication: Theory and Practice*. Springer, 2010.

[10] A. El Abbadi and S. N. Dani. A dynamic accessibility protocol for replicated databases. *Data & Knowledge Engineering*, 6(4):319 – 332, 1991.

[11] S. Elnikety, F. Pedone, and W. Zwaenepoel. Database replication providing generalized snapshot isolation. In *24th Intnl. Symp. on Reliab. Distrib. Syst. (SRDS)*, pages 73–84, Orlando, FL, USA, Oct. 2005. IEEE-CS Press.

[12] J. Gray, P. Helland, P. E. O'Neil, and D. Shasha. The dangers of replication and a solution. In *Intnl. Conf. on Manag. Data (SIGMOD)*, pages 173–182, Montreal, Quebec, Canada, June 1996. ACM Press.

[13] V. Hadzilacos and S. Toueg. Fault-tolerant broadcasts and related problems. In S. Mullender, editor, *Distributed Systems*, chapter 5, pages 97–145. ACM Press, 2nd edition, 1993.

[14] INCITS 135-1992 (R1998). *Information Systems - Database Language - SQL*. ANSI, 1992.

[15] R. Jiménez-Peris, M. Patiño-Martínez, and G. Alonso. Non-intrusive, parallel recovery of replicated data. In *21st Intnl. Symp. on Reliab. Distrib. Syst. (SRDS)*, pages 150–159, Osaka, Japan, Oct. 2002. IEEE-CS Press.

[16] J. R. Juárez-Rodríguez, J. E. Armendáriz-Iñigo, J. R. González de Mendívil, F. D. Muñoz-Escoí, and J. R. Garitagoitia. Weak voting database replication protocols providing different isolation levels. In *7th Intnl. Conf. on New Techn. of Distrib. Syst. (NOTERE)*, pages 261–268, Marrakesh, Morocco, June 2007.

[17] B. Kemme. *Database Replication for Clusters of Workstations*. PhD thesis, Swiss Federal Institute of Technology, Zürich, Switzerland, 2000.

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

[19] B. Kemme, A. Bartoli, and Ö. Babaoglu. Online reconfiguration in replicated databases based on group communication. In *Intnl. Conf. on Depend. Syst. and Netw. (DSN)*, pages 117–130, Göteborg, Sweden, July 2001. IEEE-CS Press.

[20] Y. Lin, B. Kemme, R. Jiménez-Peris, M. Patiño-Martínez, and J. E. Armendáriz-Iñigo. Snapshot isolation and integrity constraints in replicated databases. *ACM Trans. Database Syst.*, 34(2), 2009.

[21] Y. Lin, B. Kemme, M. Patiño-Martínez, and R. Jiménez-Peris. Middleware based data replication providing snapshot isolation. In *Intnl. Conf. on Manag. of Data (SIGMOD)*, pages 419–430, 2005.

[22] J. MacCormick, C. A. Thekkath, M. Jager, K. Roomp, L. Zhou, and R. Peterson. Niobe: A practical replication protocol. *Trans. Storage*, 3(4):1:1–1:43, Feb. 2008.

[23] H. K. Mak and M. H. Wong. An experimental study of semantics-based concurrency control protocols. *Data & Knowledge Engineering*, 35(1):53 – 81, 2000.

[24] Microsoft Corp. SQL Server 2012. URL: http://www.microsoft.com/sqlserver/en/us/default.aspx, May 2012.

[25] Oracle Corp. Oracle Database 11*g* release 2. URL: http://www.oracle.com/us/products/database/index.html, May 2012.

[26] F. Pedone. *The Database State Machine and Group Communication Issues*. PhD thesis, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland, 1999.

[27] F. Pedone, R. Guerraoui, and A. Schiper. Exploiting atomic broadcast in replicated databases. In *4th Intnl. Euro-Par Conf.*, volume 1470 of *Lect. Notes Comput. Sc.*, pages 513–520, Southampton, UK, Sept. 1998. Springer.

[28] F. Pedone, R. Guerraoui, and A. Schiper. The database state machine approach. *Distrib. and Paral. Databases*, 14(1):71–98, 2003.

[29] PostgreSQL Global Development Group. PostgreSQL, the world's most advanced open source database. URL: http://www.postgresql.org/, May 2012.

[30] M. I. Ruiz-Fuertes, R. de Juan-Marín, J. Pla-Civera, F. Castro-Company, and F. D. Muñoz-Escoí. A metaprotocol outline for database replication adaptability. In *2nd Intnl. Wshop. on Reliab. in Decentr. Distrib. Syst. (RDDS)*, pages 1052–1061, Vilamoura, Portugal, Nov. 2007. Springer.

[31] M. I. Ruiz-Fuertes, F. D. Muñoz-Escoí, H. Decker, J. E. Armendáriz-Iñigo, and J. R. González de Mendívil. Integrity dangers in certification-based replication protocols. In *3rd Intnl. Wshop. on Reliab. in Decentr. Distrib. Syst. (RDDS)*, pages 924–933, Monterrey, Mexico, Nov. 2008. Springer.

[32] R. Salinas-Monteagudo, J. M. Bernabé-Gisbert, F. D. Muñoz-Escoí, J. E. Armendáriz-Iñigo, and J. R. González de Mendívil. SIRC: A multiple isolation level protocol for middleware-based data replication. In *22nd Intnl. Symp. on Comput. Inf. Sc. (ISCIS)*, Ankara, Turkey, Nov. 2007. IEEE-CS Press.

[33] Transaction Processing Performance Council (TPC). TPC benchmark C. Standard Specification, 2005.

[34] S. Verma, M. L. McAuliffe, S. Listgarten, S. Haldar, and C. K. Hoang. *Patent 7243088: Database management*

system with efficient version control. Oracle Intnl. Corp., July 2007.

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

[36] M. Wiesmann, A. Schiper, F. Pedone, B. Kemme, and G. Alonso. Database replication techniques: A three parameter classification. In *19th Intnl. Symp. on Reliab. Distrib. Syst. (SRDS)*, pages 206–215, Nürnberg, Germany, 2000. IEEE-CS Press.