Correctness Criteria for Replicated Database Systems with Snapshot Isolation Replicas

J.E. Armendáriz, J.R. Juárez, J.R. González de Mendívil

Universidad Pública de Navarra, Pamplona, Spain

{enrique.armendariz, jr.juarez, mendivil}@unavarra.es

Technical Report TR-ITI-ITE-08/03

Correctness Criteria for Replicated Database Systems with Snapshot Isolation Replicas

J.E. Armendáriz, J.R. Juárez, J.R. González de Mendívil

Universidad Pública de Navarra, Pamplona, Spain

Technical Report TR-ITI-ITE-08/03

e-mail: {enrique.armendariz, jr.juarez, mendivil}@unavarra.es

Abstract

Most replicated database systems where each replica uses a Snapshot Isolation (SI) DBMS implement replication protocols based on the deferred update technique. It allows read-only transactions to be locally executed at a single replica. Moreover, as read-only transactions are executed under SI they never get blocked and, therefore, the system performance is increased whenever the majority of the workload is of this type. In this work we present the correctness criteria which a replicated database system with SI replicas must verify when deferred update protocols are used in a crash failure scenario. We have formalized them using the Input/Output Automaton Model. Our criteria proposal ensures that the replicated database system behaves like a single copy database system where transactions see a weaker form of SI, called Generalized-SI. This formal characterization allows us to study the pros and cons of the deferred update technique and it serves as a milestone to prove the correctness of a replication protocol in a rigorous way.

1 Introduction

Several database replication techniques allow read-only transactions to be executed locally at a single replica without intervention of the replication protocol. These techniques are the preferred choice when workloads are mainly composed of read-only transactions. Their performance is even more significant if the local database management systems (DBMS) employed in the replicated system execute transactions under the Snapshot Isolation (SI) level [1], since read-only transactions are never blocked by the DBMS. Recently, a significant number of database replication protocols have been proposed assuming SI replicas [5, 6, 8, 15, 18, 20]. Primary copy protocols [5, 20] distribute read-only transactions among secondary replicas (which are not allowed to perform updates) in order to improve the system performance, although they must care about failures of the primary replica to keep the availability of the system. Centralized certification protocols [6, 7] execute update transactions at any replica but their commitment must be validated by a central certifier site (i.e. no other concurrent certified transaction updated a data item written by the incoming transaction). It is clear that the certifier replica is again a single point of failure. Thus, distributed certification protocols [8, 15, 18] based on Atomic Broadcast [4] are proposed, since they can afford replica failures. Each replica itself constitutes a certifier; however, they require that certified remote transactions (a special kind of transaction containing only the updates of a given transaction) must be applied as fast as possible so as to improve scalability. Weak voting protocols [10] have also been proposed in order to combine the execution of transactions under different isolation levels. More recently, other protocols which avoid the transmission of remote transactions that will finally be aborted in the replicated system have been proposed as well [11]. All these protocols are based on the deferred update technique [19]. First transactions perform their operations at a delegate replica and when they request the commit, some kind of coordination with the rest of replicas is needed to globally commit the transaction. If the transaction is finally committed, their updates must be applied in the rest of the replicas.

Most of these protocols are implemented considering that they only see a standard DBMS interface (e.g. JDBC or ODBC) [8, 18, 15, 20, 3, 21]. For that reason, they require additional functions to be implemented on top of DBMSs core, which make the deployment of a protocol easier [21, 3, 18, 15, 20]; mainly, functions which provide the collection of transaction updates and modifications to facilitate the programming of remote transactions.

In spite of the different implementations and of the practical interest of these kind of protocols, we have not found, up to our knowledge, any work which formally specifies the correctness criteria that database replicated systems with SI replicas must fulfill. These criteria should be useful to guarantee the correctness of the protocols and to analyze their advantages and limitations.

In this paper, we propose the usage of the Input/Output Automaton Model [17, 16] to provide the specification of a database replication system with SI replicas. The presented specification stresses the properties which must fulfill the components and not their specific implementations. The replicated system is shown as the composition of an abstract replication protocol and a set of databases with extended functionalities; those ones that make easier the protocol implementation. The system explicitly considers the crash failure model scenario. Actually, most of proposed protocols are designed to tolerate at least such failure scenario. Correctness criteria require: (i) the local behavior of each database replica must be respected in the system (*Well-Formedness Conditions*); (ii) conflicting transactions executed in the system must be sequentially committed (*Conflict Serializable*); (iii) the same snapshots must be generated in the system and, hence, transactions must be committed in the very same order at all replicas (*Uniform Prefix Order Database Consistency*); (iv) the decision about a transaction is the same at all replicas (*Uniform Decision*); and, (v) local progress of transactions must be preserved at correct replicas (*Local Transaction Progress*).

As it will be shown later in this paper, these criteria allow to assure that transactions are executed in the replicated system as if there was a single database which will execute transactions under a slightly weaker isolation level than SI, known as Generalized-SI (GSI) [8]. Basically, the main difference between these isolation levels stems from the snapshot gotten by transactions. Whilst SI gives the latest committed snapshot, GSI may see any previous snapshot (including the latest one), respectively. In both cases, the *lost-updates* anomaly [1] is avoided. In this paper, we discuss some variations of the correctness criteria which can be of interest while designing replication protocols.

As we have pointed out before, the Input/Output Automaton Model is going to be used to present the components and the replicated system. For the sake of understanding, we refer the reader to the articles [17] and [16] for a concrete introduction to that model. The notation we use is based on the one presented in [17]. The rest of the paper is organized as follows. Section 2 presents an extended database module which is actually an abstraction but reflects some assumptions and features implemented in real systems [21, 3, 20, 18]. The correctness criteria for a replicated database system are given in Section 3. The justification of all of them is given in Section 4 (and more formally in an additional Appendix). Finally, we present some discussion about our correctness criteria proposal in Section 5.

2 Extended Database System

In this section, we introduce a specification of an extended database system by means of a schedule module [17], denoted EDB. This module models the performance of a database following the SI level which includes some facilities in order to simplify the control a replication protocol exercises over the database. The database contains a set of uniquely identified database items (indicated as I) which may be accessed by a group of concurrent transactions. The set of possible transactions is denoted as T, an each transaction $t \in T$ has a unique identifier. A transaction is a sequence of read and write operations over the database items starting by a *begin* operation and ending by a *committed* or *aborted* notification. The guarantees the database fulfills concerning to the transaction execution are grouped in the so called *ACID* properties [2]. Throughout this paper, we consider the *Snapshot Isolation* [1] level. SI is obtained using a multiversion concurrency control mechanism. When a transaction $t \in T$ under this level reads an item, it sees the version which was most recently committed at the time t started (notice that if t modifies an item, it sees its own most recent version). Thus, the transaction makes use of the committed state of the database, called *snapshot*, when it started. When the transaction t is finally committed and it updates the value of an item $x \in I$, a new version denoted x_t, is installed on the database. New versions are visible to other transactions

only when they are installed. In order to avoid *lost updates* [1], the transaction will not be allowed to be committed if it attempts to install a version of an item x when a new version $x_{t'}$ has been installed while t was active. If that situation occurs the transaction will be aborted. When a transaction is aborted, it has no effect on the database nor in any other transaction.

For a committed transaction $t \in T$, the set of versions installed by it in the database is called its *writeset*, denoted ws_t. The writeset of an aborted transaction is also used in some parts of the text. In that case, the ws_t is interpreted as non-installed versions by the aborted transaction. The set of versions read by the transaction from the snapshot associated to it at its beginning is its *readset*, denoted rs_t. The possible set of versions for I and T is simply represented by V; thus, each ws_t and rs_t are included in V.

In the next, we define the signature of the module EDB:

$in(EDB) = \{crash\} \cup \{commit(t, ws), apply(t, ws) \colon t \in T, ws \in 2^V\}$
$out(EDB) = \{begin(t), committed(t), aborted(t) \colon t \in T\} \cup \{deliverws(t, ws) \colon t \in T, ws \in 2^V\}$

The main actions of a transaction $t \in T$ we are most concerned with are $begin(t)$, $committed(t)$ and $aborted(t)$. By means of the action $begin(t)$ the module notifies the fact that a new transaction has been initiated. The actions $committed(t)$ and $aborted(t)$ represent the final decision about such a transaction. In this simple model, client interactions during the progress of the transaction execution are not considered. We assume they appropriately work as long as the database is correct; i.e., it has not crashed. To model the possibility of a database failure, the module includes the input action $crash$.

