Locating Objects in Mobile Computing*

Evaggelia Pitoura
Department of Computer Science
University of Ioannina
GR 45110 Ioannina, Greece
pitoura@cs.uoi.gr

George Samaras†
Department of Computer Science
University of Cyprus
CY 1678 Nicosia, Cyprus
csamara@turing.cs.ucy.ac.cy

Abstract

In current distributed systems, the notion of mobility is emerging in many forms and applications. Mobility arises naturally in wireless computing, since the location of users changes as they move. Besides mobility in wireless computing, software mobile agents are another popular form of moving objects. Locating objects, i.e., identifying their current location, is central to mobile computing. In this paper, we present a comprehensive survey of the various approaches to the problem of storing, querying, and updating the location of objects in mobile computing. The fundamental techniques underlying the proposed approaches are identified, analyzed and classified along various dimensions.

Keywords: Mobile computing, location management, location databases, caching, replication

1 Introduction

In current distributed systems, the notion of mobility is emerging in many forms and applications. Increasingly many users are not tied to a fixed access point but instead use mobile hardware such as dial-up services or wireless communications. Furthermore, mobile software, i.e., code or data that move among network locations, is emerging as a new form of building distributed network-centric applications. In the presence of mobility, the cost of communicating with a mobile user or using mobile code and data is augmented by the cost of searching for their current location.

Mobility arises naturally in wireless mobile computing [11, 15, 30] since as mobile users move, their point of attachment to the fixed network changes. Future Personal Communication Systems (PCSs) will support a huge user population and offer numerous customer

*University of Ioannina, Computer Science Department, Technical Report No: 98-020
†**Contact Author**

services. In such systems, the signaling and database traffic for locating mobile users is expected to increase dramatically [43]. Thus, deriving efficient strategies for locating mobile users, i.e., identifying their current location, is an issue central to wireless mobile computing research.

Besides mobility tied to wireless hardware, data or code may be relocated among different network sites for reasons of performance or availability. Mobile software agents [42, 1] is a popular such form of mobile software. Mobile agents are processes that may be dispatched from a source computer and be transported to remote servers for execution. Mobile agents can be launched into an unstructured network and roam around to accomplish their task [2], thus providing an efficient, asynchronous method for collecting information or attaining services in rapidly evolving networks. Other applications of moving software include the relocation of a user's personal environment to support ubiquitous computing [44], or the migration of services to support load balancing, for instance the active transfer of web pages to replication servers in the proximity of clients [7].

In this paper, we present a comprehensive survey of the various approaches to the problem of storing, querying, and updating the location of objects in mobile computing. The emphasis is on the fundamental techniques underlying the proposed approaches as well as on analyzing and classifying them along various dimensions. By identifying various parameters and classifying elemental techniques, new approaches to the problem can be developed by appropriately setting the parameters and combining the techniques.

The rest of this paper is structured as follows. In Section 2, we introduce the location problem and its variations. In Section 3, we present the two most common architectures for location directories, i.e., directories that hold the location of moving objects: one is a two-tier architecture based on a pair of home/visitor location databases; the other is a hierarchically structured one. In Section 4, we discuss optimizations and variations of these architectures. In the following sections, we introduce a number of approaches that have been proposed to reduce the cost of lookups and updates in both architectures. In particular, in Sections 5 and 6, we discuss caching and replication of location information at selected network sites and in Section 7, we present forwarding pointer techniques that only partially update the location directories. In Section 8, we present a taxonomy of the approaches presented. In Section 9, we focus on issues related to concurrency and fault-tolerance and in Section 10 on issues related to answering complex queries about the location of moving objects. We conclude in Section 11 by summarizing.

