Wireless Networks 3 (1997) 235–246


# Escrow techniques for mobile sales and inventory applications *

Narayanan Krishnakumar [a,**] and Ravi Jain [b]

[a] *Fidelity Investments, Mail Zone H4A, 82 Devonshire St., Boston, MA 02109, USA*
[b] *Applied Research, Bellcore, 331 Newman Springs Rd., Red Bank, NJ 07701, USA*

We address the design of architectures and protocols for providing mobile users with integrated Personal Information Services and Applications (PISA), such as personalized news and financial information, and mobile database access. We present a system architecture for delivery of PISA based on replicated distributed servers connected to users via a personal communications services (PCS) network. The PISA architecture partitions the geographical coverage area into *service areas*, analogous to PCS registration areas, each of which is served by a single local server. When a user moves from one service area to another, the service is provided by the new local server. This is accomplished by a service handoff, analogous to a PCS call handoff, which entails some context information transfer from the old to the new server. We focus on the mobile sales and inventory application as an example of a PISA with a well-defined market segment. We design a database management protocol for supporting both mobile and stationary salespersons. Our design uses the *site-transaction escrow* method, thus allowing faster responses to mobile clients, minimizing the amount of context information which must be transferred during a service handoff, and allowing mobile clients to operate in disconnected mode by escrowing items on their local disks. We develop a formal model for reasoning about site-transaction escrow, and develop a scheme for performing dynamic resource reconfiguration which avoids the need for time-consuming and costly database synchronization operations (i.e., a two-phase commit) when the mobile sales transaction completes. A further refinement to the scheme avoids an $n$-way two-phase commit during resource reconfiguration operations, replacing it with several simpler two-phase commits.

## 1. Introduction

An important and challenging area of mobile information systems is the design of architectures and protocols for providing mobile users with Personal Information Services and Applications (PISA). Examples of PISA include personalized financial and stock market information, electronic magazines, news clipping services, traveler information, as well as mobile shopping, banking, sales, inventory, and file access. As a concrete running example in this paper, we use mobile database access, and in particular, the mobile sales and inventory application (see section 2).

We consider the situation in which PISA are primarily provided by a commercial entity called the Information Service and Applications Provider (ISAP). The ISAP maintains a set of servers which contain the appropriate information and run applications, and which are connected to the mobile user via a personal communications services (PCS) network.

The mobile user's terminal runs application software to interact with the ISAP. These interactions are divided into logical, application-dependent segments called *sessions*. For example, in the case of the mobile sales and inventory application, a session may consist of the salesperson connecting to a remote server, downloading images and text describing a product, and then logging out. Alternatively, a session may consist of a user running an inventory transaction against the corporate database. Sessions may be initiated by the user or by the ISAP. It is desirable that when a session is in progress, the user is not aware of any disruption in service as the user moves. (Note that continuity in sessions need only be maintained at the logical level of session or application software: it is not necessary that the physical wireless link or the mobile user's terminal be continuously in use throughout the session.)

In order to meet reliability, performance and cost objectives when the number of users is large and geographically dispersed, a distributed server architecture will be necessary. In section 3 we describe the system architecture for mobile sales and inventory applications where the ISAP is organized as a distributed database (possibly replicated in parts) and uses the underlying PCS network for communicating with the user. As the user moves or network load and availability changes, the server interacting with the user may need to change. Thus, real mobility on the part of the user may result in the *virtual mobility* of the server. This is accomplished by means of a *service handoff* which is broadly analogous to a PCS call handoff. We have previously designed a service handoff protocol [10,11] and described the context information which must be transferred from the old to the new server for various classes of applications, including mobile transactions.

In section 4 we describe how the semantics of the sales application can be exploited to provide an appropriate database design with escrow protocols. We assume that the reader has some familiarity with the principles of database transactions and distributed databases [4,8], but in sections 4.1.1 and 4.1.2 we provide some background information on concurrency control for database transactions and escrow techniques for managing replicated data. We argue

* A portion of this research has appeared in preliminary form, in the proceedings of the *MOBIDATA workshop*, Rutgers University, New Brunswick, NJ, November 1 1994.

** This work was done while the author was employed at Bellcore.

that the site-transaction escrow technique is more suitable than traditional database locking schemes for the mobile sales and inventory application. The site-transaction escrow technique increases transaction throughput at the server thus allowing faster responses to mobile clients, minimizes the amount of information which must be transferred during a service handoff, and allows mobile clients to operate in disconnected mode by escrowing items on their local disks.

In section 4.3 we present a formal model of site-transaction escrow, focusing on the non-mobile case. In section 5 we extend our model to include mobile transactions using site-transaction escrow. We describe three possible implementations of the site-transaction escrow technique to support mobile users, and discuss their trade-offs. We show that, as desired, Scheme 1 makes service handoff simpler and reduces the amount of context information that must be transferred, compared to a traditional locking technique. Scheme 1 however has a two-phase commit operation at the end of the mobile sales transaction. We thus develop Scheme 2, which avoids this final two-phase commit operation by introducing two new resource reconfiguration operations, Mobile Allocate Reconfigure (MAR) and Mobile Deallocate Reconfigure (MDR). The mobile reconfiguration operations might result in "lost" updates if failures occur. To remove this possibility, Scheme 2A adds in a pairwise two-phase commit (rather than an $n$-way two-phase commit) with each mobile reconfiguration. We end with some concluding remarks in section 6.

## 2. Application scenario: Mobile sales and inventory

We consider a scenario in which the user is a mobile salesperson selling financial products (like insurance, bonds, etc.), or consumable products (e.g., doctor's office supplies like gloves, syringes, etc.). The ISAP is either the company the salesperson works for, whose products are being sold, or a third-party supplier of information. The salesperson uses a personal digital assistant (PDA) as a mobile database when discussing and completing sales. The salesperson is also referred to as the user of the ISAP's services. The PDA or other end equipment which the salesperson uses is called the mobile or the client of the ISAP's servers. The person to whom the sale is being made is referred to as the customer.

The user visits numerous customer offices during the course of a day and discusses their requirements, shows images of products, initiates new orders or queries about the status of previous orders, etc., using the PDA. The mobile database contains customer records as well as information regarding policies, prices and availability of the product being sold. The time available to the salesperson for meeting with the customer may be extremely limited [20]. In order to ensure that this time is utilized most effectively, the mobile database must have current information available about dynamically varying quantities such as inventory levels, delivery dates, etc. Thus, the mobile database is periodically updated from the ISAP's database. The user has a service profile stored with the ISAP which ensures that the mobile database is updated at appropriate times with the appropriate information. Orders or queries placed by the salesperson are transmitted to the ISAP database. Observe that this is not a far-fetched scenario. Mobile sales applications are already being tested and marketed [20,21].

## 3. System architecture

In general, mobile users will access private and corporate databases which, for reliability, performance and cost reasons will necessitate a *distributed server* architecture when the number of users and their geographic dispersion becomes large. In a distributed server architecture the information is (partially) replicated across multiple interconnected servers that function as a single logical information base.

We describe a possible system architecture using figure 1. The PCS system consists of the Public Switched Telephone Network (PSTN) connected to the network of some PCS service provider; these are indicated by the large dashed boxes in the top and bottom half of figure 1, respectively. Two ISAP servers are shown. There are several ways to interconnect the servers, e.g., using a private ISAP network or using the PCS network itself, but for simplicity we omit the ISAP network from the figure.