This module is intended to work in collaboration with a replication protocol. At some point in the execution of a transaction $t \in T$, after its action $begin(t)$, the EDB informs about the writeset the transaction is ready to install. This is done by the action $deliverws(t, ws)$. Actually, you may consider this action as a commit operation requested to the replication protocol. This output action indicates to the replication protocol that a decision about such a transaction has to be achieved. The extended database guarantees the transaction has no work left to be done; so it waits for the commit for such a transaction. The EDB allows only the replication protocol to request the commit of the transaction via the input action $commit(t, ws)$. A transaction following such a pattern of operation is called a *local transaction*. The transaction starts under the control of the extended database; and it passes the control of the transaction to the replication protocol in order to terminate it.

When the replication protocol takes the decision that a transaction $t \in T$ has to be committed, it requires either the replication protocol produces the action $commit(t, ws)$ or the database applies the updates of the transaction; i.e., its writeset. Thus, the EDB provides as input action the action $apply(t, ws)$. The extended database is responsible of programming such a transaction in the underlaying database in a transparent way to the replication protocol. A transaction following such a pattern of operation is called a *remote transaction*. The readset of a remote transaction is empty. In this case, the transaction is programmed by the replication protocol in the extended database which is in charge of terminating the transaction.

The properties of the EDB module are introduced by presenting the properties of its behaviors [17]. These properties are interpreted as assumptions in the behaviors. Recall that each behavior in $behs(EDB)$ is a finite or infinite sequence of actions from $acts(EDB)$ of the signature of EDB. A behavior $\beta \in behs(EDB)$ is denoted $\beta = \pi_1\pi_2 \ldots \pi_r \ldots$ and predicates in the assumptions make reference to this notation. The basis of our database specification is the *snapshot* concept. We must define which versions comprise the snapshot of a behavior of the EDB module at some point of the execution. To do that, the *log* of a finite behavior is firstly stated: it is the ordered sequence of writesets of committed transactions.

Definition 1 *(Log of EDB) Let β be a finite behavior of EDB. For each prefix $\beta(j)$ [1], $0 \leq j \leq |\beta|$, the log of $\beta(j)$ is defined as follows:*

$\cdot\ log(\beta(j)) =$ *empty iff* $j = 0$
$\cdot\ log(\beta(j)) = log(\beta(j-1)) \cdot \langle t, ws_t \rangle$ *iff* $\pi_j = committed(t)$ *and* $j > 0$
$\cdot\ log(\beta(j)) = log(\beta(j-1))$ *iff* $\pi_j \neq committed(t)$ *and* $j > 0$.

[1] $\beta(j)$ is the prefix of length j of β, i.e., $|\beta(j)| = j$. Further notation about sequences is introduced in the following: $\beta|\varphi$ is the subsequence of β which includes only the actions of φ in β. Finally, $\gamma \preceq \beta$ indicates that γ is a prefix of β (γ may be the empty sequence).

The snapshot of a behavior of EDB at some point of the execution contains the *latest version* of each item until that point as it has been defined for conventional SI in a DBMS [1]. We consider that for each item $x \in I$ there is an initial version x_0 as long as it has not been modified by a transaction. Thus, for a finite prefix $\beta(j)$ of β and an item $x \in I$, $latestVer(x, \beta(j))$ is x_t if $x_t \in ws_t$ and $\langle t, ws_t \rangle \in log(\beta(j))$ being $t \in T$ the latest transaction in $\beta(j)$ modifying the item x; or x_0 otherwise.

Definition 2 *(Snapshot) Let β be a finite behavior of DB. For each prefix $\beta(j)$, $0 \leq j \leq |\beta|$, the snapshot of $\beta(j)$ is defined as $Snapshot(\beta(j)) = \bigcup_{x \in I} \{latestVer(x, \beta(j))\}$*

In this paper, the set $acts(M, t)$, for some module M defined in the paper, includes the actions from $acts(M)$ having $t \in T$ as parameter. We write, $ws_t \cap ws_{t'} \neq \emptyset$ (or $rs_t \cap ws_{t'} \neq \emptyset$) if they contain some version for the same item. We also use the following shorthand predicate for a behavior β with two transactions $t', t \in T$, and two indexes $i, j \in \mathbb{Z}^+$: $conflict(t', t, i, j, \beta) \equiv \exists\, k \colon i < k < j \colon \pi_k = committed(t') \wedge ws_t \cap ws_{t'} \neq \emptyset$. The safety properties of the behaviors of EDB are presented in the next assumption.

Assumption 1 *For each behavior $\beta \in behs(EDB)$:*

1. *(Execution Integrity) $\pi_i \neq crash \Rightarrow \forall\, k \colon k < i \colon \pi_k \neq crash$*
2. *(Well-formed Transaction) For each transaction $t \in T$ the sequence $\beta | acts(EDB, t)$ is a prefix of at least one of the following sequences:*
   - *(a)* $begin(t)\ deliverws(t, ws_t)\ commit(t, ws_t)\ committed(t)$
   - *(b)* $begin(t)\ aborted(t)$
   - *(c)* $begin(t)\ deliverws(t, ws)\ aborted(t)$
   - *(d)* $begin(t)\ deliverws(t, ws)\ commit(t, ws)\ aborted(t)$
   - *(e)* $apply(t, ws_t)\ begin(t)\ committed(t)$
   - *(f)* $apply(t, ws)\ begin(t)\ aborted(t)$
3. *(Snapshot Isolation) For each transaction $t \in T$ such that $\pi_i = begin(t)$ and $\pi_j = committed(t)$ in β:*
   - *(a)* $rs_t \subseteq Snapshot(\beta(i))$
   - *(b)* $\neg conflict(t', t, i, j, \beta)$ *for all* $t' \in T$

Assumption 1.1 (Execution Integrity) indicates that after $crash$ the EDB stops its activity. Assumption 1.2 (Well-formed Transaction) indicates the correct order of transaction actions in a behavior. This last Assumption demands some additional constraints to the replication protocol using the EDB in order to build well-formed transactions in the EDB. In particular, an input action $commit(t, ws)$ for transaction t may be only possible after $deliverws(t, ws)$, this can never occur after $committed(t)$ or $aborted(t)$ actions; moreover, it only happens once in a behavior. Respectively, the $apply(t, ws)$ input action only occurs once and before its associated action $begin(t)$ in a behavior. Assumption 1.2 also indicates that the parameter ws in the actions $deliverws(t, ws)$, $commit(t, ws)$, or $apply(t, ws)$ is the ws_t of the transaction t if it is committed (sequences (*a*) or (*e*)). In other words, after $deliverws(t, ws)$ or $apply(t, ws)$ the ws will remain unchanged. In the case of an aborted transaction (sequences (*c*), (*d*) or (*f*)) we consider $ws_t \subseteq ws$ but the obtained results are not affected by this fact, so we will also consider $ws_t = ws$. Finally, Assumption 1.3 (Snapshot Isolation) specifies the requirements every committed transaction in a behavior has to verify in order to reach the SI level.

Notice that a replication protocol is only informed about the writesets of transactions. So, it may only deduce the outcome of a transaction through the order of input events and their writesets it has been received. Therefore, in order to complete the Assumption 1, it is important to attach the possible aborting causes for the actions $aborted(t)$ in Assumption 1.2. This is done in the following remark.

Remark 1 *(Abort Assumptions) Let $\beta \in behs(EDB)$*

- *There is no transaction unilateral abort.*

- *For a transaction $t \in T$ such that $\pi_i = begin(t)$ and $\pi_j = aborted(t)$, if it follows pattern (c), (d) or (f) in Assumption 1.2 then there is a transaction $t' \in T$ such that $conflict(t', t, i, j, \beta)$.*

- *$aborted(t)$ in the sequence (b) in Assumption 1.2 is possible by any abortion cause (e.g., a deadlock resolution, timeout expiration).*

From the previous remark, it is worth mentioning that a remote transaction is not equivalent to a local one. It will only abort if it is impossible to guarantee its isolation level. Other possible causes of abortion are filtered by the extended database. The conditions in Remark 1 have an important impact in the practical application of the EDB in real settings. However, we take into consideration those conditions as a first approach to provide a complete specification. Actually, similar considerations can be found in several replication protocols presented in the literature [18, 15, 8, 20, 5]. Until now, previous assumptions are only related with safety properties. In the next assumption we give some simple liveness properties of the EDB.

