A Hierarchical Proof for the One-Copy Equivalence of a Replicated Database System with Crash Failures

J.R. Juárez-Rodríguez[1], J.R. González de Mendívil[1], I. Arrieta-Salinas[1]
J.E. Arméndariz-Íñigo[1], F.D. Muñoz-Escoí[2]

[1] Depto. de Ing. Matemática e Informática
Univ. Pública de Navarra
Campus de Arrosadía, s/n
31006 Pamplona (Spain)

[2] Inst. Tecnológico de Informática
Univ. Politécnica de Valencia
Camino de Vera, s/n
46022 Valencia (Spain)

{jr.juarez,mendivil,itziar.arrieta,enrique.armendariz}@unavarra.es, fmunyoz@iti.upv.es

Technical Report TR-ITI-SIDI-2010/006

A Hierarchical Proof for the One-Copy Equivalence of a Replicated Database System with Crash Failures

J.R. Juárez-Rodríguez[1], J.R. González de Mendívil[1], I. Arrieta-Salinas[1]
J.E. Arméndariz-Íñigo[1], F.D. Muñoz-Escoí[2]

[1] Depto. de Ing. Matemática e Informática
Univ. Pública de Navarra
Campus de Arrosadía, s/n
31006 Pamplona (Spain)

[2] Inst. Tecnológico de Informática
Univ. Politécnica de Valencia
Camino de Vera, s/n
46022 Valencia (Spain)

Technical Report TR-ITI-SIDI-2010/006

e-mail: {jr.juarez,mendivil,itziar.arrieta,enrique.armendariz}@unavarra.es,
fmunyoz@iti.upv.es

January 13, 2011

Abstract

Modern database replication protocols are based on the *deferred-update technique*, i.e. each transaction is initially executed in a single node (or delegate) and later its updates are propagated and applied in the rest of replicas. This introduces an asymmetric effort model, since the delegate replica has to execute all transaction sentences, but other replicas may receive a pre-processed writeset that can be easily applied. This model is able to boost both performance and scalability.

This paper thoroughly studies the deferred-update technique from a theoretical point of view. To this end, the I/O automaton model is used, developing a hierarchical correctness proof. At each abstraction level differerent properties are analysed and proved correct. To begin with, a new set of correctness criteria to ensure one-copy equivalence is proposed. Moreover, failures are considered throughout all abstraction levels, proving that failure management cannot be added a posteriori, as a patch to a protocol that was proven correct in a system where failures were not considered.

Additionally, our assumed deferred-update replication protocols are general enough to include the management of any isolation level and support integrity constraints in those levels where they make sense. Up to our knowledge, this is the first paper able to provide a correctness framework for all the replication protocols that encompass this variety of characteristics.

Acknowledgements

This work has been supported by EU FEDER and Spanish MICINN under research grant TIN2009-14460-C03.

1 Introduction

The design and implementation of fault-tolerant applications in a distributed system is a complex endeavour. If those applications are built in a modular way, then their algorithms may be generated following formal theoretical approaches and later proven correct with rigorous mechanisms. By making use of a modular design, each element provides a well-defined interface, specifying the operations available to other

application modules. Moreover, the specification of the properties satisfied by the operations of a module allows their usage by other application components regardless of how such operations are implemented.

The *I/O automaton* formal model [1] makes use of a modular approach in its strategy for distributed algorithms, generating *hierarchical correctness proofs* [2]. This hierarchy represents a series of system (or algorithm) descriptions from different abstraction levels. The topmost layer corresponds to a precise specification of the problem to be solved. Its next refinement provides a first raw solution to that problem; for instance, a centralized solution instead of the intended distributed one. Finally, the bottom layer implements an actual solution that takes into account all distribution and communication issues. The correctness proof of a given layer is based on checking the properties of its upper layer, thus simplifying the proving tasks. Additionally, the properties of a given layer are a guide for developing the lower layer. As a result, this method is similar to deriving a solution through a series of refinements.

This paper presents a hierarchical correctness proof for a replication protocol based on the *deferred-update* technique [3] for a crash-prone replicated database system.

In the last years, there have been many implementations of data replication protocols based on the deferred-update technique, since (i) such protocols can be easily implemented in a middleware over separate database management systems (DBMS), (ii) their performance is still good when compared to other previous techniques [4].

In a database replication system with a deferred-update protocol, each transaction starts its execution in a *delegate site*. All transaction operations are served by the database of its delegate site in a transparent way; i.e., the protocol does not need to be aware of them. When this sequence of operations is completed and the transaction commit is requested, the protocol intercepts such last request and starts the replication management. At this time, the protocol collects some information about that transaction. That information comprises, among other items, the transaction writeset, which is the set of $\langle item\_id, value \rangle$ pairs that have been created, written or deleted by the transaction. The protocol manages this information, involving all system sites. When the information is received at each site, a decision on the transaction's fate must be made: (i) in the delegate site, the protocol only needs to request a commit (or abort) operation to the underlaying DBMS to complete the transaction; and (ii) in other sites, and only if the transaction is accepted, the protocol should apply the writeset updates. In order to apply the writeset, a new light instance of the transaction is created in those sites. Such light instances are known as *remote transactions*.

We assume a *partially synchronous* distributed system and a *crash* [5] failure model[1]. Each system site has a local DBMS and a *full replication* model is followed; i.e., each database replica holds a full copy of the database and all replicas share the same schema. Transactions may be started at any time and at any site, and may read or write any item. Each DBMS guarantees the *ACID* properties [6] for its transactions. Transactions may be executed under any isolation level supported by the DBMSs. Thus, applications may combine sets of transactions with different isolation levels. Moreover, databases admit the declaration of integrity constraints (data invariants) that should be respected by all committed transactions.

Most replication protocols assume a single isolation level and do not consider integrity constraints in the databases, but we break this trend and provide support for the general case outlined in the previous paragraph. Two other important issues constitute the base for this work: (i) failures have been considered along the entire work, from its start at a high-level of abstraction to its end, where several implementation details have been included; and (ii) we have followed a strict separation of concerns: the replication protocol does not repeat any managing task that could be delegated to the underlying database or to additional facilities that could be implemented as other middleware components.

This paper shows that, requiring the appropriate properties to each system component, the resulting protocol is extremely simple. It is similar to the one employed for actively replicating distributed processes [7] using *atomic broadcast* [8] as its communication support. The protocol does not store any information about the items read or written by transactions, nor about the integrity constraints to be respected, and does not repeat any of the tasks already executed by the underlying databases. It simply schedules the order in which transactions should request their commitment (when local) or to be programmed as remote ones, waiting for an answer from the underlying DBMS.

[1] The model assumes that the underlaying communication service keeps track of the active sites and detects crashes in a transparent way.

1.1 Content of the Paper

Figure 1 shows the process that has been followed in the hierarchical correctness proof, from the most abstract replicated database system to the implementation of the replication protocol. As stated before, this approach is based on the I/O automaton model. In each step, the correctness proof of a given layer consists of checking that the properties satisfied by the upper layer are also satisfied in that layer. Note that each arrow in Figure 1 indicates that the properties of the lower module verify the properties of the upper module.

The top of the hierarchical proof is the most abstract system, the $RDBS$. It consists of the composition of an extended database module EDB_n at each site of the distributed system (with $n \in \mathcal{N}$, being $\mathcal{N}$ the set of system sites), along with a very abstract (non-distributed) deferred-update replication protocol called DRP. Each EDB_n module specifies a database that satisfies the ACID properties. In fact, this model allows transactions to be executed under different isolation levels. Moreover, the database may define integrity constraints.

The EDB_n module is intended for making the replication process easier. To this end, it provides the replication protocol with the necessary mechanisms to obtain information about each transaction that is ready to commit. Such information is basically the writeset and some control information regarding the isolation level of the transaction. In addition, the EDB_n module includes some basic properties that distinguish between local and remote transactions. The abstract protocol, represented by the DRP module, simply specifies that the creation of a remote transaction t at one EDB_n derives from a local transaction trying to commit at its delegate site $EDB_{site(t)}$, where $site(t)$ is the delegate site of the transaction. Both modules consider the possibility of crash failures at each site that would stop the execution at the crashed site. This system has no obligation to be correct and serves as the basic system over which other conditions must be imposed in order to guarantee its correctness.

The correctness of replicated database systems is traditionally subjected to the notion of one-copy equivalence. The main idea is that every transaction in the replicated system behaves as if it had been executed in a logical copy of the database maintaining its isolation level and respecting the integrity constraints in case it is committed. In order to define this logical behavior for transactions, the $1CDB$ module is defined as a reference for this equivalence notion. It defines a single copy database model, which should be equivalent to the $RDBS$. However, the basic properties of the $RDBS$ do not guarantee this at all. Therefore, the $RDBS$ requires some additional properties on its behaviors to find an equivalence relation between its behaviors and the ones of the $1CDB$. As proved in this work, these properties are necessary and sufficient conditions and hence they are considered as correctness criteria. As in any specification of a distributed system these conditions are safety and liveness ones. These criteria require that: (*C1 - local transaction progress*) every transaction that starts in its delegate site eventually gives a termination response (committed or aborted) at some site of the system unless that site crashes; (*C2 - uniform decision*) if a transaction is committed (aborted) at one site, then it cannot be aborted (committed) at other sites; (*C3 - uniform prefix order consistency*) for every two distinct sites, the sequence of committed update transactions at one site is a prefix of the sequence of the committed update transactions at the other site or vice versa providing that the writesets of remote and local transactions are the same; and (*C4 - non-contradiction*) if a remote transaction t is committed, then it does not conflict with any of the transactions that were committed at t's delegate replica between the beginning and commitment of t. Although these criteria are quite intuitive, they have never been formalized or proved as valid correctness criteria for such a general model of data replication based on deferred-update techniques. The uncertainty introduced by crash failures requires Criterion C4, which has not been discussed before despite its importance. Thus, the $RDBS_{CC}$ is the $RDBS$ with Criteria C1 to C4. The $RDBS_{CC}$ is one-copy equivalent to the $1CDB$, as proven in this paper.

The following step of the hierarchical proof tries to refine the initial EDB_n and DRP modules in order to compose a new system, called $RDBS^A$, which satisfies the aforementioned correctness criteria. For this purpose, the task of guaranteeing such criteria has to be distributed between the EDB_n and the DRP. Thus, the refinements EDB_n^A and DRP^A include some new properties but still satisfy the ones of the EDB_n and the DRP. This entails that the behaviors of $RDBS^A$ are still behaviors of the $RDBS$, since they satisfy the same set of properties. The model should refrain from putting the replication protocol in charge of tasks that could be handled directly by the database in a simpler and more efficient way. Thus, in this work, we have tried to avoid including properties in the DRP^A that could be easily implemented

by the EDB_n^A. Basically, Criterion C1 is guaranteed by the collaboration of both modules; Criterion C3 is responsibility of the DRP^A by requesting the commitment of local transactions or the application of the remote transactions in the same prefix order among the sites; Criterion C4 is under the responsibility of the EDB_n^A; and Criterion C2 is derived from the collaboration of both modules when executing the transactions.

In the last step of the proof, we just provide a concrete implementation of the DRP^A module, by means of the composition of an I/O automaton RP_n at each site and a group communication system module, denoted by AB, that satisfies the atomic broadcast properties. The final replication protocol is very simple, as it only propagates the information about each local transaction to the rest of sites by means of the AB module. As the atomic broadcast primitive delivers messages in the same order at all sites, transactions are applied at remote sites (or committed at delegate sites) following the same sequence at all sites.

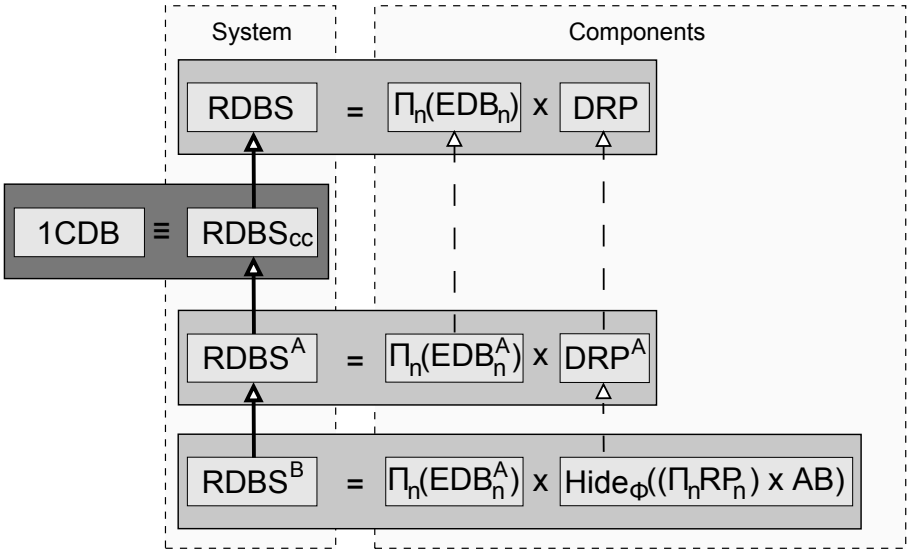

Figure 1: Organization of the hierarchical correctness proof.

1.2 Paper Roadmap

The rest of this paper is organized as follows. Section 2 introduces the specification framework used throughout this work. Section 3 explains the basic definitions for understanding single database systems. Section 4 specifies the abstract replicated database system that represents the top of the hierarchical proof. Section 5 formalizes the notion of one-copy equivalence. Section 6 is devoted to the necessary and sufficient conditions conforming the correctness criteria that must be imposed in order to achieve one-copy equivalence. In Section 7, some interesting aspects related to the correctness criteria are discussed. Section 8 includes a first refinement of the abstract replicated database system which satisfies the correctness criteria, whereas Section 9 provides a more concrete implementation of the distributed replication protocol. Finally, conclusions end the paper.

2 Specification Framework

This work describes a hierarchy of replicated database systems, whose properties are formalized by means of the *I/O automaton model* [1] with the aim to prove their correctness using a rigorous strategy. This model provides an intuitive and precise method for describing distributed systems and formulating accurate assertions about how systems behave. The I/O automaton model has been widely used for modeling and verifying distributed applications, such as concurrency control algorithms [9,10], distributed algorithms [2], network protocols [11] and data managers [12], among others.

An I/O automaton generates executions, which are alternating sequences of states and actions. Executions are assumed to be sequential, that is, actions are atomic and no two actions of the same automaton can occur simultaneously. A *behavior* is the subsequence of input and output actions of an execution, and it is independent of the internal state and actions of the automaton. Thus, an I/O automaton can be viewed as a "black box", characterized by a particular set of behaviors according to the properties of its specifications.

In order to promote a modular design, the description of each system component makes use of the properties of the rest of components regardless of any particular implementation. Thus, each component is described as an I/O automaton module. Each module M is specified by its external signature $sig(M)$ and a set of behaviors $behs(M)$ delimited by some specified safety and liveness properties. The bottom line of such a modular design is that the internal behavior of a module remains invisible to the rest of modules, whereas its external behavior is determined by the properties imposed on the module. The signature of a module M, $sig(M)$, consists of two different kinds of actions that allow M to communicate with other modules: input actions ($in(M)$) and output actions ($out(M)$). Thus, $sig(M) = (in(M), out(M))$. The set comprising all the possible actions of a module M is denoted by $acts(M) = in(M) \cup out(M)$.

Each behavior of a module M is a finite or infinite sequence of actions from $acts(M)$. The set of all acceptable behaviors of M is denoted by $behs(M)$. An infinite (finite) behavior $\beta \in behs(M)$ is denoted by $\beta = \pi_1 \cdot \pi_2 ... \pi_m ...$ ($\beta = \pi_1 \cdot \pi_2 ... \pi_m$) with $\pi_i \in acts(M)$. We say that π_i *is in* β if the i-th event in β is π_i, and that π *is in* β if there exists an index k such that $\pi_k = \pi$ and π_k is in β. For any $0 \leq j \leq |\beta|$ (where $|\beta|$ stands for the length of β), $\beta(j)$ represents the finite prefix (denoted '$\preceq$') of length j of β, i.e., $\beta(j) \preceq \beta$, $|\beta(j)| = j$ and $end(\beta(j)) = \pi_j$. By its definition, $\beta(0) =$ empty.

Let $\varphi \subseteq acts(M)$, $\beta|\varphi$ is the subsequence of β including only the actions of φ in β, i.e., $\beta|\varphi = \pi_{i_1} \cdot \pi_{i_2} ... \pi_{i_k} ...$ such that π_{i_k} is in β and $\pi_{i_k} \in \varphi$. Note that we can use the original indexes of the actions of β when working with actions of $\beta|\varphi$. When $\varphi = acts(M')$ for some module M', we can simply write $\beta|M'$.

In this paper, a replicated system is represented as the composition of a set of compatible modules [2]. A composition operation of several modules M_i whose signatures are compatible results in a module M which has a signature composed by the set of M_i signatures ($sig(M) = \Pi_{i \in I} sig(M_i)$) and a set of behaviors $behs(M)$ such that each behavior $\beta \in behs(M)$ satisfies that $\beta|M_i \in behs(M_i)$.

On the other hand, if a module M is a more detailed refinement of another module M', in order to ensure that M fulfills the requirements imposed on M', M must satisfy M' in the sense that $sig(M) = sig(M')$ and $behs(M) \subseteq behs(M')$. Therefore, the properties satisfied by M' will also be satisfied by M.

In this work, some actions of the presented modules have bounded parameters, e.g., $t \in T$ with T being the set of transactions, or $n \in \mathcal{N}$ with $\mathcal{N}$ being the set of sites of a distributed system. In the former case, we consider a mapping function, $trans\colon acts(M) \rightarrow T \cup \{udef\}$, such that $trans(\pi) = t$ if and only if action π has t as a parameter, or $trans(\pi) = udef$ otherwise. This mapping is also used for defining the set $acts(M, t) = \{\pi\colon \pi \in acts(M) \wedge trans(\pi) = t\}$, as well as the sequence of transaction identifiers of a prefix $\beta(j)$ such that $trans(\beta(j)) = trans(\beta(j-1)) \cdot trans(\pi_j)$ for $0 \leq j \leq |\beta|$ (being $trans(\beta(j)) = empty$ when $j = 0$).

Although some variables used in the formulation of the properties of the behaviors may be unbounded, it is understood that they are universaly quantified in their domains for the scope of the entire formulas, unless we explicitely specify them for a better comprehension.

3 A Database System Model

This Section presents the specification of a single database system with no failures, by means of a module denoted by DB. This module is used for introducing some basic definitions, notations and preliminary facts related to single database systems, which are used throughout this work.

3.1 Database Transactions

A *database* consists of a set of items that can be accessed by concurrent transactions. Let I be the set of database items. The set of possible values for each item $x \in I$ is represented by V_x. A transaction $t \in T$ (where T stands for the set of all possible transaction identifiers) is a sequence of read and write operations over the database items, starting with a $begin$ operation (denoted by $B(t)$) and ending with a $commit$ or $abort$ operation. Each operation op is actually a $request\_op/response\_op$ pair[2]. The response of a commit operation corresponding to a transaction t is either a $committed$ or an $aborted$ notification ($C(t)$ and $A(t)$ respectively), whereas the response of an $abort$ operation (i.e., a rollback request) always reports an $aborted$ notification.

If a transaction t completes a write operation on an item x by setting its value to v (denoted by $W(t, x, v)$) and is committed afterwards, a new version (x, v, t) is installed on the database[3]. Thus, a *version* (x, v, t) relates an item $x \in I$ to the value $v \in V_x$ installed by a commited transaction $t \in T$. Let $\mathcal{V}$ the set of all possible versions of the database. For each item $x \in I$, its initial version is the first version installed by the first committed transaction creating it. The model assumes that several versions of the same data item can be available in the database. Therefore, when a transaction t completes a read operation on an item x, it can get any version $(x, v, t') \in \mathcal{V}$ previously installed by t'. This is denoted by $R(t, (x, v, t'))$.

A behavior of a set of concurrent transactions is usually represented by an interleaved sequence of completed transaction operations with some defined restrictions that limit the set of valid behaviors in the database [14, 15]. Such a way of considering behaviors is captured in the following example:

Example 1. *Given three transactions $\{t_1, t_2, t_3\}$, a possible behavior is:*
$\triangleright\ \beta = B(t_1)\cdot W(t_1, x, v_1)\cdot C(t_1)\cdot B(t_2)\cdot W(t_2, x, v_2)\cdot B(t_3)\cdot R(t_3, (x, v_1, t_1))\cdot W(t_3, y, v_3)\cdot C(t_3)\cdot C(t_2)$
That is, t_1 writes a value v_1 on x, t_2 writes a value v_2 on x and t_3 reads the version (x, v_1, t_1) of x installed by t_1 and writes the value v_3 on y.

Given a transaction t taking place in behavior β, the *writeset* $ws_t(\beta)$ is characterized as the set of versions which t writes in β. Similarly, the *readset* $rs_t(\beta)$ represents the set of versions read by t in β. For instance, the readsets and writesets of the transactions of Example 1 are $rs_{t_1}(\beta) = rs_{t_2}(\beta) = \emptyset$, $rs_{t_3}(\beta) = \{(x, v_1, t_1)\}$, and $ws_{t_1}(\beta) = \{(x, v_1, t_1)\}$, $ws_{t_2}(\beta) = \{(x, v_2, t_2)\}$, $ws_{t_3}(\beta) = \{(y, v_3, t_3)\}$.

Let us note that $rs_t(\beta')$ and $ws_t(\beta')$ change dynamically over the prefixes β' of β. For instance, if we consider $\pi_6 = B(t_3)$ and $\pi_9 = C(t_3)$ in the behavior of Example 1, $rs_{t_3}(\beta(6)) = ws_{t_3}(\beta(6)) = \emptyset$ whereas $rs_{t_3}(\beta(9)) = \{(x, v_1, t_1)\}$ and $ws_{t_3}(\beta(9)) = \{(y, v_3, t_3)\}$.

A transaction t in β is said to be read-only if for all $j \geq 0$ $ws_t(\beta(j)) = \emptyset$; otherwise, it is called an update transaction.

The use of readsets and writesets removes the need to explicitly specify the individual read and write operations of each transaction in a behavior, as shown in the following example.

Example 2. *Given the transactions $\{t_1, t_2, t_3\}$ of Example 1, the behavior of Example 1 now turns into:*
$\triangleright\ \beta = B(t_1)\cdot C(t_1)\cdot B(t_2)\cdot B(t_3)\cdot C(t_3)\cdot C(t_2)$, *where* $rs_{t_1}(\beta) = \emptyset$, $rs_{t_2}(\beta) = \emptyset$, $rs_{t_3}(\beta) = \{(x, v_1, t_1)\}$, $ws_{t_1}(\beta) = \{(x, v_1, t_1)\}$, $ws_{t_2}(\beta) = \{(x, v_2, t_2)\}$ *and* $ws_{t_3}(\beta) = \{(y, v_3, t_3)\}$

It is worth noting that β is merely a syntactic sequence obtained from the set $\mathcal{A} = \{B(t), C(t), A(t) : t \in T\}$ whose semantics (the effects of committed transactions) can only be known by providing $ws_t(\beta)$ and $rs_t(\beta)$. In order to maintain a complete description without specifying the sequence of individual

[2] This notation has been used in other works, such as [12], where a data manager is so specified under the I/O Automaton Model.

[3] We assume that each transaction writes an item at most once. Therefore, each version of an item will correspond with the write made by the transaction that installed it in the database. This avoids considering anomalies such as "intermediate reads" [13] thus simplifying the model description.

read/write operations in a behavior, the existence of the following functions is assummed: $ws : T \times \mathcal{A}^\star \rightarrow \mathcal{V}$ and $rs : T \times \mathcal{A}^\star \rightarrow \mathcal{V}$. Thus, the description obtained for a behavior β is syntactically and semantically complete thanks to these functions. As mentioned before, the readset and writeset of a transaction may vary over the execution. However, there is some point of the execution at which these sets become meaningful and never change again from then on. At that point, they completely define the versions that the transaction has (or has tried to) read and write. The readset and writeset are defined in the following way:

$$rs_t(\beta(j)) \begin{cases} = \emptyset & \Leftrightarrow\ j = 0 \\ \in 2^{\mathcal{V}} & \Leftrightarrow\ (\pi_j \in \{C(t), A(t)\}) \wedge j > 0 \\ = rs_t(\beta(j-1)) & \Leftrightarrow\ \textit{otherwise} \end{cases}$$

$$ws_t(\beta(j)) \begin{cases} = \emptyset & \Leftrightarrow\ j = 0 \\ \in 2^{\mathcal{V}} & \Leftrightarrow\ (\pi_j \in \{C(t), A(t)\}) \wedge j > 0 \\ = ws_t(\beta(j-1)) & \Leftrightarrow\ \textit{otherwise} \end{cases}$$

This assumption does not impose any limitations on the model, since the properties that must be satisfied by correct behaviors are the ones that determine which writesets and readsets are valid. Some known concepts such as *view-equivalence* can be straightforwardly adapted to this representation. Two behaviors β and β' are view-equivalent if and only if for each $\pi_i = C(t)$ in β there exists $\pi_j = C(t)$ in β' such that $rs_t(\beta(i)) = rs_t(\beta'(j))$ and $ws_t(\beta(i)) = ws_t(\beta'(j))$, i.e., both behaviors contain the same committed transactions and these transactions have read and written the same versions.