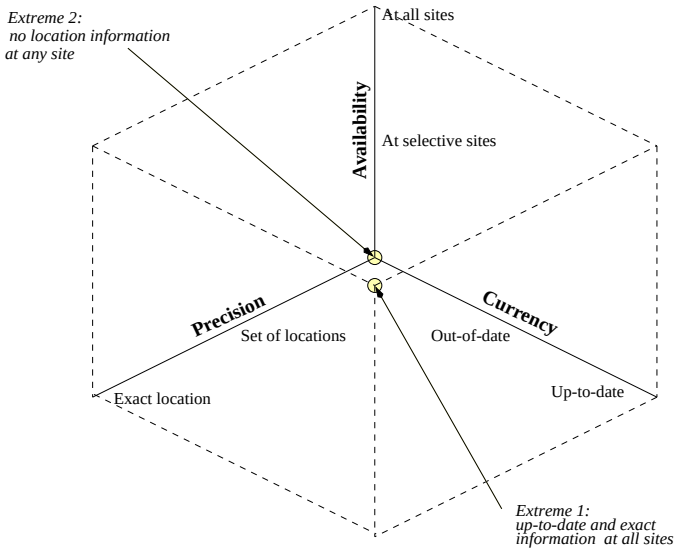

Figure 1: Approaches to Saving Location Information

2 Location Management

In mobile computing, mobile objects, e.g., mobile software or users using wireless hardware, may relocate themselves from one network location to another. To enable the efficient tracking of mobile objects, information about their current location may be stored at specific network sites. In abstract terms, location management involves two basic operations, lookups and updates. A lookup or search is invoked each time there is a need to locate a mobile object, e.g., to contact a mobile user or invoke mobile software. Updates of the stored location of a mobile object are initiated when the object moves to a new network location. In the rest of this section, we first provide an overview of the problem and then introduce network architectures that are commonly associated with mobile computing.

2.1 Overview

Approaches to storing location information range between two extremes. At one extreme, up-to-date information of the exact location of all users is maintained at each and every network location. In this case, locating a user reduces to querying a local database. On the other hand, each time the location of a user changes, a large number of associated location databases must be updated. At the other extreme, no information is stored at any site of the network. In this case, to locate a mobile user a global search at all network sites must be initiated. However, when a user moves, there is no cost associated with updating location databases.

Between these two extremes, various approaches that balance the cost of look-ups against the cost of updates are plausible. These approaches compromise the availability, precision or currency of the location information stored for each user (Figure 1). In terms of availability, choices range between saving the location at all network sites to not storing the location at all. In between these two approaches, location information may be maintained selectively at *specific* network sites. There is a wide range of selection criteria for the sites at which to save location information for each user. For example, a choice may be to save the location of users at the sites of their frequent callers. Imprecision in location information takes many forms. For instance, instead of maintaining the exact location of the user, a wider region or a set of possible locations is maintained. Currency refers to when the stored location information is updated. For instance, for highly mobile users it may make sense to defer updating the stored information about their location every time the users move. When current and precise information about a user's location is not available locally, locating the user involves a combination of some search procedure and a number of queries posed to database storing locations.

2.2 Underlying Network Architecture

The networking infrastructure for providing ubiquitous wireless communication coverage is represented by the personal communication system (PCS) also known by a number of different names such as personal communication network (PCN) and UMTS (universal mobile communication system). While the architecture of the PCS has not evolved yet, it is expected that it will be partially based on the existing digital cellular architecture (see Figure 2 adapted from [15]). This network configuration consists of fixed backbone networks extended with a number of mobile hosts (MHs) communicating directly with stationary transceivers called *mobile support stations* (MSS) or *base stations*. The area covered by an individual transceiver's signal is called a *cell*. The mobile host can communicate with other units, mobile or fixed, only through the base station at the cell in which it resides. Thus to communicate with a mobile user, the base station of the cell in which it currently resides must be located. As a mobile host moves, it may cross the boundary of a cell, and enter an area covered by a different base station. This process is called *handoff* and may involve updating any stored location information for the mobile host. It is speculated that ubiquitous communications will be provided by PCS in a hybrid fashion: heavily populated areas will be covered by cheap base stations of small radius (picocells); less populated areas will be covered by base stations of larger radius; and farm land, remote areas and highways

