1994

Distributed Lock Management for Mobile Transactions

Jin Jing

Omran Bukhres

Ahmed K. Elmagarmid
Purdue University, ake@cs.purdue.edu

Report Number:
94-073

DISTRIBUTED LOCK MANAGEMENT FOR MOBILE TRANSACTIONS

Jin Jing
Omran Bukhres
Ahmed Elmagarmid

CSD-TR-94-073
October 1994

To appear in the 15th International Conference on Distributed Computing Systems, Vancouver, BC, Canada, June 1995

Distributed Lock Management for Mobile Transactions

Jin Jing Omran Bukhres Ahmed Elmagarmid
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

Abstract

In a mobile computing environment, a user carrying a portable computer can execute a *mobile transaction* by submitting the operations of the transaction to distributed data servers from different locations. The mobility of transaction hosts introduces new issues concerning the efficiency of transaction lock management in the distributed data servers. For example, even in a fully replicated database environment, read locks of a transaction may be executed at different servers, because these operations may be submitted from different locations. The distribution of read locks implies that extra messages are required to release these locks when the mobile host decides to commit the transaction. In this paper, we present a new lock management scheme which allows a read unlock for an item to be executed at any copy site of that item; the site may be different from the copy site on which the read lock is set. The scheme, therefore, utilizes the replicated copies of data items to reduce the message costs incurred by the mobility of the transaction host. We demonstrate this idea in an optimistic locking algorithm called O2PL-MT. Like its counterpart algorithm O2PL, presented in [4] for a conventional distributed database system, O2PL-MT grants read locks immediately on demand and defers write locks until the commitment time. However, O2PL-MT requires the transmission of fewer messages than O2PL in a mobile environment in which data items are replicated. The idea presented in this paper can also be used to improve the efficiency of other distributed lock protocols (e.g., pessimistic locking) in a mobile environment, if the number of read operations dominates that of write operations.

Index terms: distributed lock management, database transaction management, replicated data, optimistic locking, mobile computing system.

1 Introduction

Advances in wireless networking technology have engendered a new computing paradigm, called *mobile computing*, in which users carrying portable devices have access to a shared infrastructure independent of their physical location.

Following the concepts and terms introduced in [5, 3, 2], a mobile computing environment consists of two distinct sets of entities: *mobile hosts* and *fixed hosts*. Some of the fixed hosts, called *Mobile Support Stations*

(MSSs), are augmented with a wireless interface to communicate with mobile hosts, which are located within a radio coverage area called a *cell.* A mobile host can move within a cell or between two cells while retaining its network connections.

The mobile computing paradigm introduces new technical issues in the area of database systems [5, 1]. For example, techniques for traditional distributed database management have been based on the assumption that the location of and connections among hosts in the distributed system do not change. However, in mobile computing, these assumptions are no longer valid. Mobility of hosts engenders a new kind of locality that migrates as hosts move. As a consequence, existing solutions for traditional distributed database management may not be readily applicable to the mobile computing environment. In particular, migrating localities may introduce extra communication costs in fixed networks.

Consider the impact of the mobility of transaction hosts on the management of read locking/unlocking in a read-one write-all concurrency control protocol. Suppose that a mobile host always issues read operations to the data server in its MSS and the server will always execute the read lock at the local copy (if it exists). Over time, read locks for a transaction may be executed in different data server sites due to the migration of the host of the transaction. The distribution of read locks requires additional message transmissions among these data servers over fixed networks for the execution of read unlock operations.

In this paper, we present a new lock management scheme which allows a read unlock for an item to be executed at any copy site of that item; the site may be different from the copy site on which the read lock is set. Our proposed scheme, therefore, utilizes the replicated copies of data items to reduce the message costs incurred by the mobility of the transaction host. We demonstrate this idea in an optimistic locking algorithm called O2PL-MT. Like its counterpart algorithm O2PL, which was presented in [4] for a conventional distributed database system, O2PL-MT grants read locks immediately on demand and defers write locks until commitment time. However, in a mobile environment where data items are replicated, O2PL-MT requires fewer messages than O2PL. The method can also be used to improve the efficiency of other distributed lock protocols (e.g., pessimistic locking) in a mobile environment where read operations outnumber write operations.

In a mobile computing system, the mobility factor is of the utmost importance in the design of a distributed algorithm. Because the physical distance between two points does not necessarily reflect the network distance, the communication path can grow disproportionately to actual movement. For example, a small movement which crosses network administrative boundaries can result in a much longer path. In a longer network path, communication traverses more intermediaries and consumes more network capacity. This mobility of hosts means that even a short transaction may involve a long communication transmission.

Some of the problems involved in supporting transaction services in a mobile environment have been identified recently in [5, 1, 8]. A prototype of transaction service for mobile hosts is currently being implemented on the Code file system [6, 8]. This prototype uses the *optimistic concurrency control* method in [7] to enforce the serializable execution of transactions submitted from mobile hosts. The optimistic concurrency control method is generally suitable for applications of low data contention.

The method presented in this paper assumes that read locks are executed immediately when read operations are performed in data servers. The transactions are thus guaranteed to fetch consistent copies of data items during their execution. This advantage is somewhat diluted by the fact that data items may be subject to lengthy blocks during periods of disconnection by the mobile hosts. This problem can be addressed through a timeout mechanism which allows data servers to unilaterally abort transactions whose hosts have been disconnected for a designated timeout period. The mechanism may accept user-specified timeout parameters so that mobile hosts can effectively support user applications during periods of disconnection.

The remainder of this paper is organized as follows. Section 2 introduces the system model and relevant terminology. In Section 3, we describe an O2PL-MT algorithm for distributed lock management that offers low message cost. Section 4 presents an analytical model of message cost which is used in Section 5 to compare the O2PL-MT and O2PL algorithms. Concluding remarks and directions for future work are offered in Section 6.

2 The Mobile Transaction Model

Figure 1 presents a general mobile database system model similar to those described in [5, 3, 2] for mobile computing systems. In this model, both a database *server* and a database are attached to each fixed host. A database server is to support basic transaction operations such as read, write, prepare, commit, and abort.

Each MSS has a *coordinator* which receives transaction operations from mobile hosts and monitors their execution in database servers within the fixed networks. Transaction operations are submitted by a mobile host to the coordinator in its MSS, which in turn sends them to the distributed database servers within the fixed networks for execution. For example, the coordinator will send a read operation to a local server if the copy to be read is in the local site. Similarly, for a commit operation, the coordinator monitors the execution of 2PC protocol over all the servers involved in the execution of the transaction.

A mobile host may submit transactions in one of two ways:

1. An entire transaction may be submitted in a single request message; the whole transaction thus becomes one submission unit. The mobile host also delivers execution control to its coordinator and awaits the return of the results of the transaction execution.

2. In contrast, the operations of a transaction may be submitted in multiple request messages. A submission unit thus consists of one operation (e.g., read) or a group of operations; the mobile host interactively submits the operations of a transaction to its coordinator. A subsequent operation can be submitted only after those previous have been executed and the results returned from the coordinator.

