DIMACS Series in Discrete Mathematics
and Theoretical Computer Science

An Adaptive Concurrency Control Protocol for Mobile Transactions

Elisa Bertino, Elena Pagani, and Gian Paolo Rossi

ABSTRACT. Wireless networks are becoming an essential component of any infrastructure for distributed data management. Unlike wired hosts, mobile hosts can be temporarily unreachable as consequence of their movements across different cells, their energy limitation or unavailable wireless channels. Mobility forces mobile hosts to alternate connected and disconnected work. When connected, they perform personal communications and access shared data and services from fixed hosts. When disconnected, they can process locally cached data. Transaction management in such an environment poses several issues and requires extensive modifications to techniques and algorithms developed for wired networks. One relevant issue is to devise caching strategies according to which data can be cached at mobile hosts to enable disconnected work. When working in disconnected mode, a mobile host performs updates on its own cached copy of data. Such updates are then applied to the data stored at the fixed hosts when the mobile host reconnects. We refer to such transactions as mobile transactions. To guarantee database consistency proper concurrency control mechanisms must be adopted to serialize mobile transactions. Two possible approaches are the pessimistic approach, under which only one mobile transaction at the time may have a cached copy of the data, and the optimistic approach, under which all mobile transactions may have a cached copy of the same data. Both approaches have severe shortcomings, that are discussed in the paper, and no one is better than the other under all application patterns. In this paper, we propose an adaptive concurrency control protocol that dynamically modulates the number of data copies that can be concurrently cached by mobile hosts. We evaluate the performance of our protocol through simulation. In particular, the evaluation shows that our protocol helps in minimizing energy expenditure by mobile hosts - an important requirement in mobile computing systems.

1. Introduction

Wireless networks are becoming an important component of a system of interconnected networks. The most rapidly growing segment of personal computers, which includes both portable computers and Personal Digital Assistants (PDAs), can find in the wireless technology the infrastructure to access data and services

1991 *Mathematics Subject Classification.* Primary 68M10, 68M20, 68P15.
Key words and phrases. mobile host networks, concurrency control, replicated databases.

available at the wired part of the Internet. Although such communication infrastructure represents an important opportunity to deploy a variety of new distributed applications (see, for instance, [**20**]), the interconnection between wired and wireless networks requires adaptations at different layers of the communication protocol stack, from the routing protocols up to the application support protocols.

While the development of wireless networks poses several challenging issues to telecommunication research [**16, 5**], several issues also arise with respect to data management research [**14, 6**]. In particular, as one of the main application areas for mobile computing systems refers to ubiquitous information use and modification, proper techniques for transaction management must be devised. Transaction management has already been widely investigated in the area of database systems (both centralized and distributed) [**12, 1**]; problems and their solutions are today well understood. Transaction management in mobile computing systems poses, however, more difficult issues because of the narrow bandwidth of the wireless communication channels, the limited power supply and the changing locations of mobile units. Such characteristics require extensive modifications to existing transaction management techniques. In particular, whereas in database systems over wired networks, the key performance metric is the response time, in wireless networks both response time and energy expenditure of mobile units are important [**10**]. As a consequence, algorithms that were designed to optimize response time only may not be any longer adequate when both response time and energy expenditure must be optimized.

To better focus our discussion, we consider the case of a transactional environment in which mobile hosts (MHs) access databases located on fixed hosts (FHs)[1]. Here, the mobility introduces four main issues: (1) Mobile hosts may need to work in *disconnected* mode, or may happen to be temporarily disconnected. As a consequence, transactions may last longer. (2) MHs are free to move among wireless networks whose technology, traffic conditions and bandwidth (ranging from Kbps to Mbps) can be very different. Thus, the network resource can be sometimes scarce. (3) The battery life-time represents an extremely critical resource for all mobile devices. The communications over the wireless links are the main responsible of the battery consumption and, therefore, the packet exchange between mobile and fixed hosts should be minimized. (4) A very large number of potential users can access the databases from thin, mobile, chip devices. Scalability is one of the primary requirements.

Unfortunately, those four issues entail contrasting requirements with respect to a transaction management system. The need of supporting disconnected work and the characteristics of the communication infrastructure (issues (1) and (2) above) require the use of a caching mechanism according to which data can be stored at MHs, thus enabling MHs autonomy. When working in disconnected mode, a MH performs updates on its own cached copy of data. Such updates are then applied to the data stored at the FHs when the MH reconnects. We refer to such transactions as *mobile transactions*. To guarantee database consistency proper concurrency control mechanisms must be adopted to serialize mobile transactions. Two possible approaches are the pessimistic approach, under which only one mobile transaction at the time may have a cached copy of the data, and the optimistic

[1] Note that other architectures have been devised such as the multi-hop architectures, where all the hosts in the system are mobile. Such architectures are out of the scope of the present paper.

approach, under which all mobile transactions may have a cached copy of the same data.

The *pessimistic* approach, adopted for instance in [**6**], avoids conflicting updates by preventing transactions from executing operations that might *collide*; this is generally achieved through the use of *locks* on data items. Such an approach minimizes the use of critical resources at the price, however, of high latency time, and low throughput. Furthermore, sudden disconnections of MHs can cause *orphan* locks and connected work can be penalized by disconnected work. On the other hand, the *optimistic* approach [**18**] has the opposite behavior by allowing the MHs to concurrently cache data item copies according to their needs. Such an approach increases transaction throughput and reduces latency time. Moreover, when the number of concurrent accesses to the same data items is low, such as in a file system, an optimistic approach fully enables disconnected work. Possible conflicts can be repaired upon reconnection [**23, 15**] through the use of transaction recovery protocols. However, the presence of a large number of MHs (issue (4) above) may drastically increase the number of concurrent transactions and, thus, the frequency of conflicts. When the number of conflicting transaction increases, MHs are repeatedly involved in the recovery of aborted transactions. Transaction recovery can be very expensive in terms of both local processing power and network resources. In this case, disconnected work can be penalized by connected work.

It is thus clear that when a transaction management system must support mobility, by ensuring at the same time scalability and resource usage optimization, with good performance, the adoption of a single concurrency control policy may result in severe shortcomings. Different policies must be adopted that dynamically range between the optimistic and the pessimistic approaches according to the dynamics of the competing transactions, as defined by the access patterns, and of the system resources status.

In this paper, we propose an *Adaptive Concurrency Control Protocol*, henceforth also called ACP, which allows the system to modulate the number of copies of data items that are concurrently cached at MHs. By dynamically varying such number, the system is able to modulate the number of concurrent operations on data items and thus to control the number of transaction aborts when critical resources have to be saved.

The paper is organized as follows. In Section 2, we describe the system model and the application environment which are considered throughout the paper. In Section 3, we describe the proposed adaptive protocol. The performance evaluation of the protocol, obtained by means of simulation, are given in Section 4. Finally, Section 5 compares our approach with related work, whereas Section 6 presents some concluding remarks and outlines future research.