Assumption 2 *For each behavior $\beta \in behs(EDB)$:*

1. $\pi_i = begin(t) \Rightarrow \exists\, k\colon k > i\colon \pi_k \in \{deliverws(t, ws), aborted(t), crash\colon ws \in 2^V\}$
2. $\pi_i = begin(t) \wedge \pi_j = deliverws(t, ws) \wedge \pi_k = committed(t') \wedge ws_{t'} \cap ws \neq \emptyset \wedge i < k$
   $\wedge\ \forall r\colon i < r < k\colon \pi_r \neq committed(t) \Rightarrow \exists\, r\colon r > j\colon \pi_r \in \{aborted(t), crash\}$
3. $\pi_i = commit(t, ws) \Rightarrow \exists\, k\colon k > i\colon \pi_k \in \{committed(t), aborted(t), crash\}$
4. $\pi_i = apply(t, ws) \Rightarrow \exists\, k\colon k > i\colon \pi_k \in \{committed(t), aborted(t), crash\}$

We will informally depict this assumption in the absence of failures. Assumption 2.1 (only for local transactions) states that if a transaction is not aborted it delivers its writeset. Assumption 2.2 indicates that a transaction will be aborted if its isolation level is not maintained. Assumption 2.3 states that after $commit(t, ws)$ the transaction terminates. Finally, Assumption 2.4 indicates that a remote transaction terminates.

One can think that we demand artificial properties in the EDB. Actually, some current implementations of replication protocols use, and do need, these extended capabilities of databases. These features are implemented in different ways over practical middleware replicated database systems; e.g. either by reattempting aborted remote transactions [15] or by early detection conflict with the help of DBMS facilities as described in [18].

To end this section, we indicate that a read-only transaction does not conflict with any other transaction and it receives the latest snapshot at is beginning. Thus, a read-only transaction always terminates in a committed status if the EDB does not crash. The execution of a read-only transaction is considered transparent to the replication protocol. In the rest, we assume that every transaction has a non empty writeset.

3 Replicated Database System: Correctness Criteria

In this section, we provide the correctness criteria for a replicated database system in which databases in the system perform independently and execute transactions under SI. The replicated database system is specified by means of a module denoted $RDBS$. The main components of this module are depicted in Figure 1.

As Figure 1 suggests, the replicated database system is the composition of an abstract replication protocol and a group of extended databases, one at each site of the distributed system. The finite set of site identifiers is denoted as N. We assume that at most f sites may fail by crash and $|N| > f$. At each site $n \in N$ there is an extended database module denoted EDB_n. Each action in the signature of EDB_n is also subscribed by n. We consider the replicated database system is *full replicated*: the set of items of each database is the same set for all $n \in N$, denoted I. The set of transactions operating in the system is T; and the set of possible versions for the items I and transactions T is V.

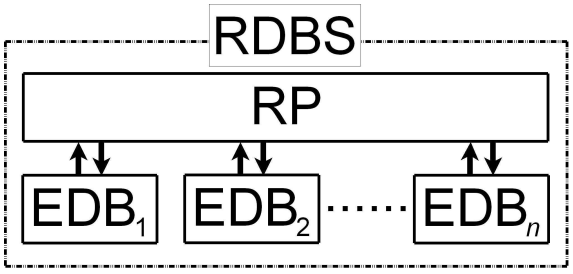

Figure 1: Replicated Database System

There is a mapping, $site\colon\ T \longrightarrow N$, which associates to each transaction, $t \in T$, a unique site, $site(t) \in N$, in the system. The $site(t)$ is called the *delegate site* of the transaction. It is the site where the transaction starts the execution. The transaction is considered as local at that replica and remote at the rest of sites.

We consider that the replication protocol is based on the *deferred update technique* [19]. In this technique, once the EDB_n completes the execution of a transaction, it submits the $deliverws_n(t, ws)$ to the protocol in order it to take the decision of commit the transaction and to apply the transaction in the rest of sites. We consider that the replication protocol does not explicitly abort the transaction if it can not provide the commit for it (in fact we have not model the action $abort(t)$ in the EDB). The protocol simply deduces the transaction will be aborted.

The replication protocol is specified by a module, denoted as RP. The main goal of the replication protocol is to guarantee the correctness criteria in the whole system. We only consider its signature because the properties of its behaviors will be abstracted in the correctness criteria. The signature of RP is:

$in(RP) = \cup_{n\in N}\ (out(EDB_n) \cup \{crash_n\})$
$out(RP) = \cup_{n\in N}\ \{commit_n(t, ws),\ apply_n(t, ws)\colon\ n \in N, t \in T, ws \in 2^V\}$

The module $RDBS$ is obtained as a result of the module composition operation [17]: $RDBS = RP \times (\Pi_{n\in N}\ EDB_n)$. This module is well defined, since the collection of signatures of the component modules is strongly compatible [17]. Thus, the signature of $RDBS$ is:

$in(RDBS) = \cup_{n\in N}\{crash_n\}$
$out(RDBS) = (\cup_{n\in N}\ out(EDB_n)) \cup out(RP)$

In the following, we present and explain the correctness criteria for the replicated database system. Let β be a behavior of $RDBS$. We use the predicate $local(t, n, \beta) \equiv begin_n(t) \preceq \beta|acts(EDB_n, t)$ to indicate that a transaction $t \in T$ has started in the site $n \in N$ as a local transaction in the behavior β. The correctness criteria are indicated in the following axioms[2]. For every behavior $\beta \in behs(RDBS)$:

1. *Well-formedness Conditions*.
   (a) $\beta|EDB_n \in behs(EDB_n)$
   (b) $local(t, n, \beta)\ \wedge\ local(t, n', \beta) \Rightarrow n = n' = site(t)$
   (c) $\pi_i = apply_n(t, ws) \Rightarrow \exists\, k\colon\ k < i\colon\ \pi_k = deliverws_{site(t)}(t, ws) \wedge n \neq site(t)$
2. *Conflict Serializable*.
   (a) $\pi_i \in \{apply_n(t, ws), commit_n(t, ws)\} \wedge \pi_j = apply_n(t', ws')\ \wedge\ i < j\ \wedge\ ws \cap ws' \neq \emptyset\ \Rightarrow$ $\exists\, k\colon\ i < k < j\colon\ \pi_k \in \{committed_n(t), aborted_n(t)\}$
   (b) $\pi_i \in \{apply_n(t, ws), commit_n(t, ws)\} \wedge \pi_{j_1} = begin_n(t') \wedge \pi_{j_2} = commit_n(t', ws') \wedge i <$ $j_2\ \wedge\ ws \cap ws' \neq \emptyset \Rightarrow \exists\, k\colon\ i < k < j_1\colon\ \pi_k \in \{committed_n(t), aborted_n(t)\}$

[2]Free variables in the expressions are universally quantified in their domains for the scope of the entire formulas

3. *Uniform Prefix Order Database Consistency.* For every finite prefix β' of β:

$$log(\beta'|EDB_n) \preceq log(\beta'|EDB_{n'}) \text{ or vice versa}$$

4. *Uniform Decision.*

(a) $\pi_i = committed_n(t) \Rightarrow \forall\, n' \in N\colon (\exists\, k\colon \pi_k \in \{commit_{n'}(t, ws_t), apply_{n'}(t, ws_t), crash_{n'}\}$

(b) $\pi_i = aborted_{site(t)}(t) \Rightarrow (\beta|\{apply_n(t, ws)\colon n \in N, ws \in 2^V\} = \text{empty})$

