A Hybrid Approach for Improving Concurrency of Frequently Disconnecting Transactions

Carmine Cesarano†, Angelo Chianese†, Vincenzo Moscato†, Antonio Picariello† and Antonio d'Acierno††

Dipartimento di Informatica e Sistemistica, University of Naples Federico II, Italy†; ISA-CNR, Avellino, Italy††

Summary
Transaction management in different contexts, such as a mobile environment, is still a challenging task. In this paper we propose an efficient integrated method in order to improve concurrency of long lived transactions. Differently from other techniques presented in the literature, we design a hybrid approach between optimistic and pessimistic models.
From one hand, our basic idea consists in the modification of the well known 2PL protocol, in order to take into account frequent disconnections or inactivity periods of mobile devices and, from the other one, we consider the semantic related to operations produced by transactions. The first condition is necessary in order to avoid an indefinite or long resource locking, by disconnecting (or idle) transactions or a high rate of preventive aborts; in the opposite, a transaction "semantic compatibility" is exploited in order to increase the concurrency of reconcilable operations on the same resources. We have implemented a middleware with the aims of emulating a classical transactional scheduler, and several experiments have been carried out.

Key words:
Mobile environments, ,Long lived transaction, 2PL protocol

1. Introduction

Specific characteristics of mobile environments (e.g. variable bandwidth, frequent disconnections, limited resources on mobile hosts) make traditional transaction management techniques (necessary to ensure *ACID* properties) no longer appropriate [1], [2], [3]. In particular, the lengthy transmission delay, frequent and unpredictable disconnections or long inactivity periods of mobile devices affect transaction duration, procreating *Long Lived Transactions*. In this framework several problems, such as low concurrency rate, deadlocks and starvation, handling of disconnections, and so on, have to be solved [7].
Traditional approaches, such as Two Phase Locking (2PL), are not efficient for this kind of transactions. The usual long life of mobile transactions forces a long time resource locking by disconnecting (or idle) transactions, or, in the opposite, a high rate of preventive aborts. To avoid a low concurrency rate, optimistic approaches allow to different transactions to immediately and concurrently operate on the various resources or by relaxing DBMS locking policies or by replicating the shared data on mobile devices. Anyway such approaches could cause the management of a high number of rollback operations on updated data when a high rate of transaction conflicts occur. More in general, in the last years several models for managing transactions in a mobile environment have been proposed.
As underlined in [4] the proposed approaches can be subdivided in two groups of models. The first group includes proposals where transactions are completely or partially executed on mobile hosts. The second group considers transactions requested by mobile hosts and executed on the wired network.
In first class of methods (e.g., Clustering, Two-Tier replication, HiCoMo, IOT, Pro-Motion, Reporting, Prewrite) the main problems are related to ACID properties support and data replication, reconciliation or synchronization techniques are requested. In the second class (e.g., Kangaroo, MDSTPM, Moflex, Pre-serialization), ACID properties are not compromised and focus is on supporting mobile host movements during transaction execution and managing disconnections.
In this paper we propose an efficient integrated method in order to improve concurrency of long lived transactions.
Differently from other techniques presented in the literature, we design a hybrid approach between optimistic and pessimistic models by supposing that transaction execution takes place entirely on fixed hosts.
From one hand, our basic idea consists in the modification of the well known 2PL protocol, in order to handle frequent disconnections or inactivity periods of mobile devices and, from the other one, we consider the semantic related to operations produced by transactions.
The first condition is necessary in order to avoid long time resource locking or a high rate of preventive aborts; in the opposite, a transaction "semantic compatibility" is exploited in order to increase the concurrency rate. More in details, in a similar manner to some optimistic approaches [8] and to different techniques diffused in distributed real time database systems realm [5], [6]. In this work we consider that in a lot of application scenarios it is not always necessary to exactly know the actual value

Manuscript received January 5, 2007
Manuscript revised January 25, 2007

of data for performing special kinds of operations and propose a transaction management model based on the concept of "similarity or compatibility" among mobile transactions in order to reduce the possibility of lock conflicts and improve concurrency of reconcilable operations on the same resources.

The paper is organized as follows. Section 2 describes a motivating example that will be used as a case study trough the rest of the paper. Section 3 describes the proposed model for transaction management. The aim of section 4 is to demonstrate, in according to the motivating example, the possible advantages of our approach, and, some implementation details. Open problems, on going work and concluding remarks are discussed in section 5.

2. Motivating Example

Let us consider the case of a classical e-commerce system selling several products on the WEB.
During a purchase, a user *ui* in general performs the following actions using a mobile device browser:

1. she reads the quantity of available products from the related list *AQui* and the price of a product;
2. she specifies the requested quantity of the selected product *RQui* for buying;
3. after the visualization of total price amount, she commits the related transaction *Ti*.