2. System model

In our work, we have used a general architecture of a mobile computing system which we show in Figure 1 [**11**]. The architecture includes a number N of mobile hosts (MHs) accessing data items located at fixed hosts (FHs). FHs are interconnected through a wired network in which a subset of the nodes (the *Mobile Support Stations*, MSSs) are equipped with a wireless interface. An MSS supports communications among MHs within a *cell* and among MHs in other cells by relying to the remote MSS. A cell is the area in which the signal generated by the MSS can

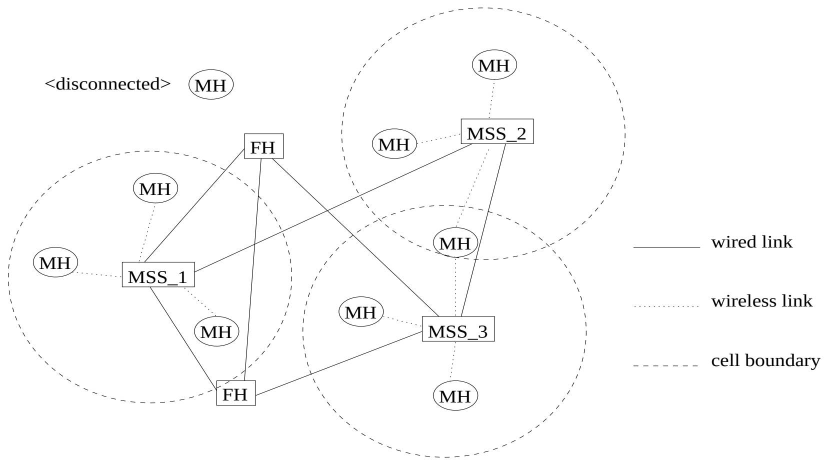


FIGURE 1. A general architecture of a mobile computing system

be received by MHs. Communications within a cell are broadcast and MHs filter the packets according to their destination address. A MH can communicate with another MH in the same cell only by relying to the local MSS.

A MH can *move* across cells, thus changing the *wireless* network topology over time. When the MH_m moves from the cell of MSS_1 to the cell of MSS_2, a *handoff* procedure is activated between MSS_1 and MSS_2 to transfer the state information about MH_m to MSS_2. A *mobility assumption* [**2, 3**] is required to ensure system liveness: MH_m remains in a cell the time needed to complete the handoff procedure and to allow the local MSS to deliver undelivered packets to MH_m; this way, message starvation is avoided.

MHs may also be temporarily *disconnected*. Disconnections occur as the consequence of *planned* events (e.g. to save energy) or of *sudden* events (e.g. moving to uncovered zones, energy exhaustion or crash failure).

We assume that all the MHs have comparable processing and memory resources. The MHs are mainly constrained by the short lifetime of the batteries, that are heavily affected by the communication over the wireless links. To reduce the waste of energy, a MH can enter the *doze mode*: a doze MH has only the network interface active. As soon as a message addressed to the MH is received, the MH is awakened to revert to the normal operation mode. This is an important facility that should be exploited by transaction management protocols designed for a mobile environment.

We consider a database as a collection of *data items*. A *version number* (*vn*) is associated with each data item. Applications can execute two kinds of operations on the data items: *transactions* and *queries*. Only transactions commit and they can commit if all the accessed data items have the version number of the most recent database copy. We make the assumption that an application performing a query can tolerate stale data; hence, the version numbers of the data items accessed by the query are not checked and the query does not have to commit.

The database, or part of it, can be replicated at the various FHs. We refer to such replication as *primary replication* as opposed to *secondary replication* (or *cached replication*) that refers to cache copies stored at MHs. Primary replicas are kept consistent by using one of the available *multiple copy update* protocols designed for distributed database systems [**9, 21**]. We assume that a MH is able to access the nearest *Replica Manager* (RM) whenever it needs to access a data item. Such an access can be transparently achieved by redirecting the MH request packet from the *home RM*, i.e. the RM where MH firstly logged on, to the current RM. Redirection can be performed by MSSs through tunneling techniques, which are often used to support communications between fixed and mobile stations (e.g. Mobile-IP, [**22**]). If the nearest RM of a MH changes as the consequence of mobility and before the termination of the MH's transactions, RMs communicate to exchange the transaction status of MHs that are moving across cells. This way, MHs are guaranteed that the nearest RMs they access are always aware of their transaction operations. For sake of simplicity, we assume, in the following, that RMs coincide with MSS stations.

To allow disconnected work, MHs *cache* a copy of the selected data items, together with their version numbers. We make the assumption that the cache is large enough to contain the needed data items and that data are cached before disconnection occurs. We use the term *disconnected transaction* to indicate a transaction processed on a disconnected MH.

3. The adaptive concurrency control protocol

In this section, we first outline the mobile transaction management protocol that we have adopted in our work. We then present the adaptive concurrency control mechanism, according to which data items copies are cached at MHs. The protocol is mainly based on a dynamically varying parameter, referred to as *number of releasable copies* (*rc* for short) which determines, for each data item, the number of cached copies of the data item that can be concurrently released to MHs. Such parameter can range between a lower and an upper bound. Heuristics for setting *rc* as well as its lower and upper bounds are also presented in this section. Table 1 lists, for the convenience of the reader, the acronyms and variables more frequently used in this paper. For each acronym and variable, the table reports a brief explanation of its meaning.

3.1. Mobile transaction management protocol. The transaction management protocol that we have used in our work is shown by the diagram in Figure 2. It is similar to the protocol proposed in [**15, 19**].

A MH_a that must process a transaction T_a requests to the nearest RM the data items that T_a requires. The RM replies with the copy of the requested data items according to the adopted concurrency control policy. These data items are cached at MH_a. MH_a may process T_a either while connected or while disconnected. In the former case, either the optimistic or the pessimistic approach can be adopted. If the optimistic approach is chosen, the *write-back* mechanism can be used to inform a MH that its cache is inconsistent[2], whereas if the pessimistic approach is chosen locks have to be acquired on the data items before accessing them. In the latter case (i.e., disconnected execution), MH_a records in a *log file* the updates transaction T_a makes to the cached data items. Upon successful completion, T_a *locally commits* at MH_a. We say that a transaction *locally commits* when it commits on the disconnected MH that processed it. Note that a locally committed transaction may still abort. Upon reconnection of the MH, the log-file is transferred to the

[2] See [**15**] for a discussion about the possible optimization of the write-back policy in the mobile environment.