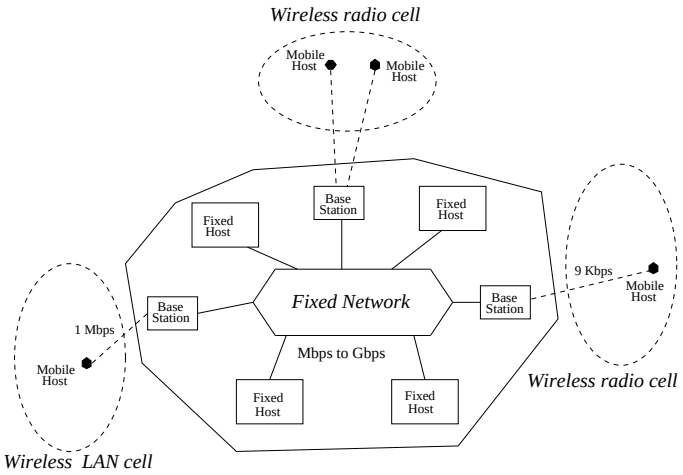

Figure 2: Wireless Computing Architecture

with satellites that will provide the bridge between these different islands of population density.

PCSs involve two types of mobility: terminal and personal mobility [26]. *Terminal mobility* allows a terminal to be identified by a unique terminal identifier independent of its point of attachment to the network. *Personal mobility* allows PCS users to make and receive calls independently of both their network point of attachment and a specific PCS terminal. Each mobile user explicitly *registers* itself to notify the system of its current location. The granularity of a registration area ranges from that of a single cell to a group of cells. Once the registration area is identified, the user can be tracked inside this area using some form of paging. Paging is the process whereby to locate a mobile user, the system issues polling signals in a number of likely locations. By changing the size of a registration area, the flexibility of any combination of registration and paging is attained [32]. If not explicitly stated otherwise, we use the term cell or *zone* as synonyms with registration area to indicate a uniquely identifiable location where a mobile user can be found.

In the cellular architecture, three levels are involved: the access, the fixed, and the intelligent network [43]. The *fixed network* is the wired backbone network. The *access network* is the interface between the mobile user and the fixed network. The *intelligent network* is the network connecting any location registers, i.e., registers used to store information about the location of mobile users. This network is used to carry traffic related to tracking mobile users. The Signaling System No. 7 (SS7) [24] and its signaling network is a good candidate to carry the signaling traffic in the intelligent network.

Location management is handled at the data link or networking layer transparently

from the layers above it [39], each time a call is placed or a change in the network point of attachment occurs. Location management is an issue present at all wireless networks (e.g., cellular, wireless LANs, satellites). Although most solutions so far relate to cellular architectures at the data link layer and to wireless LAN architectures at the networking layer, most are general enough to be applicable to different layers and architectures. In addition to handling mobility at lower layers, the need for information about the location of moving objects is encountered at the application level as well. Applications may need information about the location of mobile users to answer a variety of queries that involve location (e.g., find the nearest restaurant) [16]. Other applications may involve updating environmental parameters and selecting locally available computing resources (e.g., nearest printer) [27]. There is no standard way for applications to acquire and use location information. For example, applications may choose to maintain their own data structures for storing location information.

The cellular architecture is not the sole infrastructure for wireless mobile computing. In its absence, various techniques may be employed to identify the current location of mobile users, for instance, users may be equipped with a Global Positioning System (GPS) [13, 12]. GPSs are space-based radio positioning systems that provide three-dimensional position, velocity and time information to suitably equipped users anywhere on or near the surface of the Earth. Common applications in this area include digital battlefields in the military context and transportation systems in the civilian industry [45]. Finally, besides mobility tied to wireless hardware, the techniques presented in this paper are also applicable when the objective is to locate mobile code and data. Furthermore, similar techniques are also necessary when instead of location, the objective is to efficiently retrieve other profile information related to mobile users. This information may include QOS-related parameters or services.