From the DBMS point of view, this process is schematized as follows:

1. Begin Transaction *Ti*
2. Select AvailableQuantity *AQui* from ProductTable
3. Select ProductPrice from ProductTable
4. Update ProductTable set AvailableQuantity = AvailableQuantity - *RTui*
5. Commit *Ti*
6. End Transaction *Ti*

In a realistic environment, during transaction execution, we have to deal with concurrency problems and the isolation property has to be ensured. In a mobile scenario, the transactions could easily become very long due to variable bandwidth, frequent and unpredictable disconnections, limited resources on mobile hosts as already discussed.
Using a 2PL strategy, in a first hypothesis, we can assume that *Ti* requests a read-lock on AvailableQuantity and then promotes such lock to a write-lock when the user sets the number of requested products. In this case, if another user starts a transaction *Tj* that acquires the read-lock on AvailableQuantity, a deadlock situation can be verified (see Fig. 1) and it has to be solved by Aborted *Ti* and/or *Tj*. When the number of request increases, the number of aborted transactions could become unacceptable.

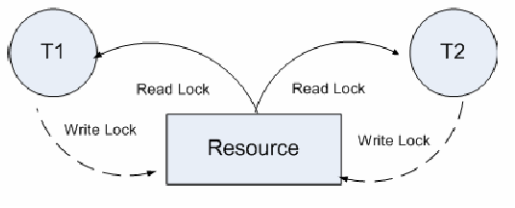

Fig. 1. Deadlock Example

Alternatively, using a strict-2PL, we can assume that we know the semantic of the transactions: in this case we can grant the write-lock to T_i on AvailableQuantity. If the user does not quickly decide to commit or abort the whole operation (e.g., due to network disconnection, inactivity periods, etc...), a long time lock may occur, and, another user transaction that wants to access to the same resource has to wait the end of all previous transactions.
Another widely adopted strategy consists of imposing precise constraints on AvailableQuantity (i.e., it has to be ≥ 0) and assuming that each user operation works in the "auto-commit" mode. Using such a strategy, no deadlocks are possible because both read and write locks are released at each step and the system exploits the highest possible concurrency level among users. A main drawback of this method is that there is no standard way to hold locks during transactions execution, so a user can find boring, and sometimes dramatic, changes to several attributes (e.g., the product price).
Eventually, optimistic approaches assume that each mobile device is provided with a "light DBMS" and purchase transactions are partially or entirely executed on mobile hosts. In this case, after local executions, data replication and reconciliation algorithms are necessary to guarantee ACID properties.
These kinds of problems are avoided in the proposed strategy. In particular, the main features of our approach are: (i) to consider the *semantic* of operations generated by mobile transactions (ensuring a high degree of concurrency among users for a certain types of operations); (ii) to introduce the concept of *disconnected or idle* transaction in the 2PL formulation, where a disconnected transaction is not a transaction to be aborted but a transaction that could reconnect and try to end its work.

3. The Proposed Solution

In the following the proposed model for long lived (mobile) transaction management is described. In the first subsection some preliminary definitions are provided, the second part is dedicated to the model description.

3.1 Preliminary Definitions

Definition 3.1 (Mobile Transaction) *Let us define Mobile Transaction MT each transaction generated by a mobile device and entirely executed by devices located on wired network.*

Definition 3.2 (Transaction Operating Data) *Let* $op_i^j(X)$ *be the i-th operation of j-th transaction on the data resource X, let us define:*

- X_{read}^j *the value of X as retrieved from the database before any kind of modifications operated by the transaction itself;*
- X_t^j *a replica of X on which all operations* $op_i^j(X)$ $(\forall\ i)$ *of MT j will operate;*
- X_{old}^j *the value of X stored in the database when MT j tries to perform the commit;*
- X_{new}^j *the value of X that has to be stored in the database if the commit of MT j is accepted.*

Now we are in the position of introduce the concept of compatibility among transaction operations.

Definition 3.3 (Transaction Operations Compatibility) *A set of operations* $\{op_i^j(X),...,op_h^k(X)\}$ *on the resource X and belonging to one or more different transactions are said to be compatible if:*

1. *assuming no constraint on X, its final value does not depend on the order in which transaction operations are executed;*
2. *there exists a reconciliating algorithm able to determine, given the transaction operating data* X_{read}^j, X_t^j, X_{old}^j *values, the final correct value* X_{new}^j *to be stored in the database at the commit moment*
3. *all transactions are assumed to be committed or aborted..*

