Correctness Criteria for Replicated Database Systems with Snapshot Isolation Replicas *

J.E. Armendáriz[1], J.R. Juárez[1], J.R. González de Mendívil[1], F.D. Muñoz[2]
[1] Universidad Pública de Navarra, Spain [2] Instituto Tecnológico de Informática, Spain
{enrique.armendariz, jr.juarez, mendivil}@unavarra.es, fmunyoz@iti.upv.es

Abstract. *In this work we present the correctness criteria which a replicated database system with SI replicas must verify when deferred update protocols are used in a crash failure scenario. Our criteria proposal ensures that the replicated database system behaves like a single copy database system where transactions see a weaker form of SI, called Generalized-SI.*

Introduction. Database replication is a well-known technique to enhance system performance and afford site failures. However, there must be some coordination mechanism so that each transaction is committed at each database replica in some consistent order; this is done by a replication protocol. The classical approach assumes that each database provides serializable, but most database systems (e.g. Oracle and PostgreSQL) provide Snapshot Isolation (SI) [1] level for the sake of performance; specially for read-only transactions that never block. Thus, several replication protocol proposals with SI replicas [2, 3, 5] have been presented. These replication protocols are a sort of distributed algorithm, though no previous work uses a formalization tool that formally specifies the correctness criteria under the simplest failure model scenario such as the crash one. These criteria should be useful to guarantee the correctness of the protocols and to analyze their advantages and limitations. In this brief announcement, due to space limitations, we outline the proposal of using the Input/Output Automaton Model [6] to provide the specification of a database replication system with SI replicas. This is thoroughly explained in [4]. The presented specification stresses the properties which must fulfill components and not their specific implementations. The replicated system is shown as the composition of an abstract replication protocol and a set of databases with extended functionalities (those ones that make easier the protocol implementation). The system explicitly considers the crash failure model scenario. Actually, most of the proposed protocols are designed to tolerate at least such failure scenario. Correctness criteria require to: respect the behavior of each database replica; apply transactions in the very same order to generate the same global set of snapshots; transactions are either committed, or aborted, at all replicas or none; transactions must be allowed to progress at correct replicas. These criteria allow us to assure that transactions are executed in the replicated system as if there was a single database which executes transactions under a slightly weaker isolation level than SI, known as Generalized-SI (GSI) [3].

Replicated Database System Abstraction. The replicated database system is specified by means of a module denoted $RDBS = RP \times (\Pi_{n \in N} EDB_n)$. This module is the composition of an abstract replication protocol and a group of extended databases, one at each site of the distributed system. The set of site identifiers is denoted as N. We assume that at most f sites may fail by crash and $|N| > f$. At each site $n \in N$ there is an extended database module denoted EDB_n. We consider a *full replicated* system. The set of transactions operating in the system is T; and the set of possible versions for the items I and transactions T is V. There is a

*This work has been supported by the Spanish Government under research grant TIN2006-14738-C02-02.

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

The EDB module models the operation of a database following the SI level which includes some facilities for simplifying the control a replication protocol exercises over the database. The signature of the EDB module is:

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

By means of the action $begin(t)$, the module notifies the fact that a new transaction has been initiated. The actions $committed(t)$ and $aborted(t)$ represent the final decision about such a transaction. This module is intended to work in collaboration with a replication protocol. At some point in the execution of a transaction $t \in T$, after its action $begin(t)$, the EDB informs about the writeset the transaction is ready to install. This is done by the action $deliverws(t, ws)$. The EDB allows only the replication protocol to request the commit of the transaction via the input action $commit(t, ws)$. A transaction following such a pattern of operation is a *local transaction*. The transaction starts under the control of the extended database; and it passes the control of the transaction to the replication protocol in order to terminate it. When the replication protocol takes the decision that a transaction $t \in T$ has to be committed, it requires either the replication protocol produces the action $commit(t, ws)$ or the database applies the updates of the transaction; i.e. its writeset. Thus, the EDB provides as input action the action $apply(t, ws)$. The extended database is responsible of programming such a transaction in the underlaying database in a transparent way for the replication protocol. A transaction following such a pattern of operation is a *remote transaction*. The properties of the EDB module are introduced by presenting properties over its behaviors [6]. EDB must provide SI and generate the proper snapshot versions comprised of a sequence of update committed transactions. It must also ensure that after a crash it will stop its activity, but it will have generated well-formed transactions until that moment. Finally, it will produce no unilateral aborts and a remote transaction will only be aborted if it can not satisfy its isolation level. Other possible causes of abortion are filtered by the replication protocol. A detailed discussion of these properties are given in [4]. On the other hand, the protocol is responsible for guaranteeing the whole correctness criteria in the whole system. The signature of RP is:

