Department of Computer Science Technical Reports

Department of Computer Science

1995

Partitioned Data Management in Mobile Environments

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

Jin Jing

Abdelsalam Helal

Rafael Alonso

Report Number:
95-080

PARTITIONED DATA MANAGEMENT IN MOBILE ENVIRONMENTS

Ahmed Elmagarmid
Jin Jing
Abdelsalam Helal
Rafael Alonso

CSD TR-95-080
December 1995

Partitioned Data Management in Mobile Environments

Ahmed Elmagarmid, Jin Jing, Abdelsalam Helal
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907 USA
{ake,jing,helal}@cs.purdue.edu

Rafael Alonso
Matsushita Information Technology
Laboratory, 2 Research Way
Princeton, NJ 08540 USA
alonso@research.panasonic.com

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. As a result of this mobility, the operations of the transaction may be executed at different servers. The distribution of operations implies that the transmission of messages (such as those involved in a two phase commit protocol) may be required among these data servers in order to coordinate the execution of these operations. In this paper, we will address the distribution of operations that update *partitioned data* in mobile environments. We show that, for operations pertaining to resource allocation, the message overhead (e.g., for a 2PC protocol) introduced by the distribution of operations is undesirable and unnecessary. We introduce a new algorithm, the *Reservation Algorithm* (RA), that does not necessitate the incurring of message overheads for the commitment of mobile transactions. We address two issues related to the RA algorithm: a *termination protocol* and a *protocol for non-partition-commutative operations*. We perform a comparison between the proposed RA algorithm and existing solutions that use a 2PC protocol.

Index terms: partitioned data, replicated data, distributed transaction management, 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 [9, 7, 5], a mobile computing environment consists of two distinct sets of entities: *mobile hosts* and *fixed hosts*. Some fixed hosts, called *Mobile Support Stations* (MSSs), are augmented with a wireless interface to communicate with mobile hosts. A mobile host can move from one *cell* (or radio coverage area) to another while retaining its network connections.

The mobile computing paradigm introduces new technical issues in the area of database systems [9, 3]. 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. A user carrying a portable computer can submit the operations of a transaction to distributed data servers from different locations. As a result of this mobility, the operations of the transaction may be executed at different servers. The distribution of operations implies that the transmission of messages (such as those involved in a two phase commit (2PC) protocol) may be required among these data servers in order to coordinate the execution of these operations. In this paper, we will address the distribution of operations that update *partitioned data* in mobile environments.

1.1 The Problem

Conventional methods for replicated data management are expensive because more than one site may be required to form the quorum necessary to run an update transaction. To overcome this restriction, some approaches reported in the literature have taken into account the semantics of applications to improve the response time and throughput of update transactions and to increase system resiliency.

One of the application classes that has recently been extensively studied and has been used to improve response time involves the problem of *resource allocation*. Consider an application where a data item represents the number of tickets to be sold. If the item is replicated, more than one site may be required to form the quorum necessary to perform an update. If the item resides in a central site, requests for tickets originating at that site can be satisfied locally, while all other sites in the system must exchange a series of messages with the central site. An alternative to either of these approaches is to *partition* the "tickets" data item among all the sites. [1] Each site is allocated a fraction of the tickets and will use them to process transactions as long as enough

[1] Some proposed approaches, including site escrow [14, 16], demarcation protocol [6, 1], and Data Value Partitioning protocol [19], can be used for dynamically partitioning data among different server sites.

tickets are locally available. As a result, the overhead associated with communications is avoided for most transactions. Therefore, by partitioning data among server sites, transactions with resource allocation operations can be performed in a single site if the allocation updates do not violate local resource constraints.

Problems involving resource allocation can also be found in such mobile application domains as mobile sales and inventory applications [15] and mobile shopping applications [4] etc. In a mobile environment, a mobile host can query or update a database, which is distributed in multiple data servers over a fixed network, from different locations. A mobile host is also likely to incur long disconnection periods due to the limitations of battery energy and the mobility of hosts. This long-disconnection characteristic may cause mobile transactions that access data from servers to be long-lived.

By exploiting the semantics of applications, it is possible to partition data items into geographically distributed servers and allow mobile clients to perform updates in local or nearby partitioned copies. The partition approach, therefore, improves not only the performance and scalability, but also the availability of transaction processing for update operations. However, traditional partitioned data management may incur extra message costs due to the mobility of hosts. The costs run counter to the motivation for the use of partitioned data and negate some of the advantages of this approach. The following example that uses the site escrow approach proposed in [14, 16] illustrates the problem:

Example 1.1 *Consider a mobile database system where X and Y are numeric objects with the resource constraints $X_{min}(= 0) \leq X \leq X_{max}(= 100)$ and $Y_{min}(= 0) \leq Y \leq Y_{max}(= 250)$, respectively. Initially, the value of X is partitioned into local numeric objects X_1 and X_2 in servers 1 and 2, respectively, with $X_1 + X_2 = X$. Similarly, Y, X_{min}, X_{max}, Y_{min}, or Y_{max} are also partitioned into servers 1 and 2 such that $Y_1 + Y_2 = Y$, $X_1^{lower} + X_2^{lower} = X_{min}$, $X_1^{upper} + X_2^{upper} = X_{max}$, $Y_1^{lower} + Y_2^{lower} = Y_{min}$, and $Y_1^{upper} + Y_2^{upper} = Y_{max}$.*

Assume that a mobile host submits a resource allocation transaction T with the following operations: [Increase X by 10], followed by [Increase Y by 10], and commit. Figure 1 shows the execution procedure of the transaction in the mobile database system. The mobile transaction host for T submits [Increase X by 10] to Server 1 from Cell 1. Server 1 checks the escrow variable $X_1^{escrow\_I}$ and makes a worst-case decision to determine whether the operation can be executed. If $X_1^{escrow\_I} + X_1 + 10 \leq X_1^{upper}$, then $X_1^{escrow\_I}$ is increased by 10. [2] *The transaction host then*

[2] Actually, an uncommitted operation on object X_1 may be logged in an *escrow log*. For simplicity, we shall use the escrow variables $X_1^{escrow\_I}$ (= a) and $X_1^{escrow\_D}$ (= a) to represent the log information for operations [Increase