For example, addition/substraction or multiplication /division of *X* for constant value (i.e., $X = X \pm c$, $X = X * c$, or $X = \frac{X}{c}$ with $c \neq 0$) are example of compatible operations. A reconciliation algorithm for the first class of operations is the following:

$$X_{new} = X_t + X_{old} - X_{real} \quad (1)$$

Table 1. Mobile Transaction State

***MT* State**	**Description**
Active	a mobile transaction is in such a state if it acquires a lock and goes on with its execution(e.g. related mobile device is still connected))
Waiting	a mobile transaction is in such a state if it is waiting for a lock
Disconnected	a mobile transaction is in such a state if it has been inactive for a period of time greater than a defined threshold (maximum allowed inactivity time)
Committed	a mobile transaction is in such a state if it has performed a commit of its operations
Aborted	a mobile transaction is in such a state if it has performed an abort of its operations

In the opposite a reconciliation algorithm for the second class of operations is:

$$X_{new} = \frac{X_t}{X_{real}} \cdot X_{old} \quad (2)$$

It easy to observe that assignment operation (i.e., $X = c$) is not compatible with any kind of operations. Moreover, for the compatible operations, we are considering that the locked data by a given transaction can be modified by other compatible transaction operations and we will assume no differences between read that are finalized to update and write operations (i.e., we will not consider read and write locks, but just *locks*).

Definition 3.4 (Mobile Transaction Conflicts) *Let MTj, MT j be two mobile transactions. They are said in conflict on a given resource X, if a transaction has the lock on X and the other one performs a lock request on the same resource.*

Definition 3.5 (Mobile Transaction Compatibility) *Let* MT_j, MT_j *be two mobile transactions in conflict on a given resource X and let* $Op_i(X)$ *and* $Op_j(X)$ *be the two set of related generated operations. The two mobile transactions are said to be compatible if the operations belonging to the set* $Op = Op_i(X) \cup Op_j(X)$ *are compatible.*

In general a mobile transaction *MT* can assume different states during its life cycle. In particular, we consider that the set of possible states that a mobile transaction can assume is: $S_{MT} \in \{Active; Waiting; Disconnected; Committed; Aborted\}$. A short description of each state is reported in table 1.

Table 2. Resource States

Resource State	Description
Free	no transaction is using such a resource
Busy	one or more transactions are operating on such a resource, and at least one transaction is not in the *Disconnected* state
BusyDisc	each transaction operating on the resource is in the *Disconnected* state

In the same vein, a given resource can assume different states S_X respect to the transaction operations. The set of possible states that a resource can assume is: $S_X \in \{Free, Busy; BusyDisc\}$. A short description of each state is reported in the table 2.
For each resource *X*, we also consider the set of operations *Op(X)* that transactions perform on *X* itself, so a resource is identified by the triple $\langle X, Op(X), S_X \rangle$

Each mobile transaction migrates from a given state to another one in correspondence of some input events. A state transition can be schematized by means of the following function:

$$\delta : S_{MT} \times I \rightarrow S_{MT} \quad (3)$$

being $I = I_{sunc} \cup I_{async}$ the set of admissible inputs (synchronous and asynchronous respect to transaction execution). In particular:

$$I_{sunc} = \{Lock\,\mathrm{Re}q(op_i^j, \langle X, Op(x), S_X \rangle), Commit\,\mathrm{Re}q, Abort\,\mathrm{Re}q\} \quad (4)$$

$$I_{async} = \{W_{tout}, D_{tout}, Unlock, Discon, \mathrm{Re}con, Abort\} \quad (5)$$

A short description of each input is reported in the table 3.

3.2 Transaction Management

Our transaction management model can be seen as a generalization of the 2PL protocol. Our purpose is to handle *asynchronous* events, such as transaction disconnections and reconnections, timeout ends or resource unlocks and to exploit transaction semantic and compatibility between transaction operations to improve concurrency. To better understanding the proposed method, we show in table 4 the function δ that describe transaction state transition by means of some algorithms.

The lock manager uses the lock policies defined in table 5 to manage lock assignments in function of the state of a resource. In the following, the lock manager behaviour is described referring to the various possible state transitions.

Cases α_1, α_2

A transaction MT_j performs a lock-request $Lock\,\mathrm{Re}q(op_i^j, \langle X, Op(X), S_X \rangle)$ on a given resource *X* that is free or locked by a set of transactions (S_x = *Free V Busy*) which operation are compatible with $op_i^j(X)$. In this cases the lock manager grants the lock to MT_j by performing the actions expressed by algorithm 1.

