Using Transaction Isolation Levels for Ensuring Replicated Database Consistency in Mobile Computing Environments

J. M. Monteiro
PUC-Rio
Rio de Janeiro, Brazil
monteiro@inf.puc-rio.br

A. Brayner
Universidade de Fortaleza
Fortaleza, Brazil
brayner@unifor.br

S. Lifschitz
PUC-Rio
Rio de Janeiro, Brazil
sergio@inf.puc-rio.br

ABSTRACT
Replicating databases over several servers is an efficient strategy to improve data availability and to increase transaction throughput. However, due to inherent limitations of mobile and other loosely-coupled environments, the replica control protocols responsible for ensuring replicated database consistency should be revisited. This paper proposes a new replica control protocol which guarantees the consistency of replicated databases in a mobile computing environment. The key benefit of the proposed protocol is to ensure consistency by means of the notion of transaction isolation levels without the use of a locking mechanism. Thus, a user may trade the degree of replica consistency for a potential increase in data availability and transaction throughput. The proposed protocol uses a read-any/write-any approach and reduces the number of messages exchanged among the replicated servers. Experimental results show the potential efficiency of the proposed approach.

Categories and Subject Descriptors
H.2.4 [Information Systems]: Database Management—distributed databases, concurrency, transaction processing

Keywords
Concurrency Control, Data Replication, Mobile Computing

1. INTRODUCTION
Mobile computing technology has driven the development of new and sophisticated database applications. Electronic commerce applications (e.g., auctions), road traffic management systems and shared access/update of medical data and patient records [10] are examples of database applications that increasingly require support from mobile computing technology.

Such applications require access of up-to-date and consistent data, even in the presence of concurrent access (to shared data) with intermittent connectivity by the mobile clients. Furthermore, several of these applications require great data availability and high rates of transaction throughput in spite of low bandwidth, frequent disconnection of mobile devices, data mobility and large numbers of clients [12].

The notion of replicated database has been widely used as a strategy to improve data availability and to maximize transaction throughput (number of committed transactions per second). The key idea behind the concept of replicated database consists in storing multiple copies of data items (data replication) in several distributed servers, called replicated servers, interconnected by a communication network. Thus applications may access data items from any of the replicated servers, i.e., they do not interrupt their executions (waiting for data access), even if some replicated servers are not available or reachable (through the network). In a replicated database all copies of a given data item should represent the same snapshot of the real world. In other words, all copies of a given data should be in the same database state. That feature is denoted data consistency.

In conventional replicated databases, data consistency is guaranteed by the following two correctness criteria:

(C1) Let DB be a database with \( n \geq 1 \) replicas \( DB_i \), \( 0 < i \leq n \), then the concurrent execution of a set \( T \) of transactions over \( k \) replicas (\( 1 < k \leq n \)) of DB should be equivalent to a serial execution of the same set \( T \) over the same database DB without replication (one-copy database).

(C2) All replicas (copies) of a given data item \( x \) will eventually converge to a unique consistent final state, independently of the operations executed on data copies of \( x \).

The first correctness criterion is called one-copy serializability (1SR) while the second one is also known as eventual consistency. In traditional replicated DBMSs, these criteria are guaranteed by a replica control protocol, which can be classified into three groups: Primary-Copy protocols (also called of pessimistic replication), Quorum-Consensus protocols and Available-Copies protocols (also known as optimistic replication) [4].

Due to potential gains in increasing data availability and transaction throughput, data replication has also been applied to mobile computing environments. A mobile replicated database is comprised of several mobile and fixed servers and clients interconnected through a wireless network. Of course, mobile replicated databases should ensure data consistency as well. However, the 1SR correctness criterion is too restrictive for mobile replicated databases. For that reason, several approaches have been proposed in which replica control protocols use correctness criteria less restrictive than 1SR. Nonetheless, most of them require either intensive message exchange [2, 5, 9, 14] or the re-execution of log-
stored transaction operations in order to ensure data consistency.

An isolation level is the degree in which the execution of a given transaction is isolated from all other concurrent transactions. A more restrictive isolation level (e.g., serializable) enables a higher data consistency degree (avoiding undesirable phenomena like dirty reads or lost updates) but at a lower concurrency degree. On the other hand, by relaxing the isolation level, one could increase transaction concurrency degree (and transaction throughput) for the price of reducing data consistency degree. It is important to note that isolation levels are implemented by a locking mechanism which delays the execution of database operations to ensure a given isolation level. The ANSI/ISO SQL-92 standard defines four isolation levels: READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ and SERIALIZABLE.