Previous representations do not determine which is the information used for establishing whether a transaction can be committed or not. This control information may be the readset or the writeset themselves, it may be inferred from these sets (e.g. the items of the writeset) or it may even be related with the execution behavior. To represent this information in a general way, let us define a function $inf : T \times \mathcal{A}^\star \rightarrow \mathcal{E}$, whose contents will depend on the restrictions imposed to a particular execution. Therefore, the set $\mathcal{E}$ will be defined according to the kind of information that inf includes. Like $rs_t(\beta)$ and $ws_t(\beta)$, $inf_t(\beta)$ is defined at some point of β and from then on it does not change.

$$inf_t(\beta(j)) \begin{cases} = \emptyset & \Leftrightarrow\ j = 0 \\ \in \mathcal{E} & \Leftrightarrow\ (\pi_j \in \{C(t), A(t)\}) \wedge j > 0 \\ = inf_t(\beta(j-1)) & \Leftrightarrow\ \textit{otherwise} \end{cases}$$

The fact that most properties related to readsets, writesets and this last control information are associated with a concrete behavior allows us to omit the parameter β in $ws_t(\beta)$, $rs_t(\beta)$ and $inf_t(\beta)$ when it is clear in the context.

The main reason for this alternative notation is that the aim of this work is to study replication protocols based on the deferred-update technique, in which the interaction between the replication protocol and the database at a site is performed at a specific point of the transaction execution, when the protocol can obtain the writeset and the information required to decide on its outcome to apply changes at other sites. As a consequence, the protocol does not need to track every individual operation performed by the transaction. In fact, it should be possible to develop a database model from the same point of view in which this kind of protocols observe the execution of transactions in the database system.

3.2 Single Database Module

The single database module DB is defined by its action signature and the set of its possible behaviors in Figure 2. The set of transaction identifiers is denoted by T, whereas functions $ws : T \times behs(DB) \rightarrow \mathcal{V}$, $rs : T \times behs(DB) \rightarrow \mathcal{V}$ and $inf : T \times behs(DB) \rightarrow \mathcal{E}$ determine the writeset, the readset and the control information of each transaction $t \in T$ in a behavior $\beta \in behs(DB)$.

By means of action $B(t)$, the DB module notifies the event concerning the beginning of a new transaction t. Actions $C(t)$ and $A(t)$ represent the database's final decision on the transaction effects.

Definition 3.1. (Well-formed Behaviors) *A behavior β of $behs(DB)$ is well-formed if for each transaction $t \in T$ the sequence $\beta|acts(DB, t)$ is a prefix of one of the following sequences: $B(t) \cdot C(t)$ or $B(t) \cdot A(t)$.*

- Signature:
  $in(DB) = \emptyset$
  $out(DB) = \{B(t), C(t), A(t) : t \in T\}$
- A set $behs(DB)$ of *well-formed* (Definition 3.1) and *legal* (Definition 3.5) behaviors.

Figure 2: Module DB

This definition ensures that after a transaction t begins, it can only be either committed or aborted, and such actions can only appear at most once in a behavior.

The database specification is based on the *committed state* concept, also called $snapshot$. A database snapshot provides a view of the installed versions of the database items existing at a certain time in a behavior. In order to determine the versions that comprise the snapshot, the log of a behavior is defined as the ordered sequence of the writesets of commited update transactions.

Definition 3.2. (Log of DB) *Let β be a behavior of DB. For each prefix $\beta(j)$ of β, with $0 \leq j \leq |\beta|$, the log of $\beta(j)$ is defined as follows:*

$$log(\beta(j)) = \begin{cases} \textit{empty} & \Leftrightarrow\ j = 0 \\ log(\beta(j-1)) \cdot \langle ws_t \rangle & \Leftrightarrow\ (\pi_j = C(t) \wedge ws_t \neq \emptyset) \wedge j > 0 \\ log(\beta(j-1)) & \Leftrightarrow\ \text{otherwise} \end{cases}$$

The log represents the set of versions that have been persistently installed on the database. This can be seen as an abstraction that ensures data durability.

The latest version of an item x for a finite prefix $\beta(j)$ is the version of that item installed by the latest committed transaction which updated (or created) its value in that prefix. This *latest version* is used for defining the concept of database snapshot in Definition 3.3.

Definition 3.3. (Database Snapshot) *Let β be a behavior of DB. For each prefix $\beta(j)$ of β, with $0 \leq j \leq |\beta|$, the snapshot of $\beta(j)$ is defined as $\mathcal{S}(\beta(j)) = \bigcup_{x \in I} lastVer(x, \beta(j))$*

where $lastVer(x, \beta(j)) = \begin{cases} \{(x, v, t)\} \Leftrightarrow \exists i \colon 0 < i \leq j \colon (x, v, t) \in ws_t \wedge \langle ws_t \rangle = end(log(\beta(i))) \wedge \\ \qquad\qquad\qquad \forall k \colon i < k < j \colon \langle ws_{t'} \rangle \in end(log(\beta(k))) \Rightarrow (x, v', t') \notin ws_{t'} \\ \emptyset \qquad\qquad \text{otherwise} \end{cases}$

3.3 Legal Database Behaviors

A database management system must guarantee all the ACID properties [16] for each transaction: atomicity, consistency, isolation and durability. Considering that the $log(\beta)$ for a behavior β represents the durability of the writesets of the committed update transactions, what remains to be defined is when a behavior satisfies the rest of the properties. In order to guarantee atomicity, the model establishes that aborted transactions must never interfere with committed transactions, i.e., the operations of aborted transactions are appropriately rolled back. The presented definitions satisfy atomicity.

Real database management systems admit the definition of a variety of isolation levels under which transactions are executed. In addition, it is possible to specify a whole range of integrity constraints to maintain data consistency. Instead of assuming a specific isolation level for each transaction, the presented database model considers weak conditions from which a variety isolation levels can be derived (within the limits of the proposed mathematical formulation). The definitions of predicates $compatible()$, $conflict()$ and $consistent()$ allow us to achieve this degree of generality.

Definition 3.4. *Let β be a behavior of DB, $t', t \in T$ be two transactions and i, j be two indexes of β such that $0 \leq i < j \leq |\beta|$:*

- $compatible(t, i, \beta(j)) \Rightarrow rs_t \subseteq (\bigcup_{i \leq k \leq j} \mathcal{S}(\beta(k))) \cup ws_t$
- $conflict(t', t, i, \beta(j)) \equiv \exists k \colon i < k < j \colon \pi_k = C(t') \wedge P(inf_{t'}, inf_t)$

- $consistent(t, \beta(j)) \equiv \forall z\colon K_z(\mathcal{S}(\beta(j-1)), ws_t)$ *where* $K_z()$ *is an integrity constraint defined in the database.*

Predicate $compatible(t, i, \beta(j))$ shows that the versions that can belong to rs_t must have been installed on the database between indexes i and j of β or must be in ws_t. From its definition, if a transaction t reads nothing ($rs_t = \emptyset$), then $compatible(t, i, \beta(j))$ is always true.

On the other hand, $conflict(t', t, i, \beta(j))$ determines the conditions that may happen in the context of a transaction t between indexes i and j of β with regard to another transaction t' that may be concurrently committed in that context. If those conditions happen then the transaction t is unable to reach the committed status (see Definition 3.5 below). By its definition, if t' is not committed between i and j, then it will never conflict with t. Note that, in this case, the predicate becomes false. If this happens for every transaction $t' \in T$, then this entails that t has been executed completely isolated from the rest of transactions between i and j. Otherwise, the control information of the involved transactions, $inf_{t'}$ and inf_t, will determine if their isolation level permits them to be concurrently committed, by means of $P(inf_{t'}, inf_t)$ in $conflict(t', t, i, \beta(j))$.

Finally, $consistent(t, \beta(j))$ holds if and only if the writeset ws_t does not infringe any integrity constraint demanded by the database at j. Each constraint depends on the previous committed state (snapshot) of the database and the ws_t to be installed. Trivially, $consistent(t, \beta(j))$ is always true for read-only transactions ($ws_t = \emptyset$).

By making use of the aforementioned predicates, Definition 3.5 provides the obligations for every committed transaction in a *legal behavior*.

Definition 3.5. (Legal Behavior) *A behavior* β *of* DB *is legal, if for each transaction* $t \in T$ *such that* $\pi_i = B(t)$ *and* $\pi_j = C(t)$ *are in* β*, the following conditions hold:*

(a) $compatible(t, i, \beta(j))$

(b) $\neg conflict(t', t, i, \beta(j))$*, for all* $t' \in T$

(c) $consistent(t, \beta(j))$

Thus, Definition 3.5 establishes that if a transaction t is committed: (a) its readset is obtained from the committed states seen within its context; (b) there is no other transaction t' conflicting with t; and (c) all the integrity constraints hold at the time the transaction is committed.

In order to better understand the proposed database model, the following two examples are provided.

Example 3. *The aim of this example is to show how different isolation levels used in replicated systems can be seen as particular cases of Definition 3.5 by imposing some restrictions on the* $compatible()$ *and* $conflict()$ *predicates. We focus on the simplest case of* $P(inf_{t'}, inf_t)$*, in which its formulation only depends on the items of the writesets and/or readsets of transactions* t' *and* t *and the conflicts between transactions are caused only by non-empty intersections of these sets. In this case,* $inf_{t'} \subseteq I$ *and* $inf_t \subseteq I$*. Table 1 presents the definition of* $P(inf_{t'}, inf_t)$ *and the corresponding restrictions for* $compatible(t, i, \beta(j))$ *for each isolation level.*

The Serial level is the most restrictive one, as transactions conflict with any concurrent one that has committed ($P(inf_{t'}, inf_t) \equiv$ *true*)*. In this case, a legal behavior consists of a succession of transactions without interleaving among them. On the contrary, in Weak Read Committed there are no conflicts between concurrent transactions* ($P(inf_{t'}, inf_t) \equiv$ *false*)*. In this case, a legal behavior allows any interleaving among transactions.*

It is also possible to represent intermediate isolation levels, such as Snapshot Isolation or Dynamic-Serializable. The former takes into account conflicts generated by update transactions that concurrently try to write on the same data items: $P(inf_{t'}, inf_t) \equiv items(ws_{t'}) \cap items(ws_t) \neq \emptyset$*. In contrast, the latter considers conflicts between reads and writes:* $P(inf_{t'}, inf_t) \equiv items(ws_{t'}) \cap (items(ws_t) \cup items(rs_t)) \neq \emptyset$*.*

With regard to predicate $compatible()$*, all the definitions shown in Table 1 satisfy Definition 3.4, that is, a transaction is able to read its own writes and reads from committed states, although each case may impose additional restrictions to its general definition. Snapshot Isolation, Dynamic-Serializable and Serial allow transactions to read from a single concrete snapshot, whereas Weak Read Committed allows values to be read from any snapshot created during the transaction execution.*

	$P(inf_{t'}, inf_t)$ in $conflict(t', t, i, \beta(j))$	$compatible(t, i, \beta(j))$
Weak Read Committed [17]	false	$rs_t \subseteq \bigcup_{i \leq k \leq j} \{\mathcal{S}(\beta(k))\} \cup ws_t$
Snapshot Isolation [18, 19]	$items(ws_{t'}) \cap items(ws_t) \neq \emptyset$	$rs_t \subseteq (\mathcal{S}(\beta(i)) \cup ws_t)$
Dynamic-Serializable [18]	$items(ws_{t'}) \cap (items(ws_t) \cup items(rs_t)) \neq \emptyset$	$rs_t \subseteq (\mathcal{S}(\beta(i)) \cup ws_t)$
Serial	true	$rs_t \subseteq (\mathcal{S}(\beta(i)) \cup ws_t)$

Table 1: Predicates depending on isolation levels

In the proposed database model, transactions in T may have the same isolation level or not. As a result, the database can behave in a heterogeneous way, due to transactions being executed under different isolation levels. In spite of this, all committed transactions must satisfy Definition 3.5. It is worth noting that Definition 3.5 limits the possible values of the readsets and writesets of committed transactions. The following example displays a behavior with concurrent transactions executed under different isolation levels.

Example 4. *Let us consider four update transactions $\{t_1, t_2, t_3, t_4\}$ such that t_1 is executed under Weak Read Committed, t_2 under Snapshot Isolation, t_3 under Dynamic-Serializable, and t_4 under Serial. Assuming that transactions satisfy all the integrity constraints and they are compatible, one of the possible behaviors could be, as shown in Figure 3:*

$\triangleright\ \beta = B(t_1) \cdot B(t_2) \cdot B(t_3) \cdot B(t_4) \cdot C(t_2) \cdot A(t_4) \cdot C(t_3) \cdot C(t_1)$

Transaction t_1 is executed under Weak Read Committed, hence $conflict(t', t_1, 1, \beta(8))$ is false for any transaction t'. As for t_2, which is executed under Snapshot Isolation, $conflict(t', t_2, 2, \beta(5))$ is also false, because there is no transaction t' that commits between $\pi_2 = B(t_2)$ and $\pi_5 = C(t_2)$. Transaction t_3, which runs under Dynamic-Serializable, would make $conflict(t_2, t_3, 3, \beta(7))$ true in case $items(ws_{t_2}) \cap (items(ws_{t_3}) \cup items(rs_{t_3})) \neq \emptyset$, since $\pi_5 = C(t_2)$. Therefore, t_3 does not have any intersections with the writeset of t_2 because $\pi_7 = C(t_3)$. Finally, transaction t_4 can never be committed, as $conflict(t_2, t_4, 4, \beta(6))$ is true because $\pi_5 = C(t_2)$. When the execution ends, the log persistently contains all the writesets of the committed transactions, i.e., $log(\beta(8)) = \langle ws_{t_2} \rangle \cdot \langle ws_{t_3} \rangle \cdot \langle ws_{t_1} \rangle$. Let us note that each time a transaction is committed, a new snapshot is obtained.

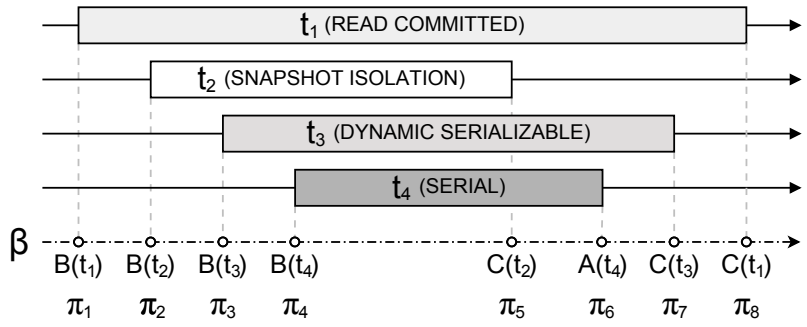

Figure 3: Example of a behavior with transactions executed concurrently under different isolation levels.

3.4 Generalized Legal Behavior

The definition of a legal behavior (see Definition 3.5) can be generalized in a very simple way to make it suitable for replicated settings. In a generalized legal behavior, a transaction is allowed to perform

operations with stale information about the database versions, as if it had been started before the time it actually did. This idea was originally introduced by [18] for the Snapshot Isolation level, under the name of Generalized Snapshot Isolation (GSI). We extend this notion to make it valid under other isolation levels.

Definition 3.6. (Generalized Legal Behaviors) *Let β be a behavior of $behs(DB)$. β is a generalized legal behavior, if for each transaction $t \in T$ such that $\pi_i = B(t)$ and $\pi_j = C(t)$ are in β, there exists $0 \leq s \leq i$ such that the following conditions hold:*

(a) $compatible(t, s, \beta(j))$

(b) $\neg conflict(t', t, s, \beta(j))$, *for all* $t' \in T$

(c) $consistent(t, \beta(j))$

Trivially, Definition 3.6 includes Definition 3.5 as a particular case when for every transaction $s = i$ holds. Moreover, if β is a generalized legal behavior, then there exists a legal behavior β' such that β is view equivalent to β'. Basically, the legal behavior β' is inductively built from β by moving action $\pi_i = B(t)$ in β to the position $\pi_s = B(t)$ of the transaction t when $i \neq s$ and adequately redefining the indexes of the new obtained behavior.

Remark 3.1. *All definitions introduced in this Section can be adapted for any other set of behaviors that satisfy the well-formedness in the sense given in Definition 3.1. These behaviors are obtained from a signature that includes $out(BD)$.*

4 An Abstract Replicated Database System

This Section provides the specification of an abstract replicated database system, represented by a module named $RDBS$. The components of this module, as depicted in Figure 4, are a replication protocol DRP and a group of extended databases called EDB_n, being n the site identifier. The set of site identifiers $\{1..N\}$ is denoted by $\mathcal{N}$. The system is crash-prone: sites may fail and stop their execution at any time. At this level of abstraction, there are no assumptions on the number of sites that may crash. The database at each site n executes transactions with independence from the rest of sites, leaving the replication protocol in charge of coordinating them. In the following, T represents the set of transaction identifiers in the $RDBS$.

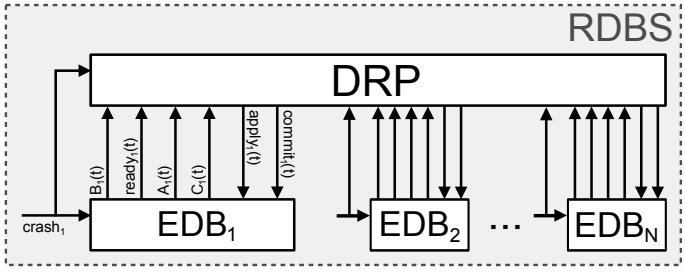

Figure 4: Replicated Database System.

This paper focuses on replication protocols based on the *deferred-update technique* [20]. In this approach, a transaction t performs all its operations on the database of the site where it starts, called *delegate site*. Transactions are said to be *local transactions* at their respective delegate sites, and *remote transactions* at the other sites. A transaction t about to be committed must be programmed on the databases of the remote sites. To this end, the protocol receives some data from the local transaction t, takes control of it and decides on the execution of the remote transactions of t at the rest of system sites. In the delegate site, the protocol may just request the commit of the transaction.

In database replication, it is convenient to avoid duplicating work that can be easily performed by databases. Several works [21–23] point out the convenience of providing databases with extended features

to simplify their replication[4]. These features are modeled by the EDB_n in an abstract way, regardless of their implementation. In fact, the EDB_n module can be interpreted as an extension of the DB module at site n. Both modules are essentially similar, but the EDB_n module presents some particular properties that model its extended operation. In particular, the EDB_n clearly distinguishes between local and remote transactions, and handles the remote ones in a special way.

All the extended databases in the system have the same set of items I and the same set of values $V_x \in \mathcal{V}$ for each item $x \in I$, as well as the same set of integrity constraints. Thus, they all have the same set of possible versions $\mathcal{V}$ for the set T. Under these conditions, full database replication is assumed.

In the replicated system, a local transaction and all its associated remote transactions share the same transaction identifier, although they are actually different transactions. This is possible because they are executed in different sites. In order to distinguish between local and remote transactions of a particular site n, there is a function $site\colon T \to \mathcal{N}$ such that $site(t)$ (the delegate site of t) is unique.

Assumption 4.1. (Unique Delegate Site) *For every transaction $t \in T$, it holds that* $site(t) = n \wedge site(t) = n' \Leftrightarrow n = n'$

There are no further assumptions restricting the way in which local transactions can appear in the system; therefore, they may begin anytime at any site and read/write any item under any isolation level.

4.1 Extended Database System

Figure 5 describes the module EDB_n for a site $n \in \mathcal{N}$. This module is intended for replicated settings and hence its specification is subject to its site identifier. Thus, functions $ws^n : T \times behs(EDB_n) \to \mathcal{V}$, $rs^n : T \times behs(EDB_n) \to \mathcal{V}$ and $inf^n : T \times behs(EDB_n) \to \mathcal{E}$ now determine the writeset, the readset and the control information of each transaction $t \in T$ in a behavior $\beta \in behs(EDB_n)$. In the following, let D be the set $2^{\mathcal{V}} \times \mathcal{E} \times (T \cup \{f_0\})$.

- Signature:
  $in(EDB_n)$=$\{crash_n, commit_n(t), apply_n(t, data)\colon t \in T, data \in D\}$
  $out(EDB_n) = \{B_n(t), ready_n(t, data), C_n(t), A_n(t)\colon t \in T, data \in D\}$
- A set of behaviors $behs(EDB_n)$ which satisfy Property 4.1, Property 4.2, Property 4.3 and Property 4.4

Figure 5: Module EDB_n

The EDB_n includes its own $crash_n$ input action in its signature to model the failure of site n. The following property indicates that after a $crash_n$ the EDB_n stops its activity and no further output actions are performed.

Property 4.1. (Execution Integrity) *For every behavior $\beta \in behs(EDB_n)$, it holds that* $\pi_i \in out(EDB_n) \Rightarrow \forall\, k\colon k < i\colon \pi_k \neq crash_n$.

The EDB_n notifies the beginning as well as the final outcome of a transaction t at site $n \in \mathcal{N}$ through the $B_n(t)$, $C_n(t)$ and $A_n(t)$ output actions.

After the beginning action $B_n(t)$ of a local transaction t with $site(t) = n$, the EDB_n can notify that the transaction t has no pending work left (and therefore, it is waiting for its commit) by means of action $ready_n(t, data)$ where $data \in D$. This action has two goals: (i) it states the point of the behavior at which the readset, writeset and control information of the transaction are defined; (ii) it allows to communicate the data of the transaction to the replication protocol.

Thus, when $\pi_j = ready_n(t, data)$ happens in a behavior β, as in the DB module, $rs^n_t(\beta(j))$, $ws^n_t(\beta(j))$ and $inf^n_t(\beta(j))$ become defined and never change again.

Action $ready(t, data)$ permits to pass some information from the transaction to the replication protocol by means of the $data$ paramater, which provides the writeset of the local transaction ($data.ws \in 2^{\mathcal{V}}$) as

[4] In [24] several mechanisms for the writeset extraction of a local transaction are presented and in [22] some ones are presented for remote transactions.

well as the control information $data.inf \in \mathcal{E}$, whose contents will depend on the isolation level of t. Besides, $data$ can provide some extra information about the transaction, $data.last \in T \cup \{f_0\}$, which will be discussed later when presenting one of the refinements of this module in Section 8.

Therefore, when $\pi_j = ready_n(t, data)$ happens in a behavior β, it explicitly provides the writeset and the information for the conflict evaluation of the transaction t at site n in that behavior; i.e., $data.ws = ws_t^n(\beta(j))$ and $data.inf = inf_t^n(\beta(j))$.

The extended database controls local transactions when they are started, and by executing $ready_n(t, data)$, the control of a local transaction t is transferred to the replication protocol, so that it can decide whether to commit it or not. The replication protocol requests the commit of the local transaction t via the input action $commit_n(t)$ to persistently install its changes on the EDB_n, if possible.

Property 4.2 defines the allowed behaviors of local transactions.

Property 4.2. (Local Transactions) *Let β be a behavior of EDB_n. For any transaction $t \in T$ such that $site(t) = n$:*

(1) *The sequence $\beta|acts(EDB_n, t)$ is a prefix of one of the following sequences:*

(a) $B_n(t) \cdot ready_n(t, data) \cdot commit_n(t) \cdot C_n(t)$ *for some* $data \in D$
(b) $B_n(t) \cdot ready_n(t, data) \cdot commit_n(t) \cdot A_n(t)$ *for some* $data \in D$
(c) $B_n(t) \cdot A_n(t)$

(2) $\pi_j = ready_n(t, data) \Rightarrow \forall k : k \geq j : rs_t^n(\beta(k)) = rs_t^n(\beta(j)) \wedge ws_t^n(\beta(k)) = ws_t^n(\beta(j)) \wedge inf_t^n(\beta(k)) = inf_t^n(\beta(j))$

(3) $\pi_j = ready_n(t, data) \Rightarrow data.ws = ws_t^n(\beta(j)) \wedge data.inf = inf_t^n(\beta(j))$

(4) $\pi_i = B_n(t) \wedge \pi_j = ready_n(t, data) \Rightarrow compatible(t, i, \beta(j))$

The first part of Property 4.2 ensures that each action of a local transaction appears at most once in a behavior $\beta \in behs(EDB_n)$ in the given order. The other parts of Property 4.2 provide some requirements that are fulfilled when $\pi_j = ready_n(t, data)$ is an action taking place in $\beta \in behs(EDB_n)$: the readset, writeset and control information of t are defined when this action happens; data from $data.ws$ and $data.inf$ match up with the values of the corresponding sets at time j, that is, $ws_t^n(\beta(j))$ and $inf_t^n(\beta(j))$; and the readset $rs_t^n(\beta(j))$ is compatible at j, i.e., $compatible(t, i, \beta(j))$.