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

In the following, we present and explain the correctness criteria for the replicated database system. Let β be a behavior of $RDBS$. We use the predicate $local(t, n, \beta) \equiv begin_n(t) \preceq \beta|acts(EDB_n, t)$ to indicate that a transaction $t \in T$ has started in the site $n \in N$ as a local transaction in the behavior β. The correctness criteria are indicated in the following axioms[1]. For every behavior $\beta \in behs(RDBS)$:

Well-formedness Conditions.
(a) $\beta|EDB_n \in behs(EDB_n)$
(b) $local(t, n, \beta)\ \wedge local(t, n', \beta) \Rightarrow n = n' = site(t)$
(c) $\pi_i = apply_n(t, ws) \Rightarrow \exists\, k\colon k < i\colon \pi_k = deliverws_{site(t)}(t, ws)\ \wedge\ n \neq site(t)$

Conflict Serializable.
(a) $\pi_i \in \{apply_n(t, ws), commit_n(t, ws)\} \wedge \pi_j = apply_n(t', ws')\ \wedge\ i < j\ \wedge\ ws \cap ws' \neq \emptyset \Rightarrow$
$\exists\, k\colon i < k < j\colon \pi_k \in \{committed_n(t), aborted_n(t)\}$

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

(b) $\pi_i \in \{apply_n(t, ws), commit_n(t, ws)\} \ \wedge \ \pi_{j_1} = begin_n(t') \ \wedge \ \pi_{j_2} = commit_n(t', ws') \ \wedge$
$i < j_2 \ \wedge \ ws \cap ws' \neq \emptyset \Rightarrow \exists\, k\colon i < k < j_1\colon \pi_k \in \{committed_n(t), aborted_n(t)\}$

Uniform Prefix Order Database Consistency.

For every finite prefix β' of β: $log(\beta'|EDB_n) \preceq log(\beta'|EDB_{n'})$ or vice versa.

Uniform Decision.

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

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

Local Transaction Progress.

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

Criterion 1 groups three different aspects: Criterion 1.(a) states that every behavior of the $RDBS$ has to respect the behavior of each EDB_n module. Criterion 1.(b) indicates that the first event of a transaction $t \in T$ in the system may only be $begin_{site(t)}(t)$ at its delegate site; t is local at that replica and remote otherwise. Criterion 1.(c) avoids the spontaneous creation of remote transactions in the system. All these previous criteria are grouped and form what we have denoted as Well-formedness Conditions. Criterion 2 (Conflict Serializable) guarantees that transactions with a non-empty writeset intersection are serialized. Criterion 3 (Uniform Prefix Database Order Consistency) imposes on the system to build the same snapshots at every database; actually, it obliges committed transactions to follow the same commit ordering at every site (not only the conflictive ones). Notice that if a database fails, this criterion ensures that the last installed snapshot is also a valid snapshot for the rest of the correct sites. The Criterion 4 (Uniform Decision) the replication protocol decides the same outcome for a transaction at all replicas (committed or aborted). Moreover, if a transaction committed at one site (correct or faulty) then the protocol would eventually apply or request the commit of the same transaction in every correct site; otherwise, if a transaction aborted at one site (correct or faulty) then it would be its delegate site and no one of its remote transactions will be programmed in the system. To conclude, Criterion 5 (Local Transaction Progress) indicates that if the replica is correct, then for each of its local associated transactions that requests the commit, the replication protocol either requests the commit or knows it will be aborted.

Conclusions. In this work it has been presented a new insight into how to think about formalizing replication protocols with the I/O automaton model in the presence of crash failures and SI replicas. Hence, it represents a case study that shows how to reason about a system in an abstract and compositional way.

References

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

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

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

[4] J. R. González de Mendívil, J. E. Armendáriz, and J. R. Juárez. Correctness criteria for replicated database systems with snapshot isolation replicas. Technical Report ITI-ITE-08/03, Mar 2008.

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

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