Algorithm 1 Lock Manager behavior in cases α_1, α_2

Create a new instance of X_{read}^j, X_t^j
$X_{read}^j := X;$
$X_t^j := X;$
$S_{MT_j} := Active;$
$S_X := Busy$

Case α_3

A transaction MT_j performs a lock-request $Lock\,\mathrm{Re}q(op_i^j, \langle X, Op(X), S_X \rangle)$ on a given resource *X* that is locked by a set of disconnected transactions (S_X=*BusyDisc*) which operations are not compatible with $op_i^j(X)$. In this cases the lock manager grants the lock to MT_j and causes the abort of disconnected transactions MT_1,..., MT_w by performing the actions expressed by algorithm 2.

Algorithm 2 Lock Manager behavior in case α_3

Create a new instance of X_{read}^j, X_t^j
$X_{read}^j := X;$
$X_t^j := X;$
$S_{MT_j} := Active;$
$S_X := Busy$
for $k = 1...w$ **do**
  $S_{MT_k} := Aborted;$
end for

Case α_4

A transaction MT_j performs a lock-request $Lock\ \mathrm{Re}\ q(op_i^j, \langle X, Op(X), S_X \rangle)$ on a given resource X that is locked by a set of active transactions (S_X = *Busy*) which current operations are not compatible with $op_i^j(X)$.

In this cases the lock manager denies the lock to MT_j and causes its migration in a waiting state by performing the actions expressed by algorithm 3

Table 3. Inputs

Input type	**Description**
$Lock\ \mathrm{Re}\ q(op_i^j, \langle X, Op(X), S_X \rangle)$	a transaction MT_j performs a lock-request on $\langle X, Op(X), S_X \rangle$ by specifying the operation to be executed $op_i^j(X)$
CommitReq	a transaction requests the commit of its execution
AbortReq	a transaction performs the abort of its execution
Wtout	asynchronous input due to the transaction stay in the *Waiting* state for a time greater than a fixed timeout
Dtout	asynchronous input due to the transaction stay in the *Disconnected* state for a time greater than a fixed timeout
Unlock	asynchronous input due to the lock release by a mobile transaction on a given resource
Discon	asynchronous input due to the disconnection of a mobile transaction
Recon	asynchronous input due to reconnection of a mobile transaction
Abort	asynchronous input due to a lock request of a active transaction MT_k on a resource X locked by a disconnected transaction MT_j and such that $op_i^j(X)$, $op_h^k(X)$ are not compatible operations.

Algorithm 3 Lock Manager behavior in case α_4

$\mathcal{S}_{\mathcal{MT}_j} := Waiting;$

Cases α_5, α_6

An active transaction MT_j performs a commit request *CommitReq*. In this cases the lock manager generates a particular transaction, called *System Transaction*, containing the correct values for each involved resource $X_{1...}X_n$ to be updated on the database, by using if necessary the reconciliation algorithms. Eventually, the system transaction is committed or aborted (for example if some integrity constraints are violated) on the database using a classical 2PL.

Algorithm 4 Lock Manager behavior in cases α_5, α_6

for $k = 1...n$ **do**
  Lock X
  Create a new instance of X^j_{old}
  Initialize X^j_{old} with the current value on the database
  Create a new instance of X^j_{new}
  X^j_{new} :=Reconciliation Algorithm $(X^j_{read}, X^j_t, X^j_{old})$;
end for
Build a System Transaction $\mathcal{ST}$ with the correct values X^j_{new}
if ($\mathcal{ST}$ Commit) **then**
  $\mathcal{S}_{\mathcal{MT}_j} := Committed;$
else
  $\mathcal{S}_{\mathcal{MT}_j} := Aborted;$
end if
for $k = 1...n$ **do**
  Deallocation of $X^j_{new}, X^j_{read}, X^j_t, X^j_{old}$;
end for

Case α_7

A transaction *MT j* performs an abort request *AbortReq*. In this case the lock manager forces the abort of MT_j by performing the actions expressed by algorithm 5.

Algorithm 5 Lock Manager behavior in case α_7

$\mathcal{S}_{\mathcal{MT}_j} := Aborted;$
for $k = 1...n$ **do**
  Deallocation of X_t^j, X_{read}^j;
end for

Cases α_8, α_{11}

A transaction MT_j migrates in a disconnected state. In this case the lock performs the actions expressed by algorithm 6.

Algorithm 6 Lock Manager behavior in case α_8, α_{11}

$\mathcal{S}_{\mathcal{MT}_j} := Disconnected;$

Case α_9

A transaction MT_j in waiting state, after an *Unlock* event, obtains the lock manager on a given resource. In this cases the lock manager grants the lock to *MT j* by performing the actions expressed by algorithm 7.