As far as remote transactions are concerned, the input action $apply_n(t, data)$ of the EDB_n module is used by the replication protocol to program a transaction $t \in T$ with $site(t) \neq n$. The $data \in D$ parameter of action $apply_n(t, data)$ contains information related to transaction t at its delegate site, $site(t)$. Basically, it includes the writeset of the transaction $data.ws \in 2^{\mathcal{V}}$ and, depending on the isolation level, $data.inf \in \mathcal{E}$. Besides, $data$ provides some extra information about the transaction, such as $data.last \in T \cup \{f_0\}$. The EDB_n is responsible for programming that remote transaction in the underlaying database, in a transparent way to the replication protocol. The EDB_n, who is in charge of the transaction termination, may have to abort other transactions to guarantee its successful ending. However, in order to be independent of the replication protocol characteristics, any transaction can be aborted and hence a programmed remote transaction is not guaranteed to be committed.

Property 4.3 establishes the permitted behaviors of remote transactions. Again, this property states that each action of a remote transaction appears at most once in a behavior $\beta \in behs(EDB_n)$ in the given order.

Property 4.3. (Remote Transactions) *Let β be a behavior of EDB_n. For any transaction $t \in T$, such that $site(t) \neq n$:*

(1) *The sequence $\beta|acts(EDB_n, t)$ is a prefix of one of the following sequences:*

(a) $apply_n(t, data) \cdot B_n(t) \cdot C_n(t)$ *for some* $data \in D$
(b) $apply_n(t, data) \cdot B_n(t) \cdot A_n(t)$ *for some* $data \in D$

(2) $\pi_j = apply_n(t, data) \Rightarrow data.ws = ws_t^n(\beta(j)) \wedge data.inf = inf_t^n(\beta(j)) \wedge \forall k\colon k \geq j\colon ws_t^n(\beta(k)) = ws_t^n(\beta(j)) \wedge inf_t^n(\beta(k)) = inf_t^n(\beta(j)) \wedge rs_t^n(\beta(k)) = \emptyset$

When $\pi_j = apply_n(t, data)$ happens in a behavior β of the EDB_n, the module knows the writeset of the remote transaction t at site n; i.e., $ws_t^n(\beta(j)) = data.ws$. The writeset is explicitly indicated in that

action. Thus, if the remote transaction is committed, $data.ws$ is the writeset that must be installed in the database. However, the EDB_n enforces the remote transaction to read nothing, that is, $rs_t^n = \emptyset$ for remote transactions. The remaining information in $data$, $data.inf$ is used for checking the $conflict()$ predicate if it is required by the isolation level of t; i.e., $inf_t^n(\beta(j)) = data.inf$.

Note that $commit_n(t)$ and $apply_n(t)$ are input actions of the EDB_n, thus Property 4.2.1 and Property 4.3.1 are well-formedness conditions for the EDB_n.

Our purpose is to keep the notation from Definition 3.5 for the legal behaviors of the EDB_n by using the $compatible()$, $conflict()$ and $consistent()$ as well as their respective properties (Definition 3.4). This is possible by Remark 3.1, Property 4.2.1 and Property 4.3.1 from which transaction well-formedness is obtained. Property 4.4 states that the behaviors of EDB_n are legal behaviors.

Property 4.4. (Legal Behaviors) *Every behavior β of $behs(EDB_n)$ is a* legal *behavior. For each transaction $t \in T$ such that $\pi_i = B(t)$ and $\pi_j = C(t)$ are in β:*

(a) $compatible(t, i, \beta(j))$

(b) $\neg conflict(t', t, i, \beta(j))$, *for all* $t' \in T$

(c) $consistent(t, \beta(j))$

Let us note that by Property 4.2.4 $compatible(t, i, k, \beta)$ holds when $\pi_k = ready_n(t, data)$ and if $\pi_i = B_n(t)$ and $\pi_j = C_n(t)$ are in β, then $i < k < j$ holds by by Property 4.2.1. As the readset never changes after π_k (see Property 4.2.2), $rs_t^n(\beta(j)) = rs_t^n(\beta(k))$ holds when $\pi_j = C(t)$, and hence $compatible(t, i, \beta(j))$ holds too.

This module only covers the properties needed to present and prove the sufficient and necessary conditions required for the replicated database system to be one-copy equivalent. Later, in Section 8, a refinement of the extended database is presented, including some additional properties to study the correctness of a particular replication protocol.

4.2 Replication Protocol: Deferred-Update Technique

This Section presents the basic properties shared by the kind of protocols which are being dealt with in this work, i.e., the ones based on the deferred-update technique. The abstract deferred-update replication protocol is specified by the module DRP.

- Signature:
  $in(DRP) = \bigcup_{n \in \mathcal{N}} out(EDB_n) \cup \{crash_n : n \in \mathcal{N}\}$
  $out(DRP) = \{commit_n(t), apply_n(t, data) : t \in T, n \in \mathcal{N}, data \in D\}$
- A set of behaviors $behs(DRP)$ which satisfy Property 4.5

Figure 6: Module DRP

The signature of DRP is presented in Figure 6 along with the set of possible behaviors. Its signature must be compatible with the signature of each EDB_n, $n \in \mathcal{N}$ and, therefore, its inputs will be the EDB_n module outputs and vice versa. The $crash_n$ actions are also included as input actions in the DRP to model the crash failure of each site.

Property 4.5. (Deferred-Update) *For every behavior $\beta \in behs(DRP)$, it holds that:*

(1) $\pi_i \in \{commit_n(t), apply_n(t, data) : t \in T, data \in D\} \Rightarrow \forall k : k < i : \pi_k \neq crash_n$ *(execution integrity).*

(2) $\pi_i = commit_n(t) \Rightarrow \exists j : j < i : \pi_j = ready_n(t, data) \wedge \forall k : j < k < i : \pi_k \neq commit_n(t)$

(3) $\pi_i = apply_n(t, data) \Rightarrow \forall k : k < i : \pi_k \notin apply_n(t, data') \wedge site(t) \neq n$

(4) $\pi_i = apply_n(t, data) \Rightarrow \exists\, k : k < i : \pi_k = ready_{site(t)}(t, data) \wedge site(t) \neq n$

The main properties which characterize the behaviors of the deferred-update protocol can be easily identified. Its behaviors must be consistent with the ones of the extended databases, i.e., outputs generated by the DRP module must not break the execution integrity or the well-formedness of the inputs of the EDB_n module (Property 4.5(1-3)). Besides, deferred-update protocols can only apply a remote transaction as long as the transaction was ready to commit at its respective delegate replica (Property 4.5.4), thus avoiding the spontaneous creation of remote transactions in the system. The $data$ in the $apply_n(t, data)$ action contains the same information as in the $ready_{site(t)}(t, data)$ action.

Let us note that explicit abort requests have not been considered by the protocol, since they are not necessary as long as the EDB_n module manages the transactions in a correct way so that conflicting transactions are aborted when others are committed. Furthermore, if explicit abort requests were allowed, it would be possible to build a trivial protocol which would abort every transaction.

This module covers the general properties of a deferred-update replication protocol. It neither differentiates between sites, nor considers any properties about how processes communicate among them. Later, a series of successive refinements is provided in order to obtain the specification of a particular replication protocol based on the primitives of a group communication system.

4.3 Module Composition

As Figure 7 shows (see also Figure 4), the $RDBS$ module is the result of the module composition [2] between the replication protocol and the group of extended databases, one at each site of the distributed system: $RDBS = DRP \times (\Pi_{n \in \mathcal{N}}\ EDB_n)$. The signature of the $RDBS$ is well-defined, since the collection of signatures of the component modules is compatible [2].

- Signature:

  $in(RDBS) = \{crash_n : n \in \mathcal{N}\}$
  $out(RDBS) = \left(\bigcup_{n \in \mathcal{N}} out(EDB_n)\right) \cup out(DRP)$
- A set of behaviors $behs(RDBS)$ which satisfy $\beta|EDB_n \in behs(EDB_n)$ and $\beta|DRP \in behs(DRP)$.

Figure 7: Module RDBS

Each behavior β of the $RDBS$ is composed by the actions that transactions generate at different sites. Due to module composition [2], every behavior β of the $RDBS$ has to comply with the behavior of each EDB_n module and the DRP module, i.e., $\beta|EDB_n \in behs(EDB_n)$ and $\beta|DRP \in behs(DRP)$.

The well-formedness properties of the modules are also guaranteed in the composition of both of them. For instance, thanks to Property 4.5.2, Property 4.2.1 is satisfied. Similarly, Property 4.5.3 guarantees the behavior specified in Property 4.3.1.

The only input actions of system $RDBS$ are the actions in $\{crash_n : n \in \mathcal{N}\}$. Throughout this work, we assume that an action $\pi_i = crash_n$ can occur in a behavior $\beta \in beh(RDBS)$ at most once. In this way, and as a result of Property 4.1 and Property 4.5.1, once an action $\pi_i = crash_n$ happens in a behavior there are no more actions of site $n \in \mathcal{N}$, as formalized in the following Lemma:

Lemma 4.1. *Let β be a behavior of the $RDBS$. Then: $\pi_i = crash_n \Rightarrow \beta|EDB_n = \beta(i)|EDB_n$.*

Behavior $\beta|EDB_n$ satisfies Property 4.1, Property 4.2, Property 4.3 and Property 4.4, and $\beta|DRP$ satisfies Property 4.5. In some cases, we will just refer to the original properties of these modules, when necessary, without referring to the module composition properties too.

In the $RDBS$, a transaction $t \in T$ may span several sites. There is only one local transaction at $site(t)$ and, possibly, several remote transactions at the other sites. A remote transaction is directly related with its local transaction by the following causal dependency among their actions.

Theorem 4.1. *Let β be a behavior of the $RBDS$. If $\pi_j \in \{C_n(t), A_n(t)\}$, $site(t) \neq n$, is in β then there are four unique actions π_{i_1}, π_{i_2}, π_{i_3} and π_{i_4} in β, with $i_1 < i_2 < i_3 < i_4 < j$ such that, for some $data \in D$:*

- $\pi_{i_1} = B_{site(t)}(t)$ *and* $\pi_{i_2} = ready_{site(t)}(t, data)$ *are in* $\beta|EDB_{site(t)}$*, and*
- $\pi_{i_3} = apply_n(t, data)$ *and* $\pi_{i_4} = B_n(t)$ *are in* $\beta|EDB_n$

Proof. By the composition, $\beta|EDB_n \in behs(EDB_n)$, $\beta|DRP \in behs(DRP)$ and $\beta|EDB_{site(t)} \in behs(EDB_{site(t)})$. By Property 4.1, Property 4.3.1 for $\beta|EDB_n$; by Property 4.5.4 for $\beta|DRP$; and Property 4.1, Property 4.2.1 for $\beta|EDB_{site(t)}$, the Theorem holds. □

As a result, the following corollaries hold.

Corollary 4.1. *Let* β *be a behavior of* $RDBS$*, it holds that:*

$$\pi_i = C_n(t) \Rightarrow \exists r : r < i : \pi_r = ready_{site(t)}(t, data) \wedge ws_t^{site(t)}(\beta(r)|EDB_{site(t)}) = end(log(\beta(i)|EDB_n))$$

Proof. If $\pi_i = C_n(t)$ and $n = site(t)$, the Corollary holds by Property 4.2.1, Property 4.2.2 and the log definition (Definition 3.2). If $\pi_i = C_n(t)$ and $n \neq site(t)$, then by Theorem 4.1 there exists $\pi_k = apply_n(t, data)$ and $\pi_r = ready_{site(t)}(t, data)$ in β with $r < k < i$. By Property 4.2.3 and Property 4.2.2, $ws_t^n(\beta(k)|EDB_n) = ws_t^{site(t)}(\beta(r)|EDB_{site(t)})$. As $\pi_i = C_n(t)$, by the log definition (Definition 3.2) the Corollary holds. □

Thus, as remote transactions have the same writeset as the local transaction (i.e., $ws_t^n = ws_t^{site(t)}$), if a remote transaction t is committed at n, it will install the same writeset as the local transaction could install at $site(t)$. Hence, all transactions with the same identifier, local and remote, provide the same data updates at every replica.

Recall that we assumed that the delegate site of t, $site(t)$ is unique in the replicated system by Assumption 4.1. This implies that the first action of every transaction $t \in T$ in the replicated system can be only $B_{site(t)}(t)$, just as Corollary 4.2 states.

Corollary 4.2. *Let* β *be a behavior of* $RDBS$*. For each transaction* $t \in T$ *such that* $\pi_i = B_{site(t)}(t)$*, it is satisfied that* $B_{site(t)}(t) \preceq \beta|acts(RDBS, t)$.

Proof. By Property 4.2.1 and Theorem 4.1 the Corollary holds. □

The following remark allows us to ignore read-only transactions for the rest of the paper.

Remark 4.1. (Read-only transactions) *If* t' *is a read-only transaction at* $site(t')$ *then it does not appear at the* $log()$ *of* $site(t')$*. Every possible remote transaction of* t' *is an empty transaction by Property 4.3.2. Thus, it is not necessary to program a read-only remote transaction; and it is sufficient to program it at its* $site(t')$ *in a transparent way for the replication protocol. However, in order for a read-only transaction to be purely a local transaction, it is mandatory that read-only transactions* t' *satisfy* $\neg conflict(t', t, i, \beta(j)|EDB_{site(t')})$ *for any* t*. Under this assumption, no control information,* $inf_{t'}^{site(t')}$ *has to be sent to remote sites (this is a consequence of one of the correctness criteria for one-copy equivalence in this paper). The global atomicity of the read-only transaction is trivially guaranteed by Property 4.2.1. In addition, every global conclusion obtained for the readsets of update transactions is applicable to the readsets of read-only transactions. From now on, we consider that every transaction* $t \in T$ *is an update transaction.*

In the following, we will apply Remark 4.2, which is derived from the previous results, to simplify the notation and make the formal reasoning easier to follow.

Remark 4.2. (Notation convention) *In a behavior* β *of the* $RDBS$*, a transaction* $t \in T$ *has globally the following semantics:* $rs_t(\beta) = \bigcup_{n \in \mathcal{N}} rs_t^n(\beta|EDB_n)$, $ws_t(\beta) = \bigcup_{n \in \mathcal{N}} ws_t^n(\beta|EDB_n)$ *and* $inf_t(\beta) = \bigcup_{n \in \mathcal{N}} inf_t^n(\beta|EDB_n)$*. By Property 4.2.2, Theorem 4.1 and Property 4.3.2:* $rs_t(\beta) = rs_t^{site(t)}(\beta|EDB_{site(t)})$, $ws_t(\beta) = ws_t^{site(t)}(\beta|EDB_{site(t)})$ *and* $inf_t(\beta) = inf_t^{site(t)}(\beta|EDB_{site(t)})$*. This is possible because* $ready_{site(t)}(t, data)$ *is unique in* β *and defines the only possible semantics for the transaction* t*, considering that remote transactions of* t *do not read anything. From this point onwards,* $rs_t(\beta)$*,* $ws_t(\beta)$ *and* $inf_t(\beta)$ *will be the readset, writeset and control information of* t *no matter if it is a*

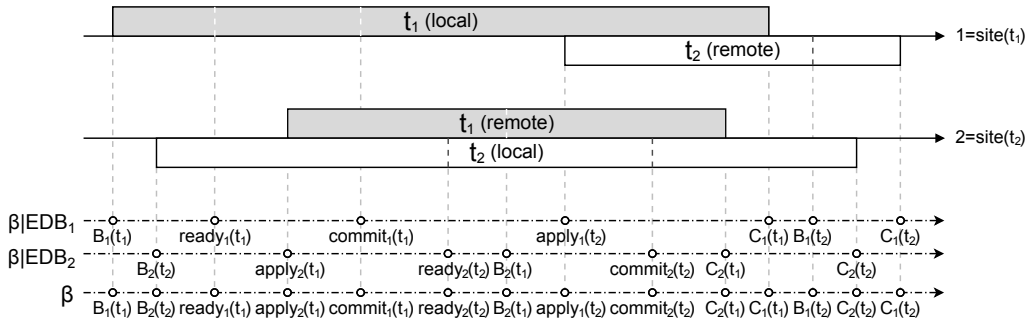

Figure 8: An example of a behavior of the RDBS module.

local or a remote transaction. Moreover, parameter β can be ommitted when it is clear within the context. This simplification is possible since these sets will be properly used in the proofs and they always use rs_t in a $compatible()$ predicate in reference to the $site(t)$. In addition, we also omit the $data$ parameter in $ready_n(t, data)$ and $apply_n(t, data)$ actions unless strictly necessary. This is possible by the Theorem 4.1 and the fact that under the previous convention $ws_t = data.ws$ and $inf_t = data.inf$ in both actions.

Finally, this Section concludes with an example of a behavior of the $RDBS$ module that illustrates how the module composition works.

Example 5. *Figure 8 presents a behavior of the $RDBS$, $\beta \in behs(RDBS)$, generated by two transactions t_1 and t_2 both executed under the Dynamic-Serializable level at two replicas 1 and 2. Transaction t_1 is local at 1 and remote at 2 and t_2 is local at 2 and remote at 1. $\beta|EDB_1$ and $\beta|EDB_2$ are the sequences of actions executed at sites 1 and 2 respectively and they must be behaviors of each EDB_n by the properties of the composition. Thus, as t_1 is committed at $\beta|EDB_2$ between $B_2(t_2)$ and $C_2(t_2)$ and t_2 is not aborted, then $items(ws_{t_1}) \cap items(ws_{t_2}) = \emptyset$ and $items(ws_{t_1}) \cap items(rs_{t_2}) = \emptyset$ must hold to satisfy Property 4.4.b and hence $\beta|EDB_2 \in behs(EDB_2)$, i.e., it is a legal behavior. The same happens at EDB_1 regarding local transaction t_1 and remote transaction t_2. Similarly, $\beta|DRP \in behs(DRP)$ must also hold and thus Property 4.5.4 must also hold in β. This forces $apply_2(t_1)$ and $apply_1(t_2)$ to appear at their remote sites after $ready_1(t_1)$ and $ready_2(t_2)$ at their local sites respectively.*

The $RDBS$ is just the composition of $N \geq 1$ extended databases with a very abstract replication protocol, $(\prod_{n \in \mathcal{N}} EDB_n) \times DRP$. In the $RDBS$ there are not any other global properties for local and remote transactions apart from the ones given in this Section. Then, any pattern is possible, e.g, although a remote transaction is committed, its local transaction may be aborted or may not give any response by the effect of a crash. Therefore, other global conditions are demanded to get a *correct* replicated database system.

5 One-Copy Equivalence

The $RDBS$ is just the composition of $N \geq 1$ extended databases with a very abstract replication protocol, $(\prod_{n \in \mathcal{N}} EDB_n) \times DRP$. In the $RDBS$ there are not any other global properties for local and remote transactions apart from the ones given in the previous Section. Then, any pattern is possible, e.g, although a remote transaction is committed, its local transaction may be aborted or may not give any response by the effect of a crash. Therefore, other global conditions are demanded to get a *correct* replicated database system.

In general, given a list of conditions φ, the $RDBS_\varphi$ module is defined as: $sig(RDBS_\varphi) = sig(RDBS)$ and $behs(RDBS_\varphi) = \{\beta : \beta \in behs(RDBS)$ and β satisfies all the conditions in $\varphi\}$. Any $RDBS_\varphi$ is called a (refined) module of the $RDBS$, since $behs(RDBS_\varphi) \subseteq behs(RDBS)$. Among all possible $RDBS_\varphi$, we need to determine which of them are correct, in other words, we must find the conditions imposed by φ that make the $RDBS$ module a correct system.

The correctness criterion commonly used to prove that a replicated database system works correctly is the one-copy equivalence notion [14]. Its main idea sets that the transactions executed in the replicated system must behave as if they were executed in a one logical copy of the database. This one-copy database is again an abstract view for a given $RDBS_\varphi$. Following this notion, a committed or aborted transaction is also committed or aborted in the one-copy database, but if no response is produced for a transaction which started in the replicated system because of a crash, then the same happens in the one-copy database. In this paper, the one-copy database attempts to provide an explanation for each transaction in a behavior of the $RDBS_\varphi$. Thus, the one-copy equivalence considers that transactions commit, abort or otherwise give no answer because the site has failed. This general one-copy equivalent model accurately reflects the actual behavior of all transactions in the replicated system.

On the other hand, when designing a replicated database system, properties satisfied by the databases and the replication protocol are engineered to obtain such equivalence. Designing a replicated system considering failures with the minimum properties that are necessary and sufficient conditions to establish one-copy equivalence would define the correctness criteria of the system. This is one of the objectives of our work, i.e., finding the necessary and sufficient conditions φ that must be fulfilled by the system specified by the $RDBS_\varphi$ in order to be one-copy equivalent.

5.1 The 1CDB Module

The $1CDB$ module is defined in Figure 9. The $1CDB$ bears some similarity to the DB module presented in Section 3. Its signature is quite similar to the DB module except for the $crash$ action included in this case as input action to consider the effect of failures. The $crash$ action included in the $1CDB$ is not exactly the same as the $crash_n$ action of the EDB_n module (see Property 4.1, execution integrity). As there are N replicas, in the $1CDB$ there are at most N chances to get the system down in the same way as in the $RDBS$ module. The $1CDB$ module is the most abstract specification of a replicated database system; therefore, it includes not only safety properties but also liveness properties in its behaviors. The $1CDB$ is used to prove that the behaviors of an $RDBS_\varphi$ are somehow *view-equivalent* to the behaviors of the $1CDB$. As it was done in other previous modules, T is the set of transaction identifiers and the functions $ws : T \times behs(1CDB) \to \mathcal{V}$, $rs : T \times behs(1CDB) \to \mathcal{V}$ and $inf : T \times behs(1CDB) \to \mathcal{E}$ determine the writeset, the readset and the control information of each transaction $t \in T$ in a behavior $\beta \in behs(1CDB)$.

- Signature:
  $in(1CDB) = \{crash\}$
  $out(1CDB) = \{B(t), A(t), C(t) \colon t \in T\}$
- A set $behs(1CDB)$ such that every $\gamma \in behs(1CDB)$ is a ***well-formed***, ***N-crash-stop***, ***generalized legal*** and ***progressive*** behavior.

Figure 9: Module 1CDB

The set of the behaviors γ are characterized by four properties, which are analyzed in the following:

- γ is a ***well-formed*** behavior; i.e., $\gamma | acts(1CDB, t)$ is a prefix sequence of $B(t) \cdot C(t)$ or $B(t) \cdot A(t)$. The transaction can only be committed or aborted after it begins, and this action only happens once in γ.

- γ is an ***N-crash-stop*** behavior. This simply states that after N crashes in γ the $1CDB$ module stops producing any output: If $|(\gamma | \{crash\})| = N$ and $\pi_j = crash$ is the last $crash$ in γ, then $\gamma = \gamma(j)$.

- γ is a ***generalized legal*** behavior in the sense given by Definition 3.6.

- γ is a ***progressive*** behavior. If $\pi_i = B(t)$, then $\exists j \colon j > i \colon \pi_j \in \{C(t), A(t), crash\}$. Therefore, a transaction t that begins but executes neither $C(t)$ nor $A(t)$ indicates the occurrence of a $crash$ action.

Let us note that the last property for γ is a liveness property. Every formal specification requires such kind of properties since a system in which nothing happens is always safe. The fact that the $1CDB$ module considers generalized legal behaviors instead of legal behaviors is because in a replicated (asynchronous distributed) setting, the updates of remote transactions may happen at different times in different sites. Thus, a transaction that begins in a site may not see the most current versions of the database items in the whole system, and the transaction may be committed locally by working with stale versions of those items.

5.2 One-copy Equivalence Definition

In an $RDBS_\varphi$ module, a transaction $t \in T$ may appear in $\beta \in behs(RDBS_\varphi)$ operating as either a local transaction or a remote transaction. However, in the $1CDB$ module each transaction $t \in T$ can only appear once without making reference to any site. Thus, it is necessary to relate the actions of a transaction t in both modules and also their semantics, i.e., the readset, writeset and control information. Consequently, in order to study the one-copy equivalence we have to define a relation between the behaviors of an $RDBS_\varphi$ and the $1CDB$.