3 Architectures of Location Databases

In this section, we describe basic architectures for distributed databases used for storing the location of moving users. The first is a two-tier scheme in which the current location of each moving user is saved at two network locations. The other is a tree-structured distributed database in which space is hierarchically decomposed in sub-regions. Finally, we describe a graph-theoretic approach that employs regional directories.

3.1 Two-tier Schemes

In two-tier schemes, a home database, termed *Home Location Register* (HLR), is associated with each mobile user. The HLR is located at a pre-specified for each user network location (zone) and maintains the current location of the user as part of the user's profile. The search and update procedures are quite simple. To locate a user x, x's HLR is identified and queried. When a user x moves to a new zone, x's HLR is contacted and updated to maintain the new location.

As an enhancement to the above scheme, *Visitor Location Registers* (VLRs) are maintained at each zone. The VLR at a zone stores copies of profiles of users not at their home location and currently located inside that zone. When a call is placed from zone i to user x, the VLR at zone i is queried first and only if the user is not found there, is x's HLR contacted. When a user x moves from zone i to j, in addition to updating x's HLR, the entry for x is deleted from the VLR at zone i, and a new entry for x is added to the VLR at zone j.

The two prevailing existing standards for cellular technologies, the Electronics Industry Association Telecommunications Industry Associations (EIA/TIA) Interim Standard 41 (IS-41) commonly used in North America and the Global System for Mobile Communications (GSM) used in Europe, both support carrying out location strategies using HLRs and VLRs [26]. At the Internet networking level, extensions of the IP protocol for routing based on the HLR/VLR scheme is provided in the IEFTP (Internet Engineering Task Force) Mobile IP protocol [21].

One problem with the home location approach is that the assignment of the home register to a mobile object is permanent. Thus, long-lived objects cannot be appropriately handled, since their home location remains fixed even when the objects permanently move to a different region. Another drawback of the two-tier approach is that it does not scale well with highly distributed systems. To contact an object, the possibly distant home location must be contacted first. Similarly, even a move to a nearby location must be registered at a potentially distant home location. Thus, locality of moves and calls is not taken advantage off.

3.2 Hierarchical Schemes

Hierarchical location schemes extend two-tier schemes by maintaining a hierarchy of location databases. In this hierarchy, a location database at a higher level contains location information for users located at levels below it. Usually, the hierarchy is tree-structured.

In this case, the location database at a leaf serves a single zone (cell) and contains entries for all users registered in this zone. A database at an internal node maintains information about users registered in the set of zones in its subtree. For each mobile user, this information is either a pointer to an entry at a lower level database or the user's actual current location. The databases are usually interconnected by the links of the intelligent signaling network, e.g., a Common Channel Signaling (CCS) network. For instance, in telephony, the databases may be placed at the telephone switches. It is often the case that the only way that two zones can communicate with each other is through the hierarchy; no other physical connection exists among them.

We introduce the following notation. We use the term $LCA(i, j)$ to denote the least common ancestor of nodes i and j. A parameter that affects the performance of most location management schemes is the relative frequency of move and call operations of each user. This is captured by the *call to mobility ratio* (CMR). Let C_i be the expected number of calls to user P_i over a time period T and U_i the number of moves made by P_i over T, then $CMR_i = C_i/U_i$. Another important parameter is the local call to mobility ratio $LCMR_{i,j}$ that also involves the origin of the calls. Let $C_{i,j}$ be the expected number of calls made from zone j to a user P_i over a time period T, then the local call to mobility ratio $LCMR_{i,j}$ is defined as $LCMR_{i,j} = C_{i,j}/U_i$. For hierarchical location schemes, the local call to mobility ratio ($LCMR_{i,j}$) for an internal node j is extended as follows: $LCMR_{i,j} = \sum_k LCMR_{i,k}$, where k is a child of j. That is, the local call to mobility ratio for a user P_i and a internal node j is the ratio of the number of calls to P_i originated from any zone at j's subtree to the the number of moves made by P_i.