While the first approach involves a single coordinator for all the operations of a transaction, the second approach may involve multiple coordinators because of the mobility of the host. For example, a mobile host may move into a new cell after it obtains the results of previously submitted operations. In the new cell, it will submit the remainder of the transaction operations to the coordinator in the appropriate new MSS. The first approach is described in [9] and related issues regarding the interface between the mobile host and the coordinator are discussed. Our proposed model employs the second approach to transaction submissions. This approach supports the interactive execution of transactions and therefore offers increased flexibility in

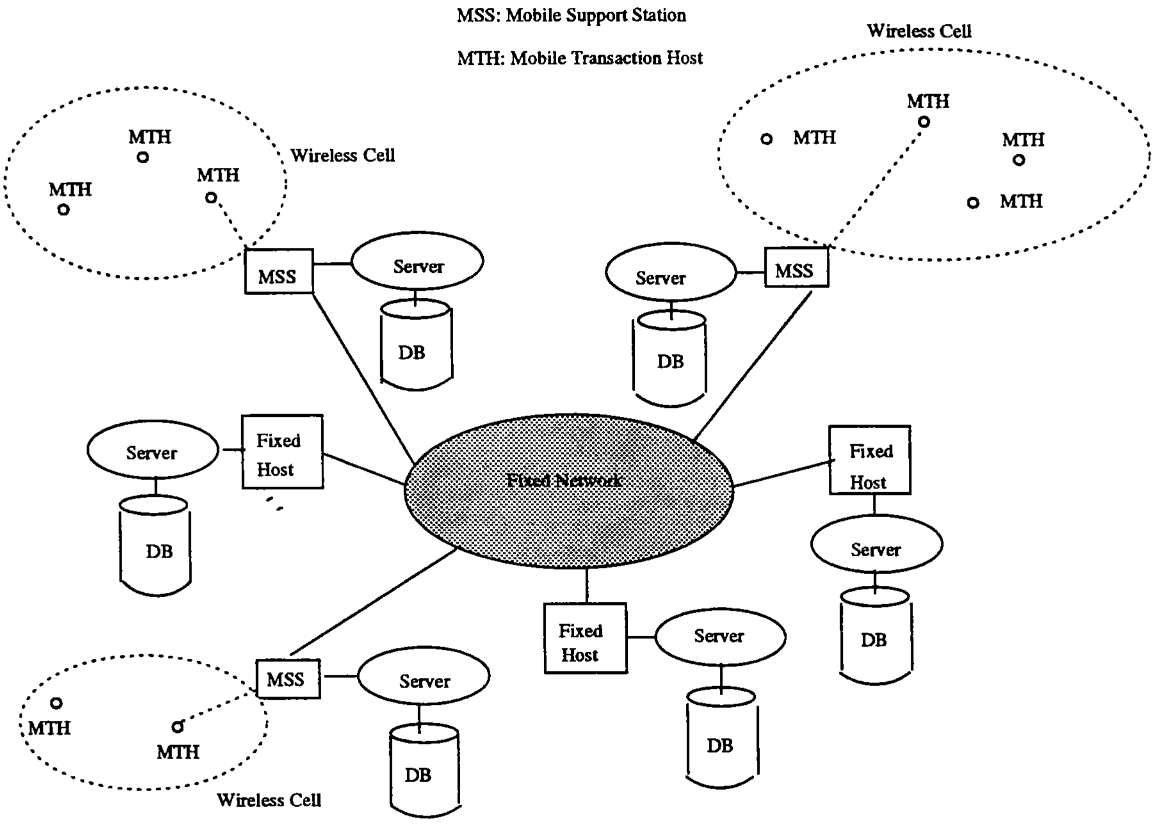


Figure 1: Mobile Database System Model

transaction computations. In this paper, we assume that a mobile host can move away from its current cell only after it has received results for all operations submitted from that cell. In practice, however, a mobile host may move at any time [2]. It may move away from its current cell after it submitted an operation and before receiving a reply from the coordinator. In this case, additional procedures are needed to locate the mobile host and convey to it the results of submitted operations. For the sake of simplicity, we will ignore this case in the rest of this paper. However, we believe that the discussion and solutions presented below will remain applicable with the incorporation of these procedures.

We also assume that only one transaction may be initialized by a mobile host at any time. That is, a mobile host can initialize a transaction only after the previous transaction has finished. The transaction submitted from the mobile host is termed a *mobile transaction* and the host is called a *mobile transaction host*. A mobile transaction consists of a set of read and write operations which are encapsulated by a $BEGIN\_TRANSACTION$ statement and an $END\_TRANSACTION$ statement.

3 An O2PL-MT Algorithm For Mobile Transactions

3.1 Motivation

In this subsection, we will provide an example to illustrate the impact of transaction host mobility on the efficiency of an optimistic two phase lock algorithm. The algorithm used in the example, O2PL, was presented in [4].

The O2PL algorithm uses an "optimistic" read-one write-all concurrency control approach. A read lock must be obtained immediately from the local or nearest copy site for each read operation; write locks for replicated copies are deferred until the beginning of the commit phase is reached. The basic idea underlying O2PL is to set locks immediately within a site (it is possible if data items are replicated), where doing so is cheap, while taking a more optimistic, less message-intensive approach across site boundaries [4].

In a mobile computing environment, however, the mobility of transaction hosts may increase message costs for the lock management approach used in the O2PC algorithm.

Example 3.1 *Consider a mobile database system, presented in Figure 2, where data items X and Y are replicated in sites A, B, and C; data item Z has one copy in site C. The mobile transaction host for T1 requests a Read(X) operation in site A and then moves to site B to request a Read(Y) operation in site B. After the host moves to site C, it requests Write(Z) and Commit operations. To read each item, the coordinator in each site immediately requests a read lock and read operation in the local server. At commit time, however, the coordinator in site C needs to send two commit (or unlock) messages to servers A and B for the release of read locks.*