We have shown only the relevant signaling network elements of the PSTN and PCS Provider Network. In the case of the PSTN, we have assumed a signaling network architecture based on the Advanced Intelligent Network (AIN) and Signaling System 7 (see [15] for a tutorial). The PSTN signaling network consists of end-office telephone exchange switches called Service Switching Points (SSP) connected via signaling links to a hierarchy of signaling packet switches called Local Signal Transfer Points (LSTP) and Regional STPs (RSTP). A two-level hierarchy of databases called the Home Location Register (HLR) and Visitor Location Register (VLR) is used to locate and deliver calls to the user (see [12] for a tutorial); these are connected to the RSTP and co-located with the SSPs, respectively.

The PCS Provider Network consists of a Mobile Switching Center (MSC) connected to several Base Station Controllers (BSC), each of which is connected to several radio Base Stations (BS) which provide the actual wireless link to the end client device. The geographical coverage area for the information service is partitioned into *service areas*, analogous to PCS registration areas. It is likely that a service area will cover several PCS registration areas. Each service area is served by a single information server, called the *local server*, analogous to the PCS network's Mobile Switching Center (MSC) or VLR database. The connection between the ISAP and the mobile user can be set up by either side dialing the other's non-geographic telephone number.

We assume that the PCS network ensures that the physical connection between the user and the ISAP is maintained

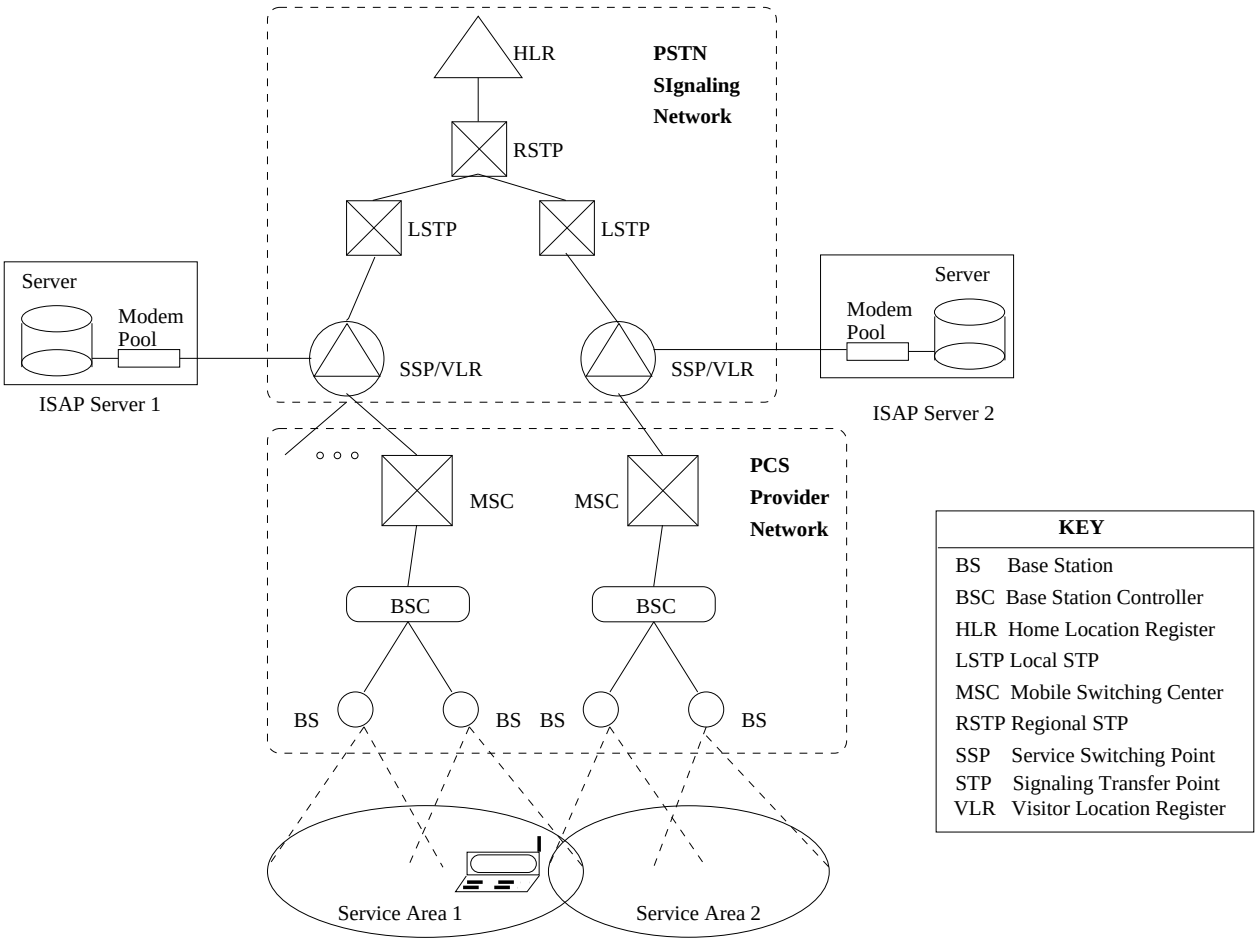



Figure 1. An example system architecture model.

without interruption during a session as the user moves, via appropriate mobility management and handoff procedures [5,12,16]. We also assume that appropriate protocols are used for efficient and reliable wireless communication; these could include a version of TCP/IP modified to deal with the idiosyncrasies of the wireless link [2], and/or a connection-oriented link-level protocol such as LAPR [19] running on top of a PCS data protocol. An example of the protocol stack running on the client is shown in figure 2. Note that several other variations are possible, including commercial packet radio services such as RAM and ARDIS or CDPD service [17], instead of the PCS data protocol. The development of efficient and reliable protocol stacks (TCP and below) for wireless data communication is an area of active research, and outside the scope of this paper; we focus on the higher-layer protocols for mobile wireless database access.

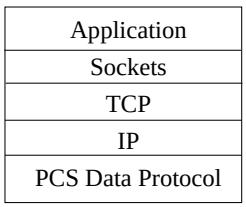



Figure 2. An example communication protocol stack.

As the user moves out of one service area into another, it is desirable that the local server at the new service area take over providing the service. This *service handoff* for the *virtual mobility* of the server is broadly analogous to the PCS call handoff procedure (except that it occurs between ISAP servers rather than PCS base stations), and also has the requirement that service appear to continue transparently without interruption. In [10,11], we have described protocols and capabilities required in both the ISAP and the PCS network to implement service handoffs. Briefly, a service handoff consists of a transfer of context information from the old server to the new server, followed by a physical connection transfer between the old and new servers. The context information depends upon the application in progress, but in general serves to inform the new server of where to pick up the session after the old server left off. For example, if the mobile user was simply reading through a file, the context information would be the name of the file and a pointer (e.g., line number or byte position) from where the new server should start sending information to the mobile.

## 4. Escrow-based replica control for partitionable data

We now describe the design of the ISAP's database. The design choices are motivated by the need to accommodate both stationary and mobile users, while minimizing aspects specific to mobile users.

### *4.1. Background*

We first provide background on how transactions may typically be handled in traditional distributed database environments, without considering mobility. (See [4,8] for further background.)

#### *4.1.1. Locking techniques for concurrency and replica control*

A transaction contains a series of operations (e.g., read a datum, write a datum, etc.). In a transaction processing environment, transactions can concurrently access shared (possibly replicated) data. Therefore, their execution has to be carefully controlled so that correctness is preserved. The traditional notion of correctness is *serializability* [4], i.e., the effect of the interleaved operations of concurrent transactions is the same as that produced by a sequential execution of the transactions. The algorithms used to ensure serializability are also referred to as concurrency control protocols.

