A Distributed Protocol for Ensuring Replicated Database Consistency in Mobile Computing Environments

Alex Costa  
University of Fortaleza  
Fortaleza, Brazil  
alexcosta@edu.unifor.br

Jose Maria Monteiro  
University of Fortaleza  
Fortaleza, Brazil  
monteiro@unifor.br

Angelo Brayner  
University of Fortaleza  
Fortaleza, Brazil  
brayner@unifor.br

ABSTRACT

A mobile replicated database is comprised of several mobile and fixed servers and clients interconnected through a wireless network. In order to ensure data consistency in mobile replicated databases, several replication control protocols have been proposed. However, most of them reveal relevant limitations, such as: possessing a single failure point and flooding the network with messages exchange. In this paper we introduce a protocol which guarantees data consistency of replicated databases in mobile computing. The proposed approach is completely distributed, avoids the existence of a single failure point, uses a read-any/write-any replication scheme, increases data availability, and reduces the number of messages exchanged among the replicated servers. Furthermore, it allows the user to choose a transaction isolation level (degree in which the execution of a given transaction is isolated from all other concurrent transactions). Experimental results show the potential efficiency of the proposed approach.

Keywords: Concurrency Control, Data Replication, Mobile Computing

1. INTRODUCTION

A mobile computing environment is characterized by having several devices distributed dynamically, with no fixed location, connected by means of a wireless network, where, in general, mobile nodes may have intermittent connectivity to the wireless network. Due to potential gains in increasing data availability and transaction throughput, data replication has been applied to mobile computing environments.

The basic idea behind database replication is to store multiple data copies (called replicas) in different servers (called replicated servers) distributed on the network. Hence, if one server becomes unavailable, an application can access data on any other active server. However, when using a data replication scheme, two properties must be ensured in order to preserve the consistency of the replicated data: eventual consistency and one-copy serializability (for short, 1SR). As described in [3], the property of eventual consistency ensures that all copies of a given item of data converge to the same final state. The property of one-copy serializability (1SR) ensures that the concurrent execution of a set of transactions over replicas of a mobile replicated database should be equivalent to a serial execution of the set over the same database without replication (one-copy database).

In replicated databases data consistency is guaranteed by replica control protocols. Conventional replica control protocols are inefficient to be applied to mobile replicated databases. This is because 1SR correctness criterion is too restrictive for mobile replicated databases [8]. For that reason, several approaches have been proposed in which replica control protocols use correctness criteria less restrictive than 1SR. Nevertheless, those approaches present several failure points. Besides, most of them require intensive message exchange and the re-execution of log-stored transaction operations in order to ensure data consistency.

This paper introduces a new protocol which ensures replicated database consistency in mobile computing environments. The proposed protocol is fully distributed thus avoiding the existence of a single failure point. Additionally, it implements a read-any/write-any replication scheme. Finally, the proposed protocol has the property of reducing the number of messages flowing in the network by selecting different levels of isolation. In order to prove the efficiency of the proposed mechanism, performance tests were undertaken.

The rest of the paper is organized as follows. The next Section addresses related work. The proposed replica control protocol is presented in Section 3. To illustrate the use of the proposed replication mechanism a running example is shown in Section 4. Section 5 presents experimental results and a discussion highlighting the advantages of the proposed replicating approach against related work. Section 6 concludes this paper.

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 [10] allows data to be updated by disconnected clients and conflicts are resolved in a pair-wise fashion during reconnection. Updates are propagated epidemiically 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 uses a weak consistency notion based...
on session guarantees, which does 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.

Golding and Long [6] proposed an approach in which each server is responsible for consolidating updates. This activity can only be 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 [5] 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 [4] 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. In scenarios where applications often access data items, which have been updated by non-committed transactions, the waiting time for executing commit operation, imposed by the Deno voting protocol, becomes unacceptably high as in the primary-commit protocol.

In our previous work [9] 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.

Finally, [8] proposes a protocol which ensures mobile replicated database consistency according to ANSI-SQL92 transaction isolation levels. Thus users can choose between database consistency degree or transaction throughput rate. Observe that by using 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. Nonetheless, that proposal has the critical drawback of having a single failure point, located in the master database, which runs a coordinator for the protocol.