This paper proposes a new replica control mechanism that guarantees replicated database consistency in mobile computing environments based on standard ANSI/ISO SQL 92 isolation levels [6] and eventual consistency. Besides improving data availability, the proposed replica control protocol allows users to specify ANSI/ISO SQL 92 isolation levels, which in turn enable users to choose between database consistency degree or transaction throughput rate. Observe that the concept of isolation levels, one can relax 1SR (e.g., read committed isolation level does not ensure 1SR). In the proposed mechanism, the different isolation levels are implemented without using a locking mechanism, an important feature for mobile databases. Finally, differently from other mobile database replication approaches, transaction operations are executed only once. Thus replicated servers do not need to log all executed operations and the number of messages exchanged among the servers is minimized.

The rest of the paper is organized as follows. The next Section addresses related work. In section 3, the transaction isolation levels are described and discussed. The proposed replica control protocol is presented in Section 4. To illustrate the use of the proposed replication mechanism a running example is shown in Section 5. Next, Section 6 presents experimental results. Moreover, a discussion highlighting the advantages of the proposed replicating approach against related work is presented. Section 7 concludes this paper and outlines future work.

2. RELATED WORK

The problem of ensuring data consistency in replicated mobile databases has been addressed by several works. The replica control mechanism proposed for the Bayou project [14] allows data to be updated by disconnected clients and conflicts are resolved in a pair-wise fashion during reconnections. Updates are propagated epidemically and conflicts are solved by bundling writes with code fragments (called conflict resolvers), which are application specific. Such a reconciliation-based protocol can only be applied for non-transactional domains like file systems. The Bayou replication mechanism does not support transactional semantics. Actually it uses a weak consistency notion based on session guarantees, which do not have clear semantics from the point of view of database consistency. Moreover, in Bayou approach each server stores a log containing all the write operations that it knows. That is because a given operation might be executed several times. executed when the server knows that the update has been received by all replicated servers. The main limitation of this approach is that the non-availability of any replicated server makes the commitment process infeasible.

Gifford’s original weight voting algorithm [7] provides low data availability. There are other voting-based approaches which allow the use of a great variety of quorums to decide for the commit of an update transaction. Particularly, Deno [5] is an approach that uses an epidemic voting protocol in order to support the data replication in a transactional framework for weakly-connected environments. Deno’s base protocol provides a weaker consistency model in which writes are not guaranteed to serialize with reads. However, Deno requires one voting round (stage) to be completely executed for each update. This may be acceptable for applications interested about the commitment process for each tentative update of a data item before executing another operation on this same data item. However, in scenarios where applications access data items, which have been updated by non-committed transactions, the wait for the commit operation, imposed by the Deno voting protocol, becomes unacceptably high as in the primary-commit protocol. In [9] the authors propose an approach, which is in fact an extension to the David Gifford’s [7] classic weighted-voting scheme for replicated data. The main limitation of the approach presented in [9] is low data availability, since the protocol doesn’t use the update anywhere semantics supported by optimistic concurrency mechanisms.

In our previous work [11] we proposed an update anywhere optimistic and centralized mechanism based on one-copy serializability and eventual consistency. However, for many database applications, serializability is too restrictive. This is the case of applications that access mobile databases. Indeed, the consistency degree may be traded for a potential gain in concurrency.

3. TRANSACTION ISOLATION LEVELS

The concept of isolation levels was first introduced in [8] under the name Degrees of Consistency (degrees 0, 1, 2 and 3). The goal of that seminal work was to increase concurrency degree by sacrificing the guarantee of perfect isolation, which in turn enforces data consistency. The idea proposed in [8] set the stage for the ANSI/ISO SQL-92 definitions for isolation levels [6], where the goal was to develop a standard that was implementation-independent. The ANSI/ISO SQL-92 approach was to avoid certain types of bad behavior, called phenomena, by applying the following premises: more restrictive consistency levels disallow more phenomena and serializability does not permit any phenomenon. The isolation levels were named READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ, and SERIALIZABLE; some of these levels were intended to correspond to the degrees of [8] (Table 1). However, these definitions are based on locking schemes and fail to meet the goals of ANSI-SQL w.r.t. implementation-independence. A subsequent paper [3] showed that the definitions provided in [6] were ambiguous and defined a more precise set of phenomena (P0, P1, P2 and P3), which should be avoided. Table 1 summarizes the isolation levels as defined in [3] and relates them to a lock-based implementation.

The approach presented in [1] provides precise and implementation-independent definitions for the ANSI SQL isolation levels [6]. These definitions are based on three kinds of read/write
for any two servers which is the state of the database in the replicated server scheme P is used U link U Thus S a multi T master approach O read T any V write T any available or reachable U communication channels are unstable and a server is frequently not replicated database servers and by weakly T connected portable

4. REPLICATING MOBILE DATABASES

The proposed data replication mechanism was designed for mobile computing environments composed by a set of replicated database servers and by weakly-connected portable devices. In such an environment, a client and a server can coexist in one host (fixed or mobile). The wireless communication channels are unstable and a server is frequently not available or reachable.