TABLE 1. Notation and terminology

Acronym	**Meaning**
ACP	Adaptive Concurrency Control Protocol
MH	mobile host
FH	fixed host
MSS	mobile support station
RM	resource manager
$as(T)$	abort set of transaction T
$vn(X)$	version number of data item X
$rc(X)$	number of releasable cache copies of data item X
$out(X)$	number of currently cached copies of data item X
$cc(X)$	"commit counter" for X
LB	lower bound for rc
UB	upper bound for rc
R	number of RMs in the system
N	number of MHs in the system
n	number of MHs without resource problems
$r = n/N$	ratio of MHs without resource problems
m	constant $\simeq N$

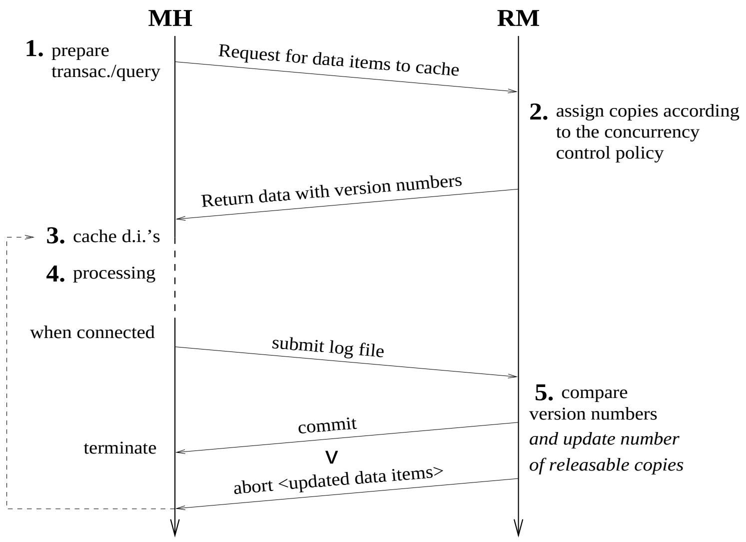


FIGURE 2. Mobile transaction management protocol

nearest RM, which forwards it to the other RMs having copies of the modified data items. All RMs check for conflicts with other possible concurrent transactions by comparing the version numbers of the data items. If no conflict occurs, all RMs redo the logged operations and the disconnected transaction *globally commits*. When a disconnected transaction globally commits, the version numbers of all the written data items are incremented. The durability of locally committed transactions is not guaranteed until they globally commit. All RMs have a consistent view of the data items and of their vn's. A RM notifies a MH whether its transaction committed or aborted. In the case of abort, a MH decides the recovery actions to undertake according to specific application requirements. To enable possible automatic re-execution of the transaction, the abort notification packet transports the copy of the current version of the transaction's data items [**19**]. The abort of a disconnected transaction T may cause the *cascading abort* of all the other disconnected transactions that locally committed after T on the same MH (see paragraph 3.2.1).

In some situations, an RM may unilateraly decide to abort a disconnected transaction. Such an abort, that we refer to as *forced abort*, is necessary in our approach to handle disconnected transactions that after a reasonable amount of time do not "connect" again. The amount of time should be large enough to let the majority of transactions complete and force the abort of a transaction only under exceptional circumstances. We refer to such amount of time as *execution timeout*. We elaborate more on its use in subsection 3.2.

Figure 3 shows the state diagram for transactions in our transaction management protocol. In the graphical representation, the state "forced to abort" denotes a forced abort of the transaction by its nearest RM because of the execution timeout has expired. In the graphical representation, an arc connecting the dashed rectangle to the "forced to abort" state denotes that a transition can occur between any state inside the rectangle and the "forced to abort" state. A RM may indeed force the abort of a transaction running at a disconnected MH, independently from the transaction status, of which the RM is unaware. The MH will be notified about the transaction abort only upon reconnection. It is important to notice that a forced abort has the same effect as a global abort with respect to data held by the aborted transaction and to transaction recovery. However, in our adaptive protocol, as we will show in the following subsection, we need to distinguish between global aborts due to concurrency control conflicts and aborts due to execution timeout expiration.[3]

Moreover, in the description of our protocol, we will use the notion of *transaction abort set*. Such a set contains the identifiers of data items for which a given transaction experienced conflicts, due to concurrent accesses, which resulted in the abort of the transaction. Let T be a transaction: its abort set is denoted as $as(T)$. For example, consider the disconnected transaction performed by MH_i in Figure 6, let us indicate it as T_i: T_i globally aborts because of conflict on the data item X; hence, $as(T_i) = \{X\}$.

3.2. Overview of the ACP protocol. The ACP protocol, used at step 2 of the transaction management protocol (Figure 2), is based on maintaining for each data item X, a parameter, denoted as $rc(X)$, indicating the number of *releasable copies* of X. Releasable copies refer to copies of X that can be cached at different MHs.

The problem of adapting to changing conditions in a mobile transactional environment can be therefore stated as the problem of dynamically modifying rc values for the various data items in the database according to the changing conditions of the environment. To this purpose, two aspects must be considered. First, the amount of the releasable copies rc of a data item should be determined according to the current access pattern. If the observed rate of conflicting transactions per data

[3]For simplicity of presentation, we assume that global aborts are only due to concurrency control conflicts. Of course, transactions may globally abort for other reasons. Our assumption can, however, be easily removed.

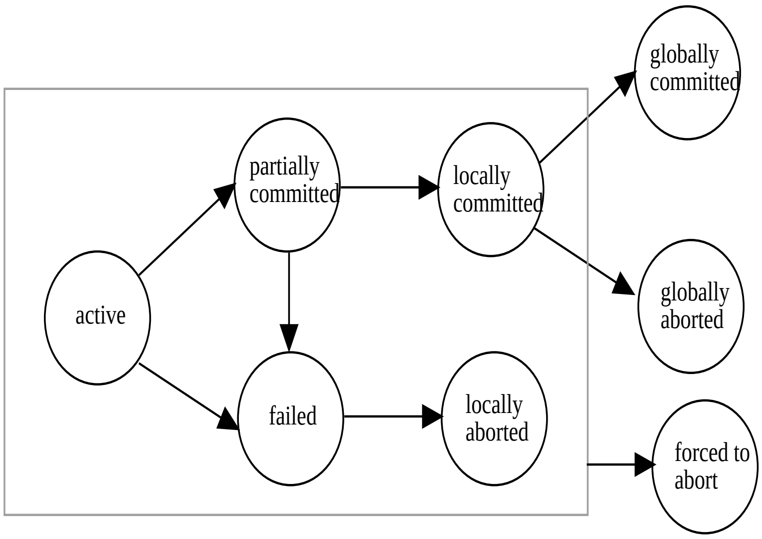


FIGURE 3. Transaction state diagram

item is low, then RMs should quickly shift towards an optimistic policy by incrementing rc. When the conflict rate grows, RMs should reduce rc accordingly, thus behaving more pessimistically. Second, for a given access pattern, the current value of rc should reflect the system status (in terms of on-board and network resources). If resources are lacking, it is necessary to control rc accordingly. The design of the adaptive mechanism must combine the above requirements into a single policy that controls the number of copies that can be concurrently cached at MHs. We elaborate on these issues in the following subsections.