One example of a concurrency control protocol is *strict two-phase locking*. In this protocol, a transaction acquires a *read lock* (*write lock*) on a data item before reading (writing) that item. Two locks on a data item are conflicting if either is a write lock, and a transaction may acquire a lock only if no other transaction holds a conflicting lock. This ensures that there is only one writer, but there can be multiple readers if there is no concurrent write. Furthermore, a transaction cannot acquire any more locks after it releases a lock. This defines a two-phase execution for a transaction with respect to the locks, where first there is a *growing phase* when locks are acquired, and then there is a *shrinking phase* when all the locks are released. All locks can be released only at when the transaction commits or aborts, ensuring that the transactions are serializable in the order in which they release locks.

The corresponding notion of correctness for replicated data (where copies of the same data item are stored at several servers) is *one-copy* serializability: the effect of the execution of a set of transactions on the replicated data is equivalent to some serial execution of those transactions on a single copy. The Quorum Consensus (Locking) Algorithm [7,9,23] can be used to preserve this property – a read operation on a data item locks the data item at a subset of replicas, called a *read quorum*, and a write operation locks the data item at a *write quorum* of replicas. One-copy serializability is guaranteed if the read and write quorums intersect.

In a replicated system, a transaction executes at a single replica; however locks might be obtained at several sites and updates might have to be installed at several sites at the end of the transaction. A co-ordination protocol is required to ensure that the transaction either commits or aborts at all sites. This co-ordination protocol is the *two-phase commit* protocol: a co-ordinator sends a prepare message to participating replicas, upon which each replica votes whether it can commit or not. If all votes are affirmative, the co-ordinator sends a commit message to the replicas, and an abort message otherwise.

The two-phase commit protocol has some disadvantages:

(a) it requires all the participants to be available at one point of time and vote yes if the transaction has to commit – any failure by any participant results in the transaction being aborted;

(b) it is a blocking protocol, i.e., if the co-ordinator fails during some window of time, the participants have to wait for the co-ordinator to recover before a decision to commit or abort can be made;

(c) it requires at least two rounds of messages between the co-ordinator and the participants before commit or abort of the transaction.

#### *4.1.2. Escrow techniques for handling replicated data*

It is possible to exploit the semantics of the sales application in order to improve the system throughput, in particular by the idea of placing instances of the item being sold in *escrow*. This scheme allows data items to be locked for small intervals of time and also avoids the two-phase commit, thereby increasing throughput. In general, an escrowable resource item refers to a resource whose instances are indistinguishable, so that the instances can be partitioned, either among transactions or among sites in a replicated database (as we see shortly).

Suppose the salesperson is selling items from inventory, where each instance of the item is indistinguishable from the others (e.g., the item is a medical supply item, and each instance is, say, one box of the item). Let $\text{Total}_m$ be a (replicated) datum in the database that indicates the total number of instances of item $m$ in stock. As sales orders are taken or canceled, salespersons launch transactions which update the number of instances sold, $\text{Sales}_m$. The problem is to ensure that the constraint $\text{Sales}_m \leqslant \text{Total}_m$ is satisfied at all times. Updates to $\text{Sales}_m$ will typically be made as operational updates instead of value updates, i.e., instead of reading and writing the actual value of the variable $\text{Sales}_m$, transactions will issue operations to increment or decrement it.

As discussed previously, if two or more salespersons launch long-running transactions which contain operational update operations, a traditional concurrency control algorithm would require that the variable $\text{Sales}_m$ be locked by each transaction, so that one transaction cannot begin until the other commits and releases the lock. In a replicated system, this further entails using a distributed protocol such as quorum locking [9] and then a two-phase commit protocol

to ensure consistency. Escrow techniques attempt to avoid locking the variable for the entire transaction and the two-phase commit. We briefly describe several escrow schemes that have been proposed.

*Transaction escrow* [18]. A transaction executes an *escrow* operation to try to place in reserve the resources that it will (potentially) use. All successful escrow operations are logged in an *escrow log*. Before executing an escrow operation, each transaction accesses the log and sees the total escrow quantities of all uncommitted transactions. The transaction then makes a worst-case decision to determine whether it can proceed.

For instance, suppose $\text{Total}_m = 100$, $\text{Sales}_m = 20$, and there are currently two uncommitted transactions each requesting one item. Let transaction $T$ wishing to reserve ten items now be initiated. Since the log indicates that $\text{Sales}_m \leqslant 22 \leqslant \text{Total}_m$, $T$ can proceed irrespective of whether the other two transactions commit or abort: the constraint is maintained in any case. Note that when transaction $T$ executes an escrow operation, $T$ obtains a short-term lock on the escrow log to access and update the log and releases the lock after the log has been updated. (The lock release need not wait for the commit or abort of $T$ as in a traditional transaction execution.) Thus any other transactions which access $\text{Sales}_m$ are forced to wait only for the duration of the log update operation, rather than for the entire duration of $T$ as would occur in a traditional scheme, thus increasing system throughput.

*Site escrow.* In site escrow algorithms [1,14,22], the total number $\text{Total}_m$ of available instances of item $m$ is partitioned across the number of sites (servers) in the system. This can be thought of as each site (as against a transaction) holding a number of instances in escrow. A transaction launched by a user runs at only one site, and can successfully complete at a site only if the number of instances it requires does not exceed the number of instances available in escrow at that site. Each operation of the transaction acquires a lock at the site when accessing the item, just as for a traditional locking scheme. However, this lock is different in two important respects. Firstly, the lock is *local* in the sense it applies only to the site where the operation is executing, and is designed to protect the operation from other transactions executing at that site. (This contrasts with traditional replica control schemes, such as quorum locking, which would require each site to lock a subset of the other sites before proceeding with the operation, and would also require a distributed two-phase commit at the end of the transaction.) Secondly, *the lock can be released on successful completion of the operation*, in contrast to traditional strict two-phase locking where the lock is released only at commit time of the entire *transaction*. By allowing each site to deplete its own escrowed instances without consulting other sites, avoiding the distributed two-phase commit, and shrinking the interval during which items are locked, the site escrow model results in higher autonomy to sites and greater throughput.

The number of instances held in escrow at each site is adjusted to reflect the consumption of instances by the transaction only if it commits; otherwise the escrow is restored to its original state. When one site requires more instances, a *redistribution* or *reconfiguration protocol* such as the point-to-point demarcation protocol [3] or a dynamic quorum-based protocol [13] is executed, so that the site can get a portion of some other sites' unassigned instances.

*Site-transaction escrow.* The two escrow schemes described above can also be combined [13]. Thus the total number $\text{Total}_m$ of available instances of item $m$ is partitioned across sites, and in addition, each transaction at a site uses a transaction escrow scheme to allocate and deallocate resources at that site. Once again, a reconfiguration protocol is used to transfer resource instances between sites as necessary.

### 4.2. Site-transaction escrow for mobile data access

The site-transaction escrow scheme provides an elegant and efficient replica control mechanism for partitionable resources, and allows sites to make allocation decisions locally as far as possible. This technique is desirable in the mobile environment due to the following reasons:

1. A mobile is usually powered by a limited power source. Suppose a mobile has established a session with a server and is trying to allocate resources. If that server could possibly allocate the resources locally, this would enable quick response to the mobile and hence less power is consumed while idling.

2. When performing service handoffs (as seen in section 5), the site escrow model permits less context information to be transferred than when a traditional concurrency/replica control protocol is used. This results in quicker service handoffs and savings in cost.

3. The wireless bandwidth between the mobile and the ISAP server is limited. Thus instead of remaining in contact with a server at all times, it might be desirable for the mobile to itself escrow some instances and allocate them locally. The site-transaction escrow scheme permits this alternative naturally.