Table 4. State Transition

Case	S_{MT}	*I*	S_{MT}
α_1	*Active*	$Lock\,\mathrm{Re}\,q(op_i^j, (X, -, Free)$	*Active*
α_2	*Active*	$Lock\,\mathrm{Re}\,q(op_i^j, \langle X, Op, Busy\rangle) op_i^j, Op.comp.$	*Active*
α_3	*Active*	$Lock\,\mathrm{Re}\,q(op_i^j, \langle X, Op, BusyDisc\rangle)$	*Active*
α_4	*Active*	$Lock\,\mathrm{Re}\,q(op_i^j, \langle X, Op, BusyDisk\rangle) op_i^j, Op.inc.$	*Waiting*
α_5	*Active*	CommitReq	*Committed*
α_6	*Active*	CommitReq	*Aborted*
α_7	*Active*	AbortReq	*Aborted*
α_8	*Active*	Discon	*Disconnected*
α_9	*Waiting*	Unlock	*Active*
α_{10}	*Waiting*	W_{tout}	*Aborted*
α_{11}	*Waiting*	Discon	*Disconnected*
α_{12}	*Disconnected*	Recon	*Active*
α_{13}	*Disconnected*	D_{tout}	*Aborted*
α_{14}	*Disconnected*	Abort	*Aborted*

Algorithm 7 Lock Manager behavior in case α_9

Create a new instance of X_{read}^j, X_t^j
$X_{read}^j := X;$
$X_t^j := X;$
$\mathcal{S}_{\mathcal{MT}_j} := Active;$
$\mathcal{S}_{\mathcal{X}} := Busy$

Cases $\alpha_{10}, \alpha_{13}, \alpha_{14}$

A transaction MT_j stays in a waiting state for a time greater than $W_{timeout}$ or stays in a disconnected state for a time greater than $D_{timeout}$, or a disconnected transaction MT_j has the lock on a resource X for the execution of an operation $op_i^j(X)$ and an other active transaction request the lock for the execution of an not compatible operation $op_h^k(X)$. In this case the lock manager forces the abort of MT_j by performing the actions expressed by algorithm 8.

Algorithm 8 Lock Manager behavior in cases α_{10}, α_{13}, α_{14}

for $k = 1...n$ **do**
  Deallocation of X_t^j, X_{read}^j;
end for
$\mathcal{S}_{\mathcal{MT}_j} := Aborted;$

Case α_{12}

A transaction MT_j migrates in a reconnected state. In this case the lock performs the actions expressed by algorithm 9.

Algorithm 9 Lock Manager behavior in case α_{12}

$\mathcal{S}_{\mathcal{MT}_j} := Active;$

In our approach when a mobile transaction is instanced, it is in an active state. After a lock request on a given resource, the transaction can stay in the same state o migrate in an another state depending on the following factors:

- current state of the resource;
- transaction operations to be executed;
- operations of mobile transactions that have locked the
- resource;
- current state of the mobile transaction.

Obtained a lock on a given resource, a generic transaction will use the replicated data X_t to perform its operations. Only at commit event, the reconciliated data will be updated on the database. It is clear how in this model a *Disconnected* transaction can reconnect (we suppose that the disconnection was due to user inactivity or to a temporary network fault) and tries to finish its work if there were not compatible operations that have requested

Table 5. Lock Management

Resource State	***Free***	***Busy***	***BusyDisc***
LockReq	Y	Y/N	Y

the lock on the same resources and operated on the same data.

3.3 Model Evaluation

In this subsection we report several experiments finalized to theoretically evaluate the transaction execution time and the abort percentage of disconnected transactions of our model respect to the classical 2PL.

In particular, the first parameter has been evaluated at the variation of: (i) number of transaction conflicts, (ii) number of not compatible transaction operations.

Let c and t_{ex} be respectively the number of transaction conflicts and the execution time in the ideal condition (no conflicts are verified) of a single transaction, we assume that the 2PL average execution time is given by the following formula:

$$\tau_{2PL}(c) = \frac{(n-c)\cdot t_{ex} + c\cdot(t_{ex} + \frac{t_{ex}}{2})}{n} \quad (6)$$

being n the number of total transactions. In fact, we suppose that the arrival time of a conflicting transaction occurs in half of execution time of the previous one. Note that no multiple conflicts are considered.

In our model we take into account both the number of transaction conflicts c and the number of not compatible transaction operations i. We can model the probability of having k not compatible conflicts among transactions as:

$$P(K) = \frac{(C_{i,k}\cdot C_{n-i,c-k})}{C_{n,c}} \quad (7)$$

being $C_{z,m} = \binom{z}{m}, z \geq m$, 0 if $z \geq m$

Considering such probability, the execution time (which behavior is reported in Fig. 2 by fixing t_{ex} = 1) in our approach is:

$$\tau_{our}(c,i) = \sum_{K=0}^{\min(i,c)} P(k)\cdot\tau_{2PL}(ex) \quad (8)$$

It is possible to observe that the 2PL execution time does not depend on the number of not compatible operations and grows in a linear manner as respect to the number of conflicts. In our approach, we observe an increase of times