5. *Local Transaction Progress.*

$$\pi_i = deliverws_{site(t)}(t, ws) \Rightarrow \exists\, k\colon k > i\colon \pi_k \in \{commit_{site(t)}(t, ws), crash_{site(t)}\}$$
$$\vee\, \pi_k \in \{commit_{site(t)}(t', ws'), apply_{site(t)}(t', ws')\colon ws \cap ws' \neq \emptyset\}$$

Criterion 1.(a) states that every behavior of the $RDBS$ has to respect the behavior of each EDB_n module. The basic consequences of this fact are that each behavior of a EDB_n module verifies the SI level, it is well-formed, and it satisfies every progress property given in Assumption 2. Criterion 1.(b) indicates that the first event of a transaction $t \in T$ in the system may only be $begin_{site(t)}(t)$ at its delegate site; t is local at that replica and remote otherwise. Criterion 1.(c) asserts that a remote transaction may appear in the system as consequence of a local transaction, with the same writeset. This criterion avoids spontaneous creation of remote transactions in the system. All these previous criteria are grouped and form what we have denoted as Well-formedness Conditions. Criterion 2 (Conflict Serializable) ensures that the replication protocol does not apply a remote transaction nor requests the commit of a local transaction if there is a previous unresolved conflicting transaction. This criterion guarantees that transactions with a non-empty intersection of their writesets are serialized. Criterion 3 (Uniform Prefix Database Order Consistency) imposes on the system to build the same snapshots at every database; actually, it obliges committed transactions to follow the same commit ordering at every site (not only the conflictive ones). In addition, every remote transaction of the same local transaction installs exactly the same writeset; i.e, the local transaction writeset. Notice that if a database fails, this criterion ensures that the last installed snapshot is also a valid snapshot for the rest of the correct sites. The Criterion 4 (Uniform Decision) is split into two kind of transactions: committed and aborted ones. Criterion 4.(a) states that if a transaction is committed at one site (correct or faulty) then the protocol eventually applies or requests the commit of the same transaction in every correct site; and Criterion 4.(b) states that if a transaction is aborted at one site (correct or faulty) then it is only aborted in the delegate site and no one of its remote transactions will be programmed in the system. To conclude, Criterion 5 (Local Transaction Progress) indicates that if the replica is correct, then for each of its local associated transactions that requests the commit (via action $deliverws(t, ws)$), the replication protocol either requests the commit or knows it will be aborted.

4 Justification

The previous correctness criteria specify, very precisely, the requirements a replicated database system has to verify. The best way to justify that they are adequate is to prove that they imply an equivalent behavior of a one-copy database system.

The most straightforward way to do that, with the I/O Automaton Model, is to prove that $behs(RDBS) \subseteq behs(DB)$ for some single database module DB. Unfortunately, due to several reasons, such as the possibility of $crash_n$ events, the lazy nature of remote transactions through $apply_n(t, ws)$ events, and the fact that each EDB_n behaves independently of each other, it is non-trivial to find out a single DB with the same signature of $RDBS$. This consideration involves an indirect way to provide the one-copy equivalence.

Other reasonable question is whether the replicated database system working with SI databases will achieve the same isolation level for the transactions executed on it. In fact, we do not enforce the system to work in a pure synchronized manner and it is possible to have two replicas with different snapshots at the same time (see Criterion 3). This means that a transaction may obtain in its delegate site a snapshot which is an older snapshot in another '*faster*' replica. A generalization of SI to include this possibility of using older snapshots was introduced in [7] under the concept of Generalized Snapshot Isolation (GSI).
Consider an ideal database module, denoted DB, such that $out(DB) = \{begin(t), committed(t)\colon\ t \in$

$T\}$, and $in(DB) = \emptyset$; in which every scheduled transaction is committed. That is, for a transaction $t \in T$ and behavior $\beta \in behs(DB)$, if $\pi_i = begin(t)$ in β then there is a $\pi_j = committed(t)$ with $i < j$. We assume that every committed transaction follows the GSI level. Therefore, for each transaction $t \in T$ such that $\pi_i = begin(t)$ and $\pi_j = committed(t)$ in β, there exists an index $0 \leq s \leq i$ such that the two next conditions hold:

1. $rs_t \subseteq Snapshot(\beta(s))$
2. $\neg conflict(t', t, s, j, \beta)$ for all $t' \in T$

Notice that a transaction t under GSI can use an older snapshot ($0 \leq s \leq i$), but it can be committed as its updates are still valid from that snapshot (recall the lost-updates phenomenon [1]). In GSI, if conditions (1) and (2) are valid for every transaction when $s = i$, then the SI definition is obtained. In the rest of this section, we give the way to extract from an arbitrary behavior of $RDBS$ an equivalent one-copy behavior of this GSI DB. In the enclosed Appendix, the proofs of the claims done in this section can be found.
Let β be a behavior of $RDBS$. Firstly, we study the performance of a transaction $t \in T$ scheduled in the system. Recall that, by Criterion 1.(b), the first event of a transaction is $begin_{site(t)}(t)$ in β. Therefore, we firstly examine the subsequence $\beta_t = \beta|\{begin_{site(t)}(t), aborted_n(t), committed_n(t) \colon\ n \in N\}$ where three different cases may arise:

$\beta_t = begin_{site(t)}(t)$ and action $crash_{site(t)}$ is given in β
$\beta_t = begin_{site(t)}(t) \cdot aborted_{site(t)}(t)$
$\beta_t = begin_{site(t)}(t) \cdot \gamma_{c_t}$ being γ_{c_t} a proper prefix of $committed_{n_1}(t) \ldots committed_{n_{|N|}}(t)$ with $(n_1, \ldots, n_{|N|})$ a permutation of the site identifiers.

This result means that a transaction terminates in the same status at every correct replica. In fact, the replication protocol will request an $apply_n(t, ws)$ or $commit_n(t, ws)$ only if it is assured that the transaction is able to become committed. Notice that the case $\beta_t = begin_{site(t)}(t)$ is only possible if there is an event $crash_{site(t)}$ in β. However, the case $\beta_t = begin_{site(t)}(t) \cdot \gamma_{c_t}$ does not exclude the replica from crashing too. If a transaction is aborted, it will have no effect in the replicated system (by previous result and Criterion 1.(a)). Thus, we will only consider those committed transactions appearing in β. We say that a *transaction $t \in T$ is committed in β*, if β_t includes an action $committed_n(t)$ for some site $n \in N$. In the previous result, the event $committed_{n_1(t)}(t)$ indicates that the site $n_1(t)$ is the first one installing the writeset ws_t of the transaction t. Every committed transaction at every replica has installed the same writeset due to Criterion 1.(a) and Criterion 1.(c). In fact, this result is consistent with Criterion 3. Thus, the rest of committed events for a transaction are still necessary to maintain the consistency of the database replicas; however, $n_1(t)$ is the first replica in the whole system in which the new snapshot is available. Therefore, we will study the properties of the following subsequence.
Let us denote as *First Committed* (FC) the set of actions $\{begin_{site(t)}(t), committed_{n_1(t)}(t) \colon\ t$ is a committed transaction in $\beta\}$. For each behavior $\beta \in behs(RDBS)$, we define the sequence $\beta_{FC} = \beta|FC$. In this ideal sequence, we assign to each transaction t: as its readset the one obtained from its delegate replica (i.e., $rs_t \subseteq Snapshot(\beta|EDB_{site(t)}(i))$ where $\pi_i = begin_{site(t)}$ in $\beta|EDB_{site(t)}$); and, as its writeset the value of ws_t at the time it committed.

This last sequence has a nice property which states that $log(\beta|EDB_n) \preceq log(\beta_{FC})$ for all $n \in N$. Therefore, β_{FC} installs the same snapshots as in the replicated system.
In the replicated system, each transaction can be submitted to any replica (actually, $site(t)$ is an arbitrary mapping) every pattern of failures is possible except for the restriction $|N| > f$, and each EDB_n performs in the same way. In addition, $n_1(t)$ is some non faulty site. Thus, for each $\beta \in behs(RDBS)$, there is some equivalent $\beta' \in behs(RDBS)$ such that each $n_1(t)$ is different to $n'_1(t)$.