The type of location information maintained in the location databases affects the relative cost of updates and lookups as well as the load distribution among the links and nodes of the hierarchy. Let's consider first the case of keeping at all internal databases forwarding pointers to lower level databases. For example, in Figure 3(left) for a user x residing at node (cell) 18, there is an entry in the database at node 0 pointing at the entry for x in the database at node 2. The entry for x in the database at node 2 points to the entry for x in the database at node 6, which in turns points to the entry for x in the database at node 18. When user x moves from zone i to zone j, the entries for x in the databases along the path from j to $LCA(i, j)$, and from $LCA(i, j)$ to i are updated. For instance, when user x moves from 18 to 20, the entries at nodes 20, 7, 2, 6, and 18 are updated. Specifically, the entry for x is deleted from the databases at nodes 18 and 6, the entry for x at the database at 2 is updated, and entries for x are added to the databases at nodes 7 and 20. When a caller

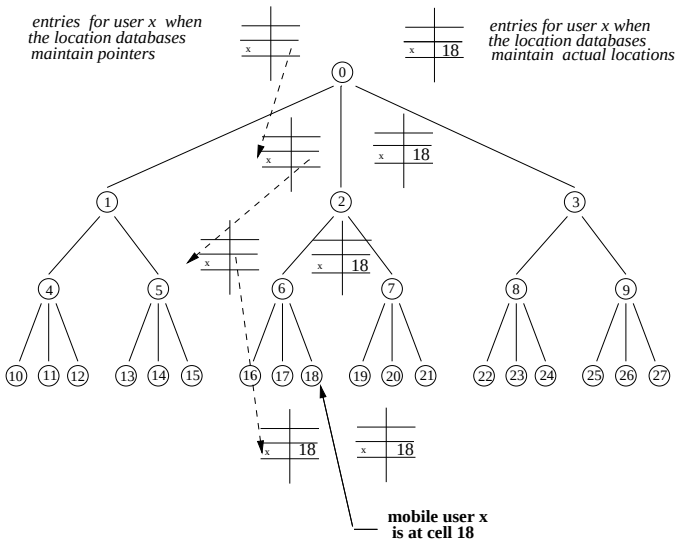

Figure 3: Hierarchical Location Schema. Location databases' entries at the left are pointers at lower level databases, while location databases' entries at the right are actual locations.

located at zone i places a call for a user x located at zone j, the lookup procedure queries databases starting from node i and proceeding upwards the tree until the first entry for x is encountered. This happens at node $LCA(i, j)$ (the least common ancestor of nodes i and j). Then, the lookup procedure proceeds downwards following the pointers to node j. For instance, a call placed from zone 21 to user x (Figure 3(left)), queries databases at nodes 21, 7 and finds the first entry for x at node 2. Then, it follows the pointers to nodes 6 and 18.

Let's now consider the case of database entries maintaining the actual location of each user. Then, for user x registered at 18 (Figure 3(right)), there are entries in the databases at nodes 0, 2, 6, and 18, each containing a pointer to location 18. In this case, a move from zone i to j causes the update of all entries along the paths from j to the root, and from the root to i. For example, a relocation of user x from node 18 to node 20, involves the entries for x at 20, 7, 0, 2, 6, and 18. After the update, entries for x exist in the databases located at nodes 0, 2, 7, and 20, each containing a pointer to 20, while the entries for x in the databases at nodes 6 and 18 were deleted. On the other hand, the cost of a call from i to j is reduced, since once the $LCA(i, j)$ is reached, there is no need to query the databases on the downward path to j. For example, a call placed from node 21 to user x (Figure 3(right)), queries databases at nodes 21, 7, 2, and then 18 directly (without querying the database at node 6).