### 4.3. A formal model of site-transaction escrow

We now develop a simple formal model for site-transaction escrow. This is based upon the model developed by Alonso and El Abbadi [1], who described variations on site escrow and applied it to non-mobile environments. In this section we extend their approach to formalize site-transaction escrow model for the non-mobile environment. In the next section we will extend the model further to accommodate mobility; we will see that relatively few enhancements are needed for this.

*4.3.1. The basic model*

Let $\mathcal{S}$ denote the set of servers in the replicated system, $t_m$ denote the total number of instances of a particular item $m$ in the system, and let $m_s$ be the number of instances at site $s$, for all $s \in \mathcal{S}$. Then $t_m = \sum_{s \in \mathcal{S}} m_s$. Let Capacity$_m$ denote the maximum number of instances of type $m$ that can be in the system. Such constraints are common in many applications; e.g., Capacity$_m$ reflects the maximum inventory space in the warehouse. Similarly, there may be a lower bound, e.g., the inventory is not allowed to fall below MinStock$_m$ in order to satisfy emergency requests. $\mathcal{F}_m$ is a predicate over $m_s$ (for all $s \in \mathcal{S}$), Capacity$_m$ and MinStock$_m$; it is used to specify the correctness (safety) condition for item $m$, i.e., the escrow scheme must ensure that $\mathcal{F}_m$ is true at all times. $\mathcal{F}_m$ is restricted to be a conjunction of terms of the form $a$ *op* $b$ where *op* is an arithmetic operator (e.g., $=$, $\leqslant$, $\neq$, etc.) and $a$ and $b$ are permitted to be natural numbers or $t_m$.

For example, suppose there are three sites which escrow instances of $m$, i.e., $\mathcal{S} = \{j, k, l\}$. Then at any given time, $t_m = m_j + m_k + m_l$. Let Capacity$_m$ be 100, and suppose MinStock$_m = 5$. The correctness condition $\mathcal{F}_m$ is given by

$$\mathcal{F}_m \equiv 5 \leqslant t_m \wedge t_m \leqslant 100. \tag{1}$$

(For ease of exposition, we will henceforth assume there is only one data item and drop the $m$ subscript when possible.)

For each $s \in \mathcal{S}$, define the *configuration set* $C_s$ as a predicate denoting the range of values that can be taken by $m_s$, e.g., $C_j \equiv 0 \leqslant m_j \leqslant 2$. A *system configuration* is denoted by $\mathcal{C} = \bigwedge_{s \in \mathcal{S}} C_s$. A system configuration is said to be *valid* if it satisfies the correctness condition $\mathcal{F}$, i.e., if $\mathcal{C} \Rightarrow \mathcal{F}$.

For example, suppose there are Total $= 90$ resource instances initially. These instances could initially be (site-) escrowed as 28 instances to site $j$, 25 to site $k$ and 37 to site $l$. Suppose further that the system-wide constraint on the system is $\mathcal{F}$ as above. An initial valid configuration with respect to $\mathcal{F}$ in equation (1) could be

$$C' \equiv (3 \leqslant m_j \leqslant 30) \wedge (0 \leqslant m_k \leqslant 30) \wedge (2 \leqslant m_l \leqslant 40).$$

An example of an invalid configuration with respect to $\mathcal{F}$ in equation (1) is

$$C'' \equiv (1 \leqslant m_j \leqslant 30) \wedge (2 \leqslant m_k \leqslant 30) \wedge (1 \leqslant m_l \leqslant 40),$$

since it is possible that $t = m_j + m_k + m_l = 4$. Informally, the configuration sets in a valid configuration indicate a set of local predicates such that together they preserve the validity of the global predicate $\mathcal{F}$. This model thereby represents two levels of partitioning: (a) the total number of resource instances being partitioned (escrowed) among the various sites, and (b) the partitioning of the minimum stock and capacity constraints (the upper and lower bounds on the total number of resources).

*4.3.2. The site-transaction escrow model*

We now extend the above model to include the notion of transaction escrow [18] and thus formalize site-transaction escrow. Two cases arise: when local resources suffice for a given transaction, and when they do not.

*Local resources suffice.* Define the *escrow value set* $E_s$ as a predicate denoting the upper and lower bounds on the actual value of $m_s$ as it is being accessed by escrow operations. Initially, $E_j \equiv (x \leqslant m_j \leqslant x)$, where $x$ is the number of resource instances assigned to server $j$, and at some point in time $E_j \equiv (u \leqslant m_j \leqslant v)$. $E_j$ can change as follows:

1. If $y$ resource instances are escrowed for allocation by a transaction $T$ at server $j$, $E_j$ becomes $(u - y) \leqslant m_j \leqslant v$. If $T$ commits, $E_j$ becomes $(u - y) \leqslant m_j \leqslant (v - y)$. If $T$ aborts, then $E_j$ becomes $u \leqslant m_j \leqslant v$.
2. Suppose $y$ resource instances are escrowed for de-allocation by $T$ at server $j$. $E_j$ becomes $u \leqslant m_j \leqslant (v + y)$. If $T$ commits, then $E_j$ becomes $(u + y) \leqslant m_j \leqslant (v + y)$. If $T$ aborts, $E_j$ becomes $u \leqslant m_j \leqslant v$.

The configuration set differs from the escrow value set. The former defines the possible range of values for $m_j$ at all points of time and is a correctness condition. The escrow value set asserts the range of values within which $m_j$ lies at a given point of time.

Given a valid configuration, $\mathcal{C}$, with configuration set $C_s$ for each $m_s$, an operation $o_s$ executed at server $s$ is defined to be *safe* [3] if for the resultant escrow value set $E_s$, $E_s \Rightarrow C_s$. For example, suppose $E_1$ is $3 \leqslant m_1 \leqslant 5$ and $C_1$ is $0 \leqslant m_1 \leqslant 30$. Then $o(m_1)$ that allocates 4 instances of $m$ is an unsafe operation, whereas $o'(m_1)$ that allocates 2 instances of $m$ is safe.

**Observation 1.** For a valid configuration $\mathcal{C}$,

$$\bigwedge_{s \in \mathcal{S}} (E_s \Rightarrow C_s) \Rightarrow \mathcal{F}.$$

*Resource reconfiguration required.* Observation 1 states that as long as operations executed at each server are safe, $\mathcal{F}$ is preserved. However, one might not be able to run safe operations all the time, since for instance, server $s$ might want to allocate or de-allocate more resources than it is allowed to. Suppose at some point of time in a valid configuration, $C_s$ is $x \leqslant m_s \leqslant y$ and $E_s$ is $x' \leqslant m_s \leqslant y'$. Suppose a transaction $T$ wishes to execute allocate($z$), i.e., allocate $z$ resources at $s$, and suppose allocate($z$) is not safe. Thus $((x' - z) \leqslant m_s \leqslant y') \not\Rightarrow (x \leqslant m_s \leqslant y)$.

It is possible that allocate($z$) can be made safe at $s$ by doing an *allocate reconfiguration operation*, AR($z$), which modifies $C_s$ as follows.

**Definition of** AR($z$). Suppose there exists $z'$ such that

$$\big((x' - z) \leqslant m_s \leqslant y'\big) \Rightarrow \big((x - z') \leqslant m_s \leqslant y\big).$$

Furthermore, suppose that for some server $t$,

(a) $C_t$ is $u \leqslant m_t \leqslant v$,

(b) $E_t \Rightarrow C_t$,

(c) $(u + z') \leqslant v$, and

(d) for $E_t$ at $t$, $E_t \Rightarrow ((u + z') \leqslant m_s \leqslant v)$.