Applications can execute read and write operations on any server with which they can establish a communication link. Thus, a multi-master approach (read-any/write-any scheme) is used.

Copies of a given data item stored in different servers may have distinct states (values). In other words, \( DB(Re_1, t) \), which is the state of the database in the replicated server \( Re_1 \) at instant of time \( t \), is not necessarily equal to \( DB(Re_2, t) \) for any two servers \( Re_1 \) and \( Re_2 \). To reach an eventual con-

Table 1: Consistency Levels and Locking ANSI-92 Isolation Levels.

<table>
<thead>
<tr>
<th>Locking Isolation Level</th>
<th>Proscribed Phenomena</th>
<th>Read Locks on Data Items and Phantoms</th>
<th>Write Locks on Data Items and Phantoms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Degree 0</td>
<td>none</td>
<td>none</td>
<td>Short write locks</td>
</tr>
<tr>
<td>Degree 1 = Locking READ UNCOMMITTED</td>
<td>P0</td>
<td>none</td>
<td>Long write locks</td>
</tr>
<tr>
<td>Degree 2 = Locking READ COMMITTED</td>
<td>P0, P1</td>
<td>Short read locks</td>
<td>Long write locks</td>
</tr>
<tr>
<td>Locking REPEATABLE READ</td>
<td>P0, P1, P2</td>
<td>Long data-item read locks</td>
<td>Long write locks</td>
</tr>
<tr>
<td>Degree 3 = Locking SERIALIZABLE</td>
<td>P0, P1, P2, P3</td>
<td>Long read locks</td>
<td>Long write locks</td>
</tr>
</tbody>
</table>

conflicts (called direct conflicts): Directly Write-Depends, Directly Read-Depends and Directly Anti-Depends (Table 2). That work defines each isolation level in terms of phenomena that must be avoided at each level. However the phenomena are prefixed by “G” to denote the fact that they are general enough to allow locking and optimistic implementations; these phenomena are named G0, G1, G2 and G3 (by analogy with P0, P1, P2 and P3 [6]). Initially, Adya et al have proposed four isolation levels: PL-1, PL-2, PL-2.99 and PL-3 (Table 3), where PL-1 is more permissive than Degree 1, PL-2 is more permissive than Degree 2, PL-3 is more permissive than Degree 3 and PL-2.99 generalize “REPEATABLE READ”.

In order to identify the isolation level for a given schedule \( S \) Adya et al [1] introduces the concept of Direct Serialization Graph (DSG). In a DSG direct write-dependencies are denoted by \( T_i \rightarrow_{WR} T_j \), direct read-dependencies are denoted by \( T_i \rightarrow_{RW} T_j \), and direct anti-dependencies by \( T_i \rightarrow_{RW} T_j \). Table 2 summarizes this notation and reviews the definitions for direct dependencies. The DSG for a given schedule \( S \) denoted DSG(S) is built as follows. Each node in the graph corresponds to a committed transaction and directed edges correspond to different types of direct conflicts. There is a read/write/anti-dependency edge from transaction \( T_i \) to transaction \( T_j \) if \( T_i \) directly read/write/anti depends on \( T_j \). Then, a schedule \( S \) is PL-I if DSG(S) no contains a directed cycle consisting entirely of write-dependency edges. A schedule \( S \) is PL-2 if DSG(S) no contains a directed cycle consisting by write-dependency and/or read-dependency edges. A schedule \( S \) is PL-3 if DSG(S) no contains a directed cycle consisting by write-dependency, read-dependency and/or anti-dependency edges. Finally, a schedule \( S \) is PL-2.99 if DSG(S) no contains a directed cycle consisting by write-dependency, read-dependency and item-anti-dependency edges.

4. REPLICATING MOBILE DATABASES

It is a well-known fact that the use of locking mechanism is prohibitive in mobile databases. In order to stay away from locking mechanisms the proposed approach for ensuring data consistency in mobile replicated databases is based on a similar strategy used by the conventional serialization graph testing protocol (SGT) [4]: the dynamic monitoring and management of an acyclic conflict graph, called serialization graph (SG). In contrast to classic serialization graph testing, our approach, called temporal serialization graph testing (TSGT), exploits temporal information w.r.t. the moment when a mobile transaction operation (read or write) is executed on a given database. Furthermore, the TSGT uses the generalized isolation level definitions, proposed in [1], to build labeled edges used to represent different types of conflicts. Next, we present a formal definition for temporal serialization graph (TSG). In Section 5 (Figure 5(b)) we illustrate how a TSG is built.