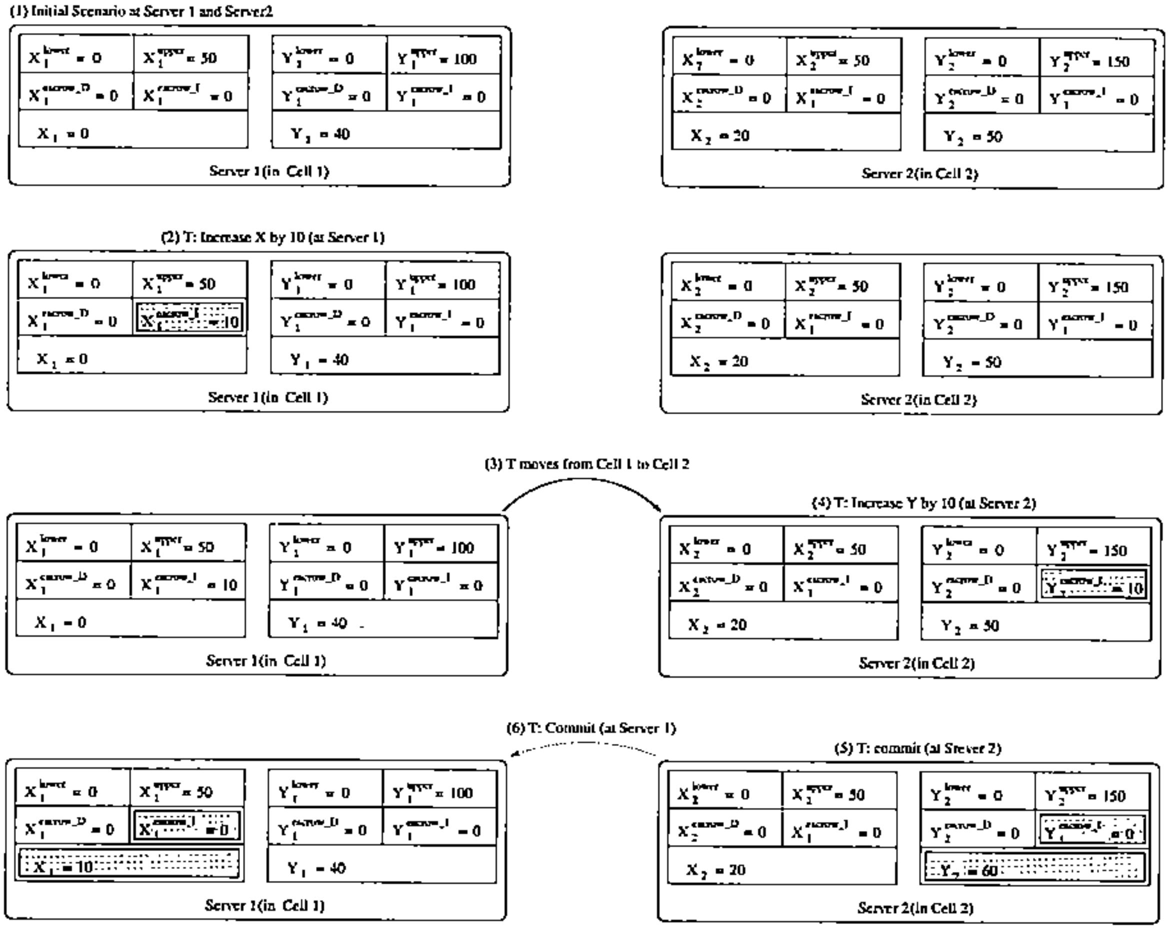

Figure 1: A Mobile Transaction Example

moves to Cell 2 to submit [Increase Y by 10] to Server 2. Similarly, Server 2 increases the escrow variable $Y_2^{escrow\_I}$ *by 10.*

As a result of this mobility, the two operations are actually executed in two different servers. At commit time, a two phase commit protocol must be used to shift the increases from the escrow variables to the local numeric objects. That is, $X_1 = X_1 + 10$, $Y_2 = Y_2 + 10$, $X_1^{escrow\_I} = X_1^{escrow\_I} - 10$, *and* $Y_2^{escrow\_I} = Y_2^{escrow\_I} - 10$.

The distribution of operations in the above example presents two problems which arise with the site escrow approach:

- the message transmissions involved in a 2PC protocol increase the traffic over the fixed networks; and
- the use of a 2PC protocol will reduce site autonomy.

These problems obviously run counter to the motivation for the use of partitioned data and negate some of the advantages of this approach. Of course, if the transaction host in Example 1.1 remains fixed, transaction T can be executed in either server without involving in message transmissions.

To avoid the use of a 2PC protocol at commit time, it was suggested in [15] that the move of a transaction host to a new cell should be accompanied by the transfer of the escrow log for the transaction to the local server under the cell. At the end of transaction, a commit operation can be executed at the local server without communication with other servers. However, the transfer procedure itself requires the use of a 2PC protocol and therefore still generate high traffic over the fixed network. When the host repeatedly moves between two cells during the execution of a transaction, the repeated log transfers between the two servers cause particularly heavy message overhead.

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, communications traverse more intermediaries and consume more network capacity. The mobility of hosts can cause that even a short transaction to involve a long communication transmission.

X by a] and [Increase X by a], respectively.

1.2 Contributions of the paper

In this paper, we present an approach that avoids both heavy message transmissions and the use of a 2PC protocol. A low message overhead among servers for each operation (including commit and abort) will improve the response time of an operation requested by a mobile host. One benefit of fast response time is that the mobile host will not need to expend precious battery resources while waiting for the acknowledgement of requested operation.

The approach we propose in this paper is called *Reservation Algorithm* (RA). In the site escrow approach, an escrow log is used for both commitment/recovery and constraint-maintaining purposes for uncommitted transactions. In contrast, this algorithm ensures resource constraints for the operations of uncommitted transactions by simply modifying bound (lower or upper) variables at local server sites. For commit and recovery purposes, the algorithm stores the operations in a *reservation log*. For example, for the operation [Increase X by 10] in Example 1.1, this algorithm needs only to decrease the bound variable X_1^{upper} by 10 and to store the operation in a reservation log at Server 1. The results of operations are returned to the mobile host along with acknowledgement messages. The mobile host stores the returned results in its reservation log for the transaction. Conceptually, the reservation log in the mobile host is a logical copy of logs maintained in server sites. At the commit time (after the host moves to Cell 2), the mobile host sends its logical reservation log, along with a commit request, to the local server 2 in the current cell. Server 2 will use the log information to perform the actual resource allocations, i.e., increasing both X_2^{upper} and X_2 by 10. This algorithm will ensure that the resource constraint, $X_{min} \leq \sum_{i=1}^{2} X_i^{lower} \leq X \leq \sum_{i=1}^{2} X_i^{upper} \leq X_{max}$, is continualy maintained.

The RA approach allows the *resource reservations* for the operations of uncommitted transactions and the actual *resource allocations* to be executed at different servers without the need for communication. The resource reservations involve the modification of bound variables and the update of reservation logs. Modifications of bound variables ensure the maintenance of resource constraints for the operations of uncommitted transactions. The process of resource allocation will restore modified bound variables and allocate resources at any partitioned data site (which may be different from the site where the reservations were performed).

Although the overall framework of the RA approach is straightforward, two interesting issues related to this approach merit deeper investigation. The first issue is the design of a *termination protocol*. In a mobile environment, an active mobile transaction may be aborted unilaterally by a data server. Such a *unilateral abortion* may be triggered by an extended long disconnection by or a total failure (destruction or loss) of the mobile host. In this case, server may decide to abort the transaction to release reserved resources. Unfortunately, the server can not make this decision on the basis only of the information in its local reservation log, since the mobile host can make a commit decision without communication with the server. The purpose of a termination protocol is to guarantee that the commit decision of a mobile host will not contradict with the unilateral abort decision of a data server.

The second issue is the development of a protocol for non-partition-commutative operations on partitioned data. Assume that a data item X is partitioned among N sites such that $X = \sum_{i=1}^{N} X_i$ where X_i is the partitioned copy of X in site i. We say an operation O is a *partition-commutative* operation (pc-operation) if $O(X) = O(X_j) + \sum_{i=1, i \neq j}^{N} X_i$ for any j $(1 \leq j \leq N)$; otherwise, it is a *non-partition-commutative* operation (npc-operation). An example of an npc-operation can be found in a banking application. In this application, both withdrawal and deposit operations are pc-operations, while an interest-posting operation is an npc-operation. It is obvious that an npc-operation on partitioned data can not be performed in any single site if the data is partitioned over more than one site. A protocol is therefore needed to coordinate the execution of such operations.

In this paper, we explore the following problems related to our proposed reservation approach:

1. Development of termination protocols that can be included in the reservation approach. These protocols should ensure that an unilateral abort by a data server and a commit by a mobile host would not be made simultaneously for a mobile transaction;

2. Determination of the effect of the proposed reservation approach on the npc-operations on partitioned data and of a protocol to permit these operations to accommodate the reservation approach.

3. Comparison of the message cost of a reservation algorithm that includes the required protocols for termination and npc-operations with that of existing site escrow or escrow log transfer algorithms.

The remainder of this paper is organized as follows. Section 2 introduces the system model and relevant terminology. In Section 3, we describe the basic reservation algorithm and the required termination protocols. Section 4 discuss a protocol for npc-operations. Section 5 presents a performance evaluation of the proposed algorithm and the traditional 2PC protocol in terms of message costs in the fixed network. Related research is discussed in Section 6, and concluding remarks are offered in Section 7.

2 The Mobile Transaction Model

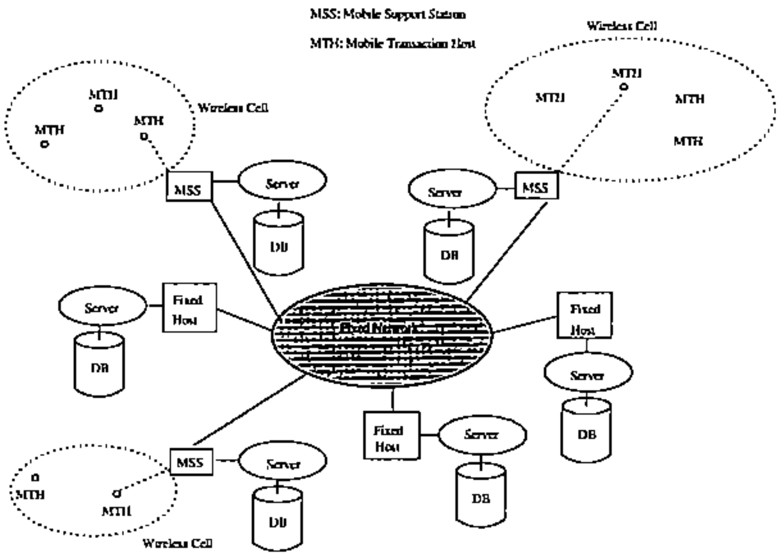

Figure 2: Mobile Database System Model

Figure 2 presents a general mobile database system model similar to those described in [9, 7, 5] for mobile computing systems. In this model, both a database *server* and a database are attached to each fixed host. A database server is intended to support basic transaction operations such as resource allocation, 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 resource allocation operation to a local server if a partitioned copy is in the local site.