Thus $(u+z') \leqslant m_s \leqslant v$ is a possible configuration set at $t$. Therefore, if $C_s$ is modified to be $((x - z') \leqslant m_s \leqslant y)$ and $C_t$ is modified to be $((u + z') \leqslant m_s \leqslant v)$, then allocate($z$) is safe at $s$.

Operationally, the reconfiguration operation AR results in $z'$ more resource instances being made available for allocation at server $s$ by altering the lower bound on allocations at $s$.

**Example 1.** Suppose $C_1$ is $(3 \leqslant m_1 \leqslant 10)$ and $E_1$ is $(6 \leqslant m_1 \leqslant 6)$ (i.e., the value of $m_1$ is exactly 6). Let $C_2$ be $(4 \leqslant m_2 \leqslant 11)$ and $E_2$ be $(9 \leqslant m_2 \leqslant 10)$. The operation, allocate(4), if initiated at server 1, is an unsafe operation. It is possible to do an AR operation such that the new $C_1$ becomes $(2 \leqslant m_1 \leqslant 10)$ and the new $C_2$ $(5 \leqslant m_2 \leqslant 9)$. This allows the safe allocation of 4 resource instances at server 1.

Now suppose a transaction $T$ wishes to execute de-allocate($z$) at $s$, i.e., de-allocate $z$ resources at $s$, and suppose de-allocate($z$) is not safe. Thus

$$\big(x' \leqslant m_s \leqslant (y' + z)\big) \not\Rightarrow (x \leqslant m_s \leqslant y).$$

A similar de-allocate reconfiguration operation can be provided, as follows.

**Definition of** DR($z$). Modify $C_s$: Suppose there exists $z'$ such that

$$\big(x' \leqslant m_s \leqslant (y' + z)\big) \Rightarrow \big(x \leqslant m_s \leqslant (y + z')\big).$$

Furthermore, suppose that for some server $t$,

(a) $C_t$ is $u \leqslant m_t \leqslant v$,

(b) $E_t \Rightarrow C_t$,

(c) $u \leqslant (v - z')$, and

(d) for $E_t$ at $t$, $E_t \Rightarrow (u \leqslant m_s \leqslant (v - z')$.

Therefore, if $C_s$ is modified to be $(x \leqslant m_s \leqslant (y + z'))$ and $C_t$ is modified to be $(u \leqslant m_s \leqslant (v - z'))$, then de-allocate($z$) is safe at $s$.

Several policies [3] can be used to determine the parameters of a reconfiguration operation, namely the number of resource instances logically transferred, or which servers participate in a reconfiguration, etc. Furthermore, it is assumed that the changes to $C_s$ and $E_s$ (and likewise $C_t$ and $E_t$) are made atomically at $s$ (likewise, $t$) during the reconfiguration. Note again that a two-phase commit is not required to accomplish the reconfiguration.

We now motivate the need for a new kind of reconfiguration operation apart from the two discussed above. We have so far dealt only with a global constraint being partitioned as local configuration sets. Suppose there are also local site constraints (on these configuration sets), such as the lower bound on $m_1$ should not drop below 2. The following example illustrates how AR and DR are not sufficient to allow desirable reconfigurations.

**Example 2.** Suppose $C_1$ is $(3 \leqslant m_1 \leqslant 10)$ and $E_1$ is $(7 \leqslant m_1 \leqslant 7)$. Let $C_2$ be $(4 \leqslant m_2 \leqslant 11)$ and $E_2$ be $(9 \leqslant m_2 \leqslant 10)$. Suppose we require that the lower bound of $C_1$ should be at least 2. Let allocate(7) be initiated at server 1. This operation is not safe at server 1. It can also be seen that no AR or DR operation can make the operation safe at server 1.

An alternative is to use the operation XFER described below, which accomplishes allocate($z$) by (logically) transferring some $z'$ instances from $t$ to $s$.

**Definition of** XFER($z$). Modify $E_s$: Suppose there exists $z'$ such that

$$\big((x' + z' - z) \leqslant m_s \leqslant (y' + z')\big) \Rightarrow (x \leqslant m_s \leqslant y).$$

Furthermore, suppose for some server $t$,

(a) $E_t$ is $(u' \leqslant m_t \leqslant v')$, and

(b) $((u' - z') \leqslant m_t \leqslant (v' - z')) \Rightarrow C_t$.

Therefore, if $E_s$ is modified to be $((x' + z') \leqslant m_s \leqslant (y' + z'))$ and $E_t$ is modified to be $((u' - z') \leqslant m_s \leqslant (v' - z'))$, then allocate($z$) is safe at $s$.

In example 2, one can do a XFER operation such that $E_1$ becomes $(10 \leqslant m_1 \leqslant 10)$ and $E_2$ becomes $(6 \leqslant m_2 \leqslant 9)$.

An operation $o(x)$ at server $t$ is said to be *unsafe* but *solvable* if the resultant escrow value set is $E'_{m_t}$ and $E'_{m_t} \not\Rightarrow C_{m_t}$, but there exists a sequence of reconfiguration operations (with possibly several sites) leading to another valid configuration $\mathcal{C}'$ in which $o(x)$ is safe. Thus, if an operation is submitted at a server and determined to be unsafe, the server can try to execute some reconfiguration operations with one or more servers such that the operation would be safe in the new configuration. An operation that is neither safe, nor unsafe and solvable is said to be *unsolvably unsafe*. Unsolvably unsafe operations cannot be successfully executed.

**Observation 2.** A reconfiguration operation preserves $\mathcal{F}_m$.

**Observation 3.** Given a set of operations, $O$, such that each operation is either safe or unsafe but solvable, $\mathcal{F}_m$ is preserved by the execution of these operations.

In summary, given a non-mobile environment as above, we note that there are five operations of interest in the site-transaction escrow protocol, namely (a) safe allocate operations, (b) safe de-allocate operations, (c) allocate reconfiguration operations, AR (d) de-allocate reconfiguration operations DR, and (e) transfer reconfiguration operations, XFER, that allow unsafe but solvable operations to proceed safely.

## 5. Mobile sales transactions

We now see how sales transactions run from a mobile at a server can be handled. In section 5.1 we consider the impact of different types of mobility on the database protocols discussed above. In section 5.2 we discuss the impact of supporting continuously mobile users with a traditional locking scheme, and in section 5.3 we discuss three possible implementations of our site-transaction escrow technique.

### *5.1. Types of mobility*

*Discrete mobility.* Suppose a user makes some sales at one service area, completes all transactions, closes the session, disconnects from the PCS network and moves to another service area. The user can now start another session with the local server of the new service area and make further sales. The sales transactions at the new service area can simply operate on the escrow values stored at the new server. The decision as to whether any operation performed by the transaction is safe or unsafe can most likely be made by the local server (and if unsafe and solvable, the server can do a reconfiguration operation to permit the operation). No change is required to accommodate mobility.

*Local and disconnected mobility.* Suppose the salesperson has an estimate of the number of resource instances of each kind he or she expects to sell. He or she could then dynamically make his mobile unit a (mobile) server, and run a reconfiguration operation with the fixed server to logically transfer some resource instances on to the mobile. Thus the mobile can itself function as a local server from this point on, and allow local transactions which the salesperson initiates. This would result in faster response times for the salesperson and the salesperson can continue selling items even while being (voluntarily or involuntarily) disconnected from the ISAP. At a later point in time, the salesperson could run another redistribution operation so that remaining resource instances on the mobile are logically transferred back to the local ISAP server. The partitioning concept of site escrow thereby permits a seamless incorporation of disconnection.

Notice that, while being disconnected, only safe operations can be run on the mobile. One drawback of the scheme above is that a fixed server that requires instances would be unable to run a reconfiguration operation with that mobile unless the mobile unit can accept incoming calls and reconfiguration operations from the ISAP.