We say that two behaviors β, β' of $RDBS$ are equivalent, $\beta \equiv \beta'$ if the following conditions hold:
- t is committed in $\beta \Leftrightarrow t$ is committed in β'. For each committed transaction t:
- rs_t in $\beta \Leftrightarrow rs_t$ in β'
- ws_t in $\beta \Leftrightarrow ws_t$ in β'
- $begin_{site(t)}(t)\ committed_{n_1(t)}(t) \preceq \beta_t \Leftrightarrow begin_{site(t)}(t)\ committed_{n'_1(t)}(t) \preceq \beta'_t$

For each $\beta \in behs(RDBS)$ we define $\beta'|FC'$ such that $\beta \equiv \beta'$ and $FC' = \{begin_{site(t)}(t), committed_{n'_1(t)}(t) \colon\ t$ is a committed transaction in β' and $n'_1(t) = site(t)\}$.

Finally, β_{1C} is obtained from $\beta'|FC'$ by renaming each action $begin_{site(t)}(t)$ as $begin(t)$ and $committed_{site(t)}(t)$ as $committed(t)$. The sequence β_{1C} is the one-copy version of β. In β_{1C}, the readset of t is rs_t of t in β and the writeset of t is ws_t of t in β.
We can prove that β_{1C} verifies that for each transaction $t \in T$ such that $\pi_i = begin(t)$ and $\pi_j = committed(t)$ in β_{1C}, there exits an index $0 \leq s \leq i$ such that the two next conditions hold: 1. $rs_t \subseteq Snapshot(\beta_{1C}(s))$ and 2. $\neg conflict(t', t, s, j, \beta_{1C})$ for all $t' \in T$.

In conclusion, for each behavior $\beta \in RDBS$ there is a one-copy version β_{1C} such that $\beta_{1C} \in behs(DB)$. In other words, an external observer that collects the first begin event and the first commit event for every committed transaction (whilst discarding the aborted ones) in the system will not distinguish the execution of the replicated database system from one derived from a one-copy database system.

5 Discussion

Most of the literature about database replication introduces a given replication protocol and then it is discussed or proved its correctness [5, 12, 15, 8, 20]. However, there has been little (or none) discussion in the literature about setting up a general correctness criteria that these replication protocols must verify [2, 22]. In the same way, most of these replication protocol proposals claim that they can afford a crash failure scenario (or a crash-recovery one) by simply forwarding transactions to another available replica. Nevertheless, it has never been formally shown, up to our knowledge, neither their correct behavior under this failure scenario nor which additional correctness criteria have to be established. We have tried to unify and propose quite general correctness criteria for replicated database systems in a crash failure scenario. The correctness criteria proposed in this work are suitable to a replicated database system where database replicas are SI and the replication protocol follows the deferred update technique. In the following, we try to thoroughly discuss different issues that can remain unclear about them. Besides, we cover some aspects about these criteria that can lead to optimizations or variations of them; actually, they can be considered in real implementations of a replicated database system.

Let us start with the *Conflict Serializable* correctness criterion (Criterion 2). Under conventional SI [1], conflicting update transactions (i.e. given t_1 and t_2 with $ws_1 \cap ws_2 \neq \emptyset$) must be serialized and, hence, it does make sense to include this fact in this correctness criterion for a replicated database system. Of course, this criterion does not say anything about the order in which non-conflicting update transactions should be applied and, thus, it is perfectly possible that a given replication protocol permits them to be executed and committed in any order at different replicas [15].

Nevertheless, we have restricted their application in order to impose a unique commit ordering throughout all replicas (Criterion 3, *Uniform Prefix Order Database Consistency*) and obtain the same global snapshots at all replicas. This does not imply that transactions are committed in a synchronous way at all replicas. Some replicas may run faster than others and transactions executed on faster ones will get more recent snapshots than others executed at slower ones; though all transactions will read from a global consistent snapshot. Thanks to this last criterion committed transactions obtain GSI [7, 8]. However, we can weaken this criterion and allow non-conflicting update transactions to be applied in any order [15]. The main advantage of this consists in increasing the number of concurrent applications of writesets. However, there exists a trade-off for this last feature: a replication protocol cannot guarantee a global consistent snapshot. Therefore, transactions should not be started as long as there are concurrent applications of writesets. The beginning of a transaction should be delayed until a given writeset (i.e. known by all replicas) is applied; the simplest scenario consists of two conflicting writesets that must always be sequentially applied. The best approach will depend on the kind of application considered in the system. On the other hand, if SI is globally desired, then the delegate replica of a transaction must delay the start of it until the latest snapshot is applied. Both approaches (obtaining SI as well as concurrent application of writesets) imply a potential blocking of the beginning of a read-only transaction. This fact goes against the basic non-blocking nature for such transactions executed under SI [1] or GSI [8] and they are not transparently executed from the protocol's point of view anymore.

Thanks to Criterion 3, the sets of committed transactions at any pair of replicas, either one constitutes the prefix of the other or vice versa, even if one of the replicas has crashed. This is particularly interesting

in a crash-recovery failure scenario, since the recovery of a failed replica consists in transferring the set of missed transactions starting from the end of its prefix at an active replica [14].

We have proposed the *Uniform Decision* criterion to set up the outcome (either committed or aborted) of a transaction in the database replicated system. One of the main features of committed transactions in the deferred update technique is that if a transaction is committed at a replica, it will eventually get committed at the rest of replicas unless they crash (Criterion 4.(a)). Another particular characteristic of the deferred update technique refers to the aborted transactions in the system. They will always get aborted at its delegate replica and will not be applied to the rest of replicas (Criterion 4.(b)), what is a great advantage. An example of this is the family of certification based replication protocols for SI replicas [8, 15, 18].

According to our specification, the replication protocol will never request the commit on behalf of a transaction which will be aborted at the local database. The *Local Transaction Progress* (Criterion 5) ensures that this transaction is going to be aborted as a consequence of another previously committed transaction. Usually, real implementations do explicitly abort these transactions (by means of an action $abort(t)$) to release database resources as soon as possible [18, 21]. These replication protocols must be pretty sure that the explicit abortion of a transaction is due to the fact that it will eventually get aborted. We have not considered these kind of actions because they are not needed in our model proposal.

Finally, we have introduced an extended database abstraction. This abstraction includes all functionalities needed by replication protocols [15, 3, 18, 20, 21]. In particular, it provides an abstraction of writeset extraction [15, 3, 20, 21] and its successful application [18, 15, 21] (with the implicit abort of local conflicting transactions). This last feature has been thoroughly studied in real implementations either with time outs and re-attempt mechanisms [15], conflict detection mechanisms [18] or aborting transactions at the time they request the commit [15]. Moreover, this abstraction hides the details of the particular implementation of the SI DBMS committing rule: either *first-committer-wins* or *first-updater-wins* rule [1, 9] (though most commercial implementations follow the former). This abstraction is a good enough one, since it reflects the common features of different real implementations. In other words, it does reflect the functionality of them while hiding all implementation details followed in different approaches.

Our work keeps certain similarities with [22] where the deferred update technique with serializable databases is formally studied using TLA+ [13]. As opposed to ours, they only cope with serializable databases and do not cover any kind of failure scenario. As future work, it will be interesting to extend these correctness criteria in different ways: to the crash-recovery model; to other isolation levels; and, formalize some replication protocols as I/O automata and verify their correctness.

References

[1] H. Berenson, P.A. Bernstein, J. Gray, J. Melton, E.J. O'Neil, and P.E. O'Neil. A critique of ANSI SQL isolation levels. In *SIGMOD*, 1995.

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

[3] N. Carvalho, A. Correia Jr., J. Pereira, L. Rodrigues, R. Oliveira, and S. Guedes. On the use of a reflective architecture to augment database management systems. *JUCS*, 13(8):1110–1135, 2007.

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

[5] K. Daudjee and K. Salem. Lazy database replication with snapshot isolation. In *VLDB*, 2006.

[6] S. Elnikety, S. Dropsho, and F. Pedone. Tashkent: Uniting durability with transaction ordering for high-performance scalable database replication. In *ACM Eurosys*, Leuven, Belgium, April 2006.

[7] S. Elnikety, F. Pedone, and W. Zwaenopoel. Generalized snapshot isolation and a prefix-consistent implementation. EPFL-Tech-Rep IC/2004/21, March 2004.