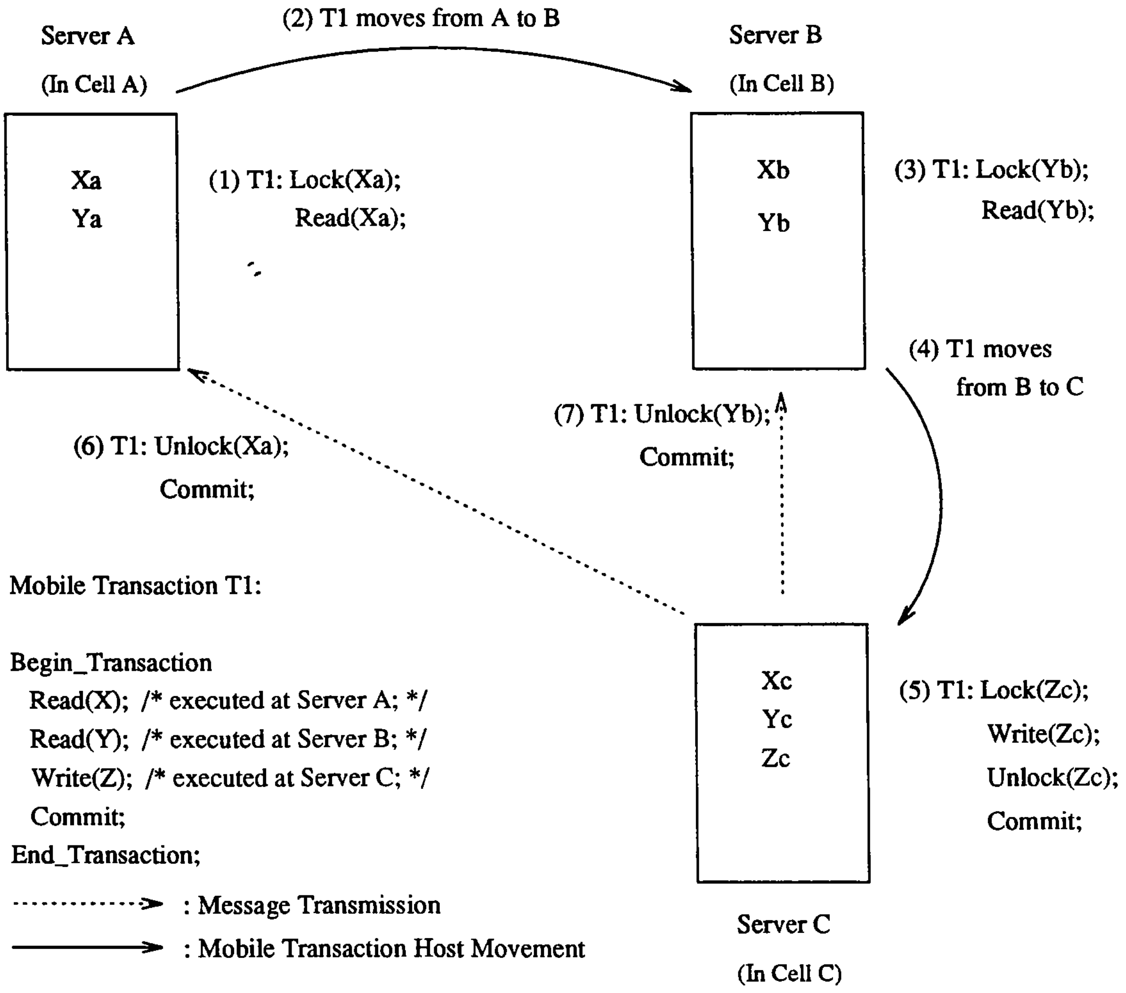
The above example shows that, with the introduction of mobile transaction hosts, the read-one write-all O2PL approach involves extra message transmissions for read operations even in a replicated database environment. In contrast, in traditional distributed database systems, the positions of transaction host and transaction coordinator are assumed to be fixed during the execution of a transaction. In this case, no extra message transmission are needed for read operations.

These additional message transmissions can obviously be avoided through an even more optimistic approach which defers read locks until commit time. Increasingly optimistic approachs, however, carry with them increasingly high transaction abort rates. This method has therefore not been pursued here.

This research therefore seeks a new algorithm for mobile transactions which requires fewer message transmissions than O2PL but retains a comparable degree of optimism. The latter stipulation means that a read lock should be obtained immediately for each read operation, while write locks are deferred until the moment of commitment.

3.2 The O2PL-MT Algorithm

In this subsection, we will describe a read-one write-all algorithm which we have called O2PL-MT (Optimistic 2 Phase Locking for Mobile Transactions). The algorithm requires fewer message transmissions than O2PL while retaining a comparable degree of optimism.



Mobile Transaction T1:

```
Begin_Transaction
  Read(X);  /* executed at Server A; */
  Read(Y);  /* executed at Server B; */
  Write(Z);  /* executed at Server C; */
  Commit;
End_Transaction;
```

Figure 2: A Mobile Transaction Example

The proposed algorithm employs a very simple method to restrict the number of extra read unlock messages. Rather than sending a read unlock to the remote copy server where the read lock is set, O2PL-MT simply allows the unlock operation to be executed at the local or nearest copy server of the item. In the case of Example 1, instead of requesting unlock(Xa) and unlock(Yb) in servers A and B, O2PL-MT will execute Unlock(Xc) and Unlock(Yc) in local server C at commit time, necessitating no message transmissions.

This method is not in itself sufficient to ensure the correctness of a read-one write-all approach. Before it can be applied to the reduction of extra unlock messages, the following issues must be explored.

- Read locks must be guaranteed to remain in effect at remote sites when the coordinator decides to release the locks and commit a transaction locally.
- An update transaction must be able to determine that the item to be updated has not been locked by other transactions for reading if the read lock and unlock are executed at different sites. Such a check should be of low message costs too.
- A mechanism must be provided to remove the pending read locks at remote sites at the proper time, if the continuation of such locks will affect the execution of other transactions.

An exploration of the first of these issues begins with the observation that, in 2PL, a lock should be held until no new lock request is needed. Without this stipulation, the serializability of transaction execution cannot be ensured. While this can be enforced by holding all locks until the commit time of a transaction, this condition cannot be confirmed for read locks set at remote sites without the use of extra message transmissions. However, we can still ensure the correct semantics of read locks if the copy version of the item to be unlocked is unchanged from its state at lock time. An unchanged copy version number implies that the item has not been updated by other transactions since it was locked. At the point of commitment, we can compare the copy version number of an item at the local site with that obtained from another copy site at lock time. If the two numbers match, we can conclude that the serialization order of transactions is the same as that in a conventional lock/unlock scheme in which the read unlock is always executed at the same copy site as the lock is set.

To address the second issue listed above, we first review how traditional read-one write-all algorithms such as O2PL request a write lock for an item from all copy sites. In these algorithms, a write lock request (and the new updated value, if any) is sent to all copy sites. Each copy site then determine whether the write lock can be granted. If no pending read lock is set on the copy, the copy site grants the request immediately and sends a reply message to the transaction coordinator (or host). Otherwise, the request is blocked at the copy site. The transaction coordinator collects reply messages from these sites. Once all sites reply to the request, the transaction coordinator concludes that there is no pending read lock at any copy site and the write lock has been granted. Thus, only one round of message exchanges is needed between the coordinator and any copy site for a write lock request.