*Continuous mobility.* Suppose salespersons run long transactions involving the allocation of multiple inventory items, and while moving between service areas.[1] In our model, we assume that a service handoff [10,11] occurs, so that the mobile starts communicating with the local server of the new service area. However, before the physical connection transfer can actually be carried out, the context related to the interactions of the user with the ISAP needs to be transferred from the old to the new server. Thus when the user moves to the new service area, the current operation in progress at the old server is completed, and then the context of the transaction is transferred to the new server. (Observe here that the mobile can continue interacting with the old server while the context is being transferred.) We will now consider the actual context information transfer required for locking schemes as well as our proposed site-transaction escrow schemes.

### *5.2. Context transfer for a traditional locking scheme*

If a traditional locking scheme had been used for concurrency control, the context information to be transferred would include (a) the transaction id, TID, (b) the list of locks held by the transaction, $L$, (c) the next operation to be performed by the new server (assuming the entire transaction has been submitted by the mobile), $N$, and (d) additional information relating to replica control, denoted $R$. Thus the context information to be transferred is $\{\text{TID}, L, N, R\}$.

To make the context information field $R$ more concrete, suppose a pessimistic quorum consensus protocol [9] with operational updates is used. In this protocol, server, $i$, before initiating an operation $o$ of a transaction $T$, locks the data items accessed by $o$ at a set of *quorum* servers. The quorum servers send the committed (timestamped) updates that they know of along with the lock grant response to $i$. Server *i* merges the responses in timestamp order with the set of committed updates which it knows about itself (i.e., locally). If the quorum is a majority of servers in the system, server $i$ is guaranteed [9] to be up to date with all the committed updates in the system and also guaranteed that since the lock for $o$ was acquired, there is no other conflicting operation executing in the system. Server $i$ appends the update operation $u_T$ to an *intentions list* of uncommitted updates for the transaction $T$. When $T$ is ready to commit, a two-phase commit is executed across all the quorum servers involved in its operations. If the decision is to commit, the intentions list is added to the list of committed updates at $i$ and the quorum servers, otherwise the list is discarded.

[1] It is important to note that the long-running transactions only imply that a connection is continuously maintained between the mobile and the server at the session or application level. In particular, it is not necessary that the wireless link be continuously in use, since the client and server can exchange occasional packets as necessary. Similarly, long-running transactions need not imply that the client machine is turned on at full power all the time; almost all modern mobile client devices slip into *doze* mode, yielding very substantial decreases in power consumption.

Thus, if the pessimistic quorum consensus protocol is used, the replica context information, $R$, transferred from the old server to the new server would be (a) the list $U$ of updates seen at the old server before the most recent operation in $T$ was executed, (b) the intentions list $I$ for $T$, and (c) the list $Q$ of quorum servers for each operation in $T$ executed so far. The entire context to be transferred is $\{\text{TID}, L, N, R\}$ where $R = \{U, I, Q\}$. The new server will have to first incorporate the list $U$ into its state, set up $I$ as an intentions list for $T$, update the lock table using list $L$ to remember the locks that $T$ has already acquired, and store the list of quorum servers $Q$ (so that they can participate in a two-phase commit when $T$ is ready to commit).

### 5.3. Context transfer with site-transaction escrow

Using site escrow methods makes context transfer and set-up much easier than the traditional locking scheme. We will first describe a simple scheme enabled by using site-transaction escrow, and then a slightly more complicated scheme which can provide improved concurrency and availability, if desired.

#### 5.3.1. Scheme 1

If site-transaction escrow is used, decisions to allocate (escrow) resources are made locally (as far as possible) so that the quorum information list $Q$ described above does not have to be maintained or transferred as context. Furthermore, since sites use transaction escrow locally, locks are released after the completion of each operation (as against each transaction). So the lock information list $L$ does not need to be transferred to the new server. In addition, suppose the operations in the sales transaction deal only with escrowable resource types. The new server does not need information about the operations already performed at the old server, since the old server made its decisions based on its locally escrowed resources. Thus the update list $U$ and the intentions list $I$ is not required. Therefore the context transferred is only $\{\text{TID}, N\}$.

However, when the transaction commits, a two-phase commit would also be required between the old server and the new server since part of the transaction has been run at the old server and the rest at the new server. (Note that the two-phase commit might be between several servers, i.e., be an $n$-way two-phase commit, if the user moves between several service areas during the duration of the transaction; we will return to this point later.) The benefit of the site-transaction escrow idea in the case of a salesperson who does not move is that most of the time, a two-phase commit would not be required *at the end of the transaction* since requests for resources would be satisfied locally. However, if the salesperson moves around a lot between service areas, then a two-phase commit is almost always required at the end of the transaction, and this is undesirable. To address this problem, we introduce a modified scheme.

#### 5.3.2. Scheme 2

We incorporate a pair of new operations into the protocol, called *mobile reconfiguration operations*, that are associated with each service handoff. We discuss these operations in the context of no failures (neither site nor communication failures).

Let $s$ and $t$ be the old and new servers, respectively. Suppose transaction $T$ has resulted in the allocation of $z$ resources of type $m$ at $s$, so that $E_s$ is of the form $((x'-z) \leqslant m_s \leqslant y')$. Suppose $E_t$ is of the form $(u' \leqslant m_t \leqslant v')$. Then the Mobile Allocate Reconfigure (MAR) operation first results in $s$ sending a message to $t$ asking it to modify $E_t$ to

$$\big((u' + z - z) \leqslant m_t \leqslant (v' + z)\big) = \big(u' \leqslant m_t \leqslant (v' + z)\big).$$

This is as if the number of instances at $t$ was increased by $z$ and the transaction $T$ resumed at $t$, i.e., these newly acquired instances are entered as having been escrowed by $T$ at $t$ (hence the "$-z$" term above). After $t$ replies to $s$ about having done the modification, $s$ modifies $E_s$ to $((x' - z) \leqslant m_s \leqslant (y' - z))$. This is as if $T$ committed at site $s$. It might be that the new $E_t$ is such that $E_t \not\Rightarrow C_t$, since $(v'+z)$ might be larger than the upper bound indicated by $C_t$. Thus, in addition, if $C_s$ was $(x \leqslant m_s \leqslant y)$ and $C_t$ was $(u \leqslant m_t \leqslant v)$, then they become respectively $(x \leqslant m_s \leqslant (y - z))$ and $(u \leqslant m_t \leqslant (v + z))$.

Note that MAR is similar to AR in that both logically transfer escrowed instances from one server to another. However, while AR affects both bounds in $E_s$ and $E_t$, MAR affects only one bound at each server. Observe that if $T$ does not allocate or deallocate items of type $m$ further and if $T$ commits at $t$, $E_t$ becomes $(u \leqslant m_t \leqslant v)$. The result is as if $T$ had run at $s$ entirely. However, if $T$ aborts, $E_t$ becomes $((u + z) \leqslant m_t \leqslant (v + z))$. The *mobile reconfiguration operation thereby optimistically assumes that transactions are likely to commit*, hence commits at $s$ the portions of $T$ which have already been done at $s$. If $T$ aborts however, $t$ finds that the number of its resource instances have increased due to the execution of $T$. (If required by the application, one can run a reconfiguration operation AR from $t$ to $s$ to restore the old situation.)