To support an adaptive behavior between a pessimistic policy and an optimistic policy, the rc parameter for a data item X is associated with an interval (LB, UB) denoting the range of values that rc can assume. In particular, LB is the lower bound corresponding to a pessimistic policy, and UB is the upper bound corresponding to an optimistic policy. (Note that different data items can have different rc). By dynamically regulating rc according to the current rate of abort notifications per data item, we can address the first of the above requirements. LB is the second dynamically changing parameter of the ACP protocol. If the reported status of the resources is not critical, LB can be kept high to emphasize the optimistic nature of the policy. As resources conditions get critical, LB is reduced, although, in our simulation experiments, we never set LB to 1. By properly modulating LB, we are able to adapt the concurrency control policies to varying conditions in the resources and thus to address the second of the above requirements.

In the remainder of this section, we describe in some detail the mechanisms adopted to determine the values of the variables rc and LB.

3.2.1. *The adaptive mechanisms of the ACP protocol.* To support the ACP protocol, each RM maintains a table with an entry for each data item, local at the RM. An entry of such a table at RM_i contains for a data item X, located at RM_i, the following information:

- the version number of X, denoted as $vn(X)$;

- the current number of copies of X, denoted as $rc_i(X)$, that can be concurrently released by RM_i;
- the number of currently cached copies of X, denoted as $out_i(X)$, released by RM_i;
- the commit counter for X, denoted as $cc_i(X)$. $cc_i(X)$ is incremented by RM_i whenever a transaction T, which accesses X and is performed by the MH using RM_i, globally commits. In the event of global abort of T due to a conflict on X, $cc_i(X)$ is set to 0.

Note that $vn(X)$ is a global variable which is consistently maintained among all RMs which store a copy of X. rc, out and cc are local to each RM and are initialized respectively to LB, 0 and 0. The total amount of copies that the system can release at the time t for data item X is obtained by adding the $rc(X)$ values of each RM at time t. Similarily, we can obtain the total amount of outstanding copies of X at time t. The value for $rc(X)$ depends on the current history of successful global commits and varies according to the following heuristics, which is implemented by each RM.

Heuristics for rc. Let X be a data item, $rc_j(X)$ and $out_j(X)$ be, respectively, the number of releasable copies and currently cached copies of X at RM_j. Then, $rc_j(X)$ is incremented by one when the number of consecutive globally committed transactions accessing X is equal to the maximum number of copies of X that can be concurrently cached, whereas when a single transaction T_j accessing X globally aborts because of a conflict on X, $rc_j(X)$ is set to LB. The way we dynamically adapt rc does not result in any additional cascading aborts. It only affects new transactions, whose execution is delayed, whereas ongoing transactions normally complete. If T_j globally aborts and $X \notin as(T_j)$, then $rc_j(X)$ is left unchanged. If, by contrast, we were to update $cc_j(X)$ and $rc_j(X)$, as if T_j had committed, there would have been the risk that too many copies of X would become releasable when the locality in accessing the data items changes, thus increasing the probability of aborts.

In Figure 4 we outline the algorithm executed by RM_j on receiving the request to globally commit T_j. By adopting this heuristics, the values of rc cannot grow too fast and the concurrency degree on a given data item also depends on the variable LB, thus allowing for the status of the system resources. Figure 5(a) shows the rc profile for given data item X according to the transaction commit history.

Each RM storing a copy of X, can locally maintain $rc(X)$ by exploiting its global knowledge about the results of the global commit procedure.

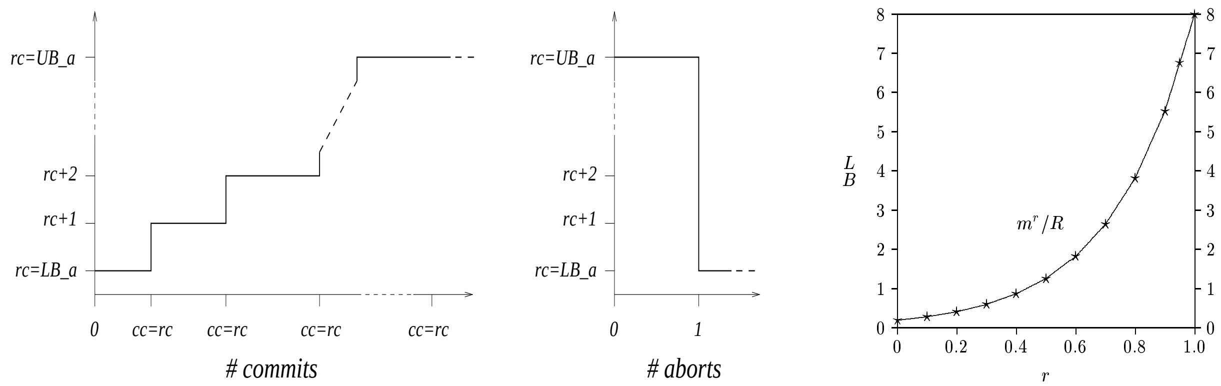
As long as $out_i(X) < rc_i(X)$, RM_i can satisfy requests for copies of X to cache. Since some transactions may last indefinitely and never release their copies of data items, the number of available copies may happen to remain indefinitely lower than rc. To cope with this event, the RM assigns an execution timeout to each transaction and forces the transaction to abort in the event of expiration of such timeout. Each MH is made aware of execution timeout for all its local transactions. This is important to allow the MH to locally discontinue a transaction after the expiration of the timeout. Because of the stringent requirements concerning energy expenditure it is important that MHs avoid useless computations by discontinuing transactions that eventually will be aborted by the RM upon reconnection. A proper dynamic tuning of the execution timeout is therefore relevant. In the current version, we assign the same execution timeout to all transactions.

ALGORITHM 3.1. **Transaction_Commit_Request**

INPUT: A request for globally committing a transaction T_j
OUTPUT: YES, if T_j globally commits
NO, otherwise
METHOD:

1. execute global commit protocol
2. **if** T_j globally commits **then**
   reply $\leftarrow$ YES;
   $\forall\ X$ accessed by T_j:
   $out_j(X) \leftarrow out_j(X) - 1$;
   $cc_j(X) \leftarrow cc_j(X) + 1$;
   if $cc_j(X) = rc_j(X)$ **then**
   $rc_j(X) \leftarrow rc_j(X) + 1$;
   $cc_j(X) \leftarrow 0$;
   otherwise /* abort */ :
   reply $\leftarrow$ NO;
   $\forall\ X$ accessed by T_j:
   if $X \in as(T_j)$ **then**
   $rc_j(X) \leftarrow LB$;
   $cc_j(X) \leftarrow 0$;
   else do nothing;
3. **return** reply

FIGURE 4. An algorithm for processing global commit requests



(a) (b)

FIGURE 5. (a) Profile of rc vs. the result of global commit. (b) Value of LB with $m = N = 40$ vs. different values of r.

The following example illustrates the approach.