When hierarchical location databases are used, there is no need for binding a user to a

(+)	No need for life-long numbering (no pre-assigned HLR)
(+)	Support for locality
(-)	Increased number of operations (database operations and communication messages)
(-)	Increased load and storage requirements at higher-levels

Table 1: Summary of the Pros and Cons of Hierarchical Architectures

home location register (HLR). The user can be located by querying the databases in the hierarchy. In the worst case, an entry for the user will be found in the database at the root.

The hierarchical scheme leads to reductions in communication cost when most calls and moves are geographically localized. In such cases, instead of contacting the HLR of the user that may be located far away from the user's current location, a small number of location databases in the user's neighborhood are accessed. However, the number of location databases that are updated and queried increases relative to the two-tier scheme. To reduce the number of lookups and updates, instead of placing databases at every node in the hierarchy, databases may be selectively placed only at particular nodes in the hierarchy. Furthermore, instead of maintaining precise location information, only coarse information may be maintained at internal nodes.

A hybrid scheme utilizing both hierarchical entries and pre-assigned home location registers (HLRs) is also possible. Assume that database entries are maintained only at selective nodes of the hierarchy and that an HLR is used. In this case, a call originating from zone i starts searching for the callee from zone i. It proceeds following the path from i to the LCA of i and the callee's HLR and then moves downwards to the callee's HLR, unless an entry for the callee is found in any database on this path. If such an entry is encountered, it is followed instead [43].

A problem with the hierarchical schemes is that the databases located at higher-level must handle a relatively large number of messages. Furthermore, they have large storage demands. One solution is to partition the databases at the high-level nodes (e.g., at the root) into smaller databases at sub-nodes so that the entries of the original database are shared appropriately among the databases at the sub-nodes [41]. Table 1 summarizes some of the pros and cons of the hierarchical architectures.

3.3 Non-tree Hierarchy: Regional Matching

The objective of the regional directories approach [5] is to favor local operations, in that moves to near-by locations or searches for near-by users cost less. The approach guarantees communication overheads that are polylogarithmic in the size (i.e., number of network sites) and the diameter (i.e., maximum distance between any two sites) of the network. The overhead is evaluated by comparing the total cost of a sequence of move and call operations against the inherent cost, i.e., the cost incurred by the operations assuming that information for the current location of each user exists at all sites for free. The comparison is done over all possible sequences of move and call operations.

Location databases called regional directories are organized in a non-tree hierarchy. In particular, a hierarchy D of δ regional directories is built, where $\delta = log\, d$, for d being the maximal distance between any two network sites. The purpose of a regional directory RD_i at level i is to enable a potential searcher to track any user residing within distance 2^i from it. Two sets of sites are associated with each site u in an RD_i directory: a readset $Read_i(u)$ and a writeset $Write_i(u)$ with the property that the readset $Read_i(u)$ and the writeset $Write_i(w)$ intersect for any pair of site u and w within a distance 2^i from each other. The two sets of sites are used as follows. Each site reports all users it hosts to every site in its writeset and upon looking for a user, it queries all sites in its readset.

Whenever a user moves to a new location at distance k away, only the $log\, k$ lowest levels of the hierarchy are updated to point directly to the new address. Directory entries at higher level directories continue pointing to the old address, where a forwarding pointer to the new location is left. To bound the length of the chain of forwarding pointers, it is guaranteed that for every user the distance $C(x)$ traveled since its address was updated at the regional directory RD_i is less or equal to $2^{i-1} - 1$ for each level i. The complete search and update procedures follow.

```
Regional Matching Search Procedure
    /* a call is placed from a user at site w to user x */
    i ← 0    address ← nil
    repeat
        i ← i + 1
        /* Search directory RD_i */
        for all sites u in Read(w)
            query u
    until address <> nil
    repeat
        follow forwarding pointers
    until reaching x
```

Regional Matching Move Procedure
/* user x moves from site v to site w */
Let RD_j be the highest directory for which $C(x) > 2^{j-1} - 1$
for i =1 *to* $max\{J, \delta\}$
/* Update directory RD_i */
for all sites u in $Write(v)$
update entry
add a forwarding pointer at RD_{i+1}

4 Placement of Databases

Maintaining location information at all nodes in the hierarchy, although resulting in cost-effective lookups, it increases the number of databases that are updated during each move operation. To reduce the update cost, database entries may be only selectively maintained at specific nodes in the tree hierarchy. In this case, during the search and update procedures, only nodes that contain location databases are queried or updated; others are skipped. For instance, when a call is made from j to i the search procedure traverse the tree from node j up to the youngest ancestor of the $LCA(i, j)$ that contains a location database.