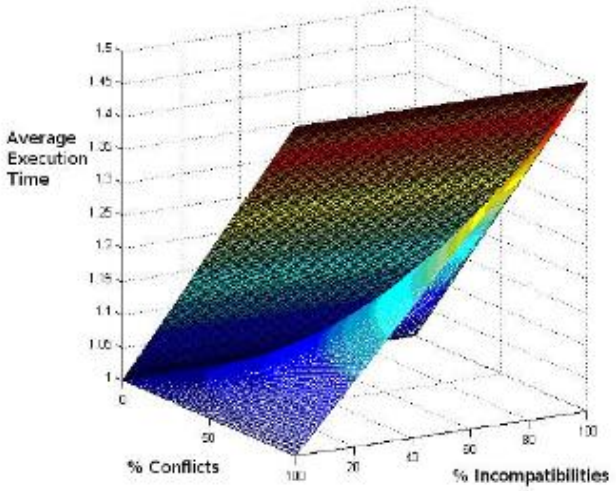

Fig. 2. Average Transaction execution Time

as respect to the number of transaction conflicts and the number of not compatible transaction operations.

However the times are lower than 2PL ones because we do not take into account the overhead due to the reconciliation operations. Our method is more suitable as respect to 2PL when we have medium-high percentage of conflicts and medium-low percentage of incompatibilities. In our best case (c = 100%, i = 0) the proposed approach

presents a theoretical time improvement of 50% respect to 2PL: such enhancement is significant for high values of t_{ex}, in other terms for long lived transactions. In the opposite the abort percentage of disconnected transactions has been analyzed at the variation of: (ii) number of transaction conflicts, (i) number of not compatible transaction operations, (iii) percentage of disconnected transactions. In the 2PL approach we can simply consider the abort percentage as function of disconnecting timeout. Instead, in our approach such percentage can be computed by product of the probabilities (percentage) of having a disconnection $P(d)$, a conflict $P(c)$ and an incompatibility $P(i)$:

$$P(\text{Abort}) = P(d) \cdot P(c) \cdot P(i) \quad (9)$$

In Fig. 3 the abort percentage is reported in function of the percentage of transaction conflicts and the percentage of disconnected transactions for increasing value of the number of not compatible transaction operations.

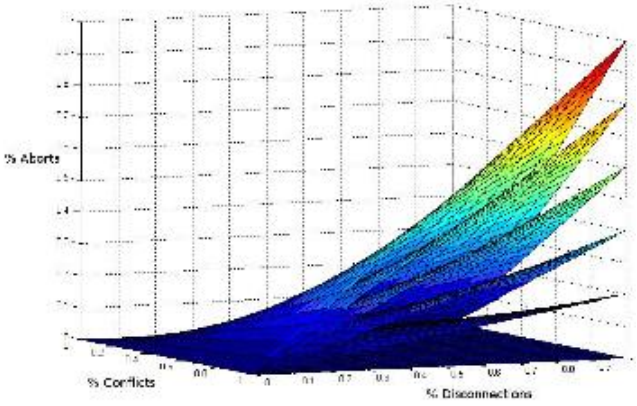

Fig. 3. Abort Percentage of disconnected transactions

4. Implementation and Use Examples

In this section some implementation details about our transaction manager middleware are reported. The developed prototype (which multi-layers infrastructure is schematized in Fig. 4) offers a set of services to manage mobile transactions and is based on the following components:

- a set of **LDBS (Local DataBase Systems)**: they contain applications data and offer the functionalities to manage and store such data, working like a traditional DataBase Management System.
- a centralized **GTM (Global Transaction Manager)**: it is the core part of the architecture. Such componentrepresents our transactional scheduler middleware. It accepts user requests in terms of mobile transactions, generated by mobile devices and then processes them in according to the previous transaction management model. In particular, it allows to manage:
  1. transaction concurrency on the base of their semantic;
  2. possible transaction deadlocks or starvation;
  3. disconnections of user clients;

  Moreover, this module is wired connected to the different LDBS servers in order to manage data on the local databases.
- a set of **Base Stations**: they allow to connect the mobile device to the GTM server. Due to the centralized approach, the management of the user mobility is totally solved, because a given base station is not delegated to track the operations executed by the users, but the mobility is directly managed by the GTM.