EXAMPLE 3.2. Figure 6 (a) shows RM_a at a given time t. RM_a released two out of three copies of data items X and Y. When the disconnected transaction T_i of MH_i tries to globally commit, it aborts because it reports an old vn for X. An updated copy of X is sent to MH_i together with the abort message. On the contrary, the transaction performed by MH_k globally commits. In Figure 6 (b), we show how the system changes after the events described in Figure 6 (a). RM_a sets $cc_a(X)$ to 0 and decreases $rc_a(X)$ to the current value of the lower bound (in this

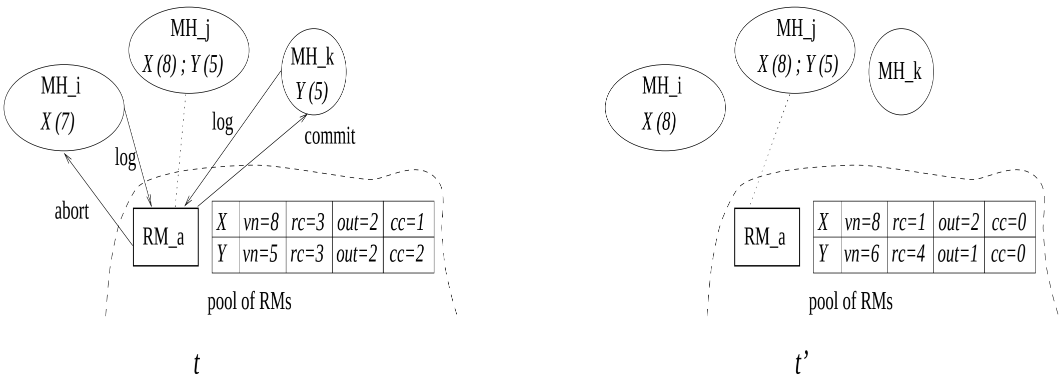


FIGURE 6. Data structures maintained by RMs: (a) at time t; (b) situation at time $t+1$

ALGORITHM 3.3. **Data_Copy_Request**

INPUT: A request for a cache copy of data item X
OUTPUT: YES, if the requested copy can be granted
NO, otherwise
METHOD:

1. **if** $out_j(X) < rc_j(X)$, **then**:
   reply $\leftarrow$ YES;
   $out_j(X) \leftarrow out_j(X) + 1$;
   otherwise:
   reply $\leftarrow$ NO;
   add request to the hold-back queue
2. **return** reply

FIGURE 7. An algorithm for processing data item requests

example, $LB = 1$). The version number of the data item Y is incremented, as well as $cc_a(Y)$. Since $cc_a(Y)$ becomes equal to $rc_a(Y)$, $rc_a(Y)$ can be incremented and $cc_a(Y)$ is reset to 0. Should a second transaction T_i' be running on MH$_i$ and should T_i' depend on T_i, then T_i' is forced to abort after the abort of T_i.

If a copy cannot be granted, the transaction request is queued into an *hold-back queue*. The MH locally running such transaction can enter the doze mode until some copy is released[4]. While waiting, a MH can move across cells. The implemented *handoff* procedure contains the exchange of any queued request being generated by the moving MH. Note that possible deadlock situations that can arise can be solved by the RMs according to usual distributed deadlock algorithms [**17**].

Figure 7 outlines an algorithm, executed by RMs, for handling data copy requests from mobile transactions.

3.2.2. *Dimensioning UB and LB.* If conflicting transactions are infrequent, disconnected transactions can globally commit and the values of rc can grow accordingly. In this event, to enable the system to behave optimistically, rc should

[4] Unless, of course, the MH has other work to do.

be able to grow until the highest concurrency degree is reached. In the current implementation of ACP, UB_i for the RM_i is set to the number of MHs for which RM_i is the nearest RM (in our case, the number of MHs in the cell of RM_i; membership information are available at each MSS).

The value of LB determines the lower bound to which rc is reset initially or whenever an abort occurs. It determines the nature of the system, optimistic or more controlled, and should be dynamically modified according to:

1. the reported status of the critical resources, such as the wireless bandwidth and battery charge;
2. the application (and the user) behavior in case of abort. In fact, if the system behaves optimistically, MHs have a relevant role in the transaction recovery, while if it behaves pessimistically, the control is left to RMs and MHs can be awaken up when the transaction globally committed. Users or application processes could be wishing to indicate their preferences.

We assume that each MH periodically sends control information about points 1 and 2 to its current RM. For sake of simplicity, this ACP implementation uses a single bit, the *status-bit*, which is piggybacked on the *Request* packet to report whether or not the MH has resource problems. RMs use *status-bits* to compute LB, whose value must be equal for all the RMs, that is, all RMs use the same LB, to ensure that a MH in a critical area has equal opportunities of a MH plenty of resources. If, by contrast, RMs in a critical area were to use a LB value lower than the one used by the other RMs, their MHs would have less probability of obtaining the required data items and would starve. To this purpose, RMs periodically exchange the received *status-bits* and compute the ratio $r = n/N$, where n is the number of MHs that *do not* report resource problems and N is the total amount of MHs in the system. Let R be the number of RMs in the system. We compute the lower bound $\boxed{LB \text{ as } m^r/R}$, where m can be set close to N. We have chosen this function for computing LB because it shows a sort of 'slow start'. It forces LB to grow slowly when most MHs have problems, while it leaves LB to grow quickly otherwise, as shown in Figure 5(b), where the values of LB are reported for $m = N = 40$, $R = 5$ and for different values of r.

The combined use of the simple heuristics to compute rc and LB satisfies the adaptation requirements of mobile transactions. In fact, if the system suffers from lack of resources and the rate of aborts is low, high performance can be achieved by letting rc to grow. However, as soon as the rate of abort increases, rc remains close to LB, thus enabling resources to be saved. On the contrary, if resources are available, LB is high and the optimistic nature of the system is emphasized.

4. Performance evaluation

4.1. Simulation environment. We measured the performance of the described protocol by means of simulation techniques; the adopted simulation package reproduces in details a single-hop network architecture up to the transport layer.