Definition 5.1. (Legal Relation) *Let $RDBS_\varphi$ be a module of $RDBS$. Let Γ be a relation in $behs(RDBS_\varphi) \times behs(1CDB)$. Γ is a legal relation if for each $\beta \in behs(RDBS_\varphi)$ there exists at least a $\gamma \in behs(1CDB)$ such that:*

(1) $rs_t(\beta) = rs_t(\gamma)$, $ws_t(\beta) = ws_t(\gamma)$ *and* $inf_t(\beta) = inf_t(\gamma)$

(2) $\exists n \in \mathcal{N}$: $B_n(t)$ *is in* $\beta \Leftrightarrow B(t)$ *is in* γ

(3) $\exists n \in \mathcal{N}$: $C_n(t)$ *is in* $\beta \Leftrightarrow C(t)$ *is in* γ

(4) $\exists n \in \mathcal{N}$: $A_n(t)$ *is in* $\beta \Leftrightarrow A(t)$ *is in* γ

(5) $|(\beta|\{crash_n : n \in \mathcal{N}\})| = |(\gamma|\{crash\})|$

To define the relation in a more general way, Definition 5.1 permits to choose arbitrarily the order of the actions in γ regardless of the order established by β for these actions. By its definition, the image of β by the legal relation Γ, denoted $\Gamma(\beta)$, satisfies $\Gamma(\beta) \subseteq behs(1CDB)$. Thus, each behavior in $\Gamma(\beta)$ must be well-formed, N-crash-stop, generalized legal and progressive (as the behaviors of $1CDB$ were defined) and contain all the transactions that were committed, aborted or did not provide any response in each behavior.

Example 6. *Regarding Example 5, shown in Figure 8, we can find several examples of $\gamma \in behs(1CDB)$ satisfying Definition 5.1:*

$$\gamma_1 = B(t_1) \cdot B(t_2) \cdot C(t_1) \cdot C(t_2),\ \gamma_2 = B(t_2) \cdot B(t_1) \cdot C(t_1) \cdot C(t_2),$$
$$\gamma_3 = B(t_1) \cdot B(t_2) \cdot C(t_2) \cdot C(t_1),\ \gamma_4 = B(t_2) \cdot B(t_1) \cdot C(t_2) \cdot C(t_1),$$
$$\gamma_5 = B(t_1) \cdot C(t_1) \cdot B(t_2) \cdot C(t_2),\ \gamma_6 = B(t_2) \cdot C(t_2) \cdot B(t_1) \cdot C(t_1)$$

However, as both transactions are executed under the Dynamic Serializable isolation level, if they also hold that $items(ws_{t_2}) \cap items(rs_{t_1}) \neq \emptyset$, then γ_4 and γ_5 are not behaviors of $1CDB$ since in this case they are not generalized legal behaviors (Definition 3.6).

This is a specific example for the β presented in Example 5. A legal relation (Definition 5.1) requires this to be possible for every $\beta \in behs(RDBS_\varphi)$.

Therefore, since transactions in $\gamma \in \Gamma(\beta)$ have the same readset rs_t, writeset ws_t and inf_t as in β, this legal relation can be somehow considered as an one-copy equivalence notion between a system characterized by $behs(RDBS_\varphi)$ and the $1CDB$ module. This allows us to define the one-copy equivalence between an $RDBS_\varphi$ module and the $1CDB$ module.

Definition 5.2. (One-Copy Equivalence) *Let $RDBS_\varphi$ be a module of $RDBS$. The $RDBS_\varphi$ module is one-copy equivalent to the $1CDB$ module if and only if there exists a legal relation $\Gamma \subseteq behs(RDBS_\varphi) \times behs(1CDB)$.*

6 Necessary and Sufficient Conditions for One-Copy Equivalence

After having explained the conditions that make an $RDBS_\varphi$ module one-copy equivalent in the previous Section, we will determine the set of properties φ that have to be imposed on the behaviors of the $RDBS_\varphi$ module to provide one-copy equivalence. As it will be proven, the proposed properties are necessary and sufficient to guarantee one-copy equivalence. For this reason, they are correctness criteria. In the following, we present and explain these correctness criteria.

Criterion 1. (C1: Local Transaction Progress) *For every behavior $\beta \in behs(RDBS)$, the following holds:* $\pi_i = B_{site(t)}(t) \Rightarrow \exists n\colon n \in \mathcal{N}\colon \exists\, k\colon k > i\colon \pi_k \in \{C_n(t), A_n(t), crash_n\}$

Criterion C1 indicates that if a transaction begins its execution, then it will be committed or aborted at least at one site, or some site will crash otherwise. This entails that if a transaction begins and does not provide any output then there must have been at least one crash in the system. Note that, despite ensuring some kind of progress of transactions in the replicated system, this criterion does not imply any local progress of the transaction at its delegate replica.

Criterion 2. (C2: Uniform Decision) *For every behavior $\beta \in behs(RDBS)$, it holds that:*

(1) $\pi_i = C_n(t) \Rightarrow \forall n'\colon n' \in \mathcal{N}\colon \forall\, k\colon \pi_k \neq A_{n'}(t)$

(2) $\pi_i = A_n(t) \Rightarrow \forall n'\colon n' \in \mathcal{N}\colon \forall\, k\colon \pi_k \neq C_{n'}(t)$

Criterion C2 considers committed and aborted transactions separately. C2.1 states that if a transaction is committed at one site (either correct or faulty), it cannot be aborted at any site, even the former one. Similarly C2.2 ensures that if a transaction is aborted at some site, then it cannot be committed at any site. Thus, C2 guarantees that the decision on the outcome of a transaction has to be the same at every site which has made a decision and that the transaction will not be both committed and aborted at different sites.

Criterion 3. (C3: Uniform Prefix Order Consistency) *For every behavior $\beta \in behs(RDBS)$, it holds that $log(\beta(j)|EDB_n) \preceq log(\beta(j)|EDB_{n'})$ or vice versa, for every $\beta(j) \preceq \beta$.*

Criterion C3 forces the system to build the same snapshots at all the databases. In fact, the same commit ordering must be followed at all sites by all committed update transactions, not only by the conflicting ones, i.e., $trans(\beta(j)|\{C_n(t)\colon t \in T\}) \preceq trans(\beta(j)|\{C_{n'}(t)\colon t \in T\})$. Recall that $trans(\beta(j)) = trans(\beta(j-1)) \cdot trans(\pi_j)$ for $0 \leq j \leq |\beta|$, where $trans(\pi) = t$ if and only if π has t as a paramter (or $trans(\pi) = udef$ otherwise). Every remote transaction t will install, if possible, the same writeset ws_t as the one defined for the local transaction (see Corollary 4.1). Note also 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.

When it comes to considering crash failures, the previous criteria may not avoid some undesirable behaviors of the replicated system. For example, if a transaction begins at its delegate site, notifies that it is ready to commit and then the site fails, none of the previous criteria will prevent its changes from being committed elsewhere by a remote transaction, even when the transaction, according to the previous criteria, has been aborted locally by another conflicting transaction, if the site had not crashed and all transactions had been committed in the same order at every replica. In other words, the behavior of a remote transaction has to be equivalent to the one of its local transaction even if it has not been able to notify its termination due to a crash. Criterion C4 avoids such potential undesirable behaviors.

To simplify the formulation of C4, we define $last(i, n, \beta)$ as the last transaction which has committed in a site n before an action π_i in a behavior $\beta \in behs(RDBS)$[5].

Definition 6.1. (Last Transaction) *Let β be a behavior of $RDBS$. The last committed transaction of β at a site n before π_i is defined as follows:*

$$last(i, n, \beta) = \begin{cases} t_{last} & \Leftrightarrow \exists j\colon j < i\colon \pi_j = C_n(t_{last}) \wedge \forall k\colon j < k < i\colon \pi_k \notin \{C_n(t)\colon t \in T\} \\ f_0 & \text{otherwise} \end{cases}$$

[5] In other parts of the paper, we also use this notation for behaviors $\beta \in behs(EDB_n)$, since the definition is also valid for such behaviors.

By Definition 6.1, either there exists a transaction t_{last} which is the last committed one just before action π_i in β, or there does not exist a previous committed transaction yet. In the latter case, in order to simplify the notation, we assume that for all $n \in \mathcal{N}$, if $\pi_j = C_n(f_0)$ then $j = 0$; i.e., a fictitious transaction f_0 has been committed at every site at the initial point.

Criterion 4. (C4: Non-Contradiction) *For every behavior $\beta \in behs(RDBS)$, the following holds:* $\pi_i = B_{site(t)}(t) \wedge \pi_j = C_{n'}(last(i, site(t), \beta)) \wedge \pi_k = C_{n'}(t) \wedge n' \neq site(t) \Rightarrow \forall t'' \in T\colon \neg conflict(t'', t, j, \beta(k)|EDB_{n'})$.

C4 prevents transactions that conflict with a transaction t which was local at a site n from being committed at another site n' between the last committed transaction at n when t began and the commit of t. Therefore, crash uncertainty is avoided, as a transaction t from a crashed site $site(t)$ would not be allowed to be committed, if any other concurrent transactions that should be committed before t conflict with it.

It is worth noting that Criterion C4 is only necessary when isolation levels may cause conflicts. Thus, if all transactions were executed under the Weak Read Committed isolation level (see Table 1), Criterion C4 would not be necessary, as it would be trivially satisfied. However, we cannot ignore Criterion C4, as the model does not assume a concrete isolation level.

Section 6.1 and Section 6.2 detail the proof that these criteria are necessary and sufficient conditions for the proposed model.

6.1 Proof of Necessity

In order to study whether these criteria are necessary conditions to get the $1CDB$ equivalent system, we will prove that such equivalence is not possible when supposing that each of the criteria does not hold separately. From Definition 5.2 an $RDBS_\varphi$ is not one-copy equivalent to $1CDB$ if there does not exist a legal relation Γ, i.e., you can find at least one $\beta \in behs(RDBS_\varphi)$ such that any γ obtained using the conditions of Definition 5.1 from any possible relation satisfies $\gamma \notin behs(1CDB)$.

Theorem 6.1. *Let $RDBS_\varphi$ be a module of $RDBS$. If Criterion C1 does not hold in $behs(RDBS_\varphi)$, then $RDBS_\varphi$ is not one-copy equivalent to $1CDB$.*

Proof. By contradiction. There exists a legal relation Γ such that $RDBS_\varphi$ is one-copy equivalent to $1CDB$. If C1 does not hold, then there exists $\beta \in behs(RDBS_\varphi)$ such that for some $t \in T\colon \pi_i = B_{site(t)}(t) \wedge \forall n \in \mathcal{N}\colon \forall k\colon k > i\colon \pi_k \notin \{C_n(t), A_n(t), crash_n\}$. Among the possible behaviors that fulfill the previous condition, there also exists a behavior β' such that $\forall k : k < i : \pi_k \notin \{crash_n : n \in \mathcal{N}\}$. Therefore, $|(\beta'|\{crash_n : n \in \mathcal{N}\})| = 0$. Since Γ is a legal relation, $\gamma \in \Gamma(\beta')$ holds that $|(\gamma|\{crash\})| = 0$. For transaction t and $\gamma \in \Gamma(\beta')$, $\gamma|\{B(t), C(t), A(t), crash\} = B(t)$. Then, for all $\gamma \in \Gamma(\beta)$, $\gamma|\{B(t), C(t), A(t), crash\} = B(t)$ and therefore γ is not progressive with regard to t. □

Theorem 6.2. *Let $RDBS_\varphi$ be a module of $RDBS$. If Criterion C2 does not hold in $behs(RDBS_\varphi)$, then $RDBS_\varphi$ is not one-copy equivalent to $1CDB$.*

Proof. By contradiction. There exists a legal relation Γ such that $RDBS_\varphi$ is one-copy equivalent to $1CDB$. If C2 does not hold, then there exists $\beta \in behs(RDBS_\varphi)$ such that for some transaction $t \in T$, $\pi_i = C_n(t)$ and $\pi_j = A_{n'}(t)$ with $n \neq n'$ by Property 4.2.1 and Property 4.3.1. Then, for any $\gamma \in \Gamma(\beta)$, it is true that $\gamma|\{B(t), C(t), A(t)\}$ is either $B(t) \cdot C(t) \cdot A(t)$ or $B(t) \cdot A(t) \cdot C(t)$ and therefore γ is not well-formed. □

Theorem 6.3. *Let $RDBS_\varphi$ be a module of $RDBS$. If C3 does not hold in $behs(RDBS_\varphi)$, then $RDBS_\varphi$ is not one-copy equivalent to $1CDB$.*

Proof. By contradiction. There exists a legal relation Γ such that $RDBS_\varphi$ is one-copy equivalent to $1CDB$. If C3 does not hold, then there exists a finite $\beta \in behs(RDBS_\varphi)$ such that $log(\beta|EDB_n) \not\preceq log(\beta|EDB_{n'})$ for some pair (n, n') with $n \neq n'$. This inequality cannot be caused by writesets of a committed transaction being different at system sites (see Corollary 4.1). Such difference may be produced because some transactions did not commit or they committed in different order at system sites. In both

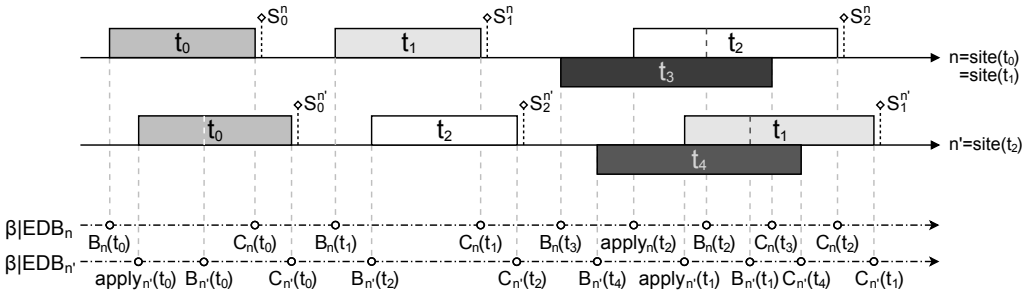

Figure 10: Example of a behavior in which transactions are committed in different order at two sites.

cases, it suffices to study the case when transactions have been committed in different order. Let us consider the β of Figure 10.

We do not take into account the intermediate actions of the local transactions since they are irrelevant to the proof. Note that $site(t_1) = n$ and $site(t_2) = n'$; i.e, t_1 and t_2 are local at n and n' respectively. In both $\beta|EDB_n$ and $\beta|EDB_{n'}$, t_0, t_1 and t_2 do not conflict among them since they are executed sequentially and they all are compatible and consistent by Property 4.4. Transactions t_3 and t_4 do not conflict with t_1 or t_2. Let S_i^k be the snapshot created by the transaction t_i: $i \in \{0, 1, 2\}$ when it is committed at site $k \in \{n, n'\}$ in $\beta|EDB_k$.

According to the system model, transactions can be executed under any isolation level and they can read/write any item at any time. Then, we establish the following additional conditions to the considered behavior β:

(1) $rs_{t_1} \subseteq S_0^n$ and $rs_{t_2} \subseteq S_0^{n'}$

(2) $items(ws_{t_1}) \nsubseteq items(ws_{t_2})$ and $items(ws_{t_2}) \nsubseteq items(ws_{t_1})$

(3) $rs_{t_3} \subseteq S_1^n$ and $rs_{t_4} \subseteq S_2^{n'}$

(4) $items(rs_{t_3}) \cap items(ws_{t_1}) \neq \emptyset \wedge items(rs_{t_3}) \cap items(ws_{t_2}) \neq \emptyset$

(5) $items(rs_{t_4}) \cap items(ws_{t_1}) \neq \emptyset \wedge items(rs_{t_4}) \cap items(ws_{t_2}) \neq \emptyset$

By the first condition, transactions t_1 and t_2 must see at their beginning the snapshots created by t_0 in n and n'. Recall that local and remote transactions commit the same writeset (see Corollary 4.1), and therefore these snapshots must be equal, i.e., $S_0 = S_0^n = S_0^{n'} = ws_{t_0}$. Thus, t_0 must be committed before t_1 and t_2 begin so that the snapshot is available at their beginning. Then, the only behaviors $\gamma \in behs(1CDB)$ that can be possible considering the transformation of Definition 5.1 and only t_0, t_1 and t_2 are the ones of Figure 11:

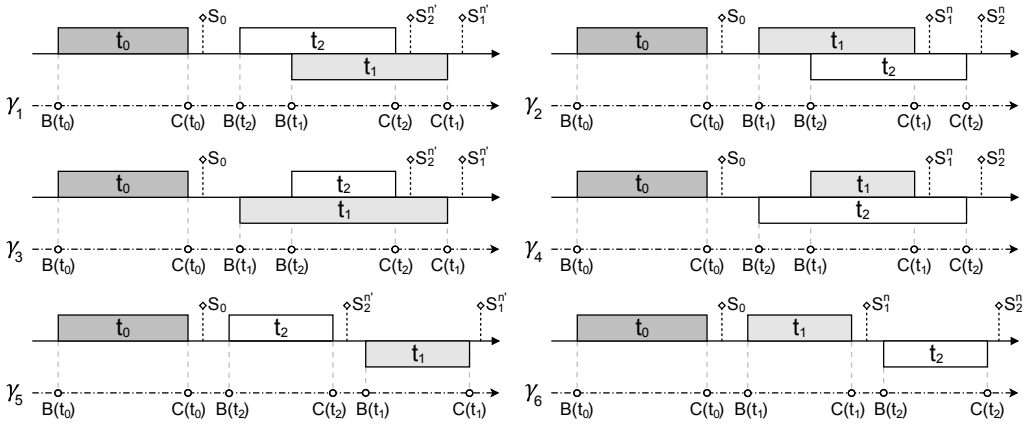

Figure 11: Possible one-copy behaviors following the transformation of Definition 5.1 for t_1, t_2 and t_3.

As a result of the second condition being applied to β, it holds that $S_1^n \neq S_2^{n'}$, $S_1^n \neq S_1^{n'}$ and $S_2^{n'} \neq$

S_2^n. Note that $S_1^{n'} \neq S_2^n$ does not necessarily hold. As the figure shows, for any possible $\gamma \in \Gamma(\beta)$, $\gamma|\{C(t_0), C(t_1), C(t_2)\}$ is either $C(t_0) \cdot C(t_1) \cdot C(t_2)$ or $C(t_0) \cdot C(t_2) \cdot C(t_1)$. In the former case $S_2^{n'}$ is never created in γ and in the latter case S_1^n is never created in γ.

By the last three conditions, both t_3 and t_4 have to read versions of some items of both ws_{t_1} and ws_{t_2} from S_1^n and $S_2^{n'}$ respectively. However, γ produces just S_1^n or $S_2^{n'}$ and besides these versions can not be obtained from S_2^n or $S_1^{n'}$ since $S_1^n \neq S_1^{n'}$ and $S_2^{n'} \neq S_2^n$. Then, for any $\gamma' \in \Gamma(\beta)$ that could be built including t_3 and t_4 over γ, either t_3 is incompatible or t_4 is incompatible and therefore $\gamma' \notin behs(1CDB)$. □

Theorem 6.4. *Let $RDBS_\varphi$ be a module of $RDBS$. If C4 does not hold in $behs(RDBS_\varphi)$, then $RDBS_\varphi$ is not one-copy equivalent to $1CDB$.*

Proof. By contradiction. There exists a legal relation Γ such that $RDBS_\varphi$ is one-copy equivalent to $1CDB$. If C4 does not hold, there exists $\beta \in behs(RDBS_\varphi)$ such that $\pi_i = B_{site(t)}(t) \wedge \pi_j = C_{n'}(last(i, site(t), \beta)) \wedge \pi_k = C_{n'}(t) \wedge site(t) \neq n' \wedge \exists t'' \in T\colon conflict(t'', t, j, \beta(k)|EDB_{n'})$. Let us consider the β of the example of Figure 12.

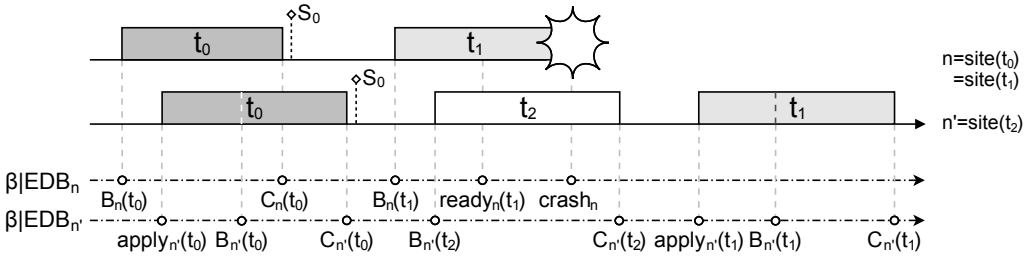

Figure 12: Example of a behavior in which a site crashes.

Let S_0 be the snapshot when $C_n(t_0)$ and $C_{n'}(t_0)$ in both $\beta|EDB_n$ and $\beta|EDB_{n'}$. As t_0, t_1 and t_2 are arbitrary transactions, then we assume a particular isolation level in which conflicts can arise, for example, the simplest one would be *Snapshot Isolation*. Then, we establish the following additional conditions for β:

(1) $rs_{t_1} \subseteq S_0$ and $rs_{t_2} \subseteq S_0$

(2) $items(ws_{t_1}) \cap items(ws_{t_2}) \neq \emptyset$

(3) $items(rs_{t_1}) \cap items(ws_{t_2}) \neq \emptyset \wedge items(rs_{t_2}) \cap items(ws_{t_1}) \neq \emptyset$

By the first condition, transactions t_1 and t_2 must see at their beginning the snapshots created by t_0 in n and n'. Therefore, t_0 must be committed before t_1 and t_2 begin. Then, the only behaviors $\gamma \in \Gamma(\beta)$ that can be possible considering the transformation of Definition 5.1 are the ones of Figure 13.

However, in the cases γ_1 and γ_3, for the transaction t_1, $conflict(t_2, t_1, 4, \gamma_1(6))$ and $conflict(t_2, t_1, 3, \gamma_3(6))$. Therefore, γ_1 and γ_3 are not generalized legal behaviors. The same happens for γ_2 and γ_4, but with t_2, i.e., $conflict(t_1, t_2, 4, \gamma_2(6))$ and $conflict(t_1, t_2, 3, \gamma_4(6))$. Therefore, γ_2 and γ_4 are not generalized legal behaviors either.

By the last condition, γ_5 is not possible since $rs_{t_1} \nsubseteq S_2$ and hence $\neg compatible(t_1, 5, \gamma_5(6))$, and γ_6 is neither possible since $rs_{t_2} \nsubseteq S_1$ and hence $\neg compatible(t_2, 5, \gamma_6(6))$.

□

6.2 Proof of Sufficiency

In the previous Subsection, we have proved that Criteria C1 to C4 are necessary conditions to obtain a one-copy equivalence of an $RDBS_\varphi$ module. In the following, we prove that they are also sufficient conditions. To this end, we denote by $RDBS_{CC}$ the module $RDBS_\varphi$ in which its behaviors satisfy C1 to C4. In order to prove their sufficiency, the criteria must ensure that any behavior of the $RDBS_{CC}$ can be transformed in such a way that the result is a behavior of the $1CDB$ module.

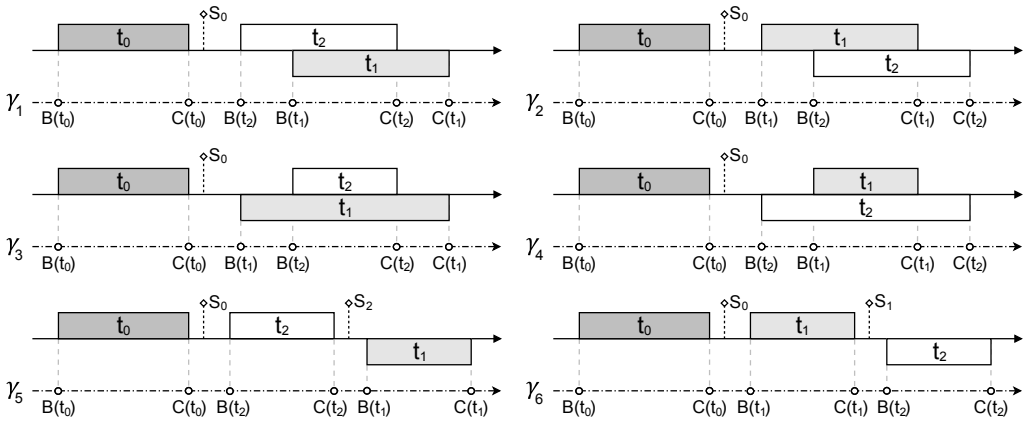

Figure 13: Possible one-copy behaviors following the transformation of Definition 5.1 for t_1, t_2 and t_3.

Next, we study the structure of a transaction in a behavior β. Let β_t be the subsequence $\beta_t = \beta|\{B_{site(t)}(t), A_n(t), C_n(t)\colon n \in \mathcal{N}\}$. For each transaction $t \in T$, the β_t sequence will always be one of the sequences defined in the next Theorem 6.5 due to the conditions enforced by Criterion C2.

Theorem 6.5. *Let β be a behavior of $RDBS_{CC}$. For each transaction $t \in T$, the sequence β_t is one of the following sequences:*

(a) $\beta_t =$ *empty*

(b) $\beta_t = B_{site(t)}(t) \cdot \gamma_{c_t}$ *with* $\gamma_{c_t} \preceq C_{n_1}(t) \ldots C_{n_N}(t)$

(c) $\beta_t = B_{site(t)}(t) \cdot \gamma_{a_t}$ *with* $\gamma_{a_t} \preceq A_{n_1}(t) \ldots A_{n_N}(t)$

where $(n_1, \ldots, n_N)$ is a permutation of the site identifiers, $1..N$.