**Definition 1.** Let \( S \) be a schedule over a set \( T = \{ T_1, T_2, \cdots , T_n \} \) of transactions, \( OP(T) \) the set of operations belonging to a transaction \( T \), and \( \prec \) a precedence relation between operations in a schedule \( S \). The Temporal Serialization Graph (TSG) for \( S \) is a directed graph \( TSG(S) = (N, E) \). The set \( N \) of nodes represents the transactions in \( T \), i.e., \( N = T \). The set \( E \) represents labeled edges of the form:

(i) \( T_i \rightarrow_{RW} T_j \), if \( T_i, T_j \in N \) and there are two operations \( p \in OP(T_i), q \in OP(T_j) \), where \( p \) conflicts with \( q, p < s q \) and \( p \) is a read operation and \( q \) is a write operation;

(ii) \( T_i \rightarrow_{RW} T_j \), if \( T_i, T_j \in N \) and there are two operations \( p \in OP(T_i), q \in OP(T_j) \), where \( p \) conflicts with \( q, p < s q \) and \( p \) is a predicate-based read operation and \( q \) is a write operation;

(iii) \( T_i \rightarrow_{RW} T_j \), if \( T_i, T_j \in N \) and there are two operations \( p \in OP(T_i), q \in OP(T_j) \), where \( p \) conflicts with \( q, p < s q \) and \( p \) is a write operation and \( q \) is a read operation;
Conflicts Name | Description \( (T_j \text{ conflicts with } T_i) \) | Notation in DSG
--- | --- | ---
Directly write-depends | \( T_i \) install \( x_z \) and \( T_j \) install \( x_z \)'s next version | \( T_i \xrightarrow{WW} T_j \)
Directly read-depends | \( T_i \) install \( x_z \) and \( T_j \) reads \( x_z \) or \( T_j \) performs a predicated-based read, \( x_z \) changes the matches of \( T_j \)'s read, and \( x_z \) is the same or an earlier version of \( x \) in \( T_i \)'s read | \( T_i \xrightarrow{WR} T_j \)
Directly anti-depends | \( T_i \) reads \( x_z \) and \( T_j \) install \( x_z \)'s next version or \( T_j \) performs a predicated-based read and \( T_j \) overwrites this read | \( T_i \xrightarrow{RW} T_j \)

Table 2: Definitions of direct conflicts between transactions.

| Level | Phenomena disallowed | Informal Description \( (T_j \) can commit only if:)
--- | --- | ---
PL-1 | \( G_0 \) | \( T_i \)'s writes are completely isolated from the writes of other transactions
PL-2 | \( G_1 \) | \( T_i \) has only read the updates of transactions that have committed by the time \( T_i \) commits (along with PL-1 guarantees)
PL-2.99 | \( G_1, G_2 \)-item | \( T_i \) is completely isolated from other transactions with respect to data items and has PL-2 guarantees for predicated-based reads
PL-3 | \( G_1, G_2 \) | \( T_i \) is completely isolated from other transactions; i.e., all operations of \( T_i \) are before or after all operations of any other transaction

Table 3: Summary of portable ANSI isolation levels.

(iv) \( T_i \xrightarrow{WW} T_j \), if \( T_i, T_j \in N \) and there are two operations \( p \in OP(T_i), q \in OP(T_j) \), where \( p \) conflicts with \( q \), \( p < q \) and \( p, q \) is a write operations.

In the proposed replication protocol, each data item \( x \) in \( DB_{Bj} \) is associated with a timestamp \( C(x) \), where \( 0 \leq i \leq n \) and \( n \) is the number of replicated servers. Moreover, each operation \( p \in r, w, p \in OP(T_i) \), executed over a database object \( x \in DB_{Bj} \), has a timestamp \( C(p(x)) \), which corresponds to the current timestamp of \( x \), i.e., \( C(p(x)) = C(x) \). In Definition 2, we describe how timestamps are generated and managed by the proposed protocol.

**Definition 2.** A timestamp is defined according to the following rules:

- A timestamp consists of an ordered pair \((z, y)\). The first part \((z)\) is called version, while the second part \((y)\) is called subversion;
- Initially \( C(z) = (0, 0) \) \( \forall x \in DB_{(primary\:\:database)} \), in all copy \( DB_{Bj} \), where \( 0 \leq j \leq n \);
- The version component of \( C(x) \) (i.e., \( z \)) is incremented for each commit of any transaction \( T_i \) which has executed a write operation on \( x \) (new version of \( x \)), \( x \in DB_{Bj} \), \( 0 \leq j \leq n \);
- The sub-version component of \( C(x) \) (i.e., \( y \)) is set to \( 0 \), i.e., \( C(x) = (v, 0) \), with \( v > 0 \), for each commit of any transaction \( T_i \) which has executed a write operation on \( x \), \( x \in DB_{Bj} \), \( 0 \leq j \leq n \);
- The sub-version component of \( C(x) \) (i.e., \( y \)) is incremented for each write operation of any active transaction \( T_i \) on \( x \), \( x \in DB_{Bj} \), \( 0 \leq j \leq n \). Note that \( T_i \) may be an uncommitted transaction.