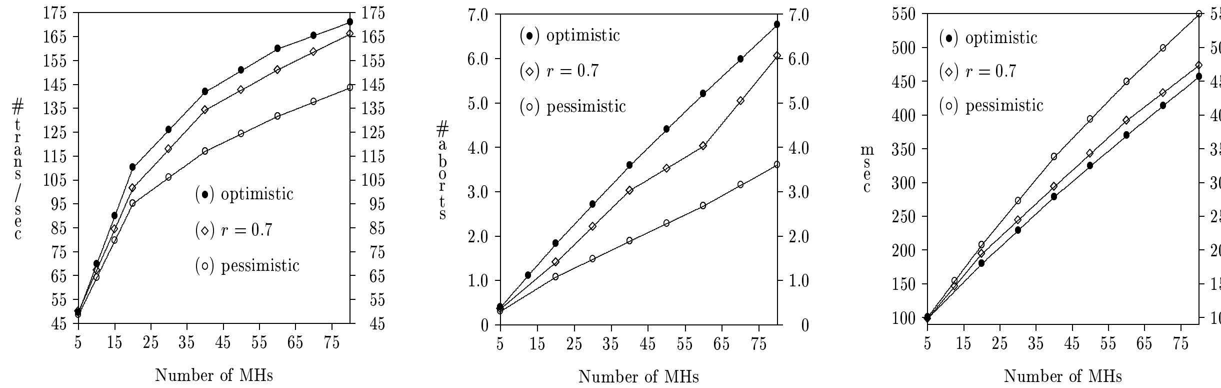
The simulated network consists of a number G of MHs and of 5 MSSs; we have evaluated the performance for different values of G. We assume that the wired network topology is completely connected. Initially the MHs are uniformly distributed in the five cells. They randomly move at a predefined speed, so that each MH enters a new cell about every second. We consider a wired network bandwidth of 10 Mbps and a wireless network bandwidth of 2 Mbps. These parameters describe a wired LAN with wireless LANs attached to the MSSs. At the transport layer we adopted the reliable multicast protocol described in [**2, 3**]. It guarantees that packets are delivered to MHs in spite of their current locality.

Each MH has a backlog of transactions and queries; a new operation on the database is generated as soon as the previous one has terminated. All the generated operations are processed in disconnected mode; we assume that during a disconnection only one transaction can be processed. After having received all the requested data items a MH disconnects. The duration of a disconnection is modeled with a random variable having a negative exponential distribution. At the end of the disconnection, the MH reconnects to a randomly chosen MSS. By varying the amount of transactions that access the same set of data items we can obtain different access patterns. For instance, a condition in which transactions have a high level of competition over a set of data items can be modeled by forcing 80% of the transactions to work on 20% of the data items. We assume that the database contains 100 data items, and that each query or transaction accesses at most 10 of them. According to the selected access pattern, 70% of the data items being accessed by a transaction are read, while 30% are written. The simulation tick is 1 msec. We assume that the packet size is 3 Kbytes; hence, it takes about 1.5 msec. to traverse a wireless link and about 0.3 msec. to traverse a wired link. The total simulation time was 50000 ticks, that corresponds to 50 sec. of simulated time.

4.2. Simulation results. Simulations have been utilized to obtain preliminary indications about the capability of the ACP to identify the trade-off between the need of saving system resources and performance. To this purpose, performance indexes have been observed under different access patterns and system conditions. We have measured:

1. the throughput, that is, the number of transactions processed by the system in the time unit;
2. the latency time, that is, the elapsed time between the start of a disconnected transaction and its global commit;
3. the number of aborts for transaction, which is expected to grow with the increase in the concurrency degree;
4. the resource utilization. To this purpose, we measured the number of exchanged messages over the wireless links and the time a MH spends (i) to process a transaction, (ii) to communicate with its RM, (iii) to wait for RM replies. The number of messages exchanged between MH and RM depends on the protocol and on the amount of aborts. If this number is low, it is possible to save the channel bandwidth and the energy. Furthermore, if the protocol behaves pessimistically, MHs wait a longer time for the RM replies, thus allowing MHs to enter a *doze* mode.

Figure 8 shows the behaviors of ACP in a general condition. In fact, we assume that only 30% of the MHs report bandwidth or battery problems and, for this reason, LB can be set and maintained quite high. However, by assuming that 80% of the transactions randomly access 20% of the data items, the high number of aborts can become the cause of resource problems. It is interesting to analyze how ACP behaves under these conditions by dynamically adapting rc and LB variables. Figure 8 shows the throughput (a), the number of aborts (b) and the



(a) Throughput (b) Number of aborts (c) Latency time

FIGURE 8. Performance results of ACP with 80% of the transactions accessing 20% of the data items.

latency time (c) of ACP compared with the optimistic and pessimistic[5] approaches. In Figure 9, the composition of the MH time utilization is reported. Figure 8 (a) shows that, under the considered conditions, ACP behaves closely to the optimistic approach. This behavior, which is confirmed by the results in Figure 8(b) and 8(c), depends on the values of LB which are maintained fairly high. It is clear that, by adopting different heuristics for LB, it is possible to obtain a protocol behavior which is more aggressive or more conservative according to the need of privileging performance or resource saving. The results in Figure 8 also indicate the good scaling capabilities of ACP. However, the reported results highlight a tendency of the protocol to approximate the behaviour of the optimistic approach with the increase in the number of MHs in the system. This event could not be acceptable in large systems where LB would grow too much, thus generating an increase of aborts and the consequent reduction of the benefits in terms of resource saving (see, for instance, in Figure 9 the computing-time bar for 80 MHs[6]). This tendency of LB to grow can be compensated, for instance, by acting on the variable m, which is used to determine LB.

Figure 9 shows that ACP succeeds in achieving energy saving with respect to the optimistic approach and, at the same time, it allows a higher number of concurrent transactions than the pessimistic approach. When the number of abort increases, under the optimistic approach, MHs have to reload the updated items and re-execute their transactions. By contrast, under ACP, MHs wait to obtain the required data items. Hence, the use of ACP lets MHs spend less time in computing and exchanging messages.

The throughput is influenced by the different settings of LB. To analyze these effects, we have considered two different system conditions (Figure 10). In the first,

[5] We have actually simulated a nearly pessimistic approach in which each RM grants up to 1 copy per data item, so that at most 5 copies are concurrently cached in the considered system (against 1 copy of a pure pessimistic approach). This is the consequence of the fact that we decide not to include, in the current version of our algorithm, any agreement protocol amongst RMs to decide which one of them can release the copy.

[6] These measurements are also affected by the discrete growing of LB, whose values are approximated to the lower or the higher integer digit.

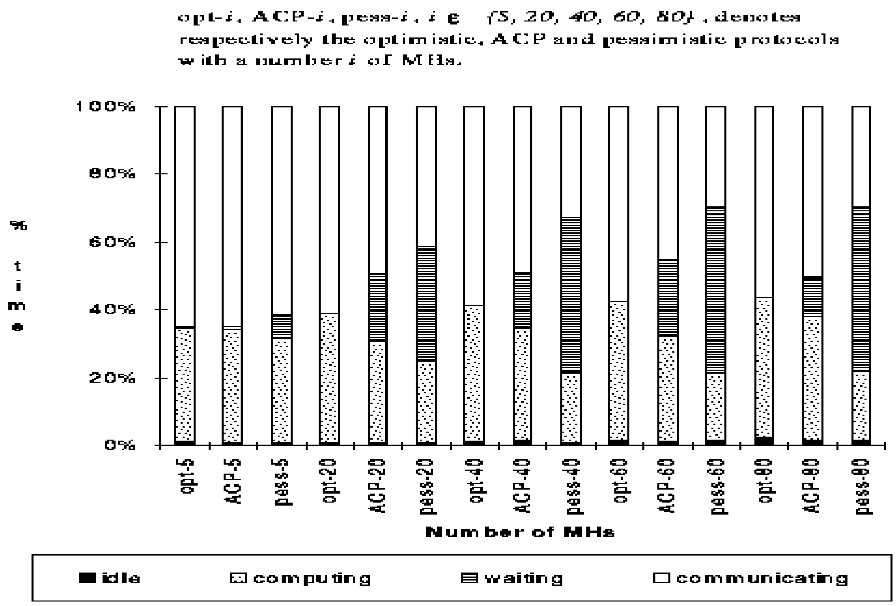


FIGURE 9. MH time utilization.