Proof. By Corollary 4.2, $B_{site(t)}(t) \preceq \beta|acts(RDBS, t)$. Therefore, $\beta_t =$ empty in case $B_{site(t)}$ is not in β. Otherwise, by the definition of β_t, it holds that $B_{site(t)}(t) \preceq \beta_t$. If $\gamma_{c_t} = \gamma_{a_t} =$ empty, the Theorem holds; if not, β_t will be $B_{site(t)}(t) \cdot \gamma$. Then, let π_i, π_j be in β such that $i < j$. Now suppose that $\pi_i = A_n(t)$ is in γ. Then, by contradiction, we assume that there also exists a $\pi_j = C_{n'}(t)$ in γ. By Property 4.2.1 and Property 4.3.1, $n \neq n'$. Since β satisfies C2, such γ is not possible. The same happens, if $\pi_i = C_n(t)$ and $\pi_j = A_{n'}(t)$. Thus, the Theorem holds. □

A transaction $t \in T$ is said to be *committed in* a behavior $\beta \in behs(RDBS_{CC})$, denoted by $t \in Committed(\beta)$, if and only if β_t has an action $C_n(t)$ for any site $n \in \mathcal{N}$, formally: $\beta_t \preceq B_{site(t)}(t) \cdot \gamma_{c_t}$ with $\gamma_{c_t} \neq empty$. In the same way, a transaction $t \in T$ *is aborted in* $\beta \in behs(RDBS_{CC})$, ($t \in Aborted(\beta)$), if and only if β_t has an action $A_n(t)$ for any site $n \in \mathcal{N}$, formally: $\beta_t \preceq B_{site(t)}(t) \cdot \gamma_{a_t}$ with $\gamma_{a_t} \neq empty$.

As a result of Theorem 6.5: (i) if $t \in Committed(\beta)$, then β_t is a prefix of $B_{site(t)}(t) \cdot C_{f_c(\beta_t)}(t)$ where $f_c(\beta_t)$ is the first site at which t is committed; and (ii) if $t \in Aborted(\beta)$, then β_t is a prefix of $B_{site(t)}(t) \cdot A_{a_c(\beta_t)}(t)$ where $f_a(\beta_t)$ is the first site at which t is aborted. Next, we define the subsequence of β which comprises the beginning and the first output (committed or aborted) of each transaction, as well as the crash actions.

Definition 6.2. (Transaction's First-Output Behavior) *Let β be a behavior of $RDBS_{CC}$. The subsequence β_F is defined as $\beta_F = \beta|F(\beta)$ where $F(\beta) = \{B_{site(t)}(t)\colon t \in T\} \cup \{C_{f_c(\beta_t)}(t)\colon t \in Committed(\beta)\} \cup \{A_{f_a(\beta_t)}(t)\colon t \in Aborted(\beta)\} \cup \{crash_n\colon n \in \mathcal{N}\}$. Moreover, $ws_t(\beta_F) = ws_t(\beta)$, $rs_t(\beta_F) = rs_t(\beta)$ and $inf_t(\beta_F) = inf_t(\beta)$.*

Example 7. *As Figure 14 shows, the subsequence β_F of the behavior of Example 5 (see also Figure 8) is $\beta = B_1(t_1) \cdot B_2(t_2) \cdot C_2(t_1) \cdot C_2(t_2)$. In this case, $f_c(\beta_{t_1}) = 2$ (i.e., t_1 is committed first at the remote site 2) and $f_c(\beta_{t_2}) = 2$ (i.e., t_2 is committed first at its local site 2). The sequences of β_{t_1} and β_{t_2} of this example are also presented in Figure 14.*

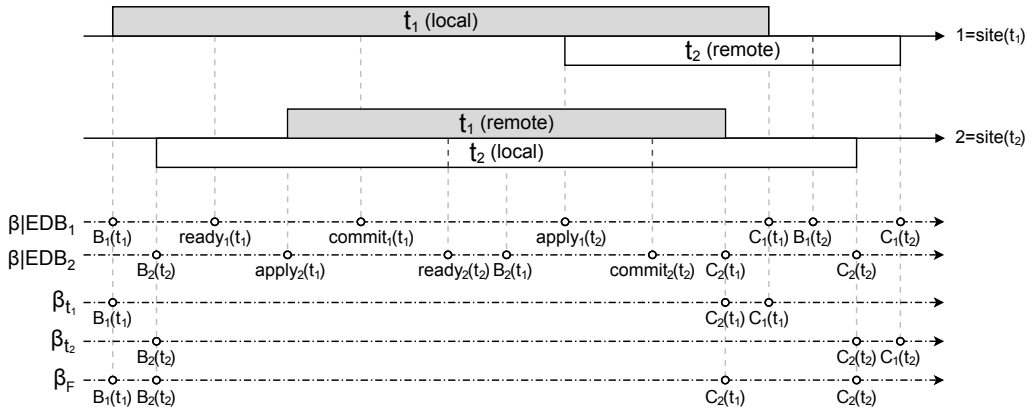

Figure 14: The β_F subsequence for the execution of Example 5.

Note that the actions in β_F for each transaction $t \in T$ (i.e., $B_{site(t)}(t)$, $C_{f_c(\beta_t)}(t)$ and $A_{f_a(\beta_t)}(t)$) are unique. The β_F sequence has some useful properties, shown in Lemma 6.1, of which we will make use later.

Lemma 6.1. *Let β be a behavior of $RDBS_{CC}$. In the subsequence β_F the following conditions hold:*

(1) $\pi_i \in \{C_{f_c(\beta_t)}(t), A_{f_a(\beta_t)}(t)\} \Rightarrow \exists k\colon k < i\colon \pi_k = B_{site(t)}(t)$

(2) *if $crash_{n_1} \cdots crash_{n_N}$ is a subsequence of β_F with $(n_1, \cdots, n_N)$ a permutation of $(1..N)$ and $\pi_j = crash_{n_N}$, then $\beta_F = \beta(j)_F$*

(3) $\pi_i = B_{site(t)}(t) \Rightarrow \exists k\colon k > i\colon \pi_k \in \{C_{f_c(\beta_t)}(t), A_{f_a(\beta_t)}(t), crash_n : n \in \mathcal{N}\}$

Proof. The first condition comes from Definition 6.2 of β_F and Theorem 6.5. The second one is proved by Property 4.1, since after the last $\pi_j = crash_{n_N}$ no action is possible after j at any site of the $RDBS_{CC}$ and hence $\beta = \beta(j)$. Finally, as $\beta \in behs(RDBS_{CC})$ satisfies C1, then again by Definition 6.2 of β_F and Theorem 6.5 the third condition holds. □

The β_F sequence has also the nice property that $log(\beta|EDB_n) \preceq log(\beta_F)$ for all $n \in \mathcal{N}$, as stated in Lemma 6.2. This means that β_F installs the same snapshots, and in the same order, as the ones installed at each replica of the replicated system. This consideration is possible due to β_F being trivially well-formed in the sense given by Definition 3.1 and the fact that each transaction in β_F has rs_t, ws_t and inf_t as its readset, writeset and control information by Remark 4.2.

Lemma 6.2. *Let β be a behavior of $RDBS_{CC}$. It holds that $log(\beta(j)|EDB_n) \preceq log(\beta(j)_F)$ for every prefix $\beta(j) \preceq \beta$ and every $n \in \mathcal{N}$.*

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

- *Basis*: $j = 0$. $\beta(0)|EDB_n$=$\beta(0)_F$=empty and, by definition, $log(\beta(0)|EDB_n)$=$log(\beta(0)_F)$=empty.
- *Hypothesis*: $j > 0$ and $log(\beta(j)|EDB_n) \preceq log(\beta(j)_F)$.
- *Induction Step*: we only consider the events π_{j+1} affecting the Lemma statement.
  - $\pi_{j+1} = C_{f_c(\beta_t)}(t)$ and $f_c(\beta_t) = n$. By Hypothesis, $log(\beta(j)|EDB_n) \preceq log(\beta(j)_F)$. The only possible case from the Hypothesis is $log(\beta(j)|EDB_n) = log(\beta(j)_F)$.
    Consider $log(\beta(j)|EDB_n) \prec log(\beta(j)_F)$. There is at least one different element $\langle t', ws_{t'}\rangle$ in $log(\beta(j)_F)$. Thus, $\beta(j)$ includes $\pi_{j'} = C_{f_c(\beta_{t'})}(t')$ with $j' < j$. This action is also in $\beta(j)_F$ but not in $\beta(j)|EDB_n$. By C3, there is some replica ($f_c(\beta_{t'}) = n'$) $n' \neq n$ 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 the log Definition 3.2, 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 C3.

As a conclusion, $log(\beta(j)|EDB_n) = log(\beta(j)_F)$. As $\beta(j+1)|EDB_n = \beta(j)|EDB_n \cdot \pi_{j+1}$ and $\beta(j+1)_F = \beta(j)_F \cdot \pi_{j+1}$, by the log Definition 3.2, $log(\beta(j+1)|EDB_n) = log(\beta(j+1)_F)$ holds.

- $\pi_{j+1} = C_{f_c(\beta_t)}(t)$ and $f_c(\beta_t) \neq n$. By Hypothesis, $log(\beta(j)|EDB_n) \preceq log(\beta(j)_F)$. Since $\beta(j+1)|EDB_n = \beta(j)|EDB_n$ and $\beta(j+1)_F = \beta(j)_F \cdot \pi_{j+1}$, then by the log Definition 3.2, $log(\beta(j+1)|EDB_n) \prec log(\beta(j+1)_F)$ holds.
- $\pi_{j+1} = C_{n_k}(t)$ and $n_k = n$, being $n_k \neq f_c(\beta_t)$. By Theorem 6.5, there exists $j' < j$ such that $\pi_{j'} = C_{f_c(\beta_t)}(t)$ is in $\beta(j)_F$. This action is in $\beta(j)_F$ but not in $\beta(j)|EDB_n$. By induction Hypothesis, $log(\beta(j')|EDB_n) \prec log(\beta(j')_F)$ and also $log(\beta(j)|EDB_n) \prec log(\beta(j)_F)$. Thus, as $\beta(j+1)|EDB_n = \beta(j)|EDB_n \cdot \pi_{j+1}$ and $\beta(j+1)_F = \beta(j)_F$, by the log Definition 3.2, $log(\beta(j+1)|EDB_n) \preceq log(\beta(j+1)_F)$.
- $\pi_{j+1} = C_{n_k}(t)$ and $n_k \neq n$, being $n_k \neq f_c(\beta_t)$. In this case, $\beta(j+1)|EDB_n = \beta(j)|EDB_n$ and $\beta(j+1)_F = \beta(j)_F$. Thus, trivially $log(\beta(j+1)|EDB_n) \preceq log(\beta(j+1)_F)$ by induction Hypothesis.

Thus, the Lemma holds. □

The actions of sequence β_F compose a behavior which somehow represents the way in which transactions behave in the replicated system. This behavior is not strictly the same as the one of a single database system (Definition 3.5), but it satisfies the generalized legal behavior of Definition 3.6, in which transactions may obtain older snapshots prior to their beginning. Theorem 6.6 covers this issue.

Theorem 6.6. *Let β be a behavior of $RDBS_{CC}$ and $\beta_F = \beta|F(\beta)$. For each transaction $t \in T$ such that $\pi_i = B_{site(t)}(t)$ and $\pi_j = C_{f_c(\beta_t)}(t)$, there exists $0 \leq s \leq i$ such that the following conditions hold:*

(a) $compatible(t, s, \beta(j)_F)$

(b) $\neg conflict(t', t, s, \beta(j)_F)$, *for all* $t' \in T$

(c) $consistent(t, \beta(j)_F)$

Proof. By the Definition 6.2 of β_F, there exists in β both $\pi_i = B_{site(t)}(t)$ and $\pi_j = C_{f_c(\beta_t)}(t)$ with $i < j$ for each considered $t \in T$. Since $\beta_F = \beta|F(\beta)$, we can use the indexes i, j of β in β_F, although the properties to be proved are related only with β_F. Let $t_0 \in T$ be the transaction such that $t_0 = last(i, site(t), \beta)$ and $\pi_{i_0} = C_{site(t)}(t_0)$. By Theorem 6.5, there exists $\pi_{i'_0} = C_{f_c(\beta_t)}(t_0)$ with $i'_0 < j$, as $log(\beta(j)|EDB_{site(t)}) \preceq log(\beta(j)|EDB_{f_c(\beta_t)})$.

- Proof of Condition (a):

  By Property 4.2.1 when $f_c(\beta_t) = site(t)$ or by Theorem 4.1 when $f_c(\beta_t) \neq site(t)$, if $\pi_j = C_{f_c(\beta_t)}(t)$ is in β, then $\pi_r = ready_{site(t)}(t)$ is in β. Thus, by Property 4.2.4, it holds that $compatible($ $t, i, \beta(r)|EDB_{site(t)})$. Note that, by its Definition 3.3, the snapshot only changes on when a transaction is committed. Thus, $\mathcal{S}(\beta(i)|EDB_{site(t)}) = \mathcal{S}(\beta(i_0)|EDB_{site(t)})$. Then, we have to be concerned about the transactions t_k which were committed between the commitment of t_0 and t at $f_c(\beta_t)$ (where t was committed first); i.e., $t_k \in T$ such that $\pi_{i'_k} = C_{f_c(\beta_t)}(t_k)$ and $i'_0 < i'_k < j$. Note that t_0 and each t_k have been committed in β. Then, let $\pi_s = C_{f_c(\beta_{t_0})}(t_0)$ and $\pi_k = C_{f_c(\beta_{t_k})}(t_k)$, it is satisfied that $s < i_0 < i$ and, by Lemma 6.2, $s < k < j$ since $log(\beta(j)|EDB_{f_c(\beta_t)}) \preceq log(\beta(j)_F)$.

  Let $\pi_{i_k} = C_{site(t)}(t_k)$ be the committed actions at $site(t)$ of each t_k with i_k: $i_0, i_1, .., i_k, .., i_m$ and $i_m < r$. The sequence of snapshots in $site(t)$ which makes $compatible(t, i, \beta(r)|EDB_{site(t)})$ true is: $\mathcal{S}(\beta(i_0)|EDB_{site(t)})\mathcal{S}(\beta(i_1)|EDB_{site(t)}) \ldots \mathcal{S}(\beta(i_k)|EDB_{site(t)}) \ldots \mathcal{S}(\beta(i_m)|EDB_{site(t)})$.

  By Property 4.2.1 if $f_c(\beta_t) = site(t)$ or by Theorem 4.1 if $f_c(\beta_t) \neq site(t)$, it is satisfied in both cases that $i_m < r < j$. By Lemma 6.2, $log(\beta|EDB_{site(t)}) \preceq log(\beta_F)$. Both behaviors have built the same snapshots. By the previous definition of π_s and π_k, then $log(\beta(i_0)|EDB_{site(t)}) = log(\beta(s)_F)$ and $log(\beta(i_k)|EDB_{site(t)}) = log(\beta(k)_F)$ for k: $1..m$. By the snapshot Definition 3.3, $\mathcal{S}(\beta(s)_F) \cdots \mathcal{S}(\beta(k)_F) \cdots \mathcal{S}(\beta(m)_F)$ is the same sequence of snapshots. Then, $compatible(t, s,$ $\beta(m)_F)$ holds and trivially $compatible(t, s, \beta(j)_F)$ holds too, since $rs_t(\beta)$ does not change after action $ready_{site(t)}(t)$ is executed, and therefore neither does $rs_t(\beta_F)$.

- Proof of Condition (b):

Recall that $ws_t(\beta)$ and $inf_t(\beta)$ of any committed transaction t are the same at all system sites no matter if t is local or remote (see Remark 4.2), thus by Definition 6.2 they are also the same for β_F. Moreover, $\pi_{i_0} = C_{site(t)}(t_0)$ and $\pi_{i'_0} = C_{f_c(\beta_t)}(t_0)$. If $f_c(\beta_t) = site(t)$, then $\pi_{i'_0} = \pi_{i_0}$ and, by Property 4.4, it holds that $\forall t'' \in T\colon\ \neg conflict(t'', t, i, \beta(j)|EDB_{f_c(\beta_t)})$. As t_0 is the last committed transaction in $site(t)$ before $\pi_i = B_{site(t)}(t)$, the $conflict()$ predicate can be extended to i'_0, i.e., it holds that $\forall t'' \in T\colon\ \neg conflict(t'', t, i'_0, \beta(j)|EDB_{f_c(\beta_t)})$. On the other hand, if $f_c(\beta_t) \neq site(t)$, then also, by C4, it holds that $\forall t'' \in T\colon\ \neg conflict(t'', t, i'_0, \beta(j)|EDB_{f_c(\beta_t)})$. Then, we have to be again concerned about the transactions $t_k \in T$ such that $\pi_{i'_k} = C_{f_c(\beta_t)}(t_k)$ and $i'_0 < i'_k < j$. By the definition of $conflict()$, these transactions satisfy $\neg conflict(t_k, t, i'_k, \beta(j)|EDB_{f_c(\beta_t)})$.

Then, recalling that $\pi_s = C_{f_c(\beta_{t_0})}(t_0)$ and $\pi_k = C_{f_c(\beta_{t_k})}(t_k)$, in β_F, by Definition 6.2, $s < k < j$. Therefore, $\neg conflict(t_k, t, s, \beta(j)_F)$. For the rest of committed transactions t' such that $\pi_{k'} = C_{f_c(\beta_{t'})}(t')$ and that $s < k' < j$ does not hold, $\neg conflict(t', t, s, \beta(j)_F)$ holds too.

- Proof of Condition (c):

  Finally, as $\pi_j = C_{f_c(\beta_t)}(t)$, by Property 4.4, $consistent(t, \beta(j)|EDB_{f_c(\beta_t)}) = \forall i\colon K_i(\mathcal{S}(\beta(j-1)|EDB_{f_c(\beta_t)}), ws_t)$. By Lemma 6.2, $log(\beta(j)|EDB_{f_c(\beta_t)}) = log(\beta(j)_F)$. Thus, $log(\beta(j-1)|EDB_{f_c(\beta_t)}) = log(\beta(j-1)_F)$, since $\pi_j = C_{f_c(\beta_t)}(t)$. Then, $\mathcal{S}(\beta(j-1)|EDB_{f_c(\beta_t)}) = \mathcal{S}(\beta(j-1)_F)$ and therefore $consistent(t, \beta(j)_F)$ holds. □

One can think that the β_F of each $\beta \in behs(RDBS_{CC})$ keeps the properties we need to prove the existence of a one-copy equivalence of the $RDBS_{CC}$. This is the conclusion drawn from Theorem 6.7, which proves that any behavior of the $RDBS_{CC}$ can be transformed in order to become a behavior of the $1CDB$ module.

Theorem 6.7. *The $RDBS_{CC}$ module satisfying Criteria C1 to C4 is one-copy equivalent to the $1CDB$.*

Proof. For each $\beta \in behs(RDBS_{CC})$, we define the following legal relation $\Gamma\colon behs(RDBS_{CC}) \rightarrow behs(1CDB)$; $\Gamma(\beta) = \mathcal{R}(\beta_F)$ where $\mathcal{R}()$ is a renaming function which removes every reference of a site in the actions of β_F related to a transaction or a crash action. That is, $B(t)$, $C(t)$, $A(t)$ or $crash$ appear in $\mathcal{R}(\beta_F)$ when $B_{site(t)}(t)$, $C_{f_c(\beta_t)}(t)$, $A_{f_a(\beta_t)}(t)$, or $crash_n$ appear in β_F. Note that $|\beta_F|\{crash_n : n \in \mathcal{N}\}| = |\mathcal{R}(\beta_F)|crash|$. Thus, β_F satisfies Lemma 6.1 and Theorem 6.6; the definition of rs_t, ws_t and inf_t does not change for the transactions in $\mathcal{R}(\beta_F)$; and the actions in $\mathcal{R}(\beta_F)$ are included in the $1CDB$ signature, i.e., $acts(1CDB)$. Therefore, $\Gamma(\beta) \in behs(1CDB)$, i.e., $\Gamma(\beta)$ is a well-formed, N-crash-stop, generalized legal and progressive behavior. □

The resulting one-copy behavior models the behavior of all the committed and aborted transactions in the replicated system $RDBS_{CC}$. In this way, transactions in the $1CDB$ behavior are committed, aborted or provide no result because of the same reasons as in the $RDBS_{CC}$ module.

In conclusion, the conditions imposed by Criteria C1 to C4 upon the $RDBS$ constitute the $RDBS_{CC}$. Such criteria are the sufficient and necessary conditions for the $RDBS_{CC}$ to be one-copy equivalent to the $1CDB$ module.

7 Discussion

Before presenting the refinements derived from the $RDBS_{CC}$ that model the implementation of an $RDBS$ in Sections 8 and 9, this Section discusses some interesting aspects related to the correctness criteria presented in Section 6.

Read-only transactions: After having stated Remark 4.1, read-only transactions have been ignored, as they can be directly executed at their respective delegate site without requiring any remote transactions to propagate the transaction to the rest of sites. In order to ignore read-only transactions, Remark 4.1 states that read-only transactions do not conflict with other transactions. Without this assumption, Criterion C2 may be violated, since Criterion C4 would not observe conflicts derived from read-only transactions.

Serializability: As far as serializability is concerned, in order to require transactions to be executed under the *Serializable* isolation level at each EDB_n, it would only be necessary to consider that committed transactions are executed in a view-equivalent way with respect to an execution where transactions are executed sequentially one after the other. Besides, module $1CDB$ can be easily transformed into a $1CSER$ module by removing the condition that requires behaviors to be generalizedly legal and replacing it with a condition requiring the sequential execution of committed transactions in its behaviors.

Even so, Criteria C1 to C4 are still necessary and sufficient conditions in order for a $RDBS_{CC}$ (with the aforementioned assumptions) to be equivalent to module $1CSER$. C1 and C2 are independent of the isolation level. As for C3 and C4, the examples used in Section 6 to prove that they both are necessary conditions may be reinterpreted for the Serializable level. This way, Criteria C3 and C4 are necessary conditions for obtaining serializability. Since serializable behaviors include the behaviors obtained for the Dynamic-Serializable level (see Table 1) and conditions C1 to C4 are sufficient conditions for obtaining one-copy equivalence under this level, it is reasonable to assume that they are also sufficient conditions for serializability (although this has not been proven).

Relaxation of the correctness criteria: As it has been proved before, the addition of Criteria C1, C2, C3 and C4 to the $RDBS$ results in a new system $RDBS_{CC}$, which is correct in the sense that it is equivalent to module $1CDB$. Such conditions are necessary and sufficient under the assumptions given for a fully replicated system:

(a) The crash failure model is considered

(b) Transactions can be executed at any place

(c) Transactions can begin at any time

(d) Transactions can read/write any item

(e) Transactions run under any isolation level defined in the database (which may also declare integrity constraints).

Evidently, if we restrict the initial assumptions of the replication model some of the correctness criteria may be relaxed with the aim to improve the system's performance. For instance, by constraining assumption (b) to its limit, we would obtain a primary-copy replication model.

By restricting condition (c), transactions would not be allowed to start at any time. Thus, a control mechanism over the start point of transactions would enable the relaxation of Criterion C3 (i.e., logs of different replicas could be disordered), since C3 is necessary in a system that fulfills condition (c). In this case, the most appropriate moment for starting each transaction should be determined (e.g., when the same snapshot is reached at different replicas).

If condition (d) were modified in order to impose certain restrictions on transactions with respect to the database items that they read/write, it would be possible to divide transactions into sets that work with disjoint sets of database items. In this context, each transaction set should fulfill C1 to C4, but would work independently with regard to the rest of transaction sets.

Finally, if condition (e) is restricted so that all transactions use the same level of isolation, it is obvious that using isolation levels that produce fewer conflicts will result in a better system performance.

Note that most of these constraints on the initial assumptions lead to better performance and improved scalability. This had been already argued in [25] regarding condition (d), with the proposal of the *shared-nothing* concept that implies a perfect database partitioning. Modern replication systems have combined such partitioning with passive replication, as in the *Discor* protocol of [26] where disjoint conflict classes (i.e., the database is logically partitioned for evaluating conflicts, although this does not demand a physical non-sharing distribution) are assigned to different master replicas (i.e., passive replication within each logical partition). This boosts the resulting scalability. These results have been further improved with an important re-design of the DBMS core and maintaining data in RAM, as in the H-Store system [27]. Moreover, all these systems still preserve *one-copy equivalence*, but reaching a level of scalability in the latter that is close to that provided in modern cloud-based systems [28].