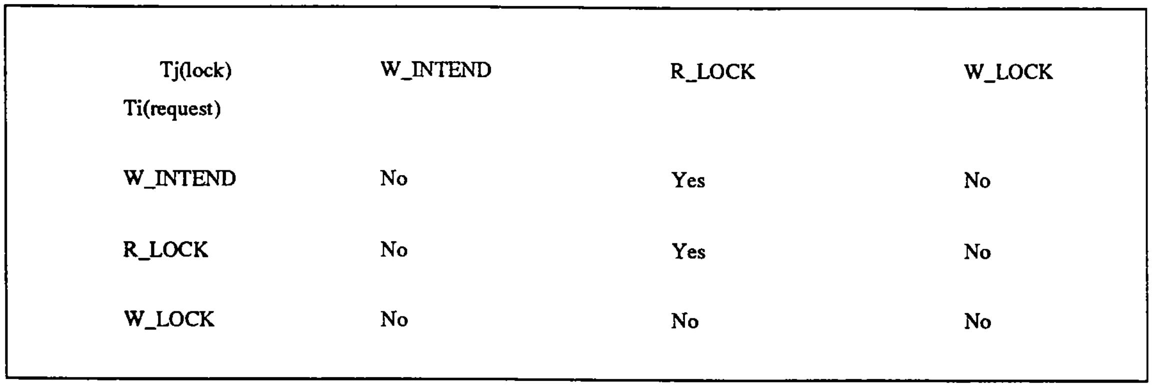
However, if the locking and the unlocking of a read operation are executed at different sites (or servers), a single message exchange between the coordinator and a copy site is insufficient. In fact, if a read lock has been set at a copy site, the write lock request can neither be granted immediately nor be blocked at that site. This complication arises because the copy site cannot know whether the read lock has been released and the read unlock has been executed at a different site. The copy site must first obtain information from the coordinator about the status of the copy before it can release the read lock and grant the write request. To gather this information, the coordinator must collect unlock information from all the copy sites involved in the first-round message exchange. If any site indicates that the unlock has been executed, the coordinator then can send a message to first copy site, allowing the write lock request to be granted.

Thus, to allow the distribution of read lock/unlock operations in different sites, two rounds of message exchanges are required. These exchanges permit the transaction coordinator to decide whether a write lock request can be granted by the copy sites. Because O2PL permits all write locks to be defered until commit time, these message exchanges can actually be merged into the regular 2PC protocol. That is, in the first phase of 2PC, the coordinator collects the read lock/unlock information from all copy sites. The coordinator cannot enter into the second phase of 2PC until all sites vote "yes" and each read lock is matched by one read unlock on the item to be updated. In the second phase, the write locks are set and the updates are enforced.

In the above method, although write locks are not granted during the first phase, all copy sites must still perform all other functions required by the standard 2PC (e.g., the checking of integrity constraints for each updated data item). As in the standard 2PC, these functions should be performed in the temporary workspace of each copy site. These actions prepare servers to enforce write operations (including the granting of write locks) at all copy sites during the second phase of 2PC.

To guarantee the enforced write operations, a new lock mode, called "write intend" or *W_INTEND*, must be used to lock the copy at all sites before these sites reply to the write request in the first-round message exchange. A requested *W_INTEND* is compatible with a granted *R_LOCK* (read lock) but not with a granted *W_INTEND* or *W_LOCK* (write lock). A granted *W_INTEND* is, however, not compatible with a requested *R_LOCK* or a requested *W_INTEND* or *W_LOCK*. When a read request is granted at a copy site, *R_LOCK* is set on the copy. When the first phase of a write request arrives at a copy site, a *W_INTEND* is set on the copy if no other *W_INTEND* or *W_LOCK* applies to that copy. Otherwise, the first phase of the write request is blocked at the site. This scenario implies that two different writes are requesting the lock. If there is a *R_LOCK* on the copy, the first phase of the write request will immediately set the *W_INTEND* on the copy. After the coordinator decides to commit the transaction in the second phase, the *W_INTEND* is upgraded to *W_LOCK* and the update is enforced at the copy. Thus, the granted *W_INTEND* will prevent any new requested *R_LOCK* or *W_LOCK* from being granted in the copy. Otherwise, the decision of the coordinator to grant the write locks will not be valid. The lock compatibility matrix appears in Figure 3.

Finally, let us consider the issue raised by the third point enumerated above. After read unlocks are executed at a copy site other than the read lock copy sites, read locks may be pending at these sites. The pending read locks can be removed when another transaction prepares to update the copy. As in the two phase commit/write lock protocol described above, a transaction can obtain a write lock and update an item only if it finds that each read lock at a copy site is matched by a read unlock at some site. Therefore, it is safe to remove the pending read lock before an update transaction can obtain the write lock on that item.



Tj(lock)
Ti(request)
	W_INTEND	R_LOCK	W_LOCK

W_INTEND	No	Yes	No
R_LOCK	No	Yes	No
W_LOCK	No	No	No

Figure 3: Lock Conflict Matrix

The pending period begins at the commitment of the read transaction and ends at the first write lock by another transaction. It is obvious that the pending read lock does not block either other read lock requests or write lock requests. Thus, there is no blocking drawback to having a pending read lock on the copy of an item until the copy is updated.

Algorithm Summary: We shall now summarize the O2PL-MT algorithm (a detailed description is available in Appendix A).

A mobile transaction host sends each read request to the coordinator in its MSS and waits for the returned copy to arrive from the coordinator. The host stores the updated values in a local workspace and defers write lock requests until the commit phase is reached. The transaction coordinator forwards read operations to the local or nearest copy server where the read locks will be set. When the commit operation is requested, the coordinator requests write locks in all updated copy servers and executes read unlock operations in the local or nearest copy servers. Due to the mobility of the host during the execution of the transaction, the read lock and the read unlock for an item may be executed on two different copy sites. A unlock operation is valid only if the version of the copy to be unlocked is the same as that at the lock time. Write locks on items are obtained through the two phase commit procedure. That is, in the first phase, the coordinator collects the read lock/unlock information from all copy servers. The coordinator can not enter into the second phase until all servers vote "yes" and each read lock is matched by one read unlock on the item to be updated. If there is a read lock that has not been matched by a read unlock on an item to be updated, the coordinator is required to wait for the unlock operation from any of the copy servers of the item. In the second phase, the write locks are set and the updates are enforced.

Table 1: Model Parameters

Parameters	Description	Values/Expressions
λ_u	Update mobile transaction arrival rate	2 update transaction/sec
λ_r	Read-only mobile transaction arrival rate	20 read-only transaction/sec
n_{op}	Average number of items accessed per trans.	10
n_u	Average number of items updated per update trans.	5
h	Probability of read hit at local site	$0 \leq h \leq 1$
p_u	Probability of each write conflicting with reads	0.2
n_c	Average number of reads conflicting with a write	2
N	Number of server sites	20
r	Average number of replicated copies per item	$1 \leq r \leq N$
N_u	Average number of sites updated by per update trans.	$r \leq N_u \leq N$
m	Probability of mobility of transaction host	$0 \leq m \leq 1$
m_0	Probability of moving away from lock server	$m[(N-1)/(N-1)]$
m_0'	Probability of moving back to lock server	$m[1/(N-1)]$
m_1	Probability of moving away from copy server	$m[(N-r)/(N-1)]$
m_1'	Probability of moving back to copy server	$m[r/(N-1)]$

4 An Analytical Model

In this section, we shall develop an analytical model to represent message costs over fixed networks for both the O2PL and O2PL-MT algorithms. This model is intended to facilitate a comparison of the magnitude of message costs between the two algorithms. In particular, we wish to demonstrate how the parameters of mobility and replication affect the message costs of two algorithms, to discover those circumstances in which O2PL-MT offers lower message costs than O2PL, and to quantify the magnitude of the cost differences.