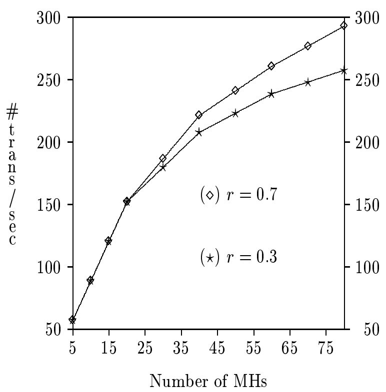


FIGURE 10. Throughput for different values of LB.

70% of the MHs are supposed to report resource problems; in the second, only 30% of MHs has resource problems. 60% of the transactions access 40% of data items; this setting has been chosen to reduce the interference of aborts in the experiments. Figure 10 shows that the throughput is nearly the same for systems of few MHs because, under these conditions, LB assumes nearly the same values. With the growing of the group membership, LB assumes different values and the throughput adapts to the different system conditions. This indicates that, by modulating LB, it is possible to properly determine the optimistic or pessimistic behaviors of ACP.

Unlike the parameter LB, rc has been introduced to dynamically adapt the number of concurrent copies of a data item according to changing access patterns.

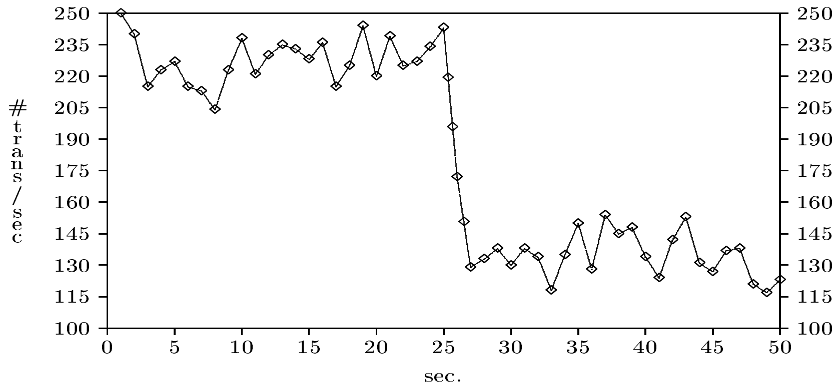


FIGURE 11. Throughput over time for changing access pattern.

To analyze this capability, in Figure 11, we report the throughput over the simulation time for a system of 40 MHs and LB set according to 30% of reported resource problems. The simulations ran a sequence of two different access patterns: in the first 25 seconds of simulated time, 60% of the transactions randomly access 40% of data items; in the second 25 seconds of simulated time, the locality of the accesses changes and 80% of the transactions access 20% of data items. Figure 11 clearly indicates how the throughput fluctuates according to the dynamics of rc and how quickly ACP adapts rc to changing access patterns. ACP requires less than 2.5 seconds to adapt to the new locality.

The combined analysis of the results reported in Figure 10 and 11 is quite encouraging and, although different heuristics can be envisaged, indicates that the use of both the parameters rc and LB is instrumental in making the ACP protocol dynamically adapting to changing access patterns and resource situations. Note that the use of only one of those parameters is not sufficient for such purpose. In fact, from the results in Figure 8 and 10, it is clear how the throughput can be affected by LB to reflect the status of the resources. rc alone would not work properly. Finally, we observe that ACP can approximate the performance of an optimistic approach, but it never behaves as a pure optimistic approach. This behaviour is due to a specific design choice; in fact, measurements indicate that an excessive amount of aborts can be the source of a bad resource utilization even when no apparent resource problem is reported.

5. Related work

Several areas are related with our work. First of all, concurrency control algorithms have been widely investigated in the area of both centralized and distributed databases. Despite, however, the large number of approaches proposed, most actual systems use locking-based mechanisms. In general, such mechanisms are not adaptive, even though some systems provide lock escalation and de-escalation techniques to dynamically convert fine-grained locks into coarse-grained locks and viceversa [**12**]. However, the concurrency control policy is always pessimistic and based on locking data before using them. A notable exception is represented by the GemStone object-oriented database system [**8**] providing two different concurrency control policies; a pessimistic one, based on locking, and an optimistic one, under which data are cached at the client and when the client tries to commit a validation test is performed. However, the choice between those two policies is left to the applications. The system makes no automatic choice neither provides any adaptive behavior, as in our approach.

Concurrency control algorithms devised specifically for mobile computing systems have also been developed. In particular, the approach proposed in [**15, 19**] to support disconnected operations in the framework of the Coda file system uses an optimistic approach, as such it can be supported as a special case of our approach. Similarly, the approach proposed in [**6**] uses a pessimistic approach and thus can be supported as a special case of our approach. A relevant shortcoming of such an approach is that it does not tolerate sudden disconnections. Both approaches differ from ours in that they only support a single policy (either optimistic or pessimistic) and do not provide any adaptive behavior.

Another relevant body of research deals with dynamic data replication approaches in distributed database systems [**24, 4**]. Those approaches develop algorithms that are able to dynamically change the data replication according to the read-write patterns of data access operations, so to optimize performance. Even though such approaches may appear relevant to mobile computing systems, they are not applicable to the problem of deciding the optimal of copies that can be cached at MHs. Those approaches assume that an host can get a missing data item from another host, when such item is not available locally. Cost functions are then formulated accordingly; in particular, the fact that an host has a certain data item may impact the access costs by other hosts. In a mobile system, the fact that a MH has a copy of a data item does not impact the access costs for the other MHs, since these MHs have anyhow to require the data item from the MSS. Moreover, those approaches seek to minimize the communication costs only, as all algorithms designed for distributed data management over fixed networks. Further, even though those approaches dynamically replicate data, they still use a single concurrency control protocol, usually based on the 2-phase locking approach or quorum approaches. As such, they do not support adaptive concurrency control protocols. It is, however, important to notice that such adaptive data replication protocols could be used to control primary data replication, that is, data replication at RMs. However, such replication is orthogonal to cache replication for MHs.

The problem of copy replication on MHs has been briefly addressed in [**13**], as a special case of a more general data replication algorithm. The approach proposed is based on an optimistic policy. Thus the same data can be freely replicated at the various MHs. Whenever a MH performs a local modification, it notifies the RM, which then is in charge of invalidating the copies that all other MHs may have. Such an approach has the serious shortcoming that in the presence of high conflict rates, the various MHs will have to frequently re-load data, which involves high energy expenditure as well high network usage. In this respect, our approach is more effective because it reduces the number of MHs that can get a data item if there is a high probability of conflict over such data item.