### 3. 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 consistency in which the servers converge to an identical state of all replicated objects, a P2P (peer to peer) protocol is used to synchronize updates. Thus, a transaction $T$ can write a new version for the data item $x$ in a server $DB_{R_1}$, and, after this, sends a commit request to a server $DB_{R_2}$, which will decide for commit or abort of $T$. Then, in case of commit, the replicated server $DB_{R_2}$ has the functionality of propagating to all other servers the new version of $x$.

Finally, it worthwhile to mention that the proposed replication mechanism avoids the problem of disconnected operation. In that case, a server $S$ can be disconnected from the rest of the system but still be used by the clients (local to the server or remote but which can communicate with $S$). When $S$ re-establishes its connection to the other replicated databases, it executes a synchronization procedure. That is an important feature, since the replicated database servers should with intermittent network connectivity.

### 3.1 Background

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 dynamic monitoring and management of an acyclic conflict graph [3]. 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).

**Definition 1.** Let $S$ be a schedule over a set $T = \{T_1, T_2, \ldots, T_n\}$ of transactions, $OP(T_i)$ the set of operations belonging to a transaction $T_i$, and $\prec_S$ 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) $T_i \xrightarrow{RW\text{-item}} T_j$, iff $T_i, T_j \in N$ and there are two operations $p \in OP(T_i), q \in OP(T_j)$, where $p$ conflicts with $q$, $p \prec_S q$ and $p$ is a read operation and $q$ is a write operation;

(ii) $T_i \xrightarrow{WR} T_j$, iff $T_i, T_j \in N$ and there are two operations $p \in OP(T_i), q \in OP(T_j)$, where $p$ conflicts with $q$, $p \prec_S q$ and $p$ is a predicate-based read operation and $q$ is a write operation;

(iii) $T_i \xrightarrow{RW} T_j$, iff $T_i, T_j \in N$ and there are two operations $p \in OP(T_i), q \in OP(T_j)$, where $p$ conflicts with $q$, $p \prec_S q$ and $p$ is a write operation and $q$ is a read operation;

(iv) $T_i \xrightarrow{WR} T_j$, iff $T_i, T_j \in N$ and there are two operations $p \in OP(T_i), q \in OP(T_j)$, where $p$ conflicts with $q$, $p \prec_S q$ and $p$ and $q$ are write operations.

In our protocol, each data item $x$ in $DB_{R_i}$ is associated with a timestamp $C(x)$, where $0 < t < n$ and $n$ is the number of replicated servers. Moreover each operation $p \in \{r, w\}$, $p \in OP(T)$, executed over a database object $x \in DB_{R_i}$ 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 \(C(x) = (0, 0) \forall x \in DB\), in all copy \(DB_{R_i}\), where \(0 < j \leq n\);
- 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 \(x\)), \(x \in DB_{R_i}, 0 < j \leq n\);
- 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 a transaction \(T_i\) which has executed a write operation on \(x\), \(x \in DB_{R_i}, 0 < j \leq n\);
- The sub-version component of \(C(x)\) (i.e., \(y\)) is incremented for each write operation of any active transaction \(T_i\) on \(x\), \(x \in DB_{R_i}, 0 < j \leq n\). Note that \(T_i\) may be an uncommitted transaction.

Definition 3 formalizes the rules to compare timestamps in our protocol.

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

(i) If \(a > c\), then \(C(q_i(x)) > C(p_i(x))\). By the same rule, if \(a < c\) then \(C(q_i(x)) < C(p_i(x))\);

(ii) If \(a = c\), the subversion value is verified. Thus,

(a) If \(b > d\), then \(C(q_i(x)) > C(p_i(x))\). By the same rule, if \(b < d\) then \(C(q_i(x)) < C(p_i(x))\).

(b) If \(b = d\), then \(C(q_i(x)) = C(p_i(x))\).

3.2 Distributed 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 1. In our approach, the replica control protocol execution is distributed among mobile clients and replicated servers. It is important to note that the (data/eventual) consistency service is provided in a P2P manner.