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 or a group of operations; the mobile host interactively submits the operations of a transaction to its coordinator. A subsequent opera-

tion 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 [20] 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 transaction computations.

We assume that a mobile host may move at any time. It may move away from its current cell after submitting an operation and before receiving a reply from the coordinator. The new coordinator will determine whether the host needs to obtain acknowledgement messages from previous coordinator after registering in the new cell. In this case, additional procedures are needed to locate the mobile host and convey to it the results of submitted operations. For the simplicity, in this paper, we assume that each service area supported be a server covers only a single cell. In reality, one service area may support more than one cell [10, 11].

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 pc-operations and npc-operations which are bracketed by a $BEGIN\_TRANSACTION$ statement and an $END\_TRANSACTION$ statement.

3 Reservation Algorithm For Mobile Transactions

3.1 Basic Structure of the Reservation Algorithm

Suppose that the value of X is partitioned into local numeric objects X_i in server i ($1 \leq i \leq N$) such that $X = \sum_{i=1}^{N} X_i$. Similarly, the bound value X_{min} (or X_{max}) is initially partitioned into X_i^{lower} (X_i^{upper}) in every server i ($1 \leq i \leq N$) such that $X_{min} = \sum_{i=1}^{N} X_i^{lower}$ (or $X_{max} = \sum_{i=1}^{N} X_i^{upper}$). A *reservation action* for [Increase X by a] (or [Decrease X by a]) in server i involves the operation $X_i^{max} = X_i^{max} - a$ when $X_i + a \leq X_i^{max}$ (or $X_i^{min} = X_i^{min} + a$ when $X_i^{min} \leq X_i - a$). A *release action* for [Increase X by a] (or [Decrease X by a]) in server i involves the operation $X_i^{max} = X_i^{max} + a$ (or $X_i^{min} = X_i^{min} - a$). A *allocation action* for [Increase X by a] (or [Decrease X by a]) in server i performs the operations $X_i = X_i + a$ and $X_i^{max} = X_i^{max} + a$ (or $X_i = X_i - a$ and $X_i^{min} = X_i^{min} - a$). An operation O in server i is *safe* if $X_i^{min} \leq O(X_i) \leq X_i^{max}$. An operation O in server i is *unsafe* but *resolvable* if $X_i^{min} \leq O(X_i) \leq X_i^{max}$ does not hold but $X^{min} \leq O(X) \leq X^{max}$ holds.

Each reservation action (release, or allocation action) should be implemented as an atomic unit. Conventional database techniques can be used at each server to ensure that the actions that change the bound and resource variables will be atomic and persistent. When an action is completed, any locks on bound and resource variables will be released. Each server will record all the executions of these actions in a reservation log.

Assuming that no abortion is invoked by the servers, the reservation algorithm follows this general format:

1. The mobile host sends each pc-operation of a mobile transaction to the coordinator in the current cell, which will forward it to a local or nearby server where a partitioned data copy resides.

2. If an pc-operation at a server is safe, the server then executes a local reservation action for the pc-operation. Otherwise, the server invokes a resource repartition procedure (such as the point-to-point demarcation protocol [6] or a dynamic quorum-based protocol [14]) to requisition additional partitioned data resources from other servers. Upon the successful completion of the resource repartition procedure, the reservation action can be executed at the local server. The result of the reservation action is returned to the mobile host that submitted this pc-operation through its coordinator. If the operation is neither safe nor resolvable, a failure message will be returned.

3. The mobile host records the results of the reservation action of each pc-operation and the pc-operation itself in a reservation log. If all pc-operations of a mobile transaction succeed from the execution of reservation actions, the mobile host sends a COMMIT message along with the reservation log of the transaction to the coordinator in the current cell. Otherwise, it sends an ABORT message along with the reservation log to the coordinator. The coordinator then submits an allocation action (for COMMIT) or release action (for ABORT) for each pc-operation in the log to the local or nearby server.

Note that, due to mobility, the server at which the allocation actions are executed may not be the same as that from which the mobile host reserved these resources for the transaction (see

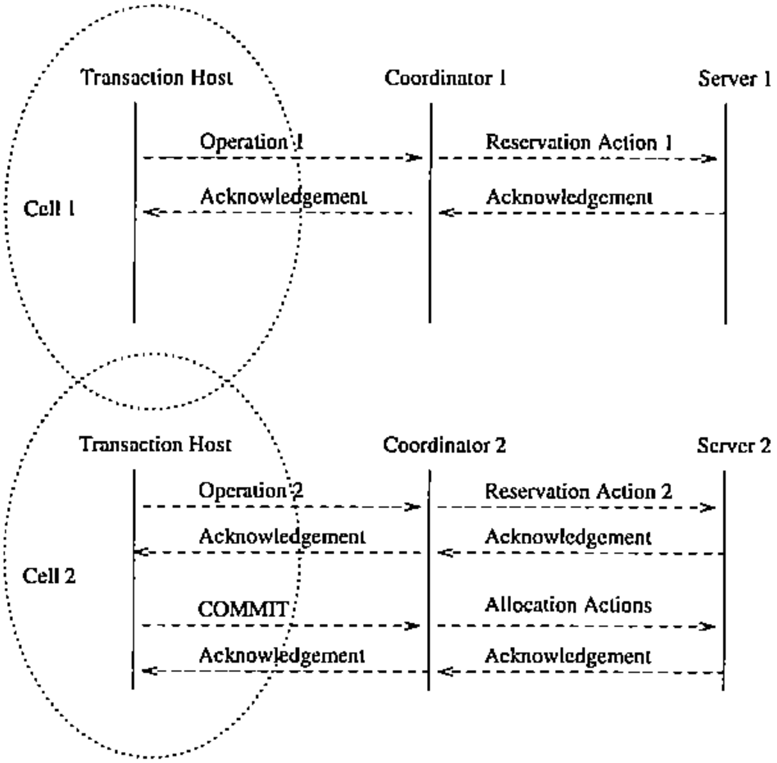

Figure 3: An Example of Basic Reservation Algorithm Execution

Figure 3).

In the escrow approach, the escrow log serves both to check local resource constraints and to commit or recover transactions. In contrast, the reservation log in the reservation approach serves only the latter purposes. In the escrow approach, for example, when an uncommitted transaction attempts to perform an pc-operation such as [Increase X by a], the server will use the escrow variable X^{escrow} in the escrow log to ascertain whether a given local resource constraint will be satisfied for the pc-operation; i.e., whether $X + X^{escrow} + a \leq X^{upper}$ holds. In the reservation algorithm, on the other hand, the maintenance of the constraint will not involve the use of the escrow variable X^{escrow}. Any effect of uncommitted transactions on resource constraints has been addressed by modifying the values of the bound variables by the reservation actions.

A mobile host may move from one cell to another at any time. In this paper, we assume that each service area supported by a server covers only a single cell. If the host has left a cell before receiving the acknowledgement message for last submitted pc-operation, this message will be forwarded by the coordinator in the new cell. After the host registers in the new cell, the coordinator will determine whether any acknowledgement messages are still outstanding from previous coordinator. To handle host mobility, each coordinator runs the following handoff algorithm:

1. If a recently arrived host has received acknowledgement messages for all submitted pc-operations, the coordinator in the new cell will require only a request of pc-operations from the host. Otherwise, before accepting new pc-operations, it will send an acknowledgement request to the coordinator in the previous cell and then forward these acknowledgements (if any) to the host.

2. If a coordinator receives an acknowledgement request from another coordinator for the pc-operations submitted by a host that has left its cell, it will forward the acknowledgement message to the requesting coordinator.