A possible placement of location databases is to maintain location entries for mobile hosts only at the leaf nodes of the zone in which they reside currently. In this case, when there is no home location register associated with a mobile host, some form of global searching in the hierarchy is needed to find its current location. In this scenario, location strategies include flat, expanding, and hybrid searches [6]. Let *home* be the zone at which a user registers initially. The *flat* search procedure starts from the root, and then in turn queries in parallel all nodes at the next level of the tree until the leaf level is reached. The *expanding* search procedure starts by querying the *home* of the callee i, then queries the parent of the *home*, which in turn queries all its children and so on. This type of search favors moves to nearby locations. Finally, the *hybrid* search procedure starts as the expanding one, but if the location is not found at the children of the parent of the callee's *home*, a flat search is initiated. The hybrid scheme can locate quickly those users that when not at *home* happen to be found far away from it.

4.1 Optimization

The placement of location databases can be seen as an optimization problem. Objective functions include minimizing: (a) the number of database updates and accesses, (b) the communication cost, (c) the sum of the traffic on the network link or links, or any combi-

nation of the above. Constraints that must be satisfied include: (a) an upper bound on the rate at which each database can be updated or accessed, (b) the capacity of links, and (c) the available storage.

Such an optimization-based approach is taken in [4]. The objective there is to minimize the number of updates and accesses per unit time given a maximum database service capacity (i.e, the maximum rate of updates and lookups that each database can service) and estimates of the call to mobility ratio. In this approach, communication is not considered, and thus, if the service capacity is sufficiently large, a single, central database at the root is the optimal placement. The problem is formulated as a combinatorial optimization problem and is solved using a dynamic programming algorithm.

4.2 Partitions

To avoid maintaining location entries at all levels of the hierarchy, and at the same time reduce the search cost, *partitions* are deployed [6]. The partitions for each user are obtained by grouping the zones (cells) among which it moves frequently and separating the zones between which it relocates infrequently. Thus, partitions exploit locality of movement. Partitions can be used in many ways. We describe next two such partition-based strategies.

For each partition, the information whether the user is currently in the partition is maintained at the least common ancestor of all nodes in the partition, called the *representative* of the partition. The representative knows that a user is in its partition but not its exact location [6]. This information is used during flat search (i.e., top-down search starting from the root) to decide which subtree in the hierarchy to search. Thus, partitions reduce the overall search cost as compared to flat search. There is an increase however on the update cost since, when a user crosses a partition, the representatives of its previous and new partitions must be informed. For example, assume that user x often moves inside four different set of nodes, i.e., partitions, and infrequently between these sets. The nodes of each partition are {10, 12, 14, 15}, {16, 18}, {19,20,21} and {22, 23, 25, 26, 27} and are depicted in Figure 4. The representative node of each partition is high-lighted. When user x is at node 14 in partition 1, the representative of the associated partition, node 1, maintains the information that the user is inside its partition. When user x moves to node 16 that is outside the current partition, both node 1, the representative of the old partition, and node 6, the representative of the new partition, are updated to reflect the movement.