Mobile transactions are submitted (by users or applications) to mobile clients. In turn, mobile clients should forward transaction operations to one mobile replicated server \(DB_{R_i}\). Then, \(DB_{R_i}\) executes and stores the received operations in a log. 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 \(DB_{R_i}\). 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 running on the replicated server \(DB_{R_i}\). The client has to wait for a reply (from the replicated server \(DB_{R_i}\)) to execute a commit or abort operation according to the decision of the \(DB_{R_i}\). 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, \(Cli_i\) can unilaterally decide to conclude \(T_i\)’s execution, without waiting for the replicated server decision.

On each replicated server \(DB_{R_i}\), the replica control protocol executes the following algorithm:

Step 1. When the replica control protocol starts running, 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 read operation \(r_i(x) \in OP(T_i)\), \(C(r_i(x)) = C(x)\).

Step 4. For each arriving write operation \(w_i(x) \in OP(T_i)\), the sub-version component of \(C(x)\), in \(DB_{R_i}\), (i.e., \(y\)) is incremented and \(C(w_i(x)) = C(x)\).

Step 5. When \(DB_{R_i}\) 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. In this case the protocol executes the protocol presented in Figure 1. In order to check the existence of any cycle in the TSG, the depicted in Figure 1 applies 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” and/or “RW-item” labels. 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” labels. When there is a cycle, \(T_i\) is aborted. The protocol sends a message to the client which has submitted \(T_i\) informing about the abort of \(T_i\). If there is not a cycle, the commit operation will be executed.

After a transaction \(T_i\) is committed, the version component of the timestamp (see Definition 2) of all data items updated by \(T_i\) is incremented, as follows: \(\forall x\), where \(x\) was updated by \(T_i\), do \(C(x) = (z + 1, 0)\). 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 undo the effect of the \(T_i\) operations, removes the edges associated with this transaction, sends the abort confirmation to client \(C_i\), and informs the abort of \(T_i\) to others replicated servers, in order to these servers update their serialization graphs and the timestamps of the data items updated by \(T_i\), in their local copies.

As already mentioned, the use of isolation levels can relax 1SR correctness criterion, for example by using PL-2 (see table 1). 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.

3.3 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 1SR 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 $T$ 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 protocol builds local temporal serialization graphs (TSG) without any type of cycle (RW, WR, WW and RW-item) for each replicated server $DB_{R_i}$. Thereafter, we show that the global graph which represents the union of all local serialization graphs (TSG) is acyclic as well. The global graph is equivalent to a temporal serialization graph generated by the protocol on the same set of transactions executed on the same database without replication. Thus, 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).

### Table 1: Summary of portable ANSI isolation levels.

<table>
<thead>
<tr>
<th>Level</th>
<th>Phenomena</th>
<th>Informal Description (T_i can commit only if:</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL-1</td>
<td>G0</td>
<td>$T_i$’s writes are completely isolated from the writes of other transactions</td>
</tr>
<tr>
<td>PL-2</td>
<td>G1</td>
<td>$T_i$ has only read the updates of transactions that have committed by the time $T_i$ commits (along with PL-1 guarantees)</td>
</tr>
<tr>
<td>PL-2.99</td>
<td>G1, G2-item</td>
<td>$T_i$ is completely isolated from other transactions with respect to data items and has PL-2 guarantees for predicated-based reads</td>
</tr>
<tr>
<td>PL-3</td>
<td>G1, G2</td>
<td>$T_i$ is completely isolated from other transactions, i.e., all operations of $T_i$ are before or after all operations of any other transaction</td>
</tr>
</tbody>
</table>

### Figure 1: Step 5 of the Replica Control Protocol.

```plaintext
For each operation $p_i(x) \in T_i$ do:
   GraphUpdate$(p_i(x))$, in $DB_{R_i}
End For
For each active replicated server $DB_{R_j}$, where $1 \leq j \leq n, j \neq i$ and $n$ represents the number of replicated servers do:
   For each data item $x$, read or written by $T_i$, do
      $DB_{R_j}$ requests for $DB_{R_i}$ the executed operations on $x$ together with the operations timestamps
   End For
   For each received operation $p_j(x) \in T_j$ do:
      GraphUpdate$(p_j(x))$, in $DB_{R_i}$
   End For
End For
Verifies if the graph has a cycle
If the graph has a cycle
   Abort $T_i$
   Informs the abort to client $C_i$
   Update the timestamps of the data items updated by $T_i$
   Informs the abort of $T_i$ to others replicated servers
Else
   Commits $T_i$
   Informs the commit to client $C_i$
   Update the timestamps of the data items updated by $T_i$
   Informs the commit of $T_i$ to others replicated servers
End if
```