In our model, without loss of generality, we assume that there are N data servers, with each server attached to an MSS. In fact, some servers may be attached to fixed hosts which have no wireless communication interface. Our model can be generalized to include this case by assigning each of these servers to its closest MSS. Update mobile transactions and read-only mobile transactions arrive in Poisson distributions with an average arrival rate of λ_u and λ_r, respectively. The data are randomly accessed or updated by a transaction. The average number of items accessed by update or read-only transaction is n_{op}. The average number of items updated by update transaction is n_u. Each update transaction only updates a subset of its read set; i.e., $n_u \leq n_{op}$.

The probability that a read operation hits a copy at the local server is h. The probability that a write operation conflicts with a read operation is p_u. When an update transaction writes a data item, the transaction may conflict with more than one transaction holding a read lock on the data item. We let n_c represent the average number of read operations conflicting with the write operation.

A data item may be replicated at several servers. The average number of replicated copies per item is represented as r. Let N_u be the average number of server sites where updates are performed by each update transaction. The average number will be between r and N.

We assume that, after each request has been performed by a server and acknowledged by a coordinator, the mobile transaction host may move away from the current cell (or server). The probability of mobility of each transaction host is m.

If a mobile transaction host were to randomly move into a new cell (or server), the probability that the host moves from a server with a copy of an item to a server with no copy of the item is $m[(N-r)/(N-1)]$, denoted by m_1. The probability that the host moves from a server with no copy of an item to a server with a copy of the item is $m[r/(N-1)]$, denoted by m_1'. The probability that the host moves away from a server where a lock is set on an item is $m[(N-1)/(N-1)]$, denoted by m_0. Finally, the probability that the host moves back to the server where a lock is set on an item is $m[1/(N-1)]$, denoted by m_0'.

Table 1 provides a summary of all the parameters described above.

We now derive the basic expressions that describe the message costs for both the O2PL and O2PL-MT algorithms. In these expressions, we ignore the message costs for aborted transactions and consider only the message costs among data servers over MSSs. Our analytical model will not treat the message costs between mobile hosts and MSSs.

O2PL: For each read request, if there is probability h that the copy is at the local server, then the probability that the request will be unfulfilled by the local server is $1-h$. The expected number of messages per second for read requests is:

$(\lambda_u+\lambda_r)n_{op}(1-h)$.

When the commit operation (the *END_TRANSACTION* operation) is requested, the deferred write locks and updates will be executed along with the procedure of 2PC protocol. The number of messages involved in the 2PC protocol, which is dependent upon the number of update transactions and the average number of servers updated by each transaction can be expressed as:

$4\lambda_u N_u$.

The coordinator also sends read unlock operations to all servers where read locks have been set for the commitment of both update and read-only transactions. The expected number of messages for those read operations for which locks are hit but unlocks are missing is:

$\lambda_u(n_{op}\text{-}n_u)hm_0 + \lambda_r n_{op} h m_0$,

and the expected number of messages for those read operations for which both locks and unlocks are missing is:

$\lambda_u(n_{op}\text{-}n_u)(1-h)(1-m_0') + \lambda_r n_{op}(1-h)(1-m_0')$.

In the above expressions, we do not count those reads which have been upgraded to writes by an update transaction. That is, for each update transaction, we need send read unlocks for only $(n_{op}\text{-}n_u)$ read operations.

Totally, the expected number of messages transmitted per second for the O2PL algorithm is given by the expression:

$$\begin{aligned} M_{O2PL} &= 4\lambda_u n_u \\ &\quad +(\lambda_u + \lambda_r) n_{op}(1-h) \\ &\quad +\lambda_u(n_{op} - n_u) h m_0 \\ &\quad +\lambda_r n_{op} h m_0 \\ &\quad +\lambda_u(n_{op} - n_u)(1-h)(1-m_0') \\ &\quad +\lambda_r n_{op}(1-h)(1-m_0') \\ &= 4\lambda_u N_u + (2R+U)(1-h) + Rhm \end{aligned} \quad (1)$$

where R = $\lambda_u(n_{op}$-$n_u)$ + $\lambda_r n_{op}$ and U = $\lambda_u n_u$.

O2PL-MT: As was the case with O2PL, the expected number of messages per second for read requests is: $(\lambda_u$+$\lambda_r)n_{op}(1-h)$.

However, the number of messages for the commitment of update transactions will be:

$4\lambda_u N_u + \lambda_u n_u p_u n_c$

where $\lambda_u n_u p_u n_c$ is the number of messages sent by copy sites to the coordinator for unlock operations in the first phase of the commit protocol. In O2PL-MT, the transaction coordinator collects all the replies for the prepare requests and waits for read unlock messages if a write is blocked by some unlocked reads during the first phase of the commit protocol.

Instead of sending unlocks to the server where the read locks are set, O2PL-MT sends read unlocks to the local or nearest copy site of the items. The expected number of messages for those read operations for which locks are hit but unlocks are missing at the local copy of the items is:

$\lambda_u(n_{op}$-$n_u)hm_1 + \lambda_r n_{op} h m_1$,

and the expected number of messages for those read operations for which both locks and unlocks are missing at any copy of the items locally is:

$\lambda_u(n_{op}$-$n_u)(1-h)(1-m_1') + \lambda_r n_{op}(1-h)(1-m_1')$.

Thus, the total expected number of messages transmitted per second for the O2PL-MT algorithm is given by the expression:

$$\begin{aligned} M_{O2PL-MT} &= 4\lambda_u n_u \\ &\quad +\lambda_u n_u p_u n_c \\ &\quad +(\lambda_u + \lambda_r) n_{op}(1-h) \\ &\quad +\lambda_u(n_{op} - n_u) h m_1 \end{aligned}$$

$$
\begin{aligned}
& +\lambda_r n_{op} h m_1 \\
& +\lambda_u (n_{op} - n_u)(1-h)(1-m_1') \\
& +\lambda_r n_{op}(1-h)(1-m_1') \\
= \; & 4\lambda_u N_u + \lambda_u n_u p_u n_c + (2R+U)(1-h) + Rm(h-(r/N)) \qquad (2)
\end{aligned}
$$

where R = $\lambda_u(n_{op}$-$n_u) + \lambda_r n_{op}$ and U = $\lambda_u n_u$.

5 Results

In this section, we analyze and compare the message costs of the O2PL and O2PL-MT algorithms on the basis of the two equations developed in the last section. Our analysis and comparison are divided into three categories according to the parameter of data replication; These categories include (1) fully replicated case, (2) non-replicated case, and (3) partially replicated case.