**Definition 3.** To compare two timestamps \( C(q_i(x)) \) and \( C(p_i(x)) \), \( C(q_i(x)) = (a, b) \) and \( C(p_i(x)) = (c, d) \), the following rules should be applied:

(i) If \( a \) \(< \) \( c \) then \( C(q_i(x)) \) \(< \) \( C(p_i(x)) \);
(ii) If \( a \) \(= \) \( c \), the subversion value is verified. Thus, if \( b \) \(>\) \( d \) then \( C(q_i(x)) \) \(>\) \( C(p_i(x)) \). By the same rule, if \( b \) \(<\) \( d \) then \( C(q_i(x)) \) \(<\) \( C(p_i(x)) \).

4.2 Replica Control Protocol

In this section, we describe how the proposed replica control protocol guarantees data and eventual consistencies according to a transaction isolation level. The supported transaction isolation levels are those shown in Table 3 (and discussed in Section 3). In our approach, the replica control protocol execution is distributed among mobile clients, replicated servers and the primary server. It is important to note that the primary server behaves as \( \text{data} \) \( \text{eventual} \) consistency service provider.

Mobile transactions are submitted (by users or applications) to mobile clients. In turn, mobile clients should forward transaction operations to one mobile replicated server. Mobile clients execute the following function of the replica control protocol. Whenever a client \( CLI_i \) receives a commit or abort operation for a mobile transaction \( T_i \), it should submit this operation to any replicated server \( DB_{Bj} \). In the case of a commit operation \( CLI_i \) should send additionally the number of operations executed by \( T_i \). This information is used by the replica control protocol’s running on the primary server (see Step 5 of the algorithm described below). The replicated server \( DB_{Bj} \) should forward the commit/abort request to the primary server. The client has to wait for a reply (from the primary server) to execute a commit or abort operation according to the decision of the primary server. If the timeout runs out while \( CLI_i \) is waiting for a decision, one of the following actions may be executed by \( CLI_i \): abort \( T_i \) or define \( T_i \) as a tentative transaction. A transaction can be defined as tentative when it has already executed all of its database operations (read and write), but
it can not commit. Additionally, if the transaction \( T_i \) is a read-only transaction, \( C_{li} \) can unilaterally decide to conclude \( T_i \)'s execution, without waiting for the primary server decision.

Besides executing transaction operations on local database, replicated servers execute the following function of the proposed replica control protocol. Periodically, each replicated database server should send to the primary server the timestamps of the read and write operations executed in its local copy.

On the primary server, the replica control protocol executes the following algorithm:

**Step 1.** When the replica control protocol starts running (on the primary server), the temporal serialization graph (see Definition 1) is created as an empty graph.

**Step 2.** As soon as the protocol receives the first operation of a new transaction \( T_i \), a new node representing that transaction is inserted in \( TSG \).

**Step 3.** For each arriving operation \( p_i(x) \in O(P(T_i)) \), it is checked if there is a conflicting operation \( q_j(x) \in O(P(T_j)) \) which has been already scheduled. In case \( q_j(x) \) exists, an edge will be inserted between the nodes \( T_i \) and \( T_j \) according to the algorithm presented in Figure 1.

**Step 4.** After the insertion of a new edge in \( TSG \), the protocol verifies if the new edge introduces a cycle, according to the following rules. If the transaction isolation level is READ UNCOMMITTED (PL-1) the protocol verifies if there is a cycle involving only edges with “WW” label. If the isolation level is READ COMMITTED (PL-2) the mechanism verifies if there is a cycle involving edges with the “WW” and/or “WR” labels. If the isolation level is REPEATABLE READ (PL-2.99), the protocol verifies if there is a cycle involving edges with the “WW,” “WR,” “RW-item” and/or “RW-label” items. If the isolation level is SERIALIZABLE (PL-3), the protocol verifies if there is a cycle involving edges with the “WW,” “WR,” “RW-item” and/or “RW-label” items. When a cycle is introduced, the (arriving) operation \( p_i(x) \) is rejected, \( T_i \) is aborted. The protocol sends a message to the client which has submitted \( T_i \) informing about the abort of \( T_i \). If the new edge does not introduce a cycle, \( p_i(x) \) is accepted and scheduled.

**Step 5.** When the protocol receives a commit operation for a transaction \( T_i \) (together with the number of operations of \( T_i \)), it verifies if all operations of \( T_i \) have already been received and scheduled and if \( T_i \) is still an active transaction.

In this case the commit operation will be executed. After the commit of \( T_i \), the version value of the timestamp of all data items updated by \( T_i \) is incremented, as follows: \( v_z \), where \( z \) was updated by \( T_i \), do \( C(z) = (z + 1, 0) \). Thereafter, the new values of data items updated by \( T_i \) with the new timestamps should be propagated to all replicated servers.