### Function GraphUpdate$(C(p_i(x)))$

```plaintext
Begin
   If $C(q_j(x)) < C(p_i(x))$
      If $q_j(x)$ and $p_i(x)$ are write operations
         Then the mechanism inserts an edge $T_i \xrightarrow{RW} T_j$
      End if
      Else
         If $C(q_j(x)) > C(p_i(x))$
            If $q_j(x)$ and $p_i(x)$ are write operations
               Then the mechanism inserts an edge $T_i \xrightarrow{WW} T_j$
            Else
               If $q_j(x)$ is a write operation and $p_i(x)$ is a read operation
                  Then the mechanism inserts an edge $T_i \xrightarrow{WR} T_j$
               End if
               Else
                  If $q_j(x)$ is a read operation and $p_i(x)$ is a write operation
                     Then the mechanism inserts an edge $T_i \xrightarrow{RW} T_j$
                  End if
                  Else
                     If $q_j(x)$ is a predicate-read operation
                        Then the mechanism inserts an edge $T_i \xrightarrow{RW-item} T_j$
                     End if
                  End if
               End if
            End if
         End if
      End if
   End if
End if
```

### Figure 2: Function GraphUpdate.

**Theorem 1.** Let $TSG_{R_i}$ be a local temporal serialization graph (see Definition 1) produced by the proposed protocol in a replicated server $R_i$ ($i > 0$) on the set $T_{R_i}$ of transactions, $GTSG$ the global serialization graph, where
\[ GTSG = \bigcup_{i=1}^{n} TSG_{R_i}, \quad TSG_{R_i} \text{ the set of all the schedules } T_{R_i} \text{ yielded by proposed protocol and CSR the set of all the conflict serializable schedules } [3] \text{ on } T_{R_i}. \] If the transaction isolation level is SERIALIZABLE (PL-3), then \( TSG_{R_i} \subseteq CSR \), 1 \( \leq i \leq n \), and GTSG has no cycle.

**Sketch of Proof.** First, we have to prove that every local temporal serialization graph (TSG) produced by the proposed replica control protocol has no cycle. It is easy to show that \( TSG_{R_i} \subseteq CSR \). We only need to observe that every schedule \( S_{R_i} \) produced by the proposed protocol in a replicated server \( R_i \) (\( i > 0 \)), when the transaction isolation level is SERIALIZABLE, has an acyclic temporal serialization graph (TSG\(_{R_i}\)), regardless the type of cycle (RW, WR, WW and RW-item). Thus, we can execute a topological sort on TSG\(_{R_i}\) to build a serial schedule which is conflict equivalent to \( S_{R_i} \).

Thus, the TSG\(_{R_i}(S)\) is acyclic. If a schedule \( S \) has an acyclic TSG\(_{R_i}(S)\), then \( S \in CSR \). To prove that TSG\(_{R_i} \subseteq CSR \), we need to show that every schedule \( S \in CSR \) can be produced by the proposed replica control protocol. This can be shown by induction in the size of \( S \), having that any operation \( p \) in \( S \) cannot generate a cycle TSG\(_{R_i}\) of any type, if the transaction isolation level is SERIALIZABLE. Second, we have to prove that GTSG has no cycle. Since GTSG is the union of all TSG\(_{R_i}\), and we have proved that there is no cycle in each TSG\(_{R_i}\), there is no cycle in GTSG. Therefore GTSG has no cycle.

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

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

**Sketch of Proof.** According to step 5 of the protocol, after the commit of a transaction \( T_i \), the protocol propagates the new value of data items written (updated) by \( T_i \) to all active replicated servers. There will be a point in future time \( t + k, k > 0 \), where no write operation is being carried out. At that point in time, all replicated servers Re\( _k \) have a (communication) connection, even temporary. Thus all replicated servers Re\( _k \) receive pending write operations and may update their local database states, making DB(Re, t) = DB(Rk, t) \( \forall 0 < k, v \leq n \), where \( n \) is the number of replicated servers and \( k \neq v \).