Case I: Fully Replicated

When each item is fully replicated at every server (i.e., $r = N$), the probability of a read hit at a local site is always equal to 1 ($h = 1$) and the number of sites updated per update transaction will be N ($N_u = N$). Thus, from Equations 1 and 2, we derive the following expressions for message cost for the fully replicated case:

$$M^f_{O2PL} = 4\lambda_u N + Rm \qquad (3)$$

and

$$M^f_{O2PL-MT} = 4\lambda_u N + \lambda_u n_u p_u n_c \qquad (4)$$

Using the parameter values provided in Table 1, we have
M^f_{O2PL} = 160 + 210m and
$M^f_{O2PL-MT}$ = 164

Figure 4 shows the message cost behavior of both algorithms as the mobility parameter rises from 0 to 1. As we can see, M^f_{O2PL} (solid line) increases as m grows. $M^f_{O2PL-MT}$ (dashed line), however, is independent of the parameter m. $M^f_{O2PL-MT}$ is slightly larger than M^f_{O2PL} only for very small m when $p_u > 0$, but not greater than M^f_{O2PL} for all m when $p_u = 0$. This can be explained as follows. In the fully replicated case, O2PL-MT does not involve extra message transmissions for read operations (including read locks and read unlocks), even though the host may move during the execution of a transaction. The only message cost for O2PL-MT is for the commit operation. The first phase of the commit protocol in O2PL-MT will collect all

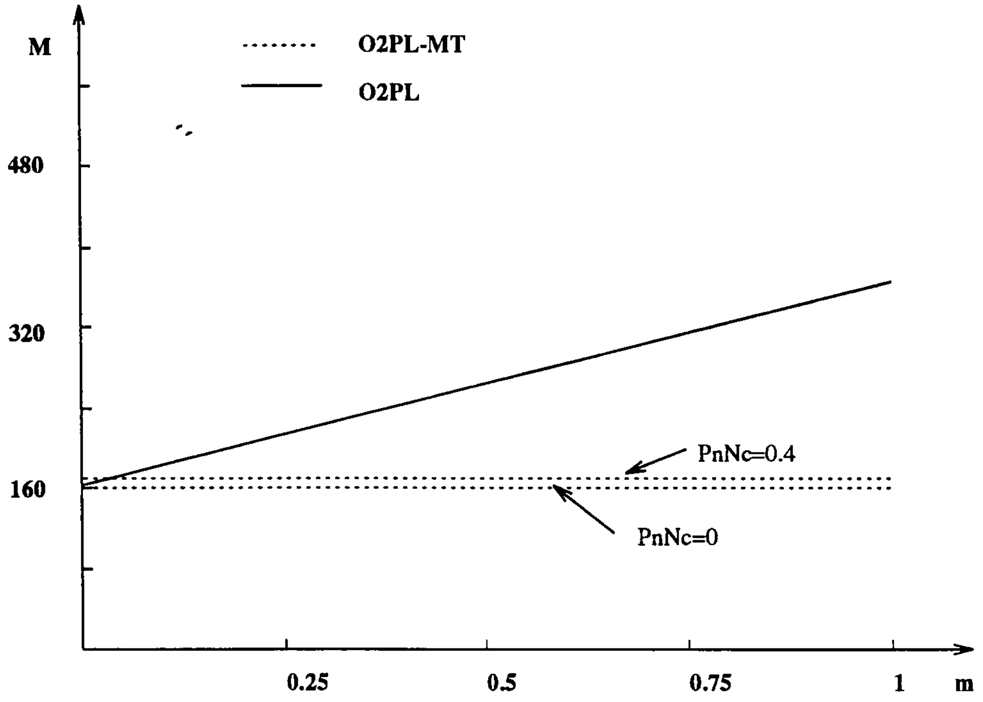


Figure 4: Results for Fully Replicated Data

unlock information from servers. This operation includes one round of message transmission between the coordinator and the servers as well as an extra transmission for each unlocked read which is in conflict with a write in the update transaction. Recall that, in O2PL-MT, after receiving replies from the servers in the first phase, the coordinator will await an additional message for the execution of a read unlock operation, if any read lock has not been matched by a read unclock. The total number of extra transmissions will be $\lambda_u n_u p_u n_c$. In contrast, the commit protocol in O2PL is a standard 2PC protocol, with exactly two rounds of message transmission. Therefore, $M^f_{O2PL-MT}$ is not greater than M^f_{O2PL} when $p_u = 0$. Furthermore, because the number of extra transmissions is small in comparison to the number required by unlock operations in O2PL due to mobility, $M^f_{O2PL-MT}$ is slightly larger than M^f_{O2PL} only for very small m.

Case II: Non-Replicated

When no item is replicated (i.e., $r = 1$), from Equations 1 and 2, we derive the following message cost expressions:

$$M^n_{O2PL} = 4\lambda_u N_u + (2R+U)(1-h) + Rhm \tag{5}$$

and

$$\begin{aligned} M^n_{O2PL-MT} &= 4\lambda_u N_u \\ &\quad + \lambda_u n_u p_u n_c \\ &\quad + (2R+U)(1-h) + Rm(h - 1/N) \\ &\approx 4\lambda_u N_u \\ &\quad + \lambda_u n_u p_u n_c \\ &\quad + (2R+U)(1-h) + Rmh(when N \gg 1) \\ &= M^n_{O2PL} + \lambda_u n_u p_u n_c \end{aligned} \tag{6}$$

When considering the scenario which corresponds a set of values: h = 0.4, $N_u = 5$ (other parameters are from Table 1), we have:

$M^n_{O2PL} =$ 84m + 298 and

$M^n_{O2PL-MT} =$ 84m + 302

Figure 5 shows the message cost behavior of both algorithms as the mobility parameter rises from 0 to 1. In the two scenarios, $M^n_{O2PL-MT}$ is very close to M^n_{O2PL} and both $M^n_{O2PL-MT}$ and M^n_{O2PL} increase at the same rate as m grows. In fact, in the non-replicated case, both O2PL and O2PL-MT should send unlock operations to remote lock sites after the host moves away from the lock site. The commit operation in O2PL-MT requires additional extra message transmissions for each conflicting unlocked read operation. The