Finally, a recent approach has been proposed supporting adaptive data broadcast over mobile computing systems [**10**]. The goal of this approach is to develop algorithms able to dynamically determine which information have to be broadcast and how often according the request patterns of mobile users. Even though the goal of this work is very different from ours, still it is important to point out that in this approach, as in ours, both response time and energy expenditure are a main concern.

6. Concluding remarks

In the transactional systems of the near future, users will use mobile hosts to access data items located at both fixed and mobile stations. Mobile hosts alternate connected and disconnected work and are free to arbitrarily move across disomogeneous wireless cells. Mobility is changing the focus of transaction management protocols from the need to optimize response time only to a new one where response time and energy/bandwidth expenditure must be both optimized and where disconnected work is performed on cached copies of the data items.

This paper considers the problem of transaction management in such an environment and proposes an adaptive concurrency control protocol that dynamically modulates the number of data copies that can be concurrently cached by mobile hosts. The introduction of adaptive mechanisms is required to address the novel problem issued by mobility: to achieve optimal performance and, at the same time, to save the resources of the network and the mobile hosts. As the initial simulation runs indicate, this is a promising approach which, however, is open to further analysis in several directions such as the definition of new heuristics to dimension rc and LB, and the way to adaptively set the execution timeout.

However, when the amount of MHs scales from hundreds to thousands, the 'system-controlled' adaptation of the concurrency degree is likely to become not sufficient to solve the problem alone. Under these conditions, a set of 'application-' and 'system-controlled' mechanisms should be placed side by side to consider, for instance, application specific wishes, MH location awareness or pre-fetching techniques of data items. When the information provided by the applications can be supported by the proper system parameters, new more efficient approaches to manage mobile transactions could be devised.

References

[1] Recovery in Database Management Systems, V. Kumar, and M. Hsu eds., Prentice-Hall, Upper Saddle River (New Jersey), 1998.

[2] Acharya A., Badrinath B. R., *"A Framework for Delivering Multicast Messages in Networks with Mobile Hosts"*, ACM/Baltzer Journal of Wireless Networks, Special Issue on *Routing in Mobile Communication Networks, Vol. 1* (1996), 199–219.

[3] ———, *"Delivering Multicast Messages in Networks with Mobile Hosts"*, Proc. 13th Intl. Conf. on Distributed Computing Systems (May 1993), 292.

[4] Awerbuch B., Bartal Y., Fiat A., *"Optimally-competitive Distributed File Allocation"*, Proceedings of the 25th Annual ACM STOC (Victoria (B.C., Canada), 1993).

[5] Badrinath B. R., Acharya A., Imielinski T., *"Impact of Mobility on Distributed Computations"*, ACM Operating Systems Review, Vol. 27, No. 2 (Apr. 1993), 15–20.

[6] Barbará D., Garcia-Molina H., *"Replicated Data Management in Mobile Environments: Anything New Under the Sun ?"*, Proc. IFIP Conf. on Applications in Parallel and Distributed Computing (Caracas, Venezuela, Apr. 1994).

[7] Bertino E., Pagani E., Rossi G. P., *"Fault-Tolerance and Recovery in Mobile Computing Systems"*, **"Recovery in Database Management Systems", ch.24, Eds. V. Kumar and M. Hsu** (Prentice-Hall, 1998), 661–696.

[8] Breitl R. et al., *"The GemStone Data Management System"*, In *Object-Oriented Concepts, Databases, and Applications*, W.Kim, and F.Lochovsky eds., Addison-Wesley, Reading (Massachusetts), 283–308.

[9] Ceri S., Pelagatti G., Distributed Databases: Principles and Systems, McGraw-Hill, 1984.

[10] Datta A., VanderMeer D., Celik A., Kumar V., *"Broadcast Protocols to Support Efficient Retrieval from Databases by Mobile Users"*, ACM Transactions on Database Systems (1998).
[11] Dunham M.H., Helal A., *"Mobile Computing and Databases: Anything New?"*, ACM SIGMOD Record, Vol. 24 (Dec. 1995), 5–9.
[12] Gray J., Reuter A., Transaction Processing: Concepts and Techniques, Morgan-Kaufmann Publishers, San Mateo (California), 1993.
[13] Huang Y., Wolfson O., *"Object Allocation in Distributed Databases and Mobile Computers"*, Proc. 10th Intl. Conf. on Data Engineering (Houston, TX, Feb. 1994), 20–29.
[14] Imielinski T., Badrinath B. R., *"Mobile Wireless Computing: Challenges in Data Management"*, Communications of the ACM, Vol. 37, No. 10 (1994), 18.
[15] Kistler J. J., Satyanarayanan M., *"Disconnected Operation in the Coda File System"*, ACM Trans. on Computer Systems, Vol. 10, No. 1 (Feb. 1992), 3–25.
[16] Kleinrock L., *"Nomadic Computing - An Opportunity"*, ACM Computing Communication Review, Vol. 25, No. 1 (Jan. 1995), 36–40.
[17] Knapp E., *"Deadlock Detection in Distributed Databases"*, ACM Computing Surveys, Vol. 19, No. 4 (Dec. 1987), 303–328.
[18] Kung H. T., Robinson J. T., *"On Optimistic Methods for Concurrency Control"*, ACM Trans. on Database Systems, Vol. 6, No. 2 (Jun. 1981), 213–226.
[19] Lu Q., Satyanarayanan M., *"Isolation-Only Transactions for Mobile Computing"*, ACM Operating Systems Review, Vol. 28, No. 2 (Apr. 1994), 81–87.
[20] Nixon P., Cahill V., eds., *"Technologies for a Distributed Society"*, IEEE Internet Computing, Vol. 2, No. 1 (1998).
[21] Ozsu T., Valduriez P., Principles of Distributed Database Systems, Prentice-Hall, 1991.
[22] Perkins C., *"Mobile Network Computing through Mobile-IP"*, IEEE Internet Computing, Vol. 2, No. 1 (1998).
[23] Pitoura E., Bhargava B., *"Maintaining Consistency of Data in Mobile Distributed Environments"*, Proc. 15th Intl. Conf. on Distributed Computing Systems (Vancouver, British Columbia, Canada, Jun. 1995), 404–413.
[24] Wolfson O., S. Jajodia, *"An Adaptive Data Replication Algorithm"*, ACM Transactions on Database Systems, Vol. 22, No. 2 (Jun. 1997), 255–314.

Computer Science Dept., Università degli Studi di Milano, via Comelico, 39. I-20135 Milano, Italy
E-mail address: bertino@dsi.unimi.it

Computer Science Dept., Università degli Studi di Milano, via Comelico, 39. I-20135 Milano, Italy
E-mail address: pagae@dsi.unimi.it

Computer Science Dept., Università degli Studi di Milano, via Comelico, 39. I-20135 Milano, Italy
E-mail address: rossi@dsi.unimi.it