[8] S. Elnikety, F. Pedone, and W. Zwaenopoel. Database replication using generalized snapshot isolation. In *SRDS*. IEEE-CS Press, 2005.

[9] A. Fekete, D. Liarokapis, E. O'Neil, P. O'Neil, and D. Shasha. Making snapshot isolation serializable. *ACM Trans. Database Syst.*, 30(2):492–528, 2005.

[10] J.R. Juárez, J.E. Armendáriz, J.R. González de Mendívil, F.D. Muñoz, and J.R. Garitagoitia. A weak voting database replication protocol providing different isolation levels. In *NOTERE'07*, 2007.

[11] J.R. Juárez, J.E. Armendáriz, F.D. Muñoz, J.R. González de Mendívil, and J.R. Garitagoitia. A deterministic database replication protocol where multicast writesets never get aborted. In *OTM Workshops (1)*, volume 4805 of *LNCS*, pages 1–2. Springer, 2007.

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

[13] L. Lamport. *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*. Addison Wesley Professional, 2002.

[14] W. Liang and B. Kemme. Online recovery in cluster databases. In *EDBT*. ACM Press, 2008. Accepted for publication.

[15] Y. Lin, B. Kemme, M. Patiño-Martínez, and R. Jiménez-Peris. Middleware based data replication providing snapshot isolation. In *SIGMOD*, 2005.

[16] N.A. Lynch. *Distributed Systems*. Morgan Kaufmann Publishers, 1996.

[17] N.A. Lynch and M.R. Tuttle. An introduction to input/output automata. *CWI-Quarterly*, 2(3):219–246, 1989.

[18] F.D. Muñoz, J. Pla, M.I. Ruiz, L. Irún, H. Decker, J.E. Armendáriz, and J.R. González de Mendívil. Managing transaction conflicts in middleware-based database replication architectures. In *SRDS*. IEEE-CS Press, 2006.

[19] F. Pedone. *The database state machine and group communication issues (Thèse N. 2090)*. PhD thesis, EPFL, Lausanne, Switzerland, 1999.

[20] C. Plattner, G. Alonso, and M. Tamer-Özsu. Extending DBMSs with satellite databases. *The VLDB Journal*, 2006.

[21] J. Salas, R. Jiménez-Peris, M. Patiño-Martínez, and B. Kemme. Lightweight reflection for middleware-based database replication. In *SRDS*. IEEE-CS Press, 2006.

[22] R. Schmidt and F. Pedone. A formal analysis of the deferred update technique. In *DISC*, volume 4731 of *LNCS*, pages 499–500. Springer, 2007.

A Appendix

Lemma 1 *Let β be a behavior of $RDBS$. It holds that*

$$\pi_i \in \{commit_n(t, ws), apply_n(t, ws)\} \Rightarrow \exists\, k\colon k > i\colon \pi_k \in \{committed_n(t), crash_n\}$$

Proof: The property is referred to the site n. By Criterion 1.(a) (Well-Formedness Conditions), $\beta|EDB_n \in behs(EDB_n)$. If we prove the property for $\beta|EDB_n$ then it is also proved for β. In the following, the indexes are referred to $\beta|EDB_n$. Let π_i in $\beta|EDB_n$ and consider that $\forall\, k\colon k > i\colon \pi_k \notin \{committed_n(t), crash_n\}$. By Assumption 3 (and 4), there is $\pi_{i_2} = aborted_n(t)$ with $i < i_2$. By Remark 1, there is a transaction $t' \in T$, such that $conflict(t', t, i_1, i_2, \beta|EDB_n)$ where $\pi_{i_1} = begin_n(t)$ with $i_1 < i_2$ (by Assumption 1.2). Therefore, there is an action $\pi_{j_2} = committed_n(t')$ such that $i_1 < j_2 < i_2$ and $ws_{t'} \cap ws \neq \emptyset$. By Assumption 1.2 there exists $\pi_j \in \{commit_n(t', ws_{t'}), apply_n(t', ws_{t'})\}$ with $j < j_2$. Any possible case, $i < j$ or $j < i$, for the actions π_i and π_j yields to a contradiction with Criterion 2 (Conflict Serializable). □

Property 1 *Let β be a behavior of $RDBS$. For each transaction $t \in T$, the first event in $\beta|acts(RDBS, t)$ is $begin_{site(t)}(t)$ if the transaction has been programmed or none, otherwise.*

Proof: If $\beta|acts(RDBS, t)$ is non empty there will be some action with parameter t in β; thus, transaction t has been programmed. Let π the first event in $\beta|acts(RDBS, t)$ such that $\pi \neq begin_{site(t)}(t)$. It cannot be $begin_n(t)$ with $site(t) \neq n$ by the Criterion 1.(b) (Well-Formedness Conditions). For any other action with $site(t) \neq n$, it is very simple to show that $begin_{site(t)}(t)$ precedes it by Criterion 1.(a), Assumption 1.2 and Criterion 1.(c). Again, by Criterion 1.(a), Assumption 1.2, any action π in $site(t)$ with parameter t is preceded by $begin_{site(t)}(t)$. The property holds. □

For each transaction $t \in T$ and behavior $\beta \in behs(RDBS)$, we define the sequence $\beta_t = \beta|\{begin_{site(t)}(t), aborted_n(t), committed_n(t)\colon n \in N\}$.

Theorem 1 *Let β be a behavior of $RDBS$. For each transaction $t \in T$, the sequence β_t is at most one of the following sequences:*

1. $\beta_t =$ *empty*
2. $\beta_t = begin_{site(t)}(t)$ *and action* $crash_{site(t)}$ *is given in* β
3. $\beta_t = begin_{site(t)}(t) \cdot aborted_{site(t)}(t)$
4. $\beta_t = begin_{site(t)}(t) \cdot \gamma_{c_t}$ *being* γ_{c_t} *a proper prefix of* $committed_{n_1}(t) \ldots committed_{n_{|N|}}(t)$ *with* $(n_1, \ldots, n_{|N|})$ *a permutation of the site identifiers.*

Proof:
(1). If β_t is empty, then by Property 1 the transaction has not been programmed in the system.
(2). Consider that there is no action $crash_{site(t)}$ in β and the transaction has been programmed. Thus, by Property 1, β_t is some sequence starting with $begin_{site(t)}(t)$. Let $\pi_i = begin_{site(t)}(t)$ in β. By Criterion 1.(a) (Well-Formedness Conditions) and Assumption 2.1, there is $\pi_{j_1} \in \{deliverws_{site(t)}(t, ws), aborted_{site(t)}(t)\}$ in β with $i < j_1$. If $\pi_{j_1} = aborted_{site(t)}(t)$ then a contradiction is obtained. Thus, $\pi_{j_1} = deliverws_{site(t)}(t, ws)$. By Criterion 5 (Local Transaction Progress), there exists in β the action $\pi_{j_2} = commit_{site(t)}(t, ws)$ or $\pi_{j_2} \in \{commit_{site(t)}(t', ws'), apply_{site(t)}(t', ws')\colon ws \cap ws' \neq \emptyset\}$ with $j_1 < j_2$. In the former case, by Lemma 1 there is $\pi_{j_3} = committed_{site(t)}(t)$ with $j_2 < j_3$. A contradiction is obtained. In the second case, by Lemma 1 there is $\pi_{j_3} = committed_{site(t)}(t')$ with $j_2 < j_3$ and $ws \cap ws_{t'} \neq \emptyset$ (by Assumption 1.2, $ws' = ws_{t'}$). Therefore, π_i, π_{j_1} and π_{j_3} happen in β. By Criterion 1.(a), the same actions are in $\beta|EDB_n$ in the same order. By Assumption 2.2, there is $\pi_{j_4} = aborted_{site(t)}(t)$. A contradiction is obtained. Therefore, $\beta_t = begin_{site(t)}(t)$ only if there is the action $crash_{site(t)}$ in β. After $crash_{site(t)}$ no other action happens in that site by Assumption 2.1.
(3). Let $\pi_i = begin_{site(t)}(t)$ in β. By the previous proof (2), if $site(t)$ is correct (no $crash_{site(t)}$ in β) then $\pi_j \in \{committed_{site(t)}(t), aborted_{site(t)}(t)\}$ with $i < j$. If $\pi_j = aborted_{site(t)}(t)$ then,