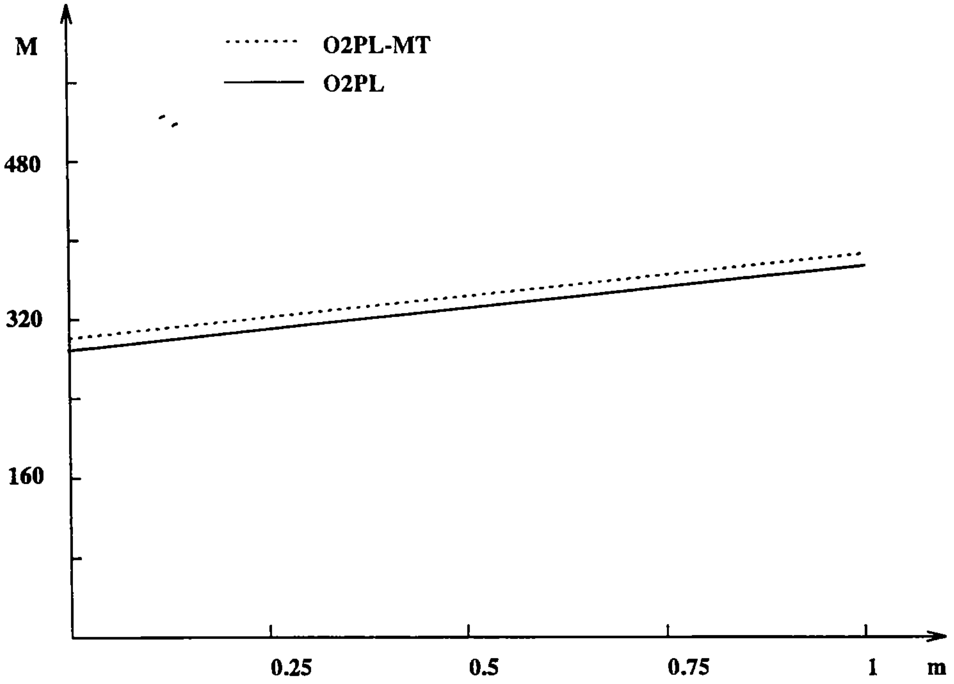


Figure 5: Results for Non-replicated Data

total number of extra message will be $\lambda_u n_u p_u n_c$. In contrast, the commit operation in O2PL involves exactly two rounds of message transmission. Thus, the difference between $M^n_{O2PL-MT}$ and M^n_{O2PL} is $\lambda_u n_u p_u n_c$ (= 4), which is independent of the parameter m.

Notice that, in the non-replicated case, an optimization can be made to eliminate the $\lambda_u n_u p_u n_c$ extra message transmission in O2PL-MT. Since, in the non-replicated case, an unlock operation must be executed at the same site as the lock operation, the O2PL-MT algorithm can adopt the approach of waiting for unlock operations in the first phase of the commit protocol. That is, a prepare request will be blocked at the site where a unlocked read is conflicting with a write request in the update transaction. Recall that, in the O2PL-MT algorithm for replicated cases, a prepare request will always be replied to, and the coordinator must await an extra message in the first phase of the commit protocol only if a read lock has not been matched by a unlock. This optimization removes any differences between the O2PL-MT and O2PL algorithms in the non-replicated case.

Case III: Partially Replicated

When items are partially replicated (i.e., $1 < r < N$), from Equations 1 and 2, we derive the following message cost expressions:

$$M^p_{O2PL} = 4\lambda_u N_u + (2R + U)(1 - h) + Rhm \qquad (7)$$

and

$$\begin{aligned} M^p_{O2PL-MT} &= 4\lambda_u N_u \\ &\quad + \lambda_u n_u p_u n_c \\ &\quad + (2R + U)(1 - h) + Rm(h - r/N) \\ &= M^p_{O2PL} + \lambda_u n_u p_u n_c - Rm(r/N) \end{aligned} \qquad (8)$$

First, consider a scenario with parameters having values of: r = 5, h = 0.5, and N_u = 10 (other parameters are from Table 1). We have:

M^p_{O2PL} = 105m + 295 and

$M^p_{O2PL-MT}$ = 105m + 295 + 4 - 52.5m = 52.5m + 299

Consider a second scenario with parameters having values of: r = 10, h = 0.8, and N_u = 15 (other parameters are from Table 1). We have:

M^p_{O2PL} = 168m + 206 and

$M^p_{O2PL-MT}$ = 168m + 206 + 4 - 105m = 63m + 210

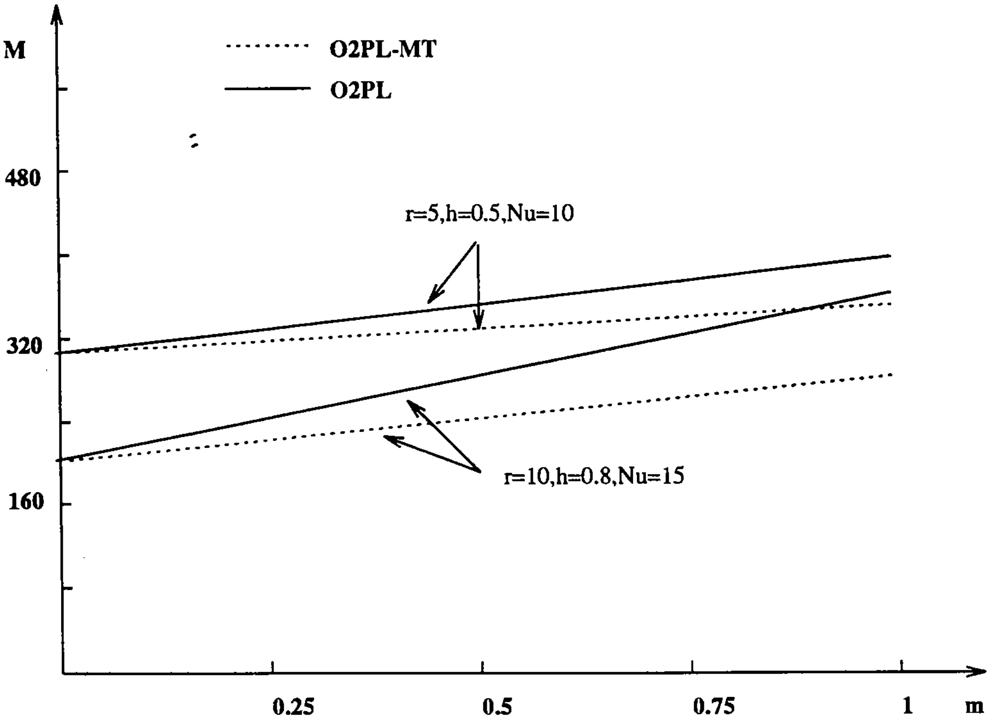


Figure 6: Results for Partially Replicated Data

Figure 6 shows the message cost behavior of both algorithms for both scenarios as m goes from 0 to 1. From Figure 6, we observe that: (1) when the replication parameter r increases, M^p_{O2PL} (or $M^p_{O2PL-MT}$) decreases; (2) for the same m, the difference between M^p_{O2PL} and $M^p_{O2PL-MT}$ increases when r increases; and (3) $M^p_{O2PL-MT}$ is smaller than M^p_{O2PL} for most values of the parameter m.

These observations can be explained as follows. First, when the replication parameter r increases, the read hit parameter h also increases. The increase in h will reduce both M^p_{O2PL} and $M^p_{O2PL-MT}$. Second, when r increases, the probability $m_1^{'}$ that the host moves back to a copy site increases. In contrast, the increase in r does not increase the probability $m_0^{'}$ that the host will move back to a lock copy server. Therefore, the increase in r will reduce the number of message transmissions for unlock operations in O2PL-MT but not in O2PL. Thus, for the same m, the difference between M^p_{O2PL} and $M^p_{O2PL-MT}$ increases when r increases. Third, once again, O2PL-MT requires fewer message transmissions for unlock operations than O2PL for most values of the parameter m, because for a replicated case (i.e., $r > 1$), $m_1^{'}$ is always larger than $m_0^{'}$. In this case, the unlock operations in O2PL-MT requires fewer message transmissions than those in O2PL.