**Step 6.** When the protocol receives an abort request of a transaction \( T_i \), it undoes the effect of the \( T_i \) operations, removes the edges associated with this transaction and sends the abort confirmation to client \( C_i \).

As already mentioned, the use of isolation levels can relax ISR correctness criterion, for example by using PL-2 (see table 3). Besides, one can easily see that the replica control protocol described above does not use any type of lock to implement the different isolation levels. We claim that such a feature can provide even more data availability and transaction throughput rate if we had just used the notion of isolation levels.

\[ \text{If } C(q_j(x)) < C(p_i(x)) \]
\[ \text{If } q_j(x) \text{ and } p_i(x) \text{ are write operations} \]
\[ \text{Then the mechanism inserts an edge } T_j \xrightarrow{W} T_i \]
\[ \text{If } q_j(x) \text{ is a read operation and } p_i(x) \text{ is a read operation} \]
\[ \text{Then the mechanism inserts an edge } T_j \xrightarrow{WR} T_i \]
\[ \text{If } q_j(x) \text{ is a read operation and } p_i(x) \text{ is a write operation} \]
\[ \text{Then the protocol verifies if there is a cycle involving edges with the “WW” label} \]
\[ \text{If } q_j(x) \text{ is a write operation and } p_i(x) \text{ is a read operation} \]
\[ \text{Then the protocol verifies if the new edge introduces a cycle} \]
\[ \text{An edge } T_j \xrightarrow{RW} T_i \text{ will be inserted on the graph} \]
\[ \text{If } q_j(x) \text{ is a write operation and } p_i(x) \text{ is a write operation} \]
\[ \text{Then the mechanism inserts an edge } T_j \xrightarrow{RW-item} T_i \]
\[ \text{Else} \]
\[ \text{Then the mechanism inserts an edge } T_j \xrightarrow{RW} T_i \]
\[ \text{Else} \]
\[ \text{If } C(q_j(x)) > C(p_i(x)) \]
\[ \text{If } q_j(x) \text{ and } p_i(x) \text{ are write operations} \]
\[ \text{Then the mechanism inserts an edge } T_j \xrightarrow{WW} T_i \]
\[ \text{If } q_j(x) \text{ is a read operation and } p_i(x) \text{ is a read operation} \]
\[ \text{Then the mechanism inserts an edge } T_j \xrightarrow{WR} T_i \]
\[ \text{If } q_j(x) \text{ is a read operation and } p_i(x) \text{ is a write operation} \]
\[ \text{Then the protocol verifies if the new edge introduces a cycle} \]
\[ \text{An edge } T_j \xrightarrow{RW} T_i \text{ will be inserted on the graph} \]
\[ \text{If } q_j(x) \text{ is a predicate-read operation} \]
\[ \text{An edge } T_j \xrightarrow{RW} T_i \text{ will be inserted on the graph} \]
\[ \text{If } q_j(x) \text{ is a predicate-read operation} \]
\[ \text{An edge } T_j \xrightarrow{RW-item} T_i \text{ will be inserted on the graph} \]
\[ \text{If } p_i(x) \text{ and } q_j(x) \text{ are executed in different copies} \]
\[ \text{If } p_i(x) \text{ is a write operation and } q_j(x) \text{ is a write operation} \]
\[ \text{Then the mechanism inserts an edge } T_j \xrightarrow{RW} T_i \]
\[ \text{on the graph} \]

Figure 1: Step 3 of the Replica Control Protocol.

4.3 Protocol Correctness

Next, we prove that the proposed replica control protocol ensures: (i) one-copy serializability (1SR), if SERIALIZABLE (PL-3) is the transaction isolation level (the most restrictive isolation level), and (ii) eventual consistency. Recall that ISR guarantees that the concurrent execution of a set \( T \) of transactions on several replicated database is equivalent to a serial execution of the same set on the same database without replication. Eventual consistency ensures that all the copies of replicated databases, eventually, will converge to one same consistent state, independently of the isolation level.

First, we prove that the proposed replica control protocol builds temporal serialization graphs (\( TSG \)) without any type of cycle (RW, WR, WW and RW-item). By proving this, we prove that the execution of a set of transactions over several replicated databases is equivalent to a serial execution over the primary database (one-copy database).

**Theorem 1.** Let \( TSG_{RT} \) be the set of all schedules on
the set \( T = \{ T_1, T_2, \ldots, T_k \} \) of transactions produced by the proposed protocol and CSR the set of all the conflict serializable schedules \( \{ q \} \) on \( T \). If the transaction isolation level is SERIALIZABLE (PL-3), then \( \text{TSG}_T = \text{CSR} \).

**Sketch of Proof.** It is easy to show that \( \text{TSG}_T \subseteq \text{CSR} \). We only need to observe that every schedule \( S \) produced by the proposed protocol, when the transaction isolation level is SERIALIZABLE, has an acyclic temporal serialization graph (TSG), regardless of the type of cycle (RW, WR, WW and RW-item).