It is clear that such an approach will maintain resource integrity constraints, provided that no data server is allowed to abort a mobile transaction. Any reservation action for a pc-operation can be performed only if the pc-operation is safe locally or resolvable globally. This safety or resolvability property implies that the bound variable updates by the reservation action always maintain resource constraints. A release action is actually the reversal of of a reservation action and is invoked only if the reservation action has been executed. An allocation action will also reverse the bound variable change made by a reservation action and will update the corresponding resource variable. The increased or decreased value for the resource variable always matches that for the bound variable. Therefore, the basic reservation algorithm maintains resource constraints. The algorithm also ensures a serializable execution of committed transactions with pc-operations without requiring locks to be held until the commit time (a lock may be needed during the execution of each action), since all pc-operations are mutually commutative.

In a mobile environment, however, a mobile host may be unreliable or may suffer a total failure such as destruction or loss. In this case, a server may take an abort action for an uncommitted mobile transaction. We will address this issue and related problems in the rest of this section.

3.2 Unilateral Abortion Anomaly

In a mobile database environment, a server may decide to abort a transaction if the mobile transaction host has disconnected from any server for an extended period. Such an abortion allows the system to release resources reserved by the host. A mobile host may be somewhat unreliable, and an unexpected long disconnection period may be caused by a total failure of the device through destruction or loss. Abortion of a transaction avoids the indefinite holdings of reserved resources.

An abortion of this sort may result in an anomaly, called a *Unilateral Abortion Anomaly*, if the mobile host attempts to commit a mobile transaction through its current coordinator without following certain protocols.

Example 3.1 *Consider a mobile database system consisting of servers 1 and 2 with a resource constraint* $X \leq 20$. *Suppose that a mobile host had submitted the operation [increase X by 10] for transaction T to server 1 before it was disconnected from the system. The host then moves and establishes a connection to server 2 after server 1 has decided to abort the transaction during its disconnect period. Because server 1 made the abort decision without following certain protocols, the mobile host, which has no knowledge of the abort, may commit the transaction at server 2. While the abort will actually release reserved resources by increasing* X_1^{upper} *by 10, the commit will consume the reserved resources by increasing* X_2 *and* X_2^{upper} *by 10 in server 2. As a consequence, the total upper bound of X becomes 30 rather than 20; i.e.* $X_1^{upper} + X_2^{upper} = 30 > 20 = X_{max}$, *which violates the resource constraint* $\sum_{i=1}^{2} X_i^{upper} \leq X_{max}$.

This anomaly arises because both the commit and abort actions are performed independently at different servers without any coordination. Recall that the coordinator allocates reserved resources at any local server site without communication with other servers from which these resources were gathered.

It is also possible that both a server and a mobile host may simultaneously decide to abort a transaction. Recall that a release action on behalf of a mobile host can be executed at any partitioned copy server. The protocols must guarantee that these reserved resources will not be released redundantly by both abortions. That is, each reserved resource can be released only once, even though these releases invoked by a mobile host can be performed at servers other than those where these resources were originally reserved.

Therefore, a protocol to avoid the unilateral abortion anomaly should ensure the following two conditions:

1. a transaction cannot be simultaneously committed by a mobile host and aborted by a data server; and

2. each reserved resource in a transaction can be released only once if a given transaction is aborted by a server and a mobile host.

3.3 Termination Protocols

In the proposed reservation algorithm, the allocation actions for the commitment of a transaction do not take the responsibility of writing the commit status into logs in other servers where reservation actions were executed. A *termination protocol* should therefore be applied so that an abort decision made by a server will not coincide with a commit decision made by the mobile host.

We assume that, when a server makes an abort decision, it can release the resources on x only if they were reserved by the transaction from the server. In other words, the server cannot release the resources on y if they were reserved from other servers. Two termination protocols which are candidates for inclusion in our reservation algorithm are:

1. All-Copy Voting Protocol: Let $N(x)$ be the set of partitioned copy sites for a data item x and $D(t)$ be the set of data items that transaction t has reserved. We assume that each item in $D(t)$ is only reserved once by transaction t. The server can abort an uncommitted transaction t and release the resources reserved on x in the server only if it receives an abort vote from each site in $N(x)$ for data item x in $D(t)$.

   A two phase protocol should be used to ensure that an abort decision and released resources will be recorded in reservation logs at all sites which have voted for the abort. In the first phase, the server sends an abort request to all the sites in $N(x)$ for reserved item x in $D(t)$. After all the sites return their votes, the server enters the second phase. If all the sites vote for the abort and the reserved item has not been released in any of these sites (by a mobile host), the server sends the abort decision to those sites and locally releases the resources reserved on x. Once a site voting for the abort receives the abort decision, the abort status for the transaction t will be recorded in its reservation log along with the names of released items. If any site has voted for commit or the reserved item has been released, the server will do nothing except recording the status into its local reservation log.

2. Transaction-Proxy Voting protocol: When a mobile transaction is initialized, the system can specify a server as the proxy for the transaction. If a server wishes to abort the uncommitted transaction and release resources reserved in the server, it must obtain a vote from the transaction proxy.

   The proxy will vote for abort only if it has not voted for commit or the reserved item has not been released. Once the transaction proxy votes for the abort, the abort status will be recorded in its reservation log along with the names of released items.

To integrate the all-copy voting protocol into our proposed reservation algorithm, each allocation action for a partitioned data item x should involve a determination of whether any partitioned copy server has voted for an abort decision. If not, the action can be executed at any local or nearby server and a commit flag (vote) can be recorded in the reservation log of the server without communication with other servers.

When a mobile host requests a commit operation for a transaction, its coordinator should execute all the allocation actions for the transaction in an atomic unit. If all the allocation actions can be executed at a single site (i.e., there is a local partitioned copy for each reserved resource), the commit operation can be performed locally. Otherwise, a 2PC protocol is needed to ensure the atomic execution of these actions. The procedure is required because each server may unilaterally abort a transaction and release reserved resources at any time. If the commit operation is not executed in a atomic unit, a server may unilaterally abort the transaction and execute a release action between two allocation actions, resulting in an undesired inconsistent termination decision. When a mobile host requests an abort operation for a transaction, the coordinator can execute the release actions for the transaction individually, without requiring atomic execution. However, each release action must ensure that the resource in question has not been previously released by an unilateral abort action by another server. This can be determined by examining the log information at the local server.

To incorporate the second termination protocol discussed above, the reservation algorithm should be modified in the following manner. Before a coordinator executes any allocation action, the server should get a commit vote from the transaction proxy. Once it obtains this vote, the allocation actions for a transaction can be executed individually; For a release action, the server should obtain an abort vote from the proxy. Furthermore, each server should inform the proxy which reserved resources to be allocated or released. Whenever other servers require a vote from the proxy, this information regarding released items should be supplied to them to prevent the redundant release of a reserved item by different servers.

A reservation algorithm that integrates either an all-copy voting or transaction-proxy voting termination protocol will be free from the unilateral abortion anomaly. Either protocol will require that the sites (or copies) voting for an abort always intersect with the sites (or copies) voting for commitment. By recording the released resources in the logs of voting sites, any reserved resource will not be released more than once by different servers.

3.4 Discussion

The two termination protocols described above could be subject to blocking even in the case of site failure. In the all-copy voting protocol, when a site which holds a copy of a reserved item fails, other servers can not execute the termination protocol to release the reserved item. In the transaction-proxy voting protocol, the potential for blocking is even higher. If the transaction proxy fails, no server can execute the termination protocol until it recovers.

Counterbalancing these blocking problems, the all-copy voting protocol offers low message overhead and supports a high degree of site autonomy for the commitment of transactions because all allocation actions can be executed locally or at nearby sites. The transaction-proxy voting protocol also offers low message overhead for the commitment or abortion of all transactions but does not support high site autonomy because both commit and abort decisions depend on the vote of a server designated prior to the execution of the transaction.

It has been generally held that non-blocking termination and efficient commitment are two incompatible goals in a distributed system. Our reservation algorithm illustrates the validity of this observation. While a reservation algorithm which incorporates either termination protocol permits a low-cost and efficient commitment of transactions, it imposes some restrictions on the execution of the termination protocol. The 3PC protocol, in contrast, involves no blocking in the event of site failure but has high message overheads for the commitment of transactions.

Finally, we note that, when a transaction is committed, this decision is not broadcast immediately to all log sites where reservation actions for the transaction were executed. In this case, some logs will still contain the *pending reservation* information about the transaction, potentially resulting the invocation of an abort request by the server. Although this will not create inconsistencies if all servers follow one of the termination protocols, it cause some unnecessary messages to be sent over the network. The pending reservation and other log information for a committed transaction can be removed if the system can periodically circulate the commit decision to other servers or piggyback the decision on other messages sent to servers.