by Criterion 1.(a) and Assumption 1.2, there will be no other action with parameter t in the $site(t)$ after $aborted_{site(t)}(t)$. If there is any other action with parameter t in other site $n \neq site(t)$, then by Criterion 1.(b) (and Property 1) the transaction is remote in such a site. By Assumption 1.2, any remote transaction for t starts with the action $apply_n(t, ws)$. By Criterion 4.(b) (Uniform Decision), $\beta|\{apply_n(t, ws)\colon n \in N, ws \in 2^V\}$ = empty; therefore, $\beta_t = begin_{site(t)}(t) \cdot aborted_{site(t)}(t)$ holds. If $site(t)$ is faulty either $\pi_j = aborted_{site(t)}(t)$ before $crash_{site(t)}$ and the property holds, or there is not $aborted_{site(t)}(t)$. In this last case either (2), with $\beta_t = begin_{site(t)}(t)$, or the sequence in (4) will be obtained.
(4). Let $\pi_i = begin_{site(t)}(t)$ in β. By the proof (2), if $site(t)$ is correct (no $crash_{site(t)}$ in β) then $\pi_j \in \{committed_{site(t)}(t), aborted_{site(t)}(t)\}$. Let $\pi_j = committed_{site(t)}(t)$ in β. By Criterion 4.(a) (Uniform Decision): $\forall n \in N\colon (\exists k\colon \pi_k \in \{commit_n(t, ws_t), apply_n(t, ws_t), crash_n\})$. By Criterion 1.(a) and Assumption 1.2, $\pi_{j_1} = commit_{site(t)}(t, ws_t)$ with $i < j_1 < j$ occurs in β. As t is local only in $site(t)$, then π_{j_1} is unique in β. Then, $\forall n \in N\colon n \neq site(t)\colon (\exists k\colon \pi_k \in \{apply_n(t, ws_t), crash_n\})$. Notice, that each possible $apply_n(t, ws_t)$ occurs after $deliverws_{site(t)}(t, ws_t)$ as Criterion 1.(c) indicates, and by Assumption 1.2, also occurs after $begin_{site(t)}(t)$. Let $\pi_k = apply(t, ws_t)$ in β. By Lemma 1, there is $\pi_{k_1} \in \{committed_n(t), crash_n\}$ with $k < k_1$. If $\pi_{k_1} = committed_n(t)$, then $i < k_1$. There is no other restriction in the potential ordering and number of $committed_n(t)$. If $crash_n$ happens, by Assumption 1.1 no event is provided by the site n. Therefore, $\beta_t = begin_{site(t)}(t) \cdot \gamma_{c_t}$ being γ_{c_t} a proper prefix of $committed_{n_1}(t) \ldots committed_{n_{|N|}}(t)$ with $(n_1, \ldots, n_{|N|})$ a permutation of the site identifiers. In the considered case γ_{c_t} includes $committed_{site(t)}(t)$.
If $site(t)$ is faulty ($crash_{site(t)}$ in β) either $\pi_j = committed_{site(t)}(t)$ in β before $crash_{site(t)}$ and the same result, $\beta_t = begin_{site(t)}(t) \cdot \gamma_{c_t}$, is obtained, or it does not occur in β. However, by Criterion 1.(c) may be $\pi_{i_3} = apply_n(t, ws)$ and $\pi_{i_2} = deliverws_{site(t)}(t, ws)$ with $i < i_2 < i_3$ in β. If there is not action $crash_n$ in β (this is possible by the fact that $|N| > f$) then, by Lemma 1, $\pi_{i_4} = committed_n(t)$ ($i < i_4$) and by Criterion 4.(a): $\forall n' \in N\colon (\exists k\colon \pi_k \in \{apply_{n'}(t, ws_t), crash_{n'}\})$. In this case, $\beta_t = begin_{site(t)}(t) \cdot \gamma_{c_t}$ but $committed_{site(t)}$ is not in γ_{c_t}. Finally, if no $appy_n(t, ws)$ occurs being n a correct site, then $\beta_t = begin_{site(t)}(t)$. □

We say that a *transaction $t \in T$ is committed in β* if β_t includes an action $committed_n(t)$ for some site $n \in N$. In β_t, the first site where the transaction t is committed is denoted as $n_1(t)$. Let FC (that stands for First Committed) be the set of actions $\{begin_{site(t)}(t), committed_{n_1(t)}(t)\colon t$ is a committed transaction in $\beta\}$. For each behavior $\beta \in behs(RDBS)$, we define the sequence $\beta_{FC} = \beta|FC$. In this ideal sequence, we assign to each transaction t as its readset the one obtained from its delegate replica (i.e., $rs_t \subseteq Snapshot(\beta|EDB_{site(t)}(i))$ where $\pi_i = begin_{site(t)}$ in $\beta|EDB_{site(t)}$); and, as its writeset the value of ws_t at the time it committed, respectively.

Lemma 2 *Let β be a behavior of $RDBS$. It holds that $log(\beta|EDB_n) \preceq log(\beta_{FC})$, for all $n \in N$.*

Proof: Let $\beta(j)$ be a finite prefix of β for some index $j \in \mathbb{Z}^+$. By induction over $j \geq 0$.

Basis. $j = 0$. $\beta(0)|EDB_n = \beta(0)|FC =$ empty, by Definition 1, $log(\beta(0)|EDB_n) = log(\beta(0)|FC) =$ empty.
Hypothesis. $j > 0$ and $log(\beta(j)|EDB_n) \preceq log(\beta(j)|FC)$.
Induction Step. We only consider the events π_{j+1} affecting the property.

- $\pi_{j+1} = committed_{n_1(t)}(t)$ and $n_1(t) = n$. By Hypothesis, $log(\beta(j)|EDB_n) \preceq log(\beta(j)|FC)$. The only possible case is $log(\beta(j)|EDB_n) = log(\beta(j)|FC)$.
Consider $log(\beta(j)|EDB_n) \prec log(\beta(j)|FC)$. There is at least one different element $\langle t', ws_{t'}\rangle$ in $log(\beta(j)|FC)$. Thus, $\beta(j)$ includes $\pi_{j'} = committed_{n_1(t')}(t')$ with $j' < j$. This action is also in $\beta(j)|FC$ but not in $\beta(j)|EDB_n$. By Criterion 3 (Uniform Prefix Order Database Consistency), there is some replica ($n_1(t') = n'$) $n' \neq n$ such that $log(\beta(j')|EDB_n) \prec log(\beta(j')|EDB_{n'})$. Then, $log(\beta(j)|EDB_n) \prec log(\beta(j)|EDB_{n'})$. By Definition 1, as $\beta(j+1)|EDB_n = \beta(j)|EDB_n \cdot \pi_{j+1}$ and $\beta(j+1)|EDB_{n'} = \beta(j)|EDB_{n'}$ then $log(\beta(j+1)|EDB_n) \not\preceq log(\beta(j+1)|EDB_{n'})$ that leads to a contradiction with Criterion 3. As a conclusion, $log(\beta(j)|EDB_n) = log(\beta(j)|FC)$. As $\beta(j+1)|EDB_n = \beta(j)|EDB_n \cdot \pi_{j+1}$

and $\beta(j+1)|FC = \beta(j)|FC \cdot \pi_{j+1}$. By Definition 1, $log(\beta(j+1)|EDB_n) = log(\beta(j+1)|FC)$ holds.

- $\pi_{j+1} = committed_{n_1(t)}(t)$ and $n_1(t) \neq n$. By Hypothesis, $log(\beta(j)|EDB_n) \preceq log(\beta(j)|FC)$. As $\beta(j+1)|EDB_n = \beta(j)|EDB_n$ and $\beta(j+1)|FC = \beta(j)|FC \cdot \pi_{j+1}$, then by Definition 1, $log(\beta(j+1)|EDB_n) \prec log(\beta(j+1)|FC)$ holds.