6 Conclusions

In this paper, we have presented a new lock management scheme suitable for the mobile computing environment. The scheme allows a read unlock for an item to be executed at any copy site, regardless of whether that site is different from the copy site in which the lock is set. The scheme utilizes the presence of replicated copies of data items to reduce the read unlock message cost incurred by the mobility of transaction hosts over fixed networks.

We have demonstrated the practicality of this idea via an optimistic locking algorithm called O2PL-MT. The idea can also be used to improve the efficiency of other distributed lock protocols such as pessimistic locking in a mobile environment, if the number of read operations dominates that of write operations. This condition is required because write locks have to be executed with two round message exchanges, although read locks and read unlocks can be executed at any item copy site.

Some enhancements may be made to the scheme presented in this paper. For example, in the first phase of the commit protocol in O2PC-MT, rather than awaiting each read unlock operation from the servers, the coordinator can wait for the commit operation of a transaction which is conflicting with the update transaction. The commit operation implies that the read unlock operations have been executed. This enhancement can reduce message cost for the write lock request, when the update transaction has conflicting operations with another transaction on more than one locked item in one site.

In this paper, we have ignored an important issue for distributed lock management in mobile environment, which is the unilateral abortion of mobile transactions by servers. A data server may decide to abort a mobile transaction which is involved in a deadlock or for which its host has disconnected from servers for a long period of time. The abortion enables the server to release the data resources locked by the transaction. In both cases, the data server needs to inform other remote servers of the abortion, if these servers are also holding locks for the transaction. Such information about the lock locations (or lock distribution), however, may not be available to each data server unless it is disseminated to servers by the mobile transaction host (or coordinator) for each operation. In a traditional distributed database system, the information can easily be obtained from the transaction coordinator, which is in a fixed host. To abort a mobile transaction, therefore, extra communication costs are entailed in searching for any remote servers that have the locks set for the transaction. Notice that, when a mobile transaction host aborts its transaction, these costs do not arise, because the host issues these transaction operations and is always aware of the locations of these locks. Possible solutions to this issue may use the search strategy or the always-inform strategy presented in [2] to locate the current transaction coordinator (or the transaction proxy) and obtain the required location information. We plan to address this issue and compare various possible alternative strategies for this problem in a future paper.

References

[1] R. Alonso and H. Korth. Database issues in nomadic computing. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 388–392, 1993.

[2] B. R. Badrinath, A. Acharya, and T. Imielinski. Structuring distributed algorithms for mobile hosts. In *Proc. of the 14th International Conference on Distributed Computing Systems*, Poznan, Poland, June 1994.

[3] D. Barbara and T. Imielinksi. Sleepers and workaholics: Caching strategies for mobile environments. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 1–12, 1994.

[4] M. J. Carey and M. Livny. Conflict detection tradeoffs for replicated data. *ACM Trans. Database Syst.*, 16(4):703–746, Dec. 1991.

[5] T. Imielinski and B. R. Badrinath. Wireless mobile computing : Challenges in data management. *Communication of ACM.* (to appear).

[6] J. Kistler and M. Satyanaranyanan. Disconnected Operation in the Coda File System. *ACM Transactions on Computer Systems*, 10(1), February 1992.

[7] H. Kung and J. Robinson. On optimistic methods for concurrency control. *ACM Trans. Database Syst.*, 6(2):213–226, June 1981.

[8] Q. Lu and M. Satyanaranyanan. Isolation-only transactions for mobile computing. *ACM Operating Systems Review*, 28(3), 1994.

[9] L. Yeo and A. Zaslavsky. Submission of transactions from mobile workstations in a cooperative multidatabase processing environment. In *Proc. of the 14th International Conference on Distributed Computing Systems*, Poznan, Poland, June 1994.

Appendix A: The O2PL-MT Algorithm

At Each Transaction Host:

- For each read operation: send a read request to the coordinate server in its MSS and wait for the returned copy from the server.
- For each write operation: store the updated value in a local workspace without any remote request.
- For each *end_transaction* operation (commit or abort): send it to the coordinate server in its MSS and wait for the acknowledgement.

At Each Transaction Coordinator:

- Whenever a read operation is received, do the following: if a copy of the data item is found in the local site, then execute the read and read lock locally. Otherwise, send the read request to the nearest copy server. The request is blocked until an acknowledgment along with the copy of the item is returned.
- Whenever a commit operation is received, do the following:

  1. send a PREPARE request to copy sites where there is at least a copy to be updated. For each data item read by the transaction, if the copy sites to be updated do not contain any copy of the data item, then a PREPARE request also needs to be sent to a local or nearest copy site of the item. A PREPARE request includes all the information about the copies to be updated and read in the transaction. From these information, the copy site will receive the necessary write lock and read unlock requests.

  2. if all of these servers answer YES for the PREPARE requests, check if a write lock for each item to be updated can be granted as following:

  (a) if each read lock operation at a copy site can be matched by a unlock operation from any other copy site for every item to be updated, then send COMMIT requests to these copy sites to be updated. That is, all of the write locks have been granted and the transaction can update the items and commit. The coordinator also informs the transaction host of the commitment of the transaction.

  (b) if any lock operation at a copy site can not be matched by a unlock operation from any other copy site for one item to be updated, then wait for another YES acknowledgement message piggybacked with new unlock information from any copy site of this item and then go back to step 2.(a). If a NO is received during the waiting, then go to step 3.

  3. if any of these copy sites returns NO, then send ABORT requests only to these sites which have voted YES for updates. The coordinator also informs the transaction host of the abortion of the transaction.

- Whenever a abort operation is received, simply send the ABORT request to all copy sites in which the operations of the transaction have been executed.

At Each Database Server:

- Whenever a read request is received from a coordinator, do the following: if the copy is not locked with a W_INTEND or a W_LOCK, then grant R_LOCK on the copy, add the lock information for the copy. and return OK acknowledgement along with the copy.
- Whenever a PREPARE request is received from a coordinator, do the following:

1. if the transaction in the site has been aborted, then reply the request with NO.
2. For each read unlock request: if the copy of item in the site has a version number which is different from that in the locked copy, then reply the request with NO and abort the transaction in the site. Otherwise, set the unlock information at the copy. The information will indicate for which transaction the unlock is requested.
3. For each write lock request: if the copy of item are not locked with any W_INTEND or W_LOCK, then grant W_INTEND on the copy. Otherwise, the request is blocked until W_INTEND or W_LOCK is released.
3. If both step (2) and step (3) finish successfully, send YES to the coordinator.

- Whenever a COMMIT request is received from a coordinator, do the following: for each copy of item to be updated, release R_LOCKs on the copy if any, clear any read lock/unlock information for the copy, upgrade W_INTEND lock to W_LOCK lock , and then enfore the update on the copy. After the update finishes, release the W_LOCK.

- Whenever a ABORT request is received from a coordinator, release any R_LOCK or W_INTEND on the copy set by for the transaction and abort the transaction in the site.

□