By definition, the conventional serialization graph for \( S \) (denoted \( \text{SG}(S) \)) is a subgraph of \( \text{TSG}(S) \), since (i) \( \text{TSG}(S) \) has the same set of nodes \( N \) (transactions in \( S \)) and; (ii) \( \text{TSG}(S) \) has more edges than \( \text{SG}(S) \) (the labeled edges). Note that the edges are constructed in the same way (by using conflicting operations) in both graphs (see Definition 1 and [4]). Thus, if the \( \text{TSG}(S) \) is acyclic, the conventional serialization graph, \( \text{SG}(S) \), also is.

A schedule \( S \in \text{CSR} \) iff \( \text{SG}(S) \) is acyclic. Therefore, for all schedule \( S \), which has an acyclic TSG, then \( S \in \text{CSR} \), consequently, \( \text{TSG}_T \subseteq \text{CSR} \). To prove that \( \text{TSG}_T \cap \text{CSR} \), we need to show that every \( S \in \text{CSR} \) schedule can be produced by the proposed replica control protocol. This can be shown using induction in the size of \( S \), having that any operation \( p \in S \) cannot generate a cycle \( \text{TSG}(S) \) of any type, if the transaction isolation level is SERIALIZABLE.

Now, we prove that all the copies eventually will converge to one same consistent state independently of the selected isolation level.

**THEOREM 2.** Let \( R = \bigcup_{i=1}^{n} R_{ei} \) be the set of the replicated servers, and \( \text{DB}(Re_i, t) \) the database state in the replicated server \( R_{ei} \), at an instant \( t \). Then, there is a point in future time \( t + k, k > 0 \), where \( \forall i, j \) (with \( i \neq j \), \( 0 < i, j \leq n \) \( \text{DB}(Re_i, t) = \text{DB}(Re_j, t) \) and \( \text{DB}(Re_i, t) \) is consistent.

**Sketch of Proof.** According to step 5 of the protocol, after the commit of a transaction \( T_c \), the protocol propagates the new value of data items written (updated) by \( T_c \), to all replicated servers where the primary server has communicated. There will be an instant \( t' = t + k, k > 0 \), where no write operation is being carried out. At that point in time, \( \text{DB}(R_c, t') \) is consistent and stable, where \( R \) is the primary server, and all replicated servers \( R_{ei} \) will have a (communication) connection, even temporary, with the primary server \( R \). Thus all replicated servers \( R_{ei} \) receive pending write operations and may update their local database states, making \( \text{DB}(R_{ei}, t) = \text{DB}(R, t) \). Moreover, according to Theorem 1, \( \text{DB}(R_{ei}, t) \) is consistent.

**5. RUNNING EXAMPLE**

To illustrate the applicability and use of our proposal, consider an e-commerce application where several products are on sale in an electronic auction.

Now, consider the following set of transactions, which read/write values of the products for sale and a schedule \( S_1 \) (Figure 3), defined over the set \( T = \{ T_1, T_2, T_3 \} \):

\[
T_1 : r_1(CEL) r_1(PALM) w_5(CEL, CEL+5) C_1; \\
T_2 : r_2(CEL) w_3(CEL, CEL+7) C_2; \\
T_3 : r_3(PALM) r_3(CEL) w_4(CEL, CEL+10) C_3;
\]

We assume that the operations had been temporally executed in the order depicted in figure 3 and that the isolation level was PL-3 (SERIALIZABLE). The notation \( r_i(CEL) \) represents a read operation of transaction \( T_i \) on the data item \( CEL \) stored in the replicated server \( R_i \). The traditional serialization graph for schedule \( S_1 (\text{SG}(S_1)) \) is depicted in figure 5 (a). \( \text{SG}(S_1) \) doesn’t have a cycle, thus the schedule \( S_1 \) would be considered conflict serializable (correct). However, \( S_1 \) produces a final state in which 6 is the value for the data item \( CEL \) in copy \( DB_{R_1} \), while in the copy \( DB_{R_2} \) and in the primary copy, \( CEL \) has a value of 16. Those final states for \( DB_{R_1} \) (CEL=6), \( DB_{PC} \), \( DB_{R_2} \) (CEL=16) could not be generated by any serial execution of transactions \( T_1, T_2 \) and \( T_3 \) over \( DB_{PC} \).