4. **RUNNING EXAMPLE**

To illustrate the applicability and use of our proposal, consider a scenario of an electronic commerce application where several products are on sale in an electronic auction and data are replicated in two hosts \( R_1 \) and \( R_2 \) (see Figure 3). Consider that the following set \( T = \{ T_1, T_2, T_3 \} \) of transactions are executed on database objects belonging to the replicated databases DB\(_{R_1}\) and DB\(_{R_2}\).

- \( T_1: r_1(CEL) r_1(PALM) w_1(CEL, CEL+5) C_1 \)
- \( T_2: r_2(CEL) w_2(CEL, CEL+7) C_2 \)
- \( T_3: r_3(PALM) r_3(CEL) w_3(CEL, CEL+10) C_3 \)

Now, consider that the transactions belonging to \( T \) have been scheduled according to the schedule \( S_1 \) depicted in Figure 4.

![Figure 3: Replicated Environment.](image)

The Temporal Serialization Graph for the schedule \( S_1 \) (Figure 4) does not present any cycle. In other words, \( S_1 \) is serializable. Albeit \( S_1 \) be serializable, it is not correct. This is because, \( S_1 \) produces the following inconsistent state: the item CEL has a value of 6 in copy DB\(_{R_1}\), while copy DB\(_{R_2}\) has a value of 16 for the same item. This phenomenon occurs due to the following fact: the read operation \( r_3(CEL) R_1 \) is not execute on the value written by \( w_2(CEL, CEL+7) R_2 \). Thus in the correct schedule the operation \( r_3(CEL) R_1 \) should precede operation \( w_2(CEL, CEL+7) R_2 \). Figure 5 shows the correct schedule \( S'_1 \).

**Figure 4: Schedule \( S_1 \).**

\[
S_1 = r_1(CEL) R_1 r_1(PALM) R_1 w_1(CEL, CEL + 5) R_1 r_2(CEL) R_1 w_2(CEL, CEL + 7) R_2 r_3(PALM) R_1 r_3(CEL) R_1 w_3(CEL, CEL + 10) R_2 C_1 C_2 C_3
\]

**Figure 5: Schedule \( S'_1 \).**

According to the proposed protocol, when a replicated server receives a commit operation of a transaction belonging to \( S_1 \), the server executes the replica control protocol to update the local temporal serialization graph. In turn the replica control protocol executes the following steps. First, it sends a request to all other replicated servers requiring information on all operations executed on the data items CEL and PALM. As soon as a message (with operations on CEL and PALM) is received the replica control protocol updates the local TSG in order to detect the existence of cycles in the TSG, according to the transaction isolation level specified by the user. By doing this, the proposed protocol aborts \( T_3 \), updates the local graph inform the other replicated servers on the abort of \( T_3 \).

5. **EVALUATION**

We have evaluated the performance of the proposed replica control protocol w.r.t. the following metrics: (i) **Abort rate**, which measures the number of aborts occurred for a set of concurrent transactions, and (ii) **Messages Overhead** for indicating the number of messages exchanged between the replicated servers. The simulation environment consisted of four mobile replicated servers and a mobile client. 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 and 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 transaction. 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. In Figure 6, one can observe that the more permissive the transaction isolation level, the lower the transaction abort rate.

Figure 7 shows that message exchange overhead has a linear behavior regarding the transaction length. Observe that the number of exchanged messages is almost the same, regardless the isolation level is being used. For that reason we advocate that our approach is scalable.

![Abort Rate](image)

**Figure 6: Abort Rate.**

![Messages Overhead](image)

**Figure 7: Messages Overhead.**

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. CONCLUSION

In this paper we presented a fully distributed replica control protocol for mobile replicated database. The proposed protocol guarantees the replicated data consistency and the replica convergence to the same consistent final state (eventual consistency) by means of transaction isolation levels and make updates propagation in a peer-to-peer (P2P) fashion. By doing this, a user can trade off degree of replica consistency for a potential increase in data availability and transaction throughput rate. Besides using the consolidated isolation level notion, the proposed protocol adopts the read-any/write-any approach, allowing higher data availability. Our approach does not deploy data locking to implement the different isolation levels. Finally, our protocol reduces the number of messages exchanged among servers (see Section 5), which minimizes in turn communication cost. The experimental results demonstrate the efficiency of the proposed protocol.

7. REFERENCES