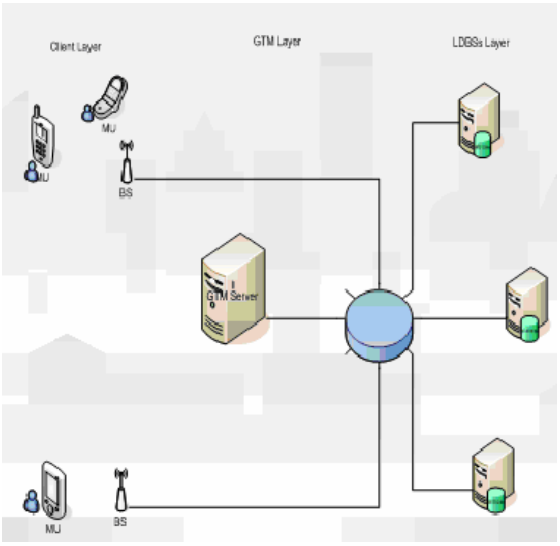

Fig. 4. System Architecture

- **a Mobile Units**: They are the set of mobile devices from which the users have access to the system.

Now some use cases are provided to show the behaviour of our system referring to different transaction data sets and to the motivating example described in section 2. The actors of the systems are two kinds of user: (i) clients that can perform purchases of different products specifying the desired quantity; (ii) administrators that can set the price or increment the available quantity of the various products.

4.1 Example 1

In this case we suppose the presence of four users: three clients that want to buy the same product *p* and one administrator that wants to increment its available quantity. No disconnections are supposed. The table 6 schematized the described situation in terms of initial and final product quantities (Q_i, Q_f) seen by the users (t_1 and t_2 are respectively the commit/abort time and the arrival time of each transactions). Because all mobile transactions generated by the different users are compatible (operations are additions or subtraction), the lock manager grants the lock to all transactions and they can operate in a concurrent manner.
At each transaction commit a system transaction is generated to update the reconciliated data on the database following the algorithm 4.

Table6. Example 1

User	t1	t2	p	Q_i	Op	Q_f
Client1	4+ γ	1	1	100	Read(Q),Q=Q-1	100
Client2	4+2 γ	2	1	100	Read(Q),Q=Q-2	100
Client3	4+3 γ	3	1	100	Read(Q),Q=Q-3	100
Admin1	4+4 γ	4	1	100	Read(Q),Q=Q+6	100

Table 7. Example 2

User	t1	t2	p	Q_i	Op	Q_f
Client1	2+ γ	1	1	100	Read(Q),Q=Q-1	94
Client2	3+ γ	2	1	100	Read(Q),Q=Q-2	94
Admin1	4+ γ	3	1	100	Read(Q),P=110	94
Client3	5	4	1	100	Read(Q),Q=Q-3	94 P=110

4.2 Example 2

In this case we suppose the presence of four users: three clients that want to buy the same product and one administrator that wants to set its price. No disconnections are supposed. The table 7 schematized the described situation in terms of initial and final product quantities seen by the users.
In the proposed scenario, the lock manager grants the lock to the first two client transactions, while administrator transaction has to wait the client operations termination for performing its task. The last client transaction operates after administrator and buys the product at new set price *P*.

4.3 Example 3

In this case we suppose the presence of four users: two clients that want to buy the same product, one administrator that wants to set its price and another client that want to buy a different product. A disconnection of the second client is supposed. The table 8 schematized the described situation in terms of initial and final product quantities seen by the users.
In the proposed scenario, the lock manager grants the lock to the first two client transactions, while administrator transaction has to wait the client operations termination for performing its task. Because the second client is disconnected, at the reconnection time, the lock manager forces his abort due to the execution of an not compatible transaction (administrator operation) on the same resources. The last client transaction operates after administrator and buys the different product.

Table 8. Example 3

User	t1	t2	p	Q_i	Op	Q_f
Client1	2+ γ	1	1	100	Read(Q1),Q1=Q1-1	99(Q1)
Client2	3+2 γ	2	1	100	Read(Q1),Q1=Q1-2	99(Q1)
Admin1	3+ γ	3	1	100	Read(Q1),P=110	-----
Client3	4+ γ	4	2	100	Read(Q2),Q2=Q2-50	50(Q2)

Table 9. Example 4

User	t1	t2	p	Q_i	Op	Q_f
Client1	2+ γ	1	1	100	Read(Q),Q=Q-1	94
Client2	4+2 γ	2	1	100	Read(Q),Q=Q-2	94
Client3	3+ γ	3	1	100	Read(Q),Q=Q-1	94
Client41	4+ γ	4	1	100	Read(Q),Q=Q-2	94

4.4 Example 4

In this case we suppose the presence of four users: all clients want to buy the same product, but the second client has a disconnection. The table 9 schematized the described situation in terms of initial and final product quantities seen by the users.
In the proposed scenario, the lock manager grants the lock to all transactions that can perform their operations in a concurrent manner. Also the second client, at the reconnection time, can continue this work because only compatible operations have operated on the data.