4 Protocol for Non-Partition-Commutative Operations

In this section, we will examine the effect of our proposed reservation algorithm on npc-operations with partitioned data and discuss the design of a protocol to accommodate the reservation algorithm with such operations. We assume that serializability is used as the correctness criterion for the execution of transactions. We first review the execution of an npc-operation in a traditional distributed environment in which the host is fixed during the execution of a transaction and each reservation action and its allocation action (or release action) are performed in the same site.

4.1 The Problem

Assume that a data item X is partitioned among n servers such that $\sum_{i=1}^{n} X_i = X$. An npc-operation *npcO* on the partitioned data X can be performed by the coordinator of a transaction in two different ways. In the first approach, the coordinator collects all the values of partitioned copies from all n servers and executes the operation over the sum of these values; i.e., $npcO(\sum_{i=1}^{n} X_i)$. At commit time, the coordinator will repartition the result of the operation and write these repartitioned copies back to the n servers. In the second method, the coordinator sends the operation directly to all n partitioned copy servers. Each server i will perform the operation over the value of the partitioned copy; i.e., $npcO(X_i)$. An operation *npcO* on X is successful (i.e., each server can write the results back to a database) if and only if the operation $npcO(X_i)$ succeeds at every server i ($1 \leq i \leq n$). The discussions in the rest of this section will be suitable to either approach.

To ensure a serializable execution, a lock protocol could be followed to coordinate the execution of pc-operations and npc-operations. If a pc-operation is to be executed on a partitioned copy X_i, a *Partition-Commutative Lock* (PC_LOCK) must be obtained from the partitioned copy X_i. If an npc-operation is to be executed, a *Non-Partition-Commutative Lock* (NPC_LOCK) should be obtained from all partitioned copies of X. A PC_LOCK is compatible with other PC_LOCKs but conflicts with another NPC_LOCK. Two NPC_LOCKs conflict with each other.

In a traditional distributed environment where the host is fixed during the execution of a transaction, all the actions of a pc-operation on X will be executed in a single partitioned copy server. A PC_LOCK should be set on the partitioned copy before these actions can executed. For an npc-operation on X, the coordinator of the transaction will send an NPC_LOCK request to all partitioned copy servers. Each server then determines whether the NPC_LOCK can be granted. If no pending PC_LOCK is set on the copy, the server grants the request and sends a reply message to the coordinator. Otherwise, the request is blocked at the server. The coordinator collects reply messages from these sites. Once all sites reply to the request, it concludes that there is no pending PC_LOCK at any server and the NPC_LOCK is granted. Thus, only one round of message exchanges is needed between the coordinator and any partitioned copy server for an NPC_LOCK request. At commit or abort time, these PC_LOCKs and NPC_LOCKs will be released.

If the transaction host may move among different cells during the execution, a reservation action and a allocation action for a pc-operation in the reservation algorithm may be executed at different servers. In this case, an interesting question is how a PC_LOCK that was set when the reservation action was requested can be released after the allocation action completes.

At the simplest level, a server executing the allocation action can immediately forward a PC_UNLOCK to the server where the PC_LOCK was set. If the transaction host moves frequently, however, this method will generate heavy message traffic. If there are m such pc-operations with reservation actions and allocation (or release) actions on different servers, m messages will be needed. The method obviously runs counter to the motivation of the use of the reservation algorithm.

A second method is to delay the forwarding of the PC_UNLOCKs until an NPC_LOCK request arrives. Each server then sends to other servers a single message containing a batch of PC_UNLOCKs that were executed since last npc-operation on the copy was completed. The method will release a PC_LOCK on a copy until an NPC_LOCK request arrives on the copy. If each partitioned copy server for a partitioned data item must forward the PC_UNLOCK messages to all other servers, then, in the worst case, the message overhead will be approximately n^2, where n is the number of partitioned copies.

In this paper, we suggest a method that requires each copy server to send all granted PC_LOCKs and delayed PC_UNLOCKs to the coordinator which is requesting a NPC_LOCK on that copy. The coordinator collects these PC_LOCKs and PC_UNLOCKs and then attempts to match a pending granted PC_LOCK with a delayed PC_UNLOCK. The message overhead will be $2n$. If the m PC_UNLOCKs for granted PC_LOCK between two npc-operations are uniformly distributed and m is greatly larger than the number n of the partitioned copies (i.e., $m >> n$), this method will prove to be more efficient than the first approach described above. If $n >> 2$, then the method is also superior to the second approach above because $n^2 > 2n$.

4.2 The Protocol

For a pc-operation, the PC_LOCK and PC_UNLOCK operations can be executed at different servers. A PC_LOCK is always executed at the server at which a reservation action is executed, while a PC_UNLOCK is always executed at the server at which an allocation action is executed. The granted PC_LOCK will actually be released until an NPC_LOCK request arrives.

To request an NPC_LOCK for an npc-operation, the coordinator can execute a protocol with two rounds of message exchanges. In the first phase, the coordinator collects the PC_LOCK/PC_UNLOCK information from all copy servers. The coordinator cannot enter into the second phase until all servers reply and each PC_LOCK is matched by one PC_UNLOCK on the item to be accessed by the npc-operation. In the second phase, the coordinator sends a confirmation message to every copy server and the server releases the PC_LOCK and sets the NPC_LOCK on the partitioned copy.

To guarantee that no other PC_LOCK will be set after the first phase of the protocol, a new lock mode, called NPC_INTEND, must be used. This lock mode locks the copy at all the partitioned copy servers before these servers reply to the NPC_LOCK request in the first phase. If this step were bypassed, the copy server would be unable to force the setting of the NPC_LOCK in the second phase if other PC_LOCKs are granted on the copy after the first phase. A requested NPC_INTEND is compatible with a granted PC_LOCK but not with a granted NPC_INTEND or NPC_LOCK. A granted NPC_INTEND is, however, incompatible with a requested PC_LOCK or a requested NPC_INTEND or NPC_LOCK.

When the first phase of an NPC_LOCK request arrives at a copy server, an NPC_INTEND is set on the copy if no other NPC_INTEND or NPC_LOCK applies to that copy. Otherwise, the first phase of the NPC_LOCK request is blocked at the server. This scenario implies that two different npc-operations are requesting the NPC_LOCK. If there is a PC_LOCK on the copy, the first phase of the NPC_LOCK request will immediately set the NPC_INTEND on the copy. On the other hand, a granted NPC_INTEND will prevent any other new requested PC_LOCK or NPC_LOCK.

If the PC_LOCKs and PC_UNLOCKs collected in the first phase are not matched, it can be concluded that some partitioned copies are locked by other pc-operations. In this case, the coordinator will wait for further PC_UNLOCK messages from partitioned copy servers. A partitioned copy server will forward any newly executed PC_UNLOCK to a coordinator if an NPC_INTEND

requested by the coordinator has been set on the copy. Once the coordinator collects matched *PC_UNLOCK*s, an *NPC_LOCK* confirmation message is sent to all the partitioned copy servers.

After the copy server receives a confirmation message from the coordinator in the second phase, all *PC_LOCK*s that have been matched by *PC_UNLOCK*s at the coordinator are removed and the *NPC_INTEND* is upgraded to an *NPC_LOCK*. The lock compatibility matrix appears in Figure 4.

Tj(lock)
Ti(request)
	NPC_INTEND	PC_LOCK	NPC_LOCK

NPC_INTEND	No	Yes	No
PC_LOCK	No	Yes	No
NPC_LOCK	No	No	No

Figure 4: Lock Compatibility Matrix

We have shown that the granted *PC_LOCK* can be removed when another transaction prepares to perform an npc-operation on the copy. The pending period of a *PC_LOCK* may last until the first *NPC_LOCK* by another transaction. It is obvious that the pending *PC_LOCK* does not block either other *PC_LOCK* requests or *NPC_LOCK* requests. Thus, no blocking is incurred by a pending *PC_LOCK* on a copy of an item until the copy is accessed by an npc-operation.