Implications of crash failures: Before presenting the successive refinements of the replicated database system and the replication protocol, we now introduce a relevant consideration regarding assumption (a) of the failure model and its influence on the design of the replication protocol. Instead of using the $RDBS_{CC}$ directly, we may consider other correct systems $RDBS_{\varphi}$; for instance, an $RDBS_{\{C'1,C'2,C'3,C'4\}}$ with $C'i \Rightarrow Ci$ $(i : 1..4)$ would also be a correct system (note that conditions $C'i$ would be sufficient but not necessary conditions for one-copy equivalence, since they are stronger properties). Among Criteria C1 to C4, the suitable candidate to be strengthened is C1, as it is a very weak criterion and therefore finding a property that guarantees a stronger progress condition than the one required by C1 is a straightforward task. Let C'1 be the following condition:

(C'1: Transaction Progress) For every behavior $\beta \in RDBS$, the following holds: $\pi_i = B_n(t) \Rightarrow \exists k : k > i : \pi_k \in \{C_n(t), A_n(t), crash_n\}$

An $RDBS$ satisfying Criteria C'1, C2, C3 and C4, denoted by $RDBS_{\{C'1,C2,C3,C4\}}$, is a correct system since $\beta \in behs(RDBS_{\{C'1,C2,C3,C4\}}) \Rightarrow \beta \in behs(RDBS_{CC})$ (as if β satisfies $C'1$ then C1 holds trivially). What is more, a system $RDBS_{\{C'1,C2,C3\}}$ (i.e. it satisfies Criteria C'1, C2 and C3) is correct in a scenario with no crash failures, because it also satisfies C4, as shown in the following theorem:

Theorem 7.1. *Let β be a behavior of $RDBS_{\{C'1,C2,C3\}}$ such that there is no action $crash_n$ in β for any $n \in \mathcal{N}$. Then, β satisfies C4.*

Proof. By contradiction. If β does not satisfy C4, then there exists a transaction $t \in T$ such that $\pi_i = B_{site(t)}(t) \wedge \pi_j = C_{n'}(last(i, site(t), \beta)) \wedge \pi_k = C_{n'}(t) \wedge n' \neq site(t) \wedge \exists t'' \in T : conflict(t'', t, j, \beta(k)|EDB_{n'})$

By C'1, $\exists s : s > i : \pi_s \in \{C_{site(t)}(t), A_{site(t)}(t)\}$ in β. By C2: $\pi_s = C_{site(t)}(t)$ since $\pi_k = C_{n'}(t)$.

By its definition there exists a $\pi_l = C_{site(t)}(t_0)$ being $t_0 = last(i, site(t), \beta)$. By Property 4.4, $\forall t'' \in T : \neg conflict(t'', t, i, \beta(s)|EDB_{site(t)})$, and therefore $\forall t'' \in T : \neg conflict(t'', t, l, \beta(s)|EDB_{site(t)})$. By C3, $log(\beta(k)|EDB_{n'}) = log(\beta(s)|EDB_{site(t)})$.

If there exists a transaction $t_m \in T$ such that $l < m < s$ and $\pi_m = C_{site(t)}(t_m)$, then there exists an m' such that $j < m' < k$ and $\pi_{m'} = C_{n'}(t_m)$. Recall that $ws_{t_m}(\beta)$ and $inf_{t_m}(\beta)$ are the same for every site. Thus, if $conflict(t_m, t, j, \beta(k)|EDB_{n'})$ is true, then a contradiction is obtained because $\neg conflict(t_m, t, l, \beta(s)|EDB_{site(t)})$ is true.

□

From this theorem, we can infer that if crash failures had not been considered, it would have been possible to develop a correct system based on Criteria C'1, C2 and C3 (without taking into account C4). However, when introducing the $RDBS_{\{C'1,C2,C3\}}$ in a crash-prone environment, it may behave incorrectly (as Criteria C4 may be violated). This result shows the importance of considering crash failures from the beginning of the specification.

8 A First Refinement of the Replicated System

One of the advantages of using the I/O automaton model is that it permits to provide accurate descriptions of a distributed system at different levels of abstraction through successive refinements [2]. In the first part of this work, a specification of a replicated database system has been presented by means of the $RDBS$ module. Its purpose was to introduce the problem specification. Then, we have described the $RDBS_{CC}$ module as an $RDBS$ module satisfying the correctness criteria (Criteria C1-C4), which guarantee the one-copy equivalence of the replicated system. The $RDBS_{CC}$ is partially distributed: the set of EDB_n corresponds to the extended databases at each location, while the DRP simply models any replication protocol responsible for ensuring the deferred-update technique.

In this Section, a new level of abstraction of the replicated system is proposed. On one hand, this refinement introduces a new module for the replication protocol, called DRP^A, which is compatible with the DRP specification. Besides, the DRP^A is closer to the possible algorithms which could be implemented in practice. On the other hand, this Section describes a new module named EDB_n^A, which consists in a refinement of the EDB_n presenting a more detailed specification of an extended database. It shares the same signature and properties with the EDB_n module, but it provides some extra properties to help the DRP^A with the replication process. Thus, we are going to study the correctness of a more specific replicated database system, which is the composition of the DRP^A module and a set of EDB_n^A modules.

8.1 Extended Database Refinement

Up until now, all the extended functionalities of the EDB_n module that have been modeled are the $apply_n(t, data)$, $ready_n(t, data)$ and $commit_n(t)$ actions as well as some basic properties which distinguish between local and remote transactions, which were enough to prove the one-copy equivalence of the $RDBS_{CC}$. However, some additional properties are necessary to study the correctness of a particular replication protocol based on the functionalities provided by the EDB_n at each site.

Thus, a new extended database module, denoted as EDB_n^A, is defined as a refinement of the EDB_n. The EDB_n^A module is the same as the EDB_n except for some properties imposed on its behaviors. This means that they both have the same signature, $sig(EDB_n^A) = sig(EDB_n)$; and that the behaviors of the EDB_n^A are a subset of the behaviors of the EDB_n, $behs(EDB_n^A) \subset behs(EDB_n)$. Therefore, the EDB_n^A satisfies Property 4.1, Property 4.2, Property 4.3, and Property 4.4. We will refer to the original properties specified for an EDB_n behavior when it is necessary to apply them for a EDB_n^A behavior. Additionally, the behaviors of the EDB_n^A satisfy Property 8.1 and Property 8.2.

Property 8.1. (Extended Database Refinement) *Each behavior $\beta \in behs(EDB_n^A)$ holds that:*

(1) $\pi_i = B_n(t) \wedge \pi_j = ready_n(t, data) \Rightarrow data.last = last(i, n, \beta)$[6]

(2) $\pi_j = apply_n(t, data) \wedge \pi_i = C_n(data.last) \wedge i < j \wedge \exists t' \in T\colon conflict(t', t, i, \beta(j)) \Rightarrow \forall k\colon k > j\colon \pi_k \neq C_n(t)$

(3) $\pi_i = B_{site(t)}(t) \wedge site(t) = n \Rightarrow \exists\, k\colon k{>}i\colon \pi_k \in \{ready_n(t, data), A_n(t), crash_n\colon data \in D\}$

(4) $\pi_i \in \{commit_n(t), apply_n(t, data)\} \Rightarrow \exists\, k\colon k > i\colon \pi_k \in \{C_n(t), A_n(t), crash_n\}$

Recall that when $ready_n(t, data)$ is generated, $data$ contains $data.ws$, $data.inf$ and $data.last$ (see Section 4.1). Property 8.1.1 is a safety property ensuring that the field $data.last$ in action $ready_n(t, data)$ will contain the transaction identifier of the last committed transaction before the beginning of transaction t. This is essential to detect conflicts when applying remote transactions in the extended database. Thus, in order to guarantee Criterion C4, Property 8.1.2 uses $data.last$ to restrict the remote transactions that can be committed in the local database. Thus, if any conflicting transaction t' has been committed before t is applied, the remote transaction t can never be committed; actually, by Property 8.1.2 and Property 8.1.4, it will eventually be aborted or the site will crash.

Property 8.1.3 and Property 8.1.4 are liveness properties. The former ensures that if a transaction t begins at a site n, the transaction will eventually be ready or will be aborted at the local database EDB_n^A, unless site n crashes. The latter ensures that if a transaction requests a commit or an apply operation at a site n, the EDB_n^A will eventually provide a response (either $C_n(t)$ or $A_n(t)$), or the site n will crash otherwise.

Aborts can happen as a result of a conflictive concurrent transaction at commit time, an integrity constraint violation or any other internal cause of the database such as deadlocks, timeouts and so on. Unilateral aborts are going to be considered as crash failures. Property 8.2 specifies which causes may be responsible for the aborted response of a transaction.

Property 8.2. (Abort Causes) *For every behavior $\beta \in behs(EDB_n^A)$, it holds that:*

(1) *There is no transaction unilateral abort or, if any, it is observed as a $crash_n$ action.*

(2) *For any local transaction $t \in T$, with $site(t) = n$:*

[6] In Definition 6.1, $last(i, n, \beta)$ is specified for $\beta \in behs(RDBS)$, but it can be redefined for any $\beta \in behs(EDB_n^A)$.

- *Let $\pi_i = B_n(t)$ and $\pi_j = A_n(t)$. If pattern (b) of Property 4.2.1 is followed, then there exists $t' \in T$ such that $conflict(t', t, i, \beta(j))$ or $\neg consistent(t, \beta(j))$.*
- *Action $A_n(t)$ in the sequence (c) of Property 4.2.1 is possible by any abortion cause with the exception of an unilateral abort (e.g., incompatible read operations, deadlock resolutions, timeout expirations, explicit client aborts or conflicting transactions).*

(3) *For any remote transaction $t \in T$ (with $site(t) \neq n$) such that $\pi_i = apply_n(t, data)$, $\pi_j = B_n(t)$ and $\pi_k(t) = A_n(t)$ follow pattern (b) in Property 4.3.1, either $\neg consistent(t, \beta(k))$, or there exists $t' \in T$ such that $conflict(t', t, l, \beta(k))$ being $\pi_l = C_n(data.last)$ with $l < i$*[7].

In fact, by the time a transaction t delegates the decision on its outcome to the replication protocol (by executing $ready_n(t, data)$), all the operations of t have already been performed successfully in the database and therefore the transaction is ready for its commit (by Property 4.2.1 $ready_n(t, data)$ must be followed by $commit_n(t)$). Therefore, from then on, t will only be aborted in case it does not fulfill the integrity constraints or if there is another concurrent conflicting transaction which causes its abortion. As for remote transactions, they will be aborted in case Property 8.1.2 is satisfied, i.e., they are not consistent.

8.2 Replication Protocol Refinement

We now introduce the DRP^A module as a refinement of the DRP module. This new level of abstraction of the DRP specification models the general characteristics which any deferred-update protocol working with an EDB_n^A module has. The new DRP^A module has the same signature as the DRP, i.e., $sig(DRP^A) = sig(DRP)$. The DRP^A only defines some additional properties (Property 8.3) to refine the specification of the DRP ($behs(DRP^A) \subset behs(DRP)$). Thus, the behaviors of DRP^A satisfy all the DRP's properties (Property 4.5).

The DRP^A explains the way in which the protocol requests the commit of a local transaction or applies a remote one. Since both actions are treated in a similar way in some of the presented properties $REQ(n) = \{apply_n(t, data), commit_n(t) \colon t \in T, data \in D\}$ and $REQ(n, t) = \{apply_n(t, data), commit_n(t) \colon data \in D\}$ to work with both of them at the same time.

Property 8.3. (Replication Protocol Refinement) *Let β be a behavior of DRP^A, it holds that:*

(1) $\pi_i \in REQ(n, t') \wedge \pi_j \in REQ(n, t) \wedge i < j \Rightarrow \exists k \colon i < k < j \colon \pi_k \in \{C_n(t'), A_n(t')\}$

(2) $trans(\beta(j)|REQ(n)) \preceq trans(\beta(j)|REQ(n'))$ *or vice versa for every prefix* $\beta(j) \preceq \beta$

(3) $\pi_i = ready_n(t, data) \Rightarrow \exists k \colon k > i \colon \pi_k \in \{commit_n(t), crash_n\}$

As Property 8.3.1 shows, a $commit_n(t)$ or $apply_n(t, data)$ can only be executed at site n once the outcome of the previous request has been decided. This property is necessary in case the underlaying database replicas do not guarantee that the outcomes of two concurrent transactions are delivered in the same order as the request operations for committing or applying them. Since Criterion C3 requires the order of committed responses to be the same at all sites, this condition is essential for the model considered in this work.

Apart from performing commit and apply requests sequentially, in order to build the same snapshots (and hence the same logs) at all the databases of the replicated system, the protocol must also ensure that the sequences of transactions that make the commit or apply requests at each site are prefixes of the others or vice versa. This is formalized by Property 8.3.2. Finally, Property 8.3.3 states that every local transaction that becomes ready will either be aborted, or will request for its commit, unless the site crashes.

8.3 Composition of the Refinements

Once presented the new modules, now we can build a new replicated system $RDBS^A$, as a composition of the DRP^A module and a set of EDB_n^A modules, i.e., $RDBS^A = DRP^A \times (\prod_{n \in \mathcal{N}} EDB_n^A)$.

By the properties of composition, the resulting $RDBS^A$ will have $crash_n$ as input action, that is, $in(RDBS^A) = \{crash_n\}$, and the union of the outputs of the DRP^A and the EDB_n^A modules as output

[7] Recall that for $l = 0$, $data.last = f_0$

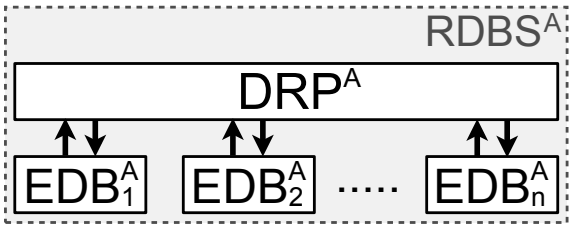

Figure 15: Composition of the Refined Replicated Database System.

actions, i.e., $out(RDBS^A) = (\bigcup_{n \in \mathcal{N}} out(EDB_n^A)) \cup out(DRP^A)$. By its definition, $sig(RDBS^A) = sig(RDBS)$. Besides, its behaviors hold that $\forall n \in \mathcal{N}\colon \beta|EDB_n^A \in behs(EDB_n^A)$ and $\beta|DRP^A \in behs(DRP^A)$.

The $RDBS^A$ can be proven correct by showing that $behs(RDBS^A) \subseteq behs(RDBS_{CC})$, as we already know that the $RDBS_{CC}$ is a correct system. However, the $RDBS_{CC}$ is specified from the $RDBS$ module including the EDB_n and DRP modules and therefore it is necessary to prove first that $behs(RDBS^A) \subseteq behs(RDBS)$.

Let $\beta \in behs(RDBS^A)$. By contradiction, if $behs(RDBS^A) \nsubseteq behs(RDBS)$, then there exists a behavior β of $RDBS^A$ such that $\beta \notin behs(RDBS)$. As $sig(RDBS^A) = sig(RDBS)$, the properties of the composition entail that $\beta|EDB_n \notin behs(EDB_n)$ or $\beta|DRP \notin behs(DRP)$. A contradiction is obtained, since for every $\beta \in behs(RDBS^A)$, $\beta|EDB_n^A \in behs(EDB_n)$ and $\beta|DRP^A \in behs(DRP)$ must hold by the definition of the EDB_n^A and DRP^A refinements.

Thus, we only need to prove that the behaviors in $behs(RDBS^A)$ satisfy Criteria C1 to C4 to guarantee that $behs(RDBS^A) \subseteq behs(RDBS_{CC})$.

8.4 Correctness Proof

The correctness proof of the $RDBS^A$ is based on a set of theorems (Theorems 8.1 to 8.4). They include assertions which satisfy every previous considered criteria. These assertions are the logical consequence of the properties specified for the new hierarchical level.

Theorem 8.1. *For every behavior $\beta \in behs(RDBS^A)$, it holds that:*

$$\pi_i = B_n(t) \Rightarrow \exists k\colon k > i\colon \pi_k \in \{C_n(t), A_n(t), crash_n\}$$

Proof. By contradiction, if $site(t) = n$, then there exists $\pi_i = B_{site(t)}(t)$ and $\forall k\colon k > i\colon \pi_k \notin \{C_{site(t)}(t), A_{site(t)}(t), crash_{site(t)}\}$. By Property 8.1.3, there exists $\pi_j = ready_{site(t)}(t, data)$ in β with $j > i$. By Property 8.3.3, there exists $\pi_r = commit_{site(t)}(t)$ in β with $r > j$. Finally, by Property 8.1.4, there exists $\pi_k \in \{C_{site(t)}(t), A_{site(t)}(t), crash_{site(t)}\}$ in β with $k > r$. Then, there is a contradiction with the initial supposition and the Theorem holds. If $site(t) \neq n$, by Property 4.3.1 and Property 8.1.4, the Theorem holds trivially. □

Lemma 8.1. *For every $\beta \in behs(RDBS^A)$, it holds that $\pi_i \in REQ(n,t) \wedge \pi_j \in \{C_n(t), A_n(t)\} \Rightarrow \forall k\colon i < k < j\colon \pi_k \notin \{C_n(t')\colon t' \in T\}$*

Proof. By contradiction, let us assume that the antecedent holds and there exists $t' \in T$ such that $\pi_k = C_n(t')$ and $i < k < j$. By Property 4.2.1, there exists $\pi_{k'} \in REQ(n,t')$ in β with $k' < k$ and $\pi_k = C_n(t')$ is unique in β. If $k' < i$, then $k' < k < i$ (by Property 8.3.1), which is a contradiction. On the other hand, if $i < k'$, then $i < j < k'$ (by Property 8.3.1), and hence $j < k' < k$, which is also a contradiction. □

Corollary 8.1. *For every $\beta \in behs(RDBS^A)$, it holds that $\pi_i \in REQ(n,t) \wedge \pi_j \in \{C_n(t), A_n(t)\} \Rightarrow \forall k\colon i < k < j\colon \pi_k \notin \{C_n(t')\colon t' \in T\} \wedge \pi_k \notin REQ(n)$*

Proof. It is proved trivially from Lemma 8.1 and Property 8.3.1. □

Lemma 8.2. *For every $\beta \in behs(RDBS^A)$, it holds that:*

(1) $\pi_i = B_n(t) \wedge \pi_j = commit_n(t) \wedge \pi_k = C_n(t)$
$\Rightarrow \forall t' \in T\colon \neg conflict(t', t, i, \beta(k)|EDB_n^A) \wedge consistent(t, \beta(j)|EDB_n^A)$

(2) $\pi_i = C_n(data.last) \wedge \pi_j = apply_n(t, data) \wedge \pi_k = C_n(t) \wedge i < j$
$\Rightarrow \forall t' \in T\colon \neg conflict(t', t, i, \beta(k)|EDB_n^A) \wedge consistent(t, \beta(j)|EDB_n^A)$

Proof. (1) By contradiction, let us assume that the antecedent is true and either there exists $t' \in T$ such that $conflict(t', t, i, \beta(k)|EDB_n^A)$ or $\neg consistent(t, \beta(j)|EDB_n^A)$. By Property 4.4, condition $\neg conflict(t', t, i, \beta(k)|EDB_n^A)$ holds for every $t' \in T$. Thus, there is a contradiction. On the other hand, by Property 4.4, $consistent(t, \beta(k)|EDB_n^A)$. By Lemma 8.1, there does not exist any $t' \in T$ such that $\pi_{k'} = C_n(t')$ and $j < k' < k$. Therefore, by the snapshot definition $\mathcal{S}(\beta(k-1)|EDB_n^A) = \mathcal{S}(\beta(j-1)|EDB_n^A)$. Thus, $consistent(t, \beta(j)|EDB_n^A)$ holds in contradiction to the initial supposition.

(2) By contradiction, assume that the antecedent is true and either there exists $t' \in T$ such that $conflict(t', t, i, \beta(k)|EDB_n^A)$ or $\neg consistent(t, \beta(j)|EDB_n^A)$. By Lemma 8.1, there cannot exist $t' \in T$ such that $conflict(t', t, j, \beta(k)|EDB_n^A)$. By Property 8.1.2, if $conflict(t', t, i, \beta(j)|EDB_n^A)$, then there cannot exist $\pi_k = C_n(t)$ with $k > j$ in β, in contradiction with the antecedent. On the other hand, if $\pi_k = C_n(t)$, by Property 4.4, $consistent(t, \beta(k)|EDB_n^A)$ holds. By Lemma 8.1, there does not exist any $t' \in T$ such that $\pi_{k'} = C_n(t')$ and $j < k' < k$. Therefore, by the snapshot definition $\mathcal{S}(\beta(k-1)|EDB_n^A) = \mathcal{S}(\beta(j-1)|EDB_n^A)$. Thus, $consistent(t, \beta(j)|EDB_n^A)$ holds in contradiction to the initial supposition. □

Lemma 8.3. *For every $\beta \in behs(RDBS^A)$, it holds that:*

(1) $\pi_i = B_n(t) \wedge \pi_j = commit_n(t) \wedge \pi_k = A_n(t) \Rightarrow (\exists t' \in T\colon conflict(t', t, i, \beta(j)|EDB_n^A) \vee \neg consistent(t, \beta(j)|EDB_n^A)) \wedge \forall t' \in T\colon \neg conflict(t', t, j, \beta(k)|EDB_n^A)$

(2) $\pi_j = apply_n(t, data) \wedge \pi_k = A_n(t) \Rightarrow ((\pi_i = C_n(data.last) \wedge i < j \wedge \exists t' \in T\colon conflict(t', t, i, \beta(j)|EDB_n^A)) \vee \neg consistent(t, \beta(j)|EDB_n^A)) \wedge \forall t' \in T\colon \neg conflict(t', t, j, \beta(k)|EDB_n^A)$

Proof. (1) By Property 8.2.2, for pattern (b) of Property 4.2.1, there exists a transaction $t' \in T$ such that $conflict(t', t, i, \beta(k)|EDB_n^A)$ or $\neg consistent(t, \beta(k)|EDB_n^A)$. By Lemma 8.1, there cannot exist $t' \in T$ such that $conflict(t', t, j, \beta(k)|EDB_n^A)$. Thus, $conflict(t', t, i, \beta(j)|EDB_n^A)$. By Lemma 8.1 and the snapshot definition, $\mathcal{S}(\beta(k-1)|EDB_n^A) = \mathcal{S}(\beta(j-1)|EDB_n^A)$. Thus, if $\neg consistent(t, \beta(k)|EDB_n^A)$, then $\neg consistent(t, \beta(j)|EDB_n^A)$.

(2) By Lemma 8.1 and the definition of $conflict()$, $\neg conflict(t', t, j, \beta(k)|EDB_n^A)$ for all $t' \in T$. Then, by Property 8.2.3 for pattern (b) of Property 4.3.1, either there exists $t' \in T$ such that $conflict(t', t, i, \beta(j)|EDB_n^A)$ with $\pi_i = C_n(data.last)$ and $i < j$, or $\neg consistent(t, \beta(k)|EDB_n^A)$. If $\neg consistent(t, \beta(k)|EDB_n^A)$, by Lemma 8.1 and the definition of snapshot, $\mathcal{S}(\beta(k-1)|EDB_n^A) = \mathcal{S}(\beta(j-1)|EDB_n^A)$. Thus, if $\neg consistent(t, \beta(k)|EDB_n^A)$, then $\neg consistent(t, \beta(j)|EDB_n^A)$.

□

Lemma 8.4. *For every $\beta \in behs(RDBS^A)$, it holds that:* $\pi_i = REQ(n, t) \wedge \pi_j \in REQ(n', t) \wedge log(\beta(i)|EDB_n^A) = log(\beta(j)|EDB_{n'}^A) \wedge \pi_k = C_n(t) \Rightarrow \forall k'\colon k' > j\colon \pi_{k'} \neq A_{n'}(t)$

Proof. By contradiction, let us consider that the antecedent holds and $\pi_{k'} = A_{n'}(t)$ with $k' > j$ is in $\beta|EDB_{n'}^A$. By Lemma 8.2, in any case of $\pi_i \in REQ(n, t)$, if $\pi_k = C_n(t)$, then $consistent(t, \beta(i)|EDB_n^A)$. By the antecedent and the log Definition 3.2, $\mathcal{S}(\beta(i)|EDB_n^A) = \mathcal{S}(\beta(j)|EDB_{n'}^A)$. Therefore, since $consistent(t, \beta(i)|EDB_n^A)$, then $consistent(t, \beta(j)|EDB_{n'}^A)$. Then, by Lemma 8.3, $A_{n'}(t)$ can only be caused by the $conflict()$ predicate. Let consider the rest of possible cases:

- Let $\pi_j = commit_{n'}(t)$ and hence $site(t) = n'$. By Property 4.2.1, $\pi_{b'} = B_{n'}(t)$ and $\pi_{r'} = ready_{n'}(t, data)$ with $b' < r' < j$ are in $\beta|EDB_{n'}^A$. By Lemma 8.3.1, since $\pi_{k'} = A_{n'}(t)$ and $consistent(t, \beta(j)|EDB_{n'}^A)$, there exists $t' \in T$ such that $conflict(t', t, b', \beta(j)|EDB_{n'}^A)$. Then, $\pi_{c'} = C_{n'}(t')$ with $b' < c' < j$ is in $\beta|EDB_{n'}^A$. By Property 8.1.1 and the $last()$ definition, $\pi_{l'} = C_{n'}(data.last)$ with $l' < b'$ is in $\beta|EDB_{n'}^A$ (or $l' = 0$ if $data.last = f_0$). By Theorem 4.1, $\pi_i = apply_n(t, data)$. Since $log(\beta(i)|EDB_n^A) = log(\beta(j)|EDB_{n'}^A)$, $\pi_l = C_n(data.last)$

(or $l = 0$) and $\pi_c = C_n(t')$ are in $\beta|EDB_n^A$. By Property 4.3.1(a), Property 8.3.2 and Corollary 8.1, $l < c < i < k$ holds. Then, $conflict(t', t, l, \beta(i)|EDB_n^A)$ is true. Thus, by Property 8.1.2, $\pi_k \neq C_n(t)$, a contradiction.