Similarly, suppose the transaction has resulted in the deallocation of $z$ resources of type $m$ at $s$, so that $E_s$ is of the form $(x' \leqslant m_s \leqslant (y' + z))$. Suppose $E_t$ is of the form $(u' \leqslant m_t \leqslant v')$. Then the Mobile Deallocate Reconfigure (MDR) operation results in $E_s$ becoming $((x' + z) \leqslant m_s \leqslant (y' + z))$. Furthermore, $E_t$ becomes

$$\big((u' - z) \leqslant m_t \leqslant (v' - z + z)\big).$$

If this creates a situation where $E_t \not\Rightarrow C_t$, then the following must be done to the configuration sets at $s$ and $t$ respectively: $C_s$ will become $((x + z) \leqslant m_s \leqslant y)$ and $C_t$ become $((u - z) \leqslant m_t \leqslant v)$.

**Example 3.** An example of a mobile sales transaction is shown in figure 3. There are two types of resource items,

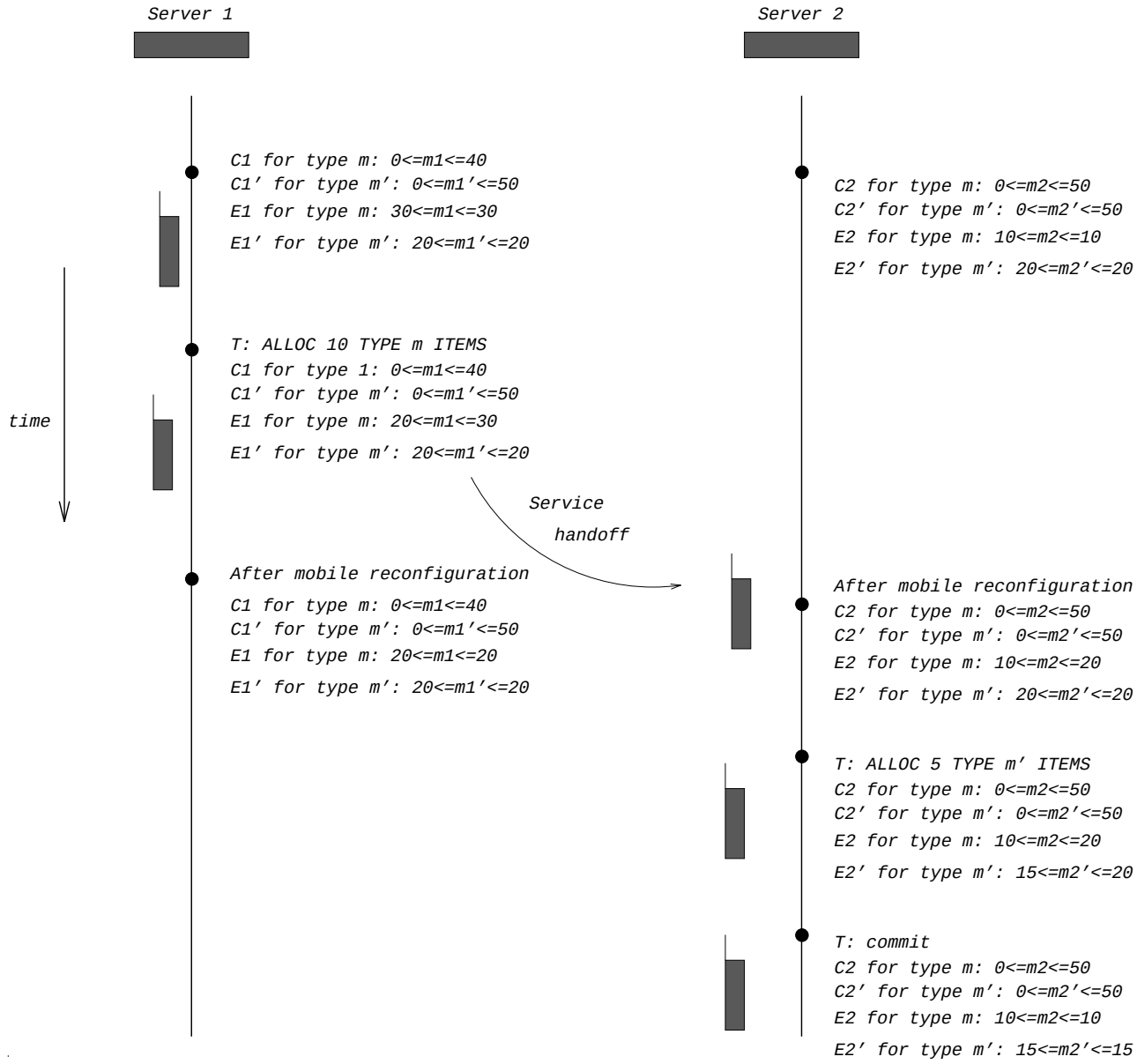



Figure 3. A mobile sales transaction example.

$m$ and $m'$. Initially, as can be seen from $E1$ and $E1'$, there are 20 and 30 resource instances of $m$ and $m'$, respectively, at site 1, and likewise, 10 and 20 instances at site 2. The mobile executes a transaction $T$ that allocates 10 instances of type $m$ at server 1, and then the mobile moves from server 1 to server 2. The mobile reconfiguration operation MAR updates $E1$ and $E2$ at the two sites, but does not affect $C1$ or $C2$ since $E2 \Rightarrow C2$. The mobile submits another operation to allocate 5 instances of type $m'$ as part of the same transaction $T$ (this would typically be run in parallel with the reconfiguration), and then commits.

**Observation 4.** Given a set of operations, $O$, such that each operation is either safe, or unsafe but solvable, and such that some of these operations are run from a mobile, $\mathcal{F}_m$ is preserved by the execution of these operations using the mobile reconfiguration operations.

The mobile reconfiguration operation can be performed independently of the context information transfer – the only requirement is that the reconfiguration complete before the transaction commits. By introducing Scheme 2 and the mobile reconfiguration operation we have increased the amount of context information which needs to be transferred in the site-transaction escrow scheme, from $\{\text{TID}, N\}$ to $\{\text{TID}, N, I\}$, in exchange for eliminating the two-phase commit which would be required at the end of the transaction.

*5.3.3. Scheme 2A*

An issue that arises with Scheme 2 is that suppose the mobile reconfiguration operation fails during a service handoff. This might be due to the loss of the message from server $t$ to server $s$ indicating that the modification of $E_t$ is complete, or it might be because server $s$ itself

fails; in either case, $s$ might not make the corresponding change to $E_s$. (Note that this does not violate the integrity constraints, $\mathcal{F}$, of the system.) The failure of the reconfiguration simply means that service handoff will not be done immediately. The mobile will continue to interact with $s$. In the meantime, after a timeout period, server $s$ can send messages to some other closer server to handoff to, and so on. However, it does mean that some resources have been made unavailable at $t$ and not made available at $s$.

To avoid this problem of "lost resources", a two-phase commit (or similar derivatives) is required during the mobile reconfigurations; otherwise it is not feasible to have the two corresponding operations at the two sites both commit or both abort. Thus we propose a "pair-wise two-phase commit" as part of each reconfiguration in Scheme 2, and we label this Scheme 2A. This two-phase commit is restricted to the pair of servers involved in the reconfiguration.

We stress that this differs from the two-phase commit required in Scheme 1 as follows. The pair-wise two-phase commit in Scheme 2A need not wait until the end of the transaction, but can be overlapped with the rest of the transaction, thus improving concurrency and performance. Further, if a mobile crosses several service areas during the course of the transaction, Scheme 1 would require an $n$-way two-phase commit to be carried out among all the servers at the end of the transaction, while Scheme 2A would require several pair-wise two-phase commits while the transaction is in process. Scheme 2A thus has some potential advantages:

1. An $n$-way two-phase commit requires that all servers be available at the end of the transaction, $T$. Even if a server that has performed a service handoff to some other server and is no longer processing $T$ crashes, transaction $T$ has to abort. In Scheme 2A, only one pair of servers needs to be available at each two-phase commit. This loosens the availability requirements of the servers.