Merging the locking protocol into the reservation algorithm is a straightforward procedure. A coordinator can send a *PC_LOCK* request with the corresponding reservation action to a partitioned copy server. The server starts to execute the reservation action only if the *PC_LOCK* is set on the copy. The *PC_UNLOCK* is set at any copy server after the completion of the allocation or release action. For an npc-operation, a coordinator sends an *NPC_LOCK* request to all partitioned copy servers. The coordinator can send any action of the npc-operation to copy servers only after the first phase of the locking protocol ends. Therefore, the npc-operation (or related actions) can be sent to copy servers with the *NPC_LOCK* confirmation message. At commit time, these *NPC_LOCK*s are released; note that a 2PC protocol is required to commit the npc-operation transaction.

5 Comparison of RES, ELT, and 2SE

As discussed in the introduction, one advantage offered by the reservation algorithm is the increased autonomy made possible by avoiding the use of a 2PC protocol. The algorithm also results in improved performance during normal execution, as no communication between server sites is needed for the commitment of some transactions. In this section, we present a comparative analysis of message costs incurred by the execution of the reservation algorithm (RES), the Escrow Log Transfering algorithm (ELT), and the 2PC-Site-Escrow algorithm (2SE) over fixed networks. The 2SE algorithm is the direct application of the site escrow method presented in [14, 16] in a mobile environment (as shown in Example 1). The ELT algorithm is a modification of a site escrow method which always transfers the escrow log for a mobile transaction to the local server in the current cell of the transaction host. This algorithm was described in [15]. Through this comparison, we wish to demonstrate the effect of the parameters of mobility and data partition on the message costs of these algorithms and to discover those circumstances in which the RES algorithm offers lower message costs than others.

5.1 An Analytical Model

We shall first describe a general equation that models the message costs incurred by the execution of various algorithms in accessing partitioned data. Let C^{alg} be the average number of messages per second required by the execution of a given algorithm alg. Then, C^{alg} can be expressed as:

$$C^{alg} = C^{alg}_{pc} + C^{alg}_{npc} + C^{alg}_{rpp} + C^{alg}_{com} + C^{alg}_{hd}$$

where C^{alg}_{pc} is the expected number of messages per second for the execution of pc-operations; C^{alg}_{npc} is the expected number of messages per second for the execution of npc-operations; C^{alg}_{rpp} is the expected number of messages per second for the execution of the repartition protocol (e.g., the site escrow protocol or the demarcation protocol); C^{alg}_{com} is the expected number of messages per second for the execution of commit operations; and C^{alg}_{hd} is the expected number of messages per second for the execution of the handoff protocol.

C^{alg} is obviously a function of such parameters as data partition, transaction host mobility, transaction rate, and transaction access pattern. To present a detail message cost equation for each algorithm, we shall now define these parameters and specify some assumptions.

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. *Partition-commutative transactions* (pc-transactions), which contain only pc-operations, and *non-partition-commutative transactions* (npc-transactions), which contain only npc-operations, arrive in Poisson distributions with an average arrival rate of λ_{pc} and λ_{npc}, respectively. The data are randomly accessed by a transaction. The average number of partitioned data items accessed by a pc-transaction is n_{pc}. The average number of partitioned data items accessed by an npc-transaction is n_{npc}.

The average number of partitioned copies per partitioned data item is represented as p (≥ 2). The probability that a pc-operation hits a partitioned copy at the local server is p/N. Let N_{npc} be the average number of server sites where the npc-operations of an npc-transaction are executed. The number will be between p and N.

We assume that a mobile transaction host can move away from the current cell (or server) only after each operation request submitted from the cell has been acknowledged by the coordinator in the same cell. In other words, we ignore the case in which the host moves to a new cell before it receives an acknowledgement for previously submitted operations. The probability of the mobility of each transaction host is m.

We now derive the basic expressions that describe the message costs for the RES, 2SE, and ELT algorithms. In these expressions, we ignore the message costs for aborted transactions and assume that no conflict exists between an npc-transaction and a pc-operation at any server. We also assume that a transaction host always submits a commit or abort operation to the local server which has all partitioned copies for the execution of the allocation or release actions. Our analytical model will not treat the message costs between mobile hosts and MSSs and consider only the message costs among data servers over MSSs.

C_{pc}^{alg}: For each pc-operation in each of the three algorithms, if there is a partitioned copy at local site, then no communication is needed. Otherwise, the operation will be sent to a nearby partitioned copy site. The expected number of messages per second for pc-operations in these algorithms is:

$$C_{pc}^{RES} = C_{pc}^{ELT} = C_{pc}^{2SE} = 2\lambda_{pc}n_{pc}(1 - p/N)$$

C_{npc}^{alg}: For each npc-transaction in the ELT and 2SE algorithms, only one round of messages is needed to obtain NPC_LOCKs, while, in the RES algorithm, two rounds of messages are needed. The operation is piggybacked along with the NPC_LOCK request messages. Because we assume that no conflict exists between an npc-transaction and a pc-operation at any server, the exact two rounds of messages will be sufficient for an NPC_LOCK request in the RES algorithm. The expected number of messages per second for pc-operations in these algorithms is:

$$C_{npc}^{RES} = 4\lambda_{npc}n_{npc}p, \quad and \quad C_{npc}^{ELT} = C_{pc}^{2SE} = 2\lambda_{npc}n_{npc}p$$

C_{com}^{alg}: When the commit operation (the $END\_TRANSACTION$ operation) is requested, a 2PC protocol will be executed for all npc-transactions in each algorithm. 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 $4\lambda_{npc}N_{npc}$. For the 2SE algorithm, a 2PC protocol will be executed for all distributed pc-transactions. Let N_{pc} be the average number of server sites where the pc-operations of a pc-transaction are executed and θ be a parameter such that $N_{pc} = n_{pc}m\theta$. Because $n_{pc}m$ is the average number of movements per pc-transaction among different servers, it is obvious that $N_{pc} < n_{pc}m$ and $0 < \theta \leq 1$. The number of messages per second involved in the commitment of distributed pc-transactions can therefore be expressed as $4\lambda_{pc}n_{pc}m\theta$. Thus, we arrive at the expected number of messages per second for the commit operation for each algorithm as follows:

$$C_{com}^{RES} = C_{com}^{ELT} = 4\lambda_{npc}N_{npc}, \quad and \quad C_{com}^{2SE} = 4\lambda_{npc}N_{npc} + 4\lambda_{pc}n_{pc}m\theta$$

C_{rpp}^{alg}: For any of the RES, ELT, and 2SE algorithms, a repartition protocol RP (e.g., the site escrow protocol or the demarcation protocol) shall be used to dynamically reallocate or repartition resources so that each pc-operation is safe. The message overhead for the execution of the repartition protocol are in general independent of the algorithm used. Instead, the overhead is a function F_{RP} of database parameters, transaction parameters, and repartition protocol parameters. The expected number of messages per second for the execution of partition protocol can be expressed as:

$$C_{rpp}^{RES} = C_{rpp}^{ELT} = C_{rpp}^{2SE} = F_{RP}$$

C_{hd}^{alg}: We assume that the mobile host moves to a new cell only after it has received acknowledgements for all operations submitted from the old cell. Therefore, in both the RES and 2SE algorithms, the current server does not need to contact the remote server for acknowledgement messages after the host moves to a new cell. However, in the ELS algorithm, a handoff protocol must be executed to transfer the context information from the previous server to the current server. The expected number of messages per second for the execution of the handoff protocol can be expressed as:

$$C_{hd}^{RES} = C_{hd}^{2SE} = 0, \quad and \quad C_{hd}^{ELT} = 4\lambda_{pc}n_{pc}m$$

Totally, the expected number of messages transmitted per second for each algorithm can be given by the expressions:

$$\begin{aligned} C^{RES} &= 2\lambda_{pc}n_{pc}(1 - p/N) \;+\; 4\lambda_{npc}n_{npc}p \;+\; 4\lambda_{npc}N_{npc} \;+\; F_{RP} \\ &= C_0 + 2\lambda_{npc}n_{npc}p \end{aligned} \tag{1}$$

$$\begin{aligned} C^{ELT} &= 2\lambda_{pc}n_{pc}(1 - p/N) \;+\; 2\lambda_{npc}n_{npc}p \;+\; 4\lambda_{npc}N_{npc} \;+\; 4\lambda_{pc}n_{pc}m \;+\; F_{RP} \\ &= C_0 + 4\lambda_{pc}n_{pc}m \end{aligned} \tag{2}$$

$$\begin{aligned} C^{2SE} &= 2\lambda_{pc}n_{pc}(1 - p/N) \;+\; 2\lambda_{npc}n_{npc}p \;+\; 4\lambda_{npc}N_{npc} \;+\; 4\lambda_{pc}n_{pc}m\theta \;+\; F_{RP} \\ &= C_0 + 4\lambda_{pc}n_{pc}m\theta \end{aligned} \tag{3}$$

where $C_0 = 2\lambda_{pc}n_{pc}(1 - p/N) + 2\lambda_{npc}n_{npc}p + 4\lambda_{npc}N_{npc} + F_{RP}$.