- Let $\pi_j = apply_{n'}(t, data)$ and hence $site(t) \neq n'$. By Lemma 8.3.2, since $\pi_{k'} = A_{n'}(t)$ and $consistent(t, \beta(j)|EDB_{n'}^A)$, there exists $t' \in T$ such that $conflict(t', t, l', \beta(j)|EDB_{n'}^A)$ with $\pi_{l'} = C_{n'}(data.last)$ (or $l' = 0$ if $data.last = f_0$) and $l' < j$. Then, $\pi_{c'} = C_{n'}(t')$ with $l' < c' < j$ is in $\beta|EDB_{n'}^A$. Since $log(\beta(i)|EDB_n^A) = log(\beta(j)|EDB_{n'}^A)$, $\pi_l = C_n(data.last)$ and $\pi_c = C_n(t')$ are in $\beta|EDB_n^A$. By Property 8.3.2 and Corollary 8.1, $l < c < i < k$ holds. Once more we have to consider two cases:
  - Let $\pi_i = commit_n(t)$ and hence $site(t) = n$. By Property 4.2.1 (a), $\pi_b = B_n(t)$ and $\pi_r = ready_n(t, data)$ with $b < r < j$ is in $\beta|EDB_n^A$. By Property 8.1.1 and the $last()$ definition, $l < b$ (or $l = 0$ if $last(b, n, \beta) = f_0$). By the $last()$ definition, it is not possible that $l < c < b$ and hence $l < b < c < i < k$ holds. Thus, by Property 4.4 $\neg conflict(t', t, b, \beta(k)|EDB_n^A$. Then, by the $conflict()$ definition, we have a contradiction since $conflict(t', t, l', \beta(j)|EDB_{n'}^A)$.
  - Let $\pi_i = apply_n(t, data)$ and hence $site(t) \neq n$. Thus, by Property 8.1.2, $\pi_k \neq C_n(t)$, a contradiction.

□

Lemma 8.5. *For every $\beta \in behs(RDBS^A)$, it holds that:* $\pi_i \in REQ(n, t) \wedge \pi_j \in REQ(n', t) \Rightarrow trans(\beta(i)|\{C_n(t) \colon t \in T\}) = trans(\beta(j)|\{C_{n'}(t) \colon t \in T\})$.

Proof. Let $trans(\beta|REQ(n))$ be the sequence of transaction identifiers $t_1, t_2, t_3, ..., t_m, ...$. We proof this Lemma by induction over the length $m \geq 1$ of this sequence.

- *Basis*: ($m = 1$) Let $\pi_{i_1} \in REQ(n, t_1)$. Then, $trans(\beta(i_1)|REQ(n)) = t_1$. If there does not exist $\pi_{j_1} \in REQ(n', t_1)$ in β the antecedent is false and the Lemma holds. Otherwise, by Property 8.3.2, $trans(\beta(j_1)|REQ(n')) = t_1$. Then, neither $t' \in T$ such that $\pi_r \in REQ(n, t')$ in $\beta(i_1)$, nor $t' \in T$ such that $\pi_{r'} \in REQ(n', t')$ in $\beta(j_1)$ exist. By Property 4.2.1(a) or by Property 4.3.1(a), neither $\pi_k \in \{C_n(t) \colon t \in T\}$ in $\beta(i_1)$ nor $\pi_{k'} \in \{C_{n'}(t) \colon t \in T\}$ in $\beta(j_1)$ exist. Thus, the Lemma holds for $m = 1$ since $trans(\beta(i_1)|\{C_n(t) \colon t \in T\}) = trans(\beta(j_1)|\{C_{n'}(t) \colon t \in T\}) =$ empty.
- *Hypothesis*: ($m \geq 1$) We assume that $\pi_{i_m} \in REQ(n, t_m) \wedge \pi_{j_m} \in REQ(n', t_m) \Rightarrow trans(\beta(i_m)|\{C_n(t) \colon t \in T\}) = trans(\beta(j_m)|\{C_{n'}(t) \colon t \in T\})$ holds. Note that, by Corollary 4.1 and Definition 3.2, $log(\beta(i_m)|EDB_n^A) = log(\beta(j_m)|EDB_{n'}^A)$ holds. Thus, $\mathcal{S}(\beta(i_m)|EDB_n^A) = \mathcal{S}(\beta(j_m)|EDB_{n'}^A)$ holds by Definition 3.3.
- *Induction Step*: Let $m + 1$ be the next step. Let $\pi_{i_{m+1}} \in REQ(n, t_{m+1})$. Then, $trans(\beta(i_{m+1})|REQ(n)) = t_1, t_2, t_3, ..., t_m, t_{m+1}$. If there does not exist $\pi_{j_{m+1}} \in REQ(n', t_{m+1})$ the antecedent is false and the Lemma holds. Otherwise, by Property 8.3.2, $trans(\beta'|REQ(n)) \preceq trans(\beta'|REQ(n'))$ or vice versa for any $\beta' \preceq \beta$ and hence $trans(\beta(j_{m+1})|REQ(n')) = t_1, t_2, t_3, ..., t_m, t_{m+1}$. By Property 8.3.1, there exists $\pi_{k_m} \in \{C_n(t_m), A_n(t_m)\}$ and $\pi_{k'_m} \in \{C_{n'}(t_m), A_{n'}(t_m)\}$. By contradiction, let $\pi_{k_m} = C_n(t_m)$ and $\pi_{k'_m} = A_{n'}(t_m)$ (the other possibility is proof by a symmetric procedure). By Hypothesis, $log(\beta(i_m)|EDB_n^A) = log(\beta(j_m)|EDB_{n'}^A)$. Thus, the conditions of Lemma 8.4 lead to a contradiction.

Thus, the Theorem holds. □

Corollary 8.2. *For every $\beta \in behs(RDBS^A)$, it holds that:*

(1) $\pi_i = REQ(n, t) \wedge \pi_j \in REQ(n', t) \wedge \pi_k = C_n(t) \Rightarrow \forall n' \colon n' \in \mathcal{N} \wedge n \neq n' \colon \forall k' \colon \pi_{k'} \neq A_{n'}(t)$

(2) $\pi_i = REQ(n, t) \wedge \pi_j \in REQ(n', t) \wedge \pi_k = A_n(t) \Rightarrow \forall n' \colon n' \in \mathcal{N} \wedge n \neq n' \colon \forall k' \colon \pi_{k'} \neq C_{n'}(t)$

Proof. (1) It is trivially proven by Lemma 8.5 and Lemma 8.4. (2) By contradiction, $\pi_{k'} = C_{n'}(t)$. By Lemma 8.5 and Lemma 8.4, $\pi_{k''} = C_n(t)$ is in β. By Property 4.2.1 or Property 4.3.1, a contradiction appears since both $\pi_{k''} = C_n(t)$ and $\pi_k = A_n(t)$ can not be in $\beta|EDB_n^A$. □

Theorem 8.2. *For every $\beta \in behs(RDBS^A)$, it holds that:*

(1) $\pi_k = C_n(t) \Rightarrow \forall n' : n' \in \mathcal{N} : \forall k' : \pi_{k'} \neq A_{n'}(t)$

(2) $\pi_k = A_n(t) \Rightarrow \forall n' : n' \in \mathcal{N} : \forall k' : \pi_{k'} \neq C_{n'}(t)$

Proof. For $n = n'$ the properties are proved trivially by Property 4.2.1 and Property 4.3.1. Thus, let $n \neq n'$. (1) By contradiction, $\pi_k = C_n(t)$ and $\pi_{k'} = A_{n'}(t)$. By Property 4.2.1 and Property 4.3.1. $\pi_j \in REQ(n,t)$ is in β. If $\pi_i \in REQ(n',t)$ is in β, then Corollary 8.2 leads to a contradiction. Otherwise, by Property 4.2.1 $n' = site(t)$ and t follows sequence (c). This means that $\pi_r = ready_{n'}(t, data)$ does not exist. Then, $n \neq site(t)$ and thus $\pi_j = apply_n(t, data)$ is in β. Therefore, by Theorem 4.1 a contradiction appears. (2) By contradiction, $\pi_k = A_n(t)$ and $\pi_{k'} = C_{n'}(t)$. We can prove it as done before, swapping n by n' and vice versa. □

Theorem 8.3. *For every behavior $\beta \in behs(RDBS^A)$, it holds that, for any prefix $\beta(j) \in \beta$, $log(\beta(j)|EDB_n^A) \preceq log(\beta(j)|EDB_{n'}^A)$ or vice versa.*

Proof. By Property 8.3.2, $trans(\beta(j)|REQ(n)) \preceq trans(\beta(j)|REQ(n'))$ or vice versa. Let us assume that $trans(\beta(j)|REQ(n)) \preceq trans(\beta(j)|REQ(n'))$. Then, there must exist $i < j$ such that $trans(\beta(j)|REQ(n)) = trans(\beta(i)|REQ(n'))$. Let $\pi_j \in REQ(n,t)$ and $\pi_i \in REQ(n',t)$. Moreover, by Lemma 8.5, $trans(\beta(j)|\{C_n(t) : t \in T\}) = trans(\beta(i)|\{C_{n'}(t) : t \in T\})$. By Corollary 4.1 and Definition 3.2, $log(\beta(j)|EDB_n^A) = log(\beta(i)|EDB_{n'}^A)$. Since $i < j$, then $log(\beta(j)|EDB_n^A) \preceq log(\beta(j)|EDB_{n'}^A)$. □

Theorem 8.4. *For every behavior $\beta \in behs(RDBS^A)$, it holds that:*

$\pi_i = B_{site(t)}(t) \wedge \pi_j = C_n(last(i, site(t), \beta)) \wedge \pi_k = C_n(t) \wedge site(t) \neq n \Rightarrow \forall t' \in T : \neg conflict(t', t, j, \beta(k)|EDB_n^A)$.

Proof. By Property 4.3.1, as $site(t) \neq n$, $\pi_r \in apply_n(t, data)$ with $r < k$ is in $\beta|EDB_n^A$. By Theorem 4.1 and Property 8.1.1 $data.last = last(i, site(t), \beta)$. Thus, by Lemma 8.2.2, we can conlude $\forall t' \in T : \neg conflict(t', t, j, \beta(k)|EDB_n^A)$. □

9 Replication Protocol Implementation

After having presented the replicated system $RDBS^A$ in Section 8, this Section presents a new refinement that models an actual implementation, considering distribution and communication issues. This new system, named $RDBS^B$, is shown in Figure 16. The database at each system site $n \in \mathcal{N}$ is modeled by means of the module EDB_n^A proposed in the previous Section. In contrast, the $RDBS^B$ extends the replication protocol DRP^A of the $RDBS^A$ so as to describe the replication protocol from an implementation-oriented point of view.

In the $RDBS^B$, the replication protocol consists of a set of modules RP_n (where each RP_n is located at site $n \in \mathcal{N}$), along with a module AB which is in charge of the communication among the RP_n modules. Each RP_n interacts with the corresponding EDB_n^A. Upon receiving a $ready_{site(t)}(t, data)$ input action from the $EDB_{site(t)}^A$, the $RP_{site(t)}$ sends a message $\langle t, data \rangle$ to all system replicas by means of the AB module, which provides an atomic broadcast communication primitive [8]. When receiving such message, at the delegate site the $RP_{site(t)}$ requests the commit of transaction t by executing $commit_{site(t)}(t)$ (which is an input action of the $EDB_{site(t)}^A$), whereas at each remote site $n' \neq site(n)$ the $RP_{n'}$ will execute $apply_{n'}(t, data)$. Once the RP_n receives a $C_n(t)$ or $A_n(t)$ input action from the EDB_n^A in response of either a $commit_n(t)$ (for $n = site(t)$) or an $apply_n(t, data)$ (for $n \neq site(t)$) action, it can process the next message. Since messages are sent using atomic broadcast, they are delivered at all sites in the same order, and therefore the $apply()$ and $commit()$ operations are executed in the same order at all sites.

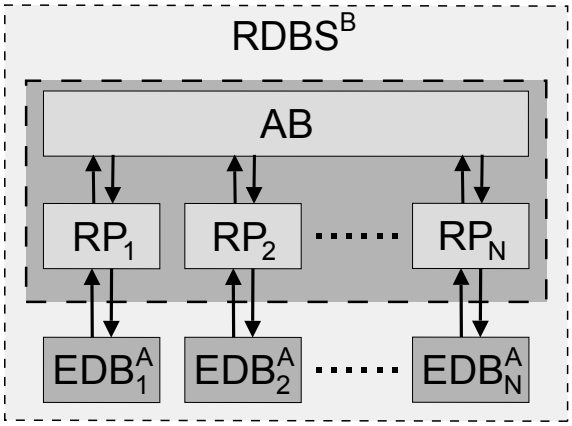

Figure 16: Composition of the Implementation of the Replicated Database System.

9.1 Atomic Broadcast

We first introduce the properties of the atomic broadcast communication primitive by means of the AB module and described in Figure 17. Variable M denotes the set of possible messages that all RP_n can send. The automaton RP_n at site $n \in \mathcal{N}$ makes use of two primitives which conform the main possible actions of this component: $send_n(m)$ and $receive_n(m)$. The former is used by the RP_n to broadcast a message m, whereas the latter allows the RP_n to receive in total order a message m that was previously broadcast by some replica. The module AB also includes an input action to model the failure of site n ($crash_n$).

- Signature:
  $in(AB) = \{crash_n, send_n(m) : n \in \mathcal{N}, m \in M\}$
  $out(AB) = \{receive_n(m) : n \in \mathcal{N}, m \in M\}$
- A set $behs(AB)$ of behaviors

Figure 17: Module AB.

In the following, we provide the assumptions verified by $behs(AB)$. First, the definition of $delivered_n$ as the sequence of messages delivered at site $n \in \mathcal{N}$ is presented.

Definition 9.1. *Let β be a behavior of AB. For every prefix $\beta(j)$, $0 \leq j \leq |\beta|$, the sequence of delivered messages at each site $n \in \mathcal{N}$ by the AB module at $\beta(j)$ is recursively defined as follows:*

$$delivered_n(\beta(j)) = \begin{cases} \textit{empty} & \Leftrightarrow\ j = 0 \\ delivered_n(\beta(j-1)) \cdot m & \Leftrightarrow\ \pi_j = receive_n(m) \wedge j > 0 \\ delivered_n(\beta(j-1)) & \Leftrightarrow\ \text{otherwise} \end{cases}$$

The replication protocol requires all messages to be delivered in the same order to all available replicas in order to have a correct behavior. This feature is provided by the atomic broadcast communication primitive of the AB module. However, the conventional uniform total order broadcast properties [8] do not avoid the contamination phenomenon [29], as they do not prevent gaps in the delivery sequence. This is a serious problem, because a faulty site could reach an inconsistent state due to delivery gaps before crashing and broadcast a message based on its inconsistent state, thus contaminating other sites. In order to avoid contamination, for any two replicas, the set of delivered messages of one must be prefix of the other or vice versa. This property, known as *prefix order* delivery [29], is formalized in Property 9.1, along with the rest of properties that model the atomic broadcast communication primitive.

Property 9.1. *(Atomic Broadcast) For each behavior $\beta \in behs(AB)$:*

(1) *(Crash Failures)* $\pi_i \in \{receive_n(m) \colon m \in M\} \Rightarrow \forall\, k \colon k < i \colon \pi_k \neq crash_n$

(2) *(Message Uniqueness)* $\pi_i = send_n(m) \wedge \pi_j = send_{n'}(m) \Rightarrow i = j$

(3) *(Delivery Integrity)* $\pi_i = receive_n(m) \Rightarrow \exists\, n' \in \mathcal{N} \colon (\exists\, k \colon k < i \colon \pi_k = send_{n'}(m))$

(4) *(No Duplication)* $\pi_i = receive_n(m) \wedge \pi_j = receive_n(m) \Rightarrow i = j$

(5) *(Prefix Order)* $delivered_n(\beta(j)) \preceq delivered_{n'}(\beta(j))$ *or vice versa, for every prefix* $\beta(j) \preceq \beta$

(6) *(Validity)* $\pi_i = send_n(m) \Rightarrow \exists\, k \colon k > i \colon \pi_k = receive_n(m) \vee \pi_k = crash_n$

(7) *(Uniform Agreement)* $\pi_i = receive_n(m) \Rightarrow \exists\, k \colon \pi_k = receive_{n'}(m) \vee \pi_k = crash_{n'}$

In the previous property, condition 1 (Crash Failures) states that after a $crash_n$ event the site n stops its activity; condition 2 (Message Uniqueness) indicates that each message is unique and different from the rest; conditions 3 (Delivery Integrity) and 4 (No Duplication) state that every site delivers each message at most once and only if it was previously sent by some site; condition 5 (Prefix Order) guarantees that messages are delivered in the same total order withoug gaps even in the case of faulty sites; condition 6 (Validity) indicates that if a correct site invokes a broadcast event, then this site will eventually deliver the message; and condition 7 (Uniform Agreement) states that if a site (correct or faulty) delivers a message, then all correct sites will eventually deliver it.

9.2 Replication Protocol: I/O Automaton $\mathrm{RP_n}$

Figure 18 presents the implementation of the replication protocol RP_n at a given replica $n \in \mathcal{N}$ by means of an I/O automaton. The components used by a RP_n are its corresponding database system EDB_n^A and the AB. This is shown in the signature of the RP_n automaton in Figure 18, as its input ($in(RP_n)$) and output ($out(RP_n)$) correspond with the actions of the EDB_n^A and AB modules.

The different state variables used by the RP_n are the following:

- Variable $site\_status$ (initially set to $alive$) monitors the state of site n and indicates whether it is alive or crashed.
- Variable to_send is a subset of messages M where $M = T \times D$ (recall that subset D was defined in Section 4.1), and initially $to\_send = \emptyset$. This variable contains messages $\langle t, data \rangle$ that correspond to all local transactions $t \in T$ (with $site(t) = n$). Every message contained in this variable must be sent using the atomic broadcast primitive to decide the outcome of t.
- Variable $received$ is a FIFO queue (initially empty) containing the previous messages in the order they were delivered by the AB module. This is the order in which the $apply_n(t, data)$ and $commit_n(t)$ operations are executed.
- Variable $sent(t)$ indicates whether a local transaction t has already sent or not its message $\langle t, data \rangle$ to the rest of replicas (initially false). Thus, the RP_n can only send the message corresponding to a transaction t at most once.
- Variable $status(t)$ monitors the states that each transaction t goes through: $\perp$ (which denotes that the state is unknown and is used as a default value that avoids the repetition of actions), $commit$ (a local transaction that has requested its commit), $apply$ (a remote transaction that is being applied), $committed$ (the transaction has been committed) and $aborted$ (the transaction has been aborted). It is worth noting the difference between the status commit and apply for a transaction t; they are used for avoiding multiple invocations of the same actions, in this case $commit_n(t)$ and $apply_n(t, data)$, respectively.

In order to define fair executions and fair behaviors [1,30], the tasks for each transaction t are also given in Figure 18. This means that if a task is continuously enabled in a fair execution (some of its respective actions are enabled, i.e. some of the preconditions are true), then it will eventually execute any of its enabled actions.

The effects of actions presented in Figure 18 are self-explanatory with the exception of some aspects that are detailed in the following. Note that $B(t)$ has no effects because the replication protocol does

AUTOMATON $\mathbf{RP_n}$

SIGNATURE:
$in(RP_n) = \{B_n(t),\ ready_n(t, data),\ C_n(t),\ A_n(t),\ receive_n(m),\ crash_n : t \in T, m \in M, data \in D\}$
$out(RP_n) = \{commit_n(t),\ apply_n(t, data),\ send_n(m) : t \in T, m \in M, data \in D\}$

STATES:
$site_status \in \{$alive, crashed$\}$, initially $site\_status =$ alive
to_send a subset of M, initially $to\_send = \emptyset$
$received$ a FIFO queue of M, initially $received =$empty
$sent(t) \in \{true, false\}$ for all $t \in T$, initially $sent(t) = false$
$status(t) \in \{\perp,$ commit, apply, aborted, committed$\}$ for all $t \in T$, initially $status(t) = \perp$

TASKS:
For every $t \in T$: $commit_n(t), apply_n(t, data), \{send_n(\langle t, data\rangle) :\ data \in D\}$

TRANSITIONS:

B$_n(t)$
eff $\equiv$

ready$_n(t, data)$
eff $to\_send \leftarrow to\_send \cup \{\langle t, data\rangle\}$

send$_n(\langle t, data\rangle)$
pre $\equiv site\_status =$ alive $\wedge$
$site(t) = n\ \wedge$
$\neg sent(t)\ \wedge$
$\langle t, data\rangle \in to\_send$
eff $\equiv to\_send \leftarrow to\_send \setminus \{\langle t, data\rangle\}$
$sent(t) \leftarrow true$

receive$_n(\langle t, data\rangle)$
eff $\equiv received \leftarrow received \cdot \langle t, data\rangle$

crash$_n$
eff $\equiv site\_status \leftarrow$ crashed

C$_n(t)$
eff $\equiv$ if $\langle t, data\rangle = head(received)$ then
$received \leftarrow tail(received)$;
$status(t) \leftarrow$ committed

A$_n(t)$
eff $\equiv$ if $status(t) \in \{$commit, apply$\} \wedge$
$\langle t, data\rangle = head(received)$ then
$received \leftarrow tail(received)$;
$status(t) \leftarrow$ aborted

commit$_n(t)$
pre $\equiv site\_status =$ alive $\wedge$
$site(t) = n\ \wedge$
$status(t) = \perp\ \wedge$
$\langle t, data\rangle = head(received)$
eff $\equiv status(t) \leftarrow$ commit

apply$_n(t, data)$
pre $\equiv site\_status =$ alive $\wedge$
$site(t) \neq n\ \wedge$
$status(t) = \perp\ \wedge$
$\langle t, data\rangle = head(received)$
eff $\equiv status(t) \leftarrow$ apply

Figure 18: Replication Protocol at site $n \in \mathcal{N}$: I/O automaton RP_n

not actually start dealing with t until $ready_n(t, data)$ is executed. Note also that a single message per transaction t is sent. Hence, it makes sense to remove the message $\langle t, data\rangle$ from the $received$ queue, in the effects of action $A_n(t)$, since it will be in that queue in case it was a local transaction and $ready_n(t, data)$ had been executed before. Moreover, the boolean variable $sent(t)$ controls that each local transaction is broadcast at most once, since the RP_n does not know in advance how many actions $ready_n(t, data)$ will the EDB_n^A invoke. Recall that the automaton RP_n is input enabled, hence these input actions can be invoked at any time and several times. Thus, the automaton is responsible for managing all possible error situations in actions $B_n(t)$, $ready_n(t, data)$, $C_n(t)$ and $A_n(t)$. However, we have not included all possible cases for the sake of simplicity, assumming that the environment has a correct behavior.

9.3 Composition and Correctness Proof

As outlined in the introduction of this Section, the system that models an actual implementation of a replicated database system, called $RDBS^B$, has the following components: a database at each system site $n \in \mathcal{N}$ modeled by the EDB_n^A module presented in Section 8, and a refinement of the DRP^A module.

Such refinement consists of a set of modules RP_n (with $n \in \mathcal{N}$), along with the AB module. Note that $(\Pi_{n\in\mathcal{N}} RP_n) \times AB$ represents a composition with the same signature as the DRP^A module, with the exception of actions $send_n(m)$ and $receive_n(m)$ (which are output actions of $(\Pi_{n\in\mathcal{N}} RP_n) \times AB$ but do not appear in the signature of DRP^A).