5. Conclusions and Future Work

In this work we have proposed a novel approach for managing concurrency of long lived transactions. We have shown that such solution presents in precise conditions different advantages both in terms of execution times and abort percentage of the disconnected transactions as respect to the 2PL. A first problem of this model could be due to possible conditions of starvation for not compatible transactions that try to access to resources locked by different compatible transactions. Possible solutions for this problem are: the introduction of a transaction priority or the lock-deny on a given resource for compatible transaction if in the resource queue there are a certain number of not compatible transactions that are in a waiting state.

A second problem is connected to the possibility of a high rate of aborts due to the violation of integrity constraints on the database during the data reconciliation process. A possible solution for this problem is to limit the number of possible concurrent and compatible transaction on a given resource in function of the current value *X* of the resource.

References

[1] Action members : EPLF, U. Grenoble, Inria-Nancy, Int-Evry, U. Montpellier 2, U. Paris 6, U. Versailles, "Mobile Databases: a Selection of Open Issues and Research Directions" SIGMOD Record, Vol. 33, No. 2, June 2004.

[2] D. Barbara', "Mobile Computing and Databases-Survey", IEEE Transactions on knowledge and data enginnering, vol. 11, n. 1, 1999.

[3] P.K. Chrysanthis, "Transaction processing in a mobile computing environment," in IEEE Workshop on Advances in Parallel and Distributed Systems (APADS), Princeton, USA, October 1993.

[4] P. Serrano, A. C. Roncacio, and M. Adiba "A Survey of Mobile Transactions" Distributed and Parallel Databases, 16, 193-230, 2004.

[5] GuoQiong Liao, YunSheng Liu, LiNa Wang and ChuJi Peng, "Concurrency control of real-time transactions with disconnections in mobile computing environment", Computer Networks and Mobile Computing, 2003. ICCNMC 2003. 2003 International Conference on 20-23 Oct. 2003 Page(s):205 - 212.

[6] Kam-Yiu Lam, Tei-Wei Kuo, Wai-Hung Tsang and Gary C. K. Law, "Concurrency control in mobile distributed real-time database systems", June 2000, Information Systems, Volume 25, Issue 4.

[7] Nitin Prabhu and Vijay Kumar, "Concurrency control in mobile database systems", Advanced Information Networking and Applications, 2004, Volume 2, pp. 83-86.

[8] S.H. Phatak and B.R. Badrinath, "Conflict resolution and reconciliation in disconnected databases", Database and Expert Systems Applications, 1999.

Carmine Cesarano received the Laurea degree in Computer Science and Engineering from the University of Napoli, Italy, in 2002. In 2002 he joined the Dipartimento di Informatica e Sistemistica of the University of Napoli "Federico II", as research follow and at the end of 2003 he started a Ph.D. program in Computer Science and Engineering and in 2006 he received the Ph.D. degree. Currently he is working with the advanced multimedia database group of the University of Naples Federico II. His research interests lie in Information Retrieval, Multimedia Information Systems, Knowledge extraction and Mobile Transactions

Angelo Chianese received the Laurea degree in electronics engineering from the University of Naples, Federico II in 1980. In 1984, he joined the Dipartimento di Informatica e Sistemistica of the University of Naples "Federico II" as an Assistant Professor. Currently he is a full professor at the University of Naples Federico II He has been active in the field of pattern recognition, optical character recognition, medical image processing, and object-oriented models for image processing. His current research interests lie in multimedia data base and multimedia content management for e-learning.

Vincenzo Moscato received the Laurea degree in computer science and engineering from the University of Naples "Federico II," Naples, Italy, in 2002. In 2005 he received the Ph.D. degree in computer science and engineering at the same university and currently he is working with the advanced multimedia database group of the University of Naples "Federico II" His research interests are in the area of image processing (active vision) and multimedia database systems (image databases, video databases, and architectures for multimedia data sources integration).

Antonio Picariello received the Laurea degree in electronics engineering and the Ph.D. degree in computer science and engineering, both from the University of Naples, Naples, Italy, in 1991 and 1998, respectively. In 1993, he joined the Istituto Ricerca Sui Sistemi Informatici Paralleli, The National Research Council, Naples, Italy. In 1999, he joined the Dipartimento di Informatica e Sistemistica, University of Naples "Federico II," and is currently an Associate Professor. He has been active in the field of computer vision, medical image processing and pattern recognition, object-oriented models for image processing, and multimedia database and information retrieval. His current research interests lie in knowledge extraction and management, multimedia integration.

Antonio d'Acierno received the Laurea degree in electronics engineering from the University of Naples Federico II. He is currently a senior researcher at the ISA-CNR of Avellino . His current research interests lie in the field of mobile transactions, information retrieval, semantic web and bioinformatics.