5.2 The Comparison

C^{2SE} is never larger than C^{ELT}, because $0 < \theta \leq 1$. From equations (1)-(3), we observe that, to make C^{RES} smaller than both C^{2SE} and C^{ELT} (i.e., $C^{RES} < C^{2SE}$ and $C^{RES} < C^{ELT}$) , the following inequality should apply:

$$\begin{aligned} & C^{RES} < C^{2SE} < C^{ELT} \\ \Rightarrow\;& 2\lambda_{npc}n_{npc}p < 4\lambda_{pc}n_{pc}m\theta < 4\lambda_{pc}n_{pc}m \\ \Rightarrow\;& \xi < 2m\theta/p \leq 2m/p \end{aligned} \tag{4}$$

where $\xi = \lambda_{npc}n_{npc}/\lambda_{pc}n_{pc}$.

The inequality in (4) illustrates the following relationships among the RES, ELT, and 2SE algorithms:

1. When $m = 0$, message costs for the RES algorithm are no better than for either the ELT and 2SE algorithms. In other words, when all transaction hosts do not move, the RES algorithm

offers no advantage over the ELT and 2SE algorithms. In fact, in this case, no additional messages are needed for pc-operations if there is a partitioned copy at the local site, while more messages are required by the RES algorithm than by the ELT and 2SE algorithms for npc-operations.

2. When $\xi = 0$, the RES algorithm always performs at least as well as the ELT and 2SE algorithms in terms of message cost. In this case, no npc-operations are to be executed.

3. When $\xi > 1$, message costs for the RES algorithm are no better than for either the ELT and 2SE algorithms. In fact, when data has been partitioned, p is always equal to or larger than 2 and $2m\theta/p$ or $2m/p$ is no larger than 1. The observation indicates that the number of npc-transactions should not be larger than the number of pc-operations. Note that, when $\xi > 1$, no partitioned-data algorithm offers lower message costs than does the algorithm for non-partitioned data. In this instance, the message costs of npc-transactions (in two or more partitioned copy sites) may offset the message savings made by pc-operations in a partitioned copy site. Non-partitioned data may actually require fewer message exchanges, because any npc-transaction or pc-operation needs at most one round of messages between the remote no-partitioned data server and the transaction server.

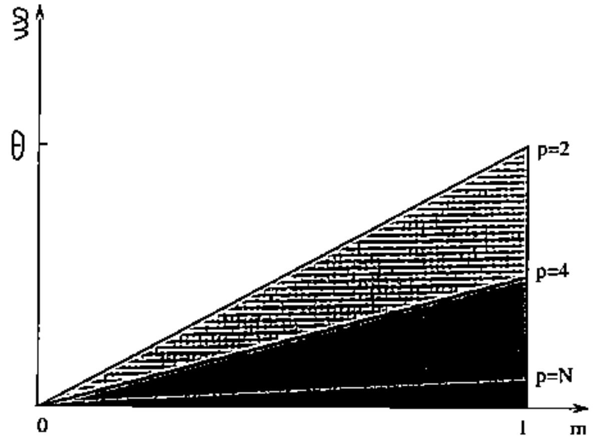

Figure 5: The relation of parameters ξ, m, and p

Figure 5 shows the relation among the parameters m, p, and ξ expressed in inequality (4) as the mobility parameter rises from 0 to 1. The shaded area in the figure indicates all the possible values of parameters ξ and m for a given p for which the the RES algorithm performs better than both ELT and 2SE algorithms in terms of message cost.

5.3 Satisfiability

Assume that C^{CEN} is a message cost function for non-partitioned and centralized algorithm in which no data is partitioned and all pc-operations or npc-operations are sent to a central site that stores data copies. The cost equation can be expressed as:

$$\begin{aligned} C^{CEN} &= C^{CEN}_{pc} + C^{CEN}_{npc} + C^{CEN}_{com} \\ &= 2\lambda_{pc}n_{pc}((N-1)/N) \;\; + \;\; 2\lambda_{npc}n_{npc}((N-1)/N) \;\; + \;\; 2(\lambda_{pc}+\lambda_{npc})((N-1)/N) (5) \end{aligned}$$

where $2\lambda_{pc}n_{pc}((N-1)/N)$ is the expected number of messages per second for pc-operations, $2\lambda_{npc}n_{npc}((N-1)/N)$ for npc-operations and $2(\lambda_{pc}+\lambda_{npc})((N-1)/N)$ for commit operations.

We now examine the *satisfiability* of C^{RES} with respect to C^{CEN} under the condition of inequality (4). Let C^X be a cost function for the algorithm X with parameter vector V. We say C^A is *satisfiable* with respect to C^B if there is a vector value v' in the domain of the parameter V such that $C^A < C^B$ with the vector value v' for the parameter V.

The satisfiability of C^{RES} with respect to C^{CEN} implies that in some situations the RES algorithm will involve lower message overhead than the CEN algorithm. However, if C^{RES} is NOT satisfiable with respect to C^{CEN} under inequality (4), the algorithm RES may not be valuable because it will involve higher message overhead than the CEN algorithm. In other words, in this case, $C^{RES} < C^{CEN}$ can not be satisfied even though $C^{RES} < C^{2SE} < C^{ELT}$ with $\xi < 2m\theta/p \leq 2m/p$. Our expectation would be that the RES algorithm would not only offer lower message costs than the ELT or 2SE algorithms but would also prove superior to the CEN algorithm in non-partitioned data environments.

Let now examine a scenario in which $C^{RES} < C^{CEN}$ under condition (4). Consider a database consisting of a fully partitioned data item X with no resource constraint over two servers (i.e., the constraint can be expressed as $-\infty < X < \infty$, $p = N = 2$, and each server has a partitioned copy). Assume that each pc-transaction has two pc-operations, i.e., $n_{pc} = 2$, and each npc-transaction has only one npc-operation, i.e. $n_{npc} = 1$. In this database, for any transaction accessing data item X, the message cost for repartition protocol is zero; i.e., $C^{RES}_{rpp} = 0$, as any pc-operation is always safe. C^{RES}_{pc} is also equal to zero, as $p/N = 1$. When $\xi = 0$ (i.e., there is no npc-operation), $C^{RES} = 0$ but $C^{CEN} \neq 0$. That is, $C^{RES} < C^{CEN}$. Assume $\xi > 0$. We compute the inequality (i.e., $C^{RES} < C^{CEN}$) from equations (1) and (5) as follows:

$$2\lambda_{pc}n_{pc}(1-p/N) \;\; + \;\; 4\lambda_{npc}n_{npc}p \;\; + \;\; 4\lambda_{npc}N_{npc} <$$

$$2\lambda_{pc}n_{pc}((N-1)/N) \;+\; 2\lambda_{npc}n_{npc}((N-1)/N) \;+\; 2(\lambda_{pc}+\lambda_{npc})((N-1)/N)$$

$$\Rightarrow\; 4\xi p \;+\; 4\xi N_{npc}/n_{npc} \;<\; 1 \;+\; \xi \;+\; 1/n_{pc} \;+\; \xi/n_{npc}$$

$$\Rightarrow\; \xi < 3/28$$

From this computation, we have the inequality $\xi < 3/28$. That is, when $0 < \xi < 3/28$, $C^{RES} < C^{CEN}$ can be satisfied. So, when $0 < \xi < \min(3/28, 2m\theta/p)$, $C^{RES} < C^{RES}$ and $C^{RES} < C^{2SE} < C^{ELT}$. Therefore, we have the following theorem:

Theorem 5.1 *C^{RES} is satisfiable with respect to C^{CEN} with either $\xi = 0$ or $0 < \xi < 2m\theta/p$.*

6 Related Work

Some of the problems involved in supporting transaction services and distributed data management in a mobile environment have been identified recently in [9, 3]. The management of distributed data has been identified in [9] as a research area on which the mobility of host has a large impact. In [3], it is predicted that future applications of mobile computing will demand various transactional and transaction-like services.