In order for the refinement of the DRP^A to have the same signature as the DRP^A, it is only necessary to hide these two actions as if they were internal actions. To this end, we make use of the hiding operation described in [31], which merely classifies a certain set of actions as internal actions. Thus, we define a new module that represents the implementation of the replication protocol, denoted by IRP, with $IRP = Hide_\phi((\Pi_{n\in\mathcal{N}} RP_n) \times AB)$, being $\phi = \{send_n(m), receive_n(m) : m \in M\}$. Once this new module IRP has been introduced, the $RDBS^B$ can be formally defined as $RDBS^B = (\Pi_{n\in\mathcal{N}} EDB_n^A) \times IRP = (\Pi_{n\in\mathcal{N}} EDB_n^A) \times Hide_\phi((\Pi_{n\in\mathcal{N}} RP_n) \times AB)$.

In order to ensure that the $RDBS^B$ is correct, we must prove that $behs(RDBS^B) \subseteq behs(RDBS^A)$. Since $sig(IRP) = sig(DRP^A)$ (interpreting the IRP module as an automaton), it suffices to prove that $fairbehs(IRP) \subseteq behs(DRP^A)$, i.e. $fairbehs(IRP)$ satisfies Property 4.5 and Property 8.3 (where $fairbehs(IRP)$ stands for the set of fair behaviors [1] of the IRP automaton). Recall that these properties are the ones that allow the DRP^A, in composition with $\Pi_{n\in\mathcal{N}} EDB_n^A$, to form the correct system $RDBS^A$.

According to the I/O Automaton model, an execution α of an automaton is described as a sequence $s_0\pi_1 s_1 \cdots \pi_k s_k \cdots$, where each s_i is a state, each π_i is an action, and (s_i, π_i, s_{i+1}) denotes a transition. In our case, $\alpha \in execs(IRP)$, and for each state s_i, $s_i[n]$ represents the state of the RP_n. The fair executions satisfy the progress properties of the IRP defined by the tasks of each RP_n and the progress properties of the AB module (assuming that the AB module is implemented as an I/O automaton).

As the IRP has $\Pi_{n\in\mathcal{N}} EDB_n^A$ as its environment, the proof assumes some of the well-formedness conditions of $\Pi_{n\in\mathcal{N}} EDB_n^A$ for the sake of simplicity. For instance, it is considered that $ready_n(t, data)$ occurs at most once in $\alpha \in execs(IRP)$. The same assumption is made in the case of $C_n(t)$ or $A_n(t)$ (see Properties 4.1, 4.2.1 and 4.3.1). In any case, the IRP must not violate the well-formedness of $\Pi_{n\in\mathcal{N}} EDB_n^A$ when it behaves according to its specification.

The following lemmas help us show that $fairbehs(IRP)$ satisfies Property 4.5 (as proven in Theorem 9.1) and Property 8.3 (as shown in Theorem 9.2).

Lemma 9.1. *Let α be an execution of IRP. It holds that:*

(1) $\pi_i = send_n(m) \Rightarrow \forall\, k\colon k < i\colon \pi_k \neq crash_n$

(2) $\pi_i = send_n(\langle t, data\rangle) \wedge \pi_j = send_{n'}(\langle t, data'\rangle) \Rightarrow i = j \wedge n = n' = site(t) \wedge data = data' \wedge \exists k\colon k < i\colon \pi_k = ready_{site(t)}(t, data)$

Proof. Let us prove each assertion separately:

- By contradiction, $\pi_i = send_n(m)$ and $\pi_k = crash_n$ with $k < i$. Then, $s_k[n].site\_status =$ crashed. No other action can change that status. So, $send_n(m)$ is disabled at every reachable state of α from s_k and hence $\pi_i \neq send_n(m)$.
- By the preconditions of π_i and π_j, $site(t) = n$ and $site(t) = n'$. By Assumption 4.1, the delegate site of t is unique. Then, $n = n' = site(t)$. If $i \neq j$ and $i < j$, then $s_i[n].sent(t) =$ true. No other action can change that value. So, $send_n(\langle t, data'\rangle)$ is disabled at every reachable state of α. Therefore, $i = j$ and then $data = data'$. That is, it is the only $send_n(\langle t, data\rangle)$ action for transaction t. Finally, by the preconditions of π_i, $\langle t, data\rangle \in s_{i-1}[n].to\_send$. The only acion that makes this happen is $ready_n(t, data)$. Thus, as $n = site(t)$, $\exists k\colon k < i\colon \pi_k = ready_{site(t)}(t, data)$ holds.

□

Lemma 9.2. *Let α be an execution of IRP. It holds that:*

(1) $\pi_i = receive_n(\langle t, data\rangle) \wedge \pi_j = receive_n(\langle t, data'\rangle) \Rightarrow i = j \wedge data = data'$

(2) $\langle t, data\rangle = head(s_i[n].received) \Rightarrow \exists j\colon j < i\colon \pi_j = ready_{site(t)}(t, data)$

Proof. Let us prove each assertion separately:

- By Property 9.1.3, $\pi_k = send_{n'}(\langle t, data\rangle)$ and $\pi_m = send_{n''}(\langle t, data'\rangle)$. By Lemma 9.1, $\pi_k = \pi_m$ and $data = data'$. Then, by Property 9.1.4, $\pi_i = \pi_j$ and $i = j$.
- The only action that makes at some previous state s_r in α that $\langle t, data\rangle \in s_r[n].received$ is $\pi_r = receive_n(\langle t, data\rangle)$. By Property 9.1.3, $\pi_m = send_{n'}(\langle t, data\rangle)$ with $m < r$ happened in α. By its preconditions, $site(t) = n'$ and $\langle t, data\rangle \in s_{m-1}[n'].to\_send$. The only action making $\langle t, data\rangle \in s_{m-1}[n'].to\_send$ is $\pi_k = ready_{n'}(t, data)$ with $k < m$ and $n' = site(t)$.

□

Theorem 9.1. *Let α be an execution of IRP. It holds that:*

(1) $\pi_i \in REQ(n) \Rightarrow \forall k\colon k < i\colon \pi_k \neq crash_n$

(2) $\pi_i = commit_n(t) \Rightarrow \exists j\colon j < i\colon \pi_j = ready_n(t, data) \wedge \forall k\colon i < k < j\colon \pi_k \neq commit_n(t)$

(3) $\pi_i = apply_n(t, data) \Rightarrow \forall k\colon k < i\colon \pi_k \neq apply_n(t, data') \wedge site(t) \neq n$

(4) $\pi_i = apply_n(t, data) \Rightarrow \exists k\colon k < i\colon \pi_k = ready_{site(t)}(t, data) \wedge site(t) \neq n$

Proof. Let us prove each assertion separately:

- By contradiction, $\pi_i \in REQ(n)$ and $\pi_k = crash_n$ with $i < k$. Then, $s_k[n].site\_status =$ crashed. No other action can change that status. Any $\pi_i \in REQ(n)$ is disabled at every reachable state of α from s_k and hence $\pi_i \notin REQ(n)$.
- By the precondition of π_i, $\langle t, data\rangle = head(s_{i-1}[n].received)$ and $site(t) = n$. By Lemma 9.2.2, there exists $\pi_j = ready_n(t, data)$ with $j < i$ for some $data \in D$. Let us assume by contradiction that $\pi_k = commit_n(t)$ with $j < k < i$. By its effects, $s_k[n].status(t) =$ commit. Thus, π_i is disabled since there is no action that can set $status(t)$ to $\perp$ again. A contradiction.
- By contradiction, $\pi_i = apply_n(t, data)$ and $\pi_k = apply_n(t, data')$ with $k < i$. By the preconditions of π_k, $site(t) \neq n$. By the effects of π_k, $s_k[n].status(t) =$ apply and no other action makes $status(t) = \perp$ again. Then, $\pi_i = apply_n(t, data)$ is disabled at every reachable state of α from s_k and hence $\pi_i \neq apply_n(t, data)$.
- By the precondition of π_i, $\langle t, data\rangle = head(s_{i-1}[n].received)$ and $site(t) \neq n$. By Lemma 9.2.2, $\pi_k = ready_{site(t)}(t, data)$ with $k < i$ is in α. As $site(t) \neq n$ the property holds.

□

Lemma 9.3. *Let α be a fair execution of IRP. It holds that:* $\langle t, data\rangle = head(s_i[n].received) \wedge s_i[n].site\_status =$ *alive* $\wedge \forall j\colon j < i\colon \pi_j \notin REQ(n, t) \Rightarrow \exists k\colon k > i\colon \pi_k \in REQ(n, t) \vee \pi_k = crash_n$

Proof. By contradiction, neither $\pi_k \in REQ(n, t)$ nor $\pi_k = crash_n$ with $k > i$ are in α. If $site(t) \neq n$ and $\pi_k = apply_n(t, data)$ is not in α, then by Property 4.3 there is no other action from $acts(EDB_n, t)$. If $site(t) = n$, as $\langle t, data\rangle = head(s_i[n].received)$, by Lemma 9.2.2, there exists $\pi_q = ready_n(t, data)$ with $q < i$ in α and then, by Property 4.2, there cannot be other action from $acts(EDB_n, t)$.

Therefore, in any case, $status(t)$ is never modified and hence $s_r[n].status(t) = \perp$. Besides, for any reachable state s_r with $r > i$, $s_r[n].site\_status =$ alive (as there is no $crash_n$ in α); and $\langle t, data\rangle = head(s_r[n].received)$, as there is no $\pi_k \in \{C_n(t), A_n(t)\}$ that removes $\langle t, data\rangle$ from $head(received)$. Thus, for any state s_r with $r > i$ all the preconditions of either $commit_n(t)$ if $site(t) = n$ or $apply_n(t, data)$ if $site(t) \neq n$ hold. As α is a fair behavior, either $commit_n(t)$ if $site(t) = n$ or $apply_n(t, data)$ if $site(t) \neq n$ must be eventually executed. Thus, we get a contradiction.

□

Before the following Lemma, we introduce some additional notation. Let us denote the distance of a given message $m \in M$ ($m = \langle t, data\rangle$) to the head of queue $received$ at a given replica $n \in \mathcal{N}$ in a given state s_i as $D(m, head(s_i[n].received))$.

Lemma 9.4. *Let $\alpha = s_0\pi_1 s_1 ... s_{i-1}\pi_i s_i$ be a fair execution of IRP. Then, it holds that:*

$$D(m, head(s_i[n].received)) > 0 \wedge s_i[n].site\_status = \textit{alive} \Rightarrow$$
$$\exists k\colon k > i\colon D(m, head(s_k[n].received)) < D(m, head(s_i[n].received)) \vee \pi_k = crash_n$$

Proof. Let $\langle t, data\rangle = head(s_i[n].received)$. By contradiction, $\pi_k = crash_n$ with $k > i$ never happens in α and $D(m, head(s_k[n].received)) = D(m, head(s_i[n].received))$ for any reachable state $s_k > s_i$ at α. Then, we can consider two cases:

- If there exists $\pi_j \in REQ(n,t)$ with $j < i$, then by Property 8.1.4, $\pi_k \in \{C_n(t), A_n(t)\}$ with $k > j$ happens in α.
- If there does not exist $\pi_j \in REQ(n,t)$ with $j < i$, then by Lemma 9.3 there exists $\pi_j \in REQ(n,t)$ with $j > i$ in α. By Property 8.1.4, there exists $\pi_k \in \{C_n(t), A_n(t)\}$ in α with $k > j > i$.

By the effects of π_j, $s_{j-1}[n].status(t) \in \{\text{commit}, \text{apply}\}$. Then, by Property 4.2 or Property 4.3, there is no other action until s_{k-1} that modifies $status(t)$ and hence by the effects of any case of $\pi_k \in \{C_n(t), A_n(t)\}$, $\langle t, data\rangle \notin s_k[n].received$. Thus, if $k > i$, then we get a contradiction since $\langle t, data\rangle$ is removed from $head(received)$ at a state s_k with $k > i$ and hence $D(m, head(s_k[n].received)) < D(m, head(s_i[n].received))$. Otherwise, if $k < i$, by Property 9.1.4 $\langle t, data\rangle \notin s_{k'}[n].received$ for any reachable state $s_{k'} > s_k$. However, $\langle t, data\rangle = head(s_i[n].received)$, in contradiction. □

Theorem 9.2. *Let α be an fair execution of IRP and β the fair behavior of α. It holds that:*

(1) $\pi_i \in REQ(n,t') \wedge \pi_j \in REQ(n,t) \wedge i < j \Rightarrow \exists k\colon i < k < j\colon \pi_k \in \{C_n(t'), A_n(t')\}$

(2) $trans(\beta(j)|REQ(n)) \preceq trans(\beta(j)|REQ(n'))$ *or vice versa for every prefix* $\beta(j) \preceq \beta$

(3) $\pi_i = ready_n(t, data) \Rightarrow \exists k\colon k > i\colon \pi_k \in \{commit_n(t), crash_n\}$

Proof. Let us prove each assertion separately:

- By contradiction, $\pi_k \notin \{C_n(t'), A_n(t')\}$ with $i < k < j$. By the preconditions of π_i and π_j, $head(s_{i-1}[n].received) = \langle t', data'\rangle$ and $head(s_{j-1}[n].received) = \langle t, data\rangle$. By Theorem 9.1.2 and 9.1.3, as $i \neq j$, then $t \neq t'$. The only action that removes $\langle t', data'\rangle$ from $head(received)$ is $\pi_k \in \{C_n(t'), A_n(t')\}$. A contradiction.
- Let $delivered$ be a history variable defined in the states of RP_n at each $n \in \mathcal{N}$ where $s_0[n].delivered =$ empty and $s_i[n].delivered = s_{i-1}[n].delivered \cdot \langle t, data\rangle$ when $\pi_i = received_n(\langle t, data\rangle)$. Let $trans(\beta(j)|REQ(n)) = t_1 ... t_k ... t_m$, then it holds that $s_{j-1}[n].delivered = \langle t_1, data_1\rangle ... \langle t_k, data_k\rangle ... \langle t_m, data_m\rangle$ since $\langle t_k, data_k\rangle$ has been in $head(s_k[n].received)$ for some $k \leq j-1$ by the preconditions of the actions in $REQ(n)$. In the same way, $trans(\beta(j)|REQ(n'))$ implies that $s_{j-1}[n'].delivered = \langle t'_1, data'_1\rangle ... \langle t'_k, data'_k\rangle ... \langle t'_m, data'_m\rangle$. Moreover, by contradiction, $trans(\beta(j)|REQ(n)) \not\preceq trans(\beta(j)|REQ(n'))$ implies that $s_{j-1}[n].delivered \not\preceq s_{j-1}[n'].delivered$. Since $s_{j-1}[n].delivered$ satisfies Definition 9.1 for every $n \in \mathcal{N}$, then by Property 9.1.5, $delivered_n(\beta(j)|AB) \preceq delivered_{n'}(\beta(j)|AB)$ or vice versa. Therefore, $s_{j-1}[n].delivered \preceq s_{j-1}[n'].delivered$ or vice versa. A contradiction.
- By contradiction, $\pi_k \notin \{commit_n(t), crash_n\}$ with $k > i$ in α. If there is no $crash_n$ in α, $site\_status =$ alive for any reachable state of α from s_i. By Lemma 9.1.2, $\pi_{k'} = send_n(\langle t, data'\rangle)$ with $k' < i$ cannot happen in α. As $send_n(\langle t, data\rangle)$ is the only action that modifies $sent(t)$ and initially $s_0[n].sent(t) =$ false, then $s_i[n].sent(t) =$ false. By the effects of π_i, $\langle t, data\rangle \in s_i[n].to\_send$. By Property 4.2, the only action that can follow the $ready_n(t, data)$ is a $commit_n(t)$ and hence there is no other action in $acts(EDB_n, t)$ that changes the initial value of $status(t)$, that is, $status(t) = \perp$ for any reachable state of α from s_i. Finally, by Property 4.2, $site(t) = n$. Thus, all the preconditions of $send_n(\langle t, data\rangle)$ are enabled from s_i on and therefore $\pi_j = send_n(\langle t, data\rangle)$ with $j > i$ will be executed eventually. By Property 9.1.6, $\pi_r = receive_n(\langle t, data\rangle)$ with $r > j$ happens in α. By the effects of π_r, $\langle t, data\rangle \in s_r[n].received$. By Lemma 9.4, $\langle t, data\rangle$ will become $head(s_m[n].received)$ at some state s_m of α with $m > r$. Thus, all the preconditions of $commit_n(t)$ are enabled from s_m and therefore $\pi_k = commit_n(t)$ with $k > m > r > j > i$ will be executed eventually. A contradiction.

□

Remark 9.1. *Theorem 9.1 and Theorem 9.2 guarantee the correction of the $RDBS^B$ and, therefore, the implementation of the replication protocol by means of the composition of the set of RP_n modules and the AB module is correct for the databases considered in this work, i.e., databases that run under a variety of*

isolation levels and with integrity constraints. The proposed implementation, which makes use of atomic broadcast, provides a property related to uniformity. Thanks to Property 9.1.7 (Uniform Agreement), which has not been used so far, uniformity of committed transactions is an straightforward property, as formalized in the following Lemma.

Lemma 9.5. *Let α be an execution of the IRP. It holds that:* $\pi_i = C_n(t) \Rightarrow \forall n' \in \mathcal{N} : \exists k : \pi_k = C_{n'}(t) \vee \pi_k = crash_{n'}$

10 Conclusions

Deferred-update replication protocols have been regularly used in the database replication field, since they are appropriate for update-everywhere approaches, i.e. in systems where every replica may receive and directly serve different client transactions. These protocols execute the transaction operations in a delegate site, propagating later its writeset to other replicas in total order. This ensures a high level of consistency and an asymmetric workload management that boosts performance, since writeset application at remote sites requires less effort than a local transaction service.

Up to our knowledge, there exists no general formalization of this kind of replication protocols that considers transactions running under different isolation levels, supporting both integrity constraints and crash failures. We have provided a detailed specification of a replicated database system that fills this void. To this end, we have used the Input/Output Automaton model.

The specification has been modularly structured; i.e., it distinguishes different components that need to interact for managing transactions in any distributed architecture: the DBMS being used in each replica, the replication protocol and the communication support. The use of the I/O Automaton model allows us to develop hierarchical proofs. Starting with an abstract level of specification, we can refine and prove other modules with lower levels of abstraction, which correspond to a coarse problem solution and finally to a specific solution of the problem. The correctness proof for a given level is based on satisfying the properties of the previous abstraction level. This simplifies the proofs.

The properties stated at the top-most level of this hierarchical refinement provide the correctness criteria ensuring that the replicated system will behave as a single one. So, they can be considered as a new proposal for one-copy equivalence criteria, specifically tailored for deferred-update replication protocols. Several relaxations of these criteria and their corresponding constraints on the assumptions taken in the current paper (full replication, crash failures, update-everywhere server architecture, non-constrained transaction start time, no database partitioning, support for multiple isolation levels) have been analysed. This analysis shows that those additional restrictions on the system assumptions do not compromise the obtained correctness criteria. So, they would still be applicable to systems with, e.g., partial replication, passive replication or partitioned databases; thus improving the scalability of the resulting deferred-update protocols.

References

[1] N. A. Lynch and M. R. Tuttle, "An introduction to input/output automata," *CWI-Quarterly*, vol. 2, no. 3, pp. 219–246, 1989.

[2] N. A. Lynch and M. R. Tuttle, "Hierarchical correctness proofs for distributed algorithms," in *PODC '87: Proceedings of the 6th ACM Symposium on Principles of Distributed Computing*, (New York, NY, USA), pp. 137–151, ACM, 1987.

[3] J. Gray, P. Helland, P. O'Neil, and D. Shasha, "The dangers of replication and a solution," in *SIGMOD '96: Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, (New York, NY, USA), pp. 173–182, ACM, 1996.

[4] M. Wiesmann and A. Schiper, "Comparison of database replication techniques based on total order broadcast," *IEEE Trans. on Knowl. and Data Eng.*, vol. 17, no. 4, pp. 551–566, 2005.

[5] M. Fischer, "The consensus problem in unreliable distributed systems (a brief survey)," in *Proc. 1983 International Conference on Foundations of Computations Theory, Lecture Notes in Computer Science*, vol. 158, pp. 127–140, Springer-Verlag, 1983.

[6] T. Härder and A. Reuter, "Principles of transaction-oriented database recovery," *ACM Comput. Surv.*, vol. 15, no. 4, pp. 287–317, 1983.

[7] F. B. Schneider, "Implementing fault-tolerant services using the state machine approach: A tutorial," *ACM Comput. Surv.*, vol. 22, no. 4, pp. 299–319, 1990.

[8] G. V. Chockler, I. Keidar, and R. Vitenberg, "Group communication specifications: a comprehensive study," *ACM Comput. Surv.*, vol. 33, no. 4, pp. 427–469, 2001.

[9] N. A. Lynch, M. Merritt, W. Weihl, and A. Fekete, *Atomic Transactions: In Concurrent and Distributed Systems*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[10] A. Fekete, N. Lynch, M. Merritt, and W. Weihl, "Commutativity-based locking for nested transactions," *J. Comput. Syst. Sci.*, vol. 41, no. 1, pp. 65–156, 1990.

[11] A. Feteke, "Formal models of communication services: A case study," *IEEE Computer*, vol. 26, no. 8, pp. 37–47, 1993.

[12] D. Kuo, "Model and verification of a data manager based on ARIES," *ACM Trans. Database Syst.*, vol. 21, no. 4, pp. 427–479, 1996.

[13] A. Adya, *Weak Consistency: A Generalized Theory and Optimistic Implementations for Distributed Transactions*. PhD thesis, MIT, 1999.

[14] P. A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1987.

[15] C. H. Papadimitriou, *The Theory of Database Concurrency Control*. Computer Science Press, 1986.

[16] J. Gray and A. Reuter, *Transaction Processing: Concepts and Techniques*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1992.

[17] J. Bernabé-Gisbert, J. Armendáriz-Iñigo, R. de Juan-Marín, and F. Muñoz-Escoí., "Providing read committed isolation level in non-blocking rowa database replication protocols," in *JCSD '07: Proceedings of XV Jornadas de Concurrencia y Sistemas Distribuidos*, 2007.

[18] S. Elnikety, W. Zwaenepoel, and F. Pedone, "Database replication using generalized snapshot isolation," in *IEEE Symposium on Reliable Distributed Systems*, (Washington, DC, USA), pp. 73–84, IEEE Computer Society, 2005.

[19] Y. Lin, B. Kemme, M. Patiño-Martínez, and R. Jiménez-Peris, "Middleware based data replication providing snapshot isolation," in *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, (New York, NY, USA), pp. 419–430, ACM, 2005.

[20] F. Pedone, *The Database State Machine and Group Communication Issues (Thèse N. 2090)*. PhD thesis, École Polytecnique Fédérale de Lausanne, Lausanne, Switzerland, 1999.

[21] J. Salas, R. Jiménez-Peris, M. Patiño-Martínez, and B. Kemme, "Lightweight reflection for middleware-based database replication," in *IEEE Symposium on Reliable Distributed Systems*, (Washington, DC, USA), pp. 377–390, IEEE Computer Society, 2006.

[22] F. D. Muñoz-Escoí, J. Pla-Civera, M. I. Ruiz-Fuertes, L. Irún-Briz, H. Decker, J. E. Armendáriz-Íñigo, and J. R. González de Mendívil, "Managing transaction conflicts in middleware-based database replication architectures," in *IEEE Symposium on Reliable Distributed Systems*, (Washington, DC, USA), pp. 401–420, IEEE Computer Society, 2006.

[23] 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," *Journal of Universal Computer Science*, vol. 13, no. 8, pp. 1110–1135, 2007.

[24] R. Jiménez-Peris, M. Patiño-Martínez, G. Alonso, and B. Kemme, "Are quorums an alternative for data replication?," *ACM Trans. Database Syst.*, vol. 28, no. 3, pp. 257–294, 2003.

[25] M. Stonebraker, "The case for shared nothing," *IEEE Database Eng. Bull.*, vol. 9, no. 1, pp. 4–9, 1986.

[26] M. Patiño-Martínez, R. Jiménez-Peris, B. Kemme, and G. Alonso, "MIDDLE-R: Consistent database replication at the middleware level," *ACM Trans. Comput. Syst.*, vol. 23, no. 4, pp. 375–423, 2005.

[27] R. Kallman, H. Kimura, J. Natkins, A. Pavlo, A. Rasin, S. B. Zdonik, E. P. C. Jones, S. Madden, M. Stonebraker, Y. Zhang, J. Hugg, and D. J. Abadi, "H-store: a high-performance, distributed main memory transaction processing system," *PVLDB*, vol. 1, no. 2, pp. 1496–1499, 2008.

[28] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker, "A comparison of approaches to large-scale data analysis," in *SIGMOD Conference*, pp. 165–178, 2009.

[29] X. Défago, A. Schiper, and P. Urbán, "Total order broadcast and multicast algorithms: Taxonomy and survey," *ACM Comput. Surv.*, vol. 36, no. 4, pp. 372–421, 2004.

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

[31] N. A. Lynch and M. Tuttle, "Hierarchical correctness proofs for distributed algorithms," tech. rep., Cambridge, MA, USA, 1987.