The aforementioned phenomenon stems from the fact that the value read by the operation \( r_1(CEL) \) does not take into account the value written by \( w_2(CEL, CEL+7) \). In fact the value read by the operation \( r_1(CEL) \) is previous to the value generated by the operation \( w_2(CEL, CEL+7) \). In other words, \( r_3(CEL) \) should precede \( w_2(CEL, CEL+7) \) in \( S_1 \). The temporally correct schedule \( S_1' \) is shown in figure 4, while the correct temporal serialization graph for the schedule \( S_1' \) is presented in Figure 5 (b). Of course, such a phenomenon should be avoided because it may generate inconsistent database states. For the proposed replica control protocol, the temporal serialization graph (TSG) contains a cycle and the transaction \( T_2 \) aborted, according to Step 4. In this case the final value of the item \( CEL \) in \( DB_{R_2} \) would be 6, while in the copy \( DB_{PC} \) and in the primary copy this item would have a value of 13. The value 13 would then be propagated by the primary copy to the other copies (Step 5), which is in fact a consistent value, since it would be generated by any serial execution of the transactions \( T_1 \) and \( T_2 \). Recall that \( T_3 \) is aborted by the replica control protocol in order to produce a consistent final state in all replicas and in the primary copy (database).

It’s important to note that if the messages with information on reads and writes operations executed in the several replicated servers (copies) arrive at the primary server in a different order from the order in which the operations had actually been executed, the replica control protocol continues generating correct temporal serialization graph and identifying the cycle, which means correct schedules.

**Figure 2: Replicated Environment.**

**Figure 3: Schedule \( S_1 \).**
Now, consider that the specified isolation level was PL-1 (READ UNCOMMITTED). The correct TGS($S_i$) (figure 4) is similar to the graph presented in figure 5 (b). However, in this case, there isn’t a cycle involving only “WW” labeled edges and the schedule $S_i$ is correct schedule regarding the PL-1 isolation level. It is important to note that the final value for “CEL” is 16. That is because $T_3$’s operations over ride $T_2$’s operations, that is because the phenomena “dirty read” and “lost update” (see Table 3). Nevertheless, it’s not a problem, since those phenomena are typical of the PL-1 isolation level.

6. EVALUATION

6.1 Experimental Results

In order to evaluate the quality of the proposed replica control protocol we considered performance with respect to the following metrics: (i) Transaction Response Time, which indicates the time interval, measured in seconds, between the moment a transaction $T$ is submitted and the moment that $T$ commits (including the time involved in restarts); (ii) Abort rate, which measures the number of aborts occurred for a set of concurrent transactions, and; (iii) Messages Overhead for indicating the number of messages exchanged between the master server and replicated servers.

The simulation environment consisted of a master server, three mobile replicated servers and a mobile client. Those hosts were connected by means of a 802.11g wireless network. The client host was responsible for submitting 200 mobile transactions to be executed in more than one replicated servers. In turn, each replicated server has submitted 40 server transactions which had to be executed locally. Server transactions had a fixed length of 6 operations. On the other hand, client transactions could have variable lengths (2, 4, 6, 8 and 10 operations). Clients and servers executes read-write transactions. A small database (300 objects) helped to intensify data conflicts by creating a hot-spot effect. The object access was determined using a random distribution function.

Figures 6, 8, 7 show the results of our simulation experiments. The four different curves in each Figure represents the four isolation levels, thus curve 1 represents PL-1, 2 represents PL-2, curve 3 corresponds to PL-2.99 and curve 4 represents PL-3 (the most restrictive one). Figure 6 shows that our protocol presents a linear behavior for the abort rate with respect to the length of mobile transactions. A low abort rate means that throughput and data availability are high, important properties for transaction management in mobile computing environments. The linear behavior holds for the four transaction isolation levels. Such a feature shows that the proposed replication mechanism is scalable w.r.t. the length of transactions. Looking more closely to Figure 6, one can observe that that the more permissive the transaction isolation level, the lower the transaction abort rate.

Figure 7 shows that transactions are executed in short time intervals. It shows also that with short transactions (e.g., with 4 operations) the response time is almost the same for the four possible transaction isolation levels. Figure 8 shows that message exchange overhead tends to be linear w.r.t transaction length. The number of exchanged messages is almost the same, regardless the isolation level is being used. This is another demonstrations that our approach is scalable.

Finally, it is important to emphasize that the proposed replica control protocol is quite efficient to be used in mobile computing environments. This is because, it guarantees low abort rates, linear message overhead and small response time. For that reason, client transactions can commit more quickly, which increases the system throughput.

6.2 DISCUSSION

The mobile database replication mechanism presented in this paper can be classified as optimistic (read-any/write-only model) and centralized (uses a master copy to maximize the number of successful commit operations). The proposed protocol allows the use of transaction isolation levels to adapt transactions to the scenario they are being exe-
The proposed protocol does not deploy the idea of data locking to implement the different isolation levels. Finally, our approach reduces the number of messages exchanged among servers, which minimizes in turn communication channels cost. The results obtained in the simulations demonstrate the efficiency of the proposed protocol.

We are currently working on extending the proposed protocol in order to make updates propagation in a peer-to-peer (P2P) fashion.

8. REFERENCES