A prototype of transaction service for mobile hosts is currently being implemented on the Coda file system [13, 18] to support continued services in a disconnection mode. This prototype uses the *optimistic concurrency control* method presented in [17] to enforce the serializable execution of transactions submitted from mobile hosts. The optimistic concurrency control method is generally suitable for applications, such as those in a file system environment, of low data contention. The prototype, however, did not address the issue of the mobility of transaction hosts and its effect on the management of distributed data.

The impact of mobility on distributed algorithms has recently been investigated in [5]. This research also emphasizes the reduction of the message costs in networks in which a mobile host involved in the execution of distributed algorithms moves across different cells. Unlike the work presented here, that research did not utilize the semantics of data to minimize the message overhead caused by the mobility of hosts.

As stated previously, the notion of using partitioned data to reduce message overhead and increase system throughput in distributed database environments has been investigated in the literature [2, 6, 14, 16, 19, 1]. These efforts address principally the efficient repartition or reconfiguration of a partitioned data item among different sites so that an operation on the data item can be performed at a local site. The research presented here, in contrast, utilizes the partitioned data to efficiently deal with the distribution of operations caused by the mobility of a transaction host. This problem did not arise in a traditional distributed environment with a fixed location of transaction host.

Some commonalities are present between previous work on repartition protocols and our efforts toward a reservation algorithm. The execution of a mobile transaction in the reservation algorithm can be thought of as involving a series of repartition procedures. A reservation action is a repartition procedure that moves a portion of the partitioned data from a local server to a mobile host, and the allocation (or release) action involves a repartition procedure that moves a portion of partitioned data from a mobile host to a local server. Specifically, like the demarcation protocol presented in [6], these repartition procedures are performed by updating the bound variables of a partitioned data item. However, the requirements for reliability procedures are quite different. In the previous approach, a repartition procedure is performed only among relatively reliable distributed servers in fixed networks. The procedure can be executed as a atomic unit. In our work, a repartition procedure is performed between a data server and a mobile host, and the series of repartition procedures for a mobile transaction is executed as an atomic unit. Guaranteeing the atomicity of the series of repartition procedures therefore poses an additional issue, particularly in the development of an atomic protocol which can handle the problems introduced by the failure and extended long disconnections of the mobile host.

In [15], the problem of using the site escrow method (2SE) for mobile transactions was discussed. To avoid the use of a 2PC protocol at commit time, the authors in [15] suggested the Escrow Log Transfering (ELT) method. The method executes a handoff protocol to move the escrow log of a transaction from the server in previous cell to the new server in the current cell before the transaction can continue its execution. This method carries with it a heavy message overhead when a mobile host moves frequently across cells.

Our approach to a pending partition-commutative lock is similar to that explored in [12] for a pending read lock in a replicated database for mobile transactions. In [12], the commutative semantics of read locks are utilized to reduce message overhead for the distributed read operations of a mobile transaction. The partition-commutative lock is applied to prevent a conflicting non-partition-commutative operation from accessing a partitioned copy, while a read lock is used to prevent a write operation from updating a replicated copy.

The issue of termination for a 2PC or 3PC protocol has been well studied (see [8] for details). A server will execute a termination protocol only after it enters the first phase of a 2PC or 3PC protocol but before it receives the commit decision from the coordinator. In our algorithm, a server is always required to execute a termination protocol before it can make an abort decision. The termination protocol in our algorithm, like a 2PC protocol, may be blocked if there is a link failure or a site failure.

7 Conclusions

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. A small movement which crosses network administrative boundaries can result in a much longer path. In a longer network path, communications traverse more intermediaries and consume more network capacity. The mobility of hosts can cause even a short transaction to involve a long communication transmission.

A low message overhead among servers for each operation (including commit and abort) will improve the response time of an operation requested by a mobile host. One benefit of fast response time is that the host will not need to expend precious battery resources while waiting for the acknowledgement of the requested operation.

In this paper, we have addressed the issue of the distribution of operations that update *partitioned data* in mobile environments. We have shown that, for operations pertaining to resource allocation, the message overhead (e.g., for a 2PC protocol) introduced by the distribution of operations is undesirable and unnecessary.

We have introduced a new algorithm, the *Reservation Algorithm* (RA), that does not necessitate the incurring of message overhead for the commitment of mobile transactions. We have discussed two issues related to the RA algorithm: *termination protocol* and *protocol for non-partition-commutative operations.* The algorithm ensures a serializable execution of transactions. We have performed a comparison between the proposed RA algorithm and existing solutions that use a 2PC protocol.

The algorithm proposed in this paper requires the transmission of reservation log information from a mobile host to its current coordinator through a wireless channel when the host decides to commit a transaction. These transmissions do not usually involve additional message exchanges, as they are piggybacked on the commit request message of the transaction. These transmissions can be structured to consume only minimal bandwidth on wireless channels by representing the reservation log by logical operations rather than physical pages.

Although the algorithm discussed in this paper applies directly only to operations over partitioned data, it can be merged into other locking algorithms to support operations on non-partitioned data. For example, in [12], we have shown that, if non-partitioned data are replicated among different servers, then a read unlock for an non-partitioned data item can be executed at any copy site, including sites other than that on which the read lock is set. The locking schema utilizes the replicated copies of data items to reduce the message costs incurred by the mobility of the transaction host. Therefore, the reservation algorithm can be augmented with the locking schema to support operations on both partitioned and non-partitioned data. Such an augmented algorithm can improve the efficiency of concurrency control protocols in a mobile environment if the number of read operations on non-partitioned data and pc-operations on partitioned data dominates that of write operations on non-partitioned data and npc-operations on partitioned data.

Finally, we note that the algorithm presented in this paper is a pessimistic concurrency control protocol. Each pc-operation obtains reserved resources when it is invoked by a mobile host. The message overhead which arises from the mobility of hosts can obviously be avoided through a more optimistic approach which defers reservation actions until commit time. Increasingly optimistic approaches, however, carry with them increasingly high transaction abort rates, offering a tradeoff between the message overhead and the abort rate. The choice of a pessimistic or an optimistic approach therefore depends on the requirements of specific applications and the parameters of system environments.

References

[1] G. Alonsa and A. E. Abbadi. Partitioned data objects in distributed databases. *the International Journal on Distributed and Parallel Databases*, 3(1), 1995.

[2] R. Alonso, D. Barbará, and H. Garcia-Molina. Data caching issues in an information retrieval system. *ACM Transactions on Database Systems*, 15(3):359–384, September 1990.

[3] 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.

[4] A. Asthana, M. Cravatts, and P. Krzyzanowski. An indoor wireless system for personalized shopping assistance. In *1994 Workshop on Mobile Computing Systems and Applications*, 1994.

[5] 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.

[6] D. Barbará, , and H. Garcia-Molina. The demarcation protocol: A technique for maintaining constraints in distributed database systems. *The VLDB Journal*, 3(3):325–354, 1994.

[7] D. Barbará 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.

[8] P. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Databases Systems.* Addison-Wesley Publishing Co., 1987.

[9] T. Imielinski and B. R. Badrinath. Wireless mobile computing : Challenges in data management. *Communication of ACM*, 37(10), 1994.

[10] R. Jain and N. Krishnakumar. Network support for personal information services to pcs users. In *Proceedings of the IEEE conference on Networks for Personal Communications (NPC '94)*, Long Branch, NJ, Mar. 1994.

[11] R. Jain and N. Krishnakumar. Service handoffs and virtual mobility for delivery of personal information services to mobile users. Technical Report TM-24696, Bellcore, Dec. 1994.

[12] J. Jing, O. Bukhres, and A. Elmagarmid. Distributed lock management for mobile transactions. In *Proc. of the 15th International Conference on Distributed Computing Systems*, Vancouver, Canada, June 1995.

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

[14] N. Krishnakumar and A. J. Bernstein. High throughput escrow algorithm for replicated databases. In *Proceedings of the Eighteenth International Conference on Very Large Data Bases*, 1992.

[15] N. Krishnakumar and R. Jain. Protocols for maintaining inventory databases and user service profiles in mobile sales applications. In *Proceedings of the Mobidata Workshop*, Rutgers University, Nov. 1994.

[16] A. Kumar and M. Stonebraker. Semantics-based transaction management techniques for replicated data. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, 1988.

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

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

[19] N. Soparkar and A. Siberschatz. Data-value partitioning and virtual messages. In *Proceedings of the Conference on Principles of Database Systems*, 1990.

[20] 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.