- $\pi_{j+1} = committed_{n_k}(t)$ and $n_k = n$, being $n_k \neq n_1(t)$. By Theorem 1, there exits $j' < j$ such that $\pi_{j'} = committed_{n_1(t)}(t)$ in $\beta(j)|FC$. This action is in $\beta(j)|FC$ but not in $\beta(j)|EDB_n$ (Criterion 1.(a), Assumption 1.2). By induction Hypothesis, $log(\beta(j')|EDB_n) \prec log(\beta(j')|FC)$ and also $log(\beta(j)|EDB_n) \prec log(\beta(j)|FC)$. Thus, as $\beta(j+1)|EDB_n = \beta(j)|EDB_n \cdot \pi_{j+1}$ and $\beta(j+1)|FC = \beta(j)|FC$, by Definition 1 $log(\beta(j+1)|EDB_n) \preceq log(\beta(j+1)|FC)$.

- $\pi_{j+1} = committed_{n_k}(t)$ and $n_k \neq n$, being $n_k \neq n_1(t)$. In this case, $\beta(j+1)|EDB_n = \beta(j)|EDB_n$ and $\beta(j+1)|FC = \beta(j)|FC$. Thus, trivially $log(\beta(j+1)|EDB_n) \preceq log(\beta(j+1)|FC)$ by induction Hypothesis.

Thus, in the limit $j \to \infty$: $log(\beta|EDB_n) \preceq log(\beta|FC)$ holds. □

We say that two behaviors β, β' of $RDBS$ are equivalent, $\beta \equiv \beta'$ if the following conditions hold:
- t is committed in $\beta \Leftrightarrow t$ is committed in β'. For each committed transaction t:
- rs_t in $\beta \Leftrightarrow rs_t$ in β'
- ws_t in $\beta \Leftrightarrow ws_t$ in β'
- $begin_{site(t)}(t) committed_{n_1(t)}(t) \preceq \beta_t \Leftrightarrow begin_{site(t)}(t) committed_{n'_1(t)}(t) \preceq \beta'_t$

In the replicated system, each transaction can be submitted to any replica (actually, $site(t)$ is an arbitrary mapping); every pattern of failures is possible except for the restriction $|N| > f$, and each EDB_n behaves in the same way, though independent of each other. In addition, Theorem 1 indicates that $n_1(t)$ is some non faulty site. Thus, for each $\beta \in behs(RDBS)$ there is some $\beta' \in behs(RDBS)$ such that $\beta \equiv \beta'$ being each $n_1(t)$ different to $n'_1(t)$. For each $\beta \in behs(RDBS)$ we define $\beta'|FC'$ such that $\beta \equiv \beta'$ and $FC' = \{begin_{site(t)}(t), committed_{n'_1(t)}(t) : t$ is a committed transaction in β' and $n'_1(t) = site(t)\}$. Finally, β_{1C} is obtained from $\beta'|FC'$ by renaming each action $begin_{site(t)}(t)$ as $begin(t)$ and $committed_{site(t)}(t)$ as $committed(t)$. The sequence β_{1C} is the one-copy version of β. In β_{1C}, the readset of t is rs_t of t in β and the writeset of t is ws_t of t in β.

Theorem 2 *Let β be a behavior of $RDBS$. β_{1C} verifies that for each transaction $t \in T$ such that $\pi_i = begin(t)$ and $\pi_j = committed(t)$ in β_{1C}, there exits an index $0 \leq s \leq i$ such that the two next conditions hold:*

1. $rs_t \subseteq Snapshot(\beta_{1C}(s))$
2. $\neg conflict(t', t, s, j, \beta_{1C})$ *for all* $t' \in T$

Proof: Let $\pi_i = begin(t)$. It verifies $rs_t \subseteq Snapshot(\beta'(i')|EDB_{site(t)})$ by definition of β_{1C} and the given equivalence $\beta \equiv \beta'$ being $\pi_{i'} = begin_{site(t)}$ in β'. By Lemma 2: $log(\beta'|EDB_{site(t)}) \preceq log(\beta_{1C})$. Therefore, by Definition 1 (Log) and Definition 2 (Snapshot), it follows that there is some index $0 \leq s \leq i$, such that $Snapshot(\beta'(i')|EDB_{site(t)}) = Snapshot(\beta_{1C}(s))$. The condition (1) holds for every t.
In order to prove the condition (2) we firstly prove the following: Consider there is $\pi_k = committed(t')$ such that $i < k < j$ and $ws_t \cap ws_{t'} \neq \emptyset$. Let $\pi_{k'} = committed_{site(t')}(t')$ and $\pi_{j'} = committed_{site(t)}(t)$ in β'. By Criterion 3: $log(\beta'(k')|EDB_{site(t)}) \prec log(\beta'(k')|EDB_{site(t')})$ and $log(\beta'(j')|EDB_{site(t')}) \prec log(\beta'(j')|EDB_{site(t)})$. This is because $\pi_{k'}$ and $\pi_{j'}$ are the first committed actions in the system. Therefore, the two previous conditions are only possible if $k' < j'$ and $\langle t', ws_{t'} \rangle$ is also in $log(\beta'(j')|EDB_{site(t)})$; i.e., $committed_{site(t)}(t')$ is before $committed_{site(t)}(t)$ in $\beta'(j')|EDB_{site(t)}$. There exists $\pi_{k''} = committed_{site(t)}(t')$ in β' such that $k' < k'' < j'$. In addition, as β_{1C} preserves the relative order of actions in β', $i' < k' < j'$ holds. Then, $i' < k' < k'' < j'$. In conclusion, $\beta'(j')|EDB_{site(t)}$ will contain $begin_{site(t)}(t)$, $committed_{site(t)}(t')$ and $committed_{site(t)}(t)$ in such order with $ws_{t'} \cap ws_t \neq \emptyset$. By Criterion 1.(a),

$\beta'|EDB_{site(t)} \in behs(EDB_{site(t)})$, and by Assumption 1.3 every committed transaction in $site(t)$ verifies the Snapshot Isolation level. A contradiction with the fact that $ws_t \cap ws_{t'} \neq \emptyset$ In conclusion: $\neg conflict(t', t, i, j, \beta_{1C})$ for all $t' \in T$.

If a transaction t verifies the condition (1) for $s = i$ then by the previous proof it also verifies the condition (2). Thus, we consider a transaction t such that condition (1) is verified for $0 \leq s < i$. In this case, we need to prove $\neg conflict(t', t, s, j, \beta_{1C})$ for every transaction t'. If condition (1) holds for $0 \leq s < i$, by Definition 2, it must be a transaction t' in β_{1C} such that $\pi_k = committed(t')$ with $s < k < i < j$ and $rs_t \cap ws_{t'} \neq \emptyset$. Thus, in β' there are the actions $\pi_{k'} = committed_{site(t')}(t')$, $\pi_{i'} = begin_{site(t)}(t)$, $\pi_{j'} = committed_{site(t)}(t)$ such that $k' < i' < j'$. Again by Criterion 3: $log(\beta'(k')|EDB_{site(t)}) \prec log(\beta'(k')|EDB_{site(t')})$ and $log(\beta'(j')|EDB_{site(t')}) \prec log(\beta'(j')|EDB_{site(t)})$. This is because $\pi_{k'}$ and $\pi_{j'}$ are the first committed actions in the system. Therefore, there exists $\pi_{k''} = committed_{site(t)}(t')$ in β' such that $k' < k'' < j'$. However, the $Snapshot(\beta'(i')|EDB_{site(t)})$ does not contain the version $\pi_{k''}$ produces. Thus, $k' < i' < k'' < j'$ holds. As $\beta'(j')|EDB_{site(t)}$ will contain $begin_{site(t)}(t)$, $committed_{site(t)}(t')$ and $committed_{site(t)}(t)$ in such order, then by By Criterion 1.(a), $\beta'|EDB_{site(t)} \in behs(EDB_{site(t)})$, and by Assumption 1.3 every committed transaction in $site(t)$ verifies the Snapshot Isolation level. In conclusion: $ws_{t'} \cap ws_t = \emptyset$ and $\neg conflict(t', t, s, j, \beta_{1C})$. Any other transaction t'' that is committed after π_k and before π_j falls between i' and j' in β' and also verifies $\neg conflict(t'', t, s, j, \beta_{1C})$. The Theorem holds. □