2. An $n$-way two-phase commit is at least as slow as the slowest server involved, and is a form of barrier synchronization. In Scheme 2A a slow server would not hold up the commit.

3. Pairwise two-phase commits allow each old server to completely hand off service for the transaction, releasing CPU and other resources, e.g., space in internal tables.

There are clearly tradeoffs involved in the schemes we have introduced, and their evaluation depends upon the specifics of the application and architecture used. However, using the ideas of escrow, service handoffs and reconfiguration of resources, we have provided a clean transaction framework for performing mobile sales transactions.

## 6. Conclusions

We have presented a database system design based on the site-transaction escrow method, that is suitable for sales and inventory applications supporting both stationary and mobile users. We provided a formal model for reasoning about site-transaction escrow, and discussed several reconfiguration operations in the non-mobile environment. We dealt with several scenarios in a mobile environment and discussed how the site-transaction escrow protocol is a natural fit for these scenarios. Furthermore, we identified a mobile reconfiguration operation to prevent having to do a two-phase commit at the end of a transaction, which was executed as a mobile was handed off between servers. We thereby outlined how the site-transaction escrow protocol provides a simple mechanism for performing service handoffs when a mobile user moves from one service area to another.

We are currently investigating some of the issues discussed in this paper further. We have previously discussed [10,11] how mobile transactions which access distinguishable instances, for which site escrow methods are typically not appropriate, can be supported. We are considering the feasibility of light-weight protocols for combining such transactions with those accessing indistinguishable resource instances [1]. Furthermore, the treatment in this paper has been restricted to partitionable data items and constraints that are numeric in nature. Data structures such as queues and stacks are also partitionable, and escrow techniques are applicable in such cases too [6].

## References


[1] G. Alonso and A. El Abbadi, Partitioned data objects in distributed databases, Technical Report TRCS-93-06, University of California, Santa Barbara (1993).

[2] H. Balakrishnan, S. Seshan, E. Amir and R.H. Katz, Improving TCP/IP performance over wireless networks, in: *Proceedings of MobiCom '95* (November 1995).

[3] D. Barbara and H. Garcia-Molina, The Demarcation Protocol: A technique for maintaining arithmetic constraints in distributed database systems, in: *Proceedings of International Conference on Extending Data Base Technology* (1992).

[4] P.A. Bernstein, V. Hadzilacos and N. Goodman, *Concurrency Control and Recovery in Database Systems* (Addison-Wesley, 1987).

[5] Cellular radiotelecommunications intersystem operations, Rev. B, EIA/TIA (July 1991).

[6] P. Chrysanthis and G. Walborn, Personal communication (1994).

[7] D.K. Gifford, Weighted voting for replicated data, in: *Proceedings of the Seventh ACM Symposium on Operating Systems Principles* (1979) pp. 150–159.

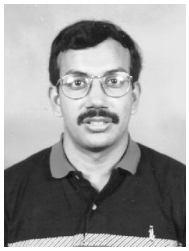
[8] J. Gray and A. Reuter, *Transaction Processing: Concepts and Techniques* (Morgan Kaufmann, 1993).

[9] M.P. Herlihy, Concurrency vs. availability: Atomicity mechanisms for replicated data, ACM TOCS 5(3) (August 1987) 249–274.

[10] R. Jain and N. Krishnakumar, Network support for personal information services to PCS users, in: *Proceedings of IEEE Conf. Networks for Personal Comm.* (*NPC*), Long Branch, NJ (March 1994).

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

[12] R. Jain, Y.-B. Lin and S. Mohan, Location strategies for personal communications services, in: *Mobile Communications Handbook*, ed. J. Gibson (CRC Press, 1996).
[13] N. Krishnakumar and A. Bernstein, High throughput escrow algorithms for replicated databases, in: *Proceedings of 18th Internat. Conf. on Very Large Data Bases* (August 1992) pp. 175–186.
[14] A. Kumar and M. Stonebraker, Semantics based transaction management techniques for replicated data, in: *Proceedings of the ACM SIGMOD International Conference on Management of Data* (1988) pp. 379–388.
[15] A.R. Modaressi and R.A. Skoog, Signaling system No. 7: A tutorial, IEEE Comm. Mag. (July 1990) 19–35.
[16] M. Mouly and M.-B. Pautet, *The GSM System for Mobile Communications* (49 rue Louise Bruneau, Palaiseau, France, 1992).
[17] N.J. Muller, *Wireless Data Networking* (Artech House, 1995).
[18] P.E. O'Neil, The escrow transactional model, ACM Transactions on Database Systems 11(4) (December 1986) 405–430.
[19] A.R. Noerpel, L.F. Chang and D.J. Harasty, Radio link access procedure for a wireless access communications system, in: *Proceedings of Internat. Conf. Comm.* (May 1994).
[20] V. Schnee, An excellent adventure, Wireless (March/April 1994) 40–43.
[21] J. Schwartz, Upgrade lets salespeople share data, Comm. Week (May 24, 1994) 47–48.
[22] N. Soparkar and A. Silberschatz, Data-value partitioning and virtual messages, in: *Proceedings of 9th ACM SIGACT-SIGMOD Symposium on Principles of Database Systems* (1990) pp. 357–367.
[23] R.H. Thomas, A majority consensus approach to concurrency control for multiple copy databases, ACM Transactions on Database Systems 4(2) (June 1979) 180–209.



**Narayanan Krishnakumar** received the B. Tech degree in computer science from the Indian Institute of Technology, Madras, in 1987 and the M.S. and Ph.D. degrees in computer science from the State University of New York at Stony Brook in 1989 and 1992, respectively. Krishnakumar has been a Research Scientist at Bellcore in New Jersey and is currently a Technical Director at Fidelity Investments, Boston. He also teaches computer systems courses at Brandeis University in Waltham, MA. Krishnakumar's interests are in high-performance transaction processing architectures, distributed object computing, workflow systems and mobile computing. Recently, Krishnakumar has been engaged in developing architectural object frameworks for scalable distributed computing at Fidelity. Krishnakumar has been a guest co-editor for the Distributed and Parallel Databases Journal special issue on Databases and Mobile Computing, and has also served on the committees of conferences and workshops.
E-mail: narayanan.krishnakumar@fmr.com

**Ravi Jain** received the Ph.D. in computer science from the University of Texas at Austin in 1992. Prior to his Ph.D. degree he worked at Syntrex Inc., SES Inc. and the Schlumberger Laboratory for Computer Science on developing communications and systems software, performance modeling, and parallel programming. In 1992 Jain joined Bellcore as a Research Scientist in the Personal Communications Services (PCS) area. His research interests include design and analysis of algorithms, architectures and protocols for mobile computing and communications, including techniques for locating mobile users, mobile database access, and mobile sales applications. He was also technical team leader for the SCOUT system, which delivers personalized road traffic information to mobile users; this project is being incorporated into Bellcore's AirBoss product line. Recently Jain has been engaged in implementing a mobile IP testbed as well as research on supporting mobile users with fixed ATM backbone networks and wireless Internet access. Jain is guest co-editor of the IEEE Journal of Selected Areas in Communications special issue on "Networking and Performance Issues of Personal Mobile Communications" and an area editor for MONET, MC2R and IEEE Personal Communications. He has one US patent issued and several patents pending, mostly in the area of wireless and personal communications. Jain is a member of the Upsilon Pi Epsilon and Phi Kappa Phi honorary societies, a senior member of IEEE, and a member of ACM and INFORMS.
E-mail: rjain@bellcore.com