A slightly different use of partitions called *redirection trees* is proposed in [9]. A single partition, called local region, is defined by including all nodes between which the user

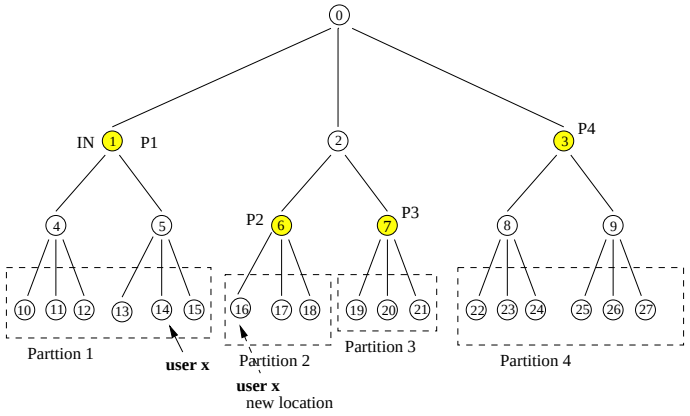

Figure 4: Partitions

often moves. The representative of the local region called a redirection agent maintains the location of all users that have appointed it as their redirection agent. When the user is located in its local region, its redirection agent redirects any calls passing through it during any type of search (e.g., flat or using HLRs) to the current location of the user. Movements inside a local region are recorded in the redirection agent and not necessarily at location servers outside the region.

5 Caching

Caching is based on the premise that after a call is resolved, the information about the current location of the callee should be reused by any subsequent calls originated from the same region. To this end, in two-tier architectures, every time a user x is called, x's location is cached at the VLR in the caller's zone, so that any subsequent call to x originated from that zone can reuse this information [19]. Caching is useful for those users who receive calls frequently relative to the rate at which they relocate. Similar to the idea of exploiting locality of file accesses, the method exploits the spatial and temporal locality of calls received by users.

To locate a user, the cache at the VLR of the caller's zone is queried first. If the location of the user is found at the cache, then a query is launched to the indicated location without contacting the user's HLR. Otherwise, the HLR is queried. Regarding cache invalidation, there are various approaches. In *eager caching*, every time a user moves to a new location, all cache entries for this user's location are updated. Thus, the cost of move operations increases for those users whose address is cached. In *lazy caching*, a move operation signals no cache updates. Then, when at lookup a cache entry is found there are two cases: either

the user is still in the indicated location and there is a cache hit, or it has moved out, in which case a cache miss is signaled. In the case of a cache miss, the usual procedure is followed: the HLR is contacted and after the call is resolved the cache entry is updated. Thus, in lazy caching, the cached location for any given user is updated only upon a miss. The basic overhead involved in lazy caching is in cases of cache misses, since the cached location must be visited first. So, for lazy caching to produce savings over the non-caching scheme, the hit ratio p for any given user at a specific zone must exceed a hit ratio threshold $p_T = C_H/C_B$, where C_H is the cost of a lookup when there is a hit and C_B the cost of the lookup in the non-caching scheme. Among other factors, C_H and C_B depend on the relative cost of querying HLR's and VLR's.

A performance study for lazy caching is presented in [19, 14]. There, an estimation of C_H and C_B is computed for a given signaling architecture based on a Common Channel Signaling network that uses the SS7 protocol [24] to set up calls. Conclusions are drawn on the benefits of caching based on which of the factors participating in C_H and C_B dominate. The hit ratio for the cache of user's i location at zone j can also be directly related to the $LCMR_{i,j}$ of the user [19]. For instance, when the incoming calls follow a Poisson distribution with arrival rate λ and the intermove times are exponentially distributed with mean μ, then $p = \lambda/(\lambda + \mu)$ and the minimum $LCMR$, denoted $LCMR_T$, required for caching to be beneficial is $LCMR_T = p_T/(1 - p_T)$. So, caching can be selectively done per user i at zone j, when the $LCM_{i,j}$ is larger than the $LCMR_T$ bound. In general, this threshold is lower when users accept calls more frequently from users located near by. In practice, it is expected that $LCMR_T > 7$ [19].

When the cache size is a concern, cache replacement policies, such as replacing the least recently used (LRU) location, may be used. Another issue is how to initialize the cache entries. User profiles and other types of domain knowledge may be used to initially populate the cache with the locations of the users most likely to be called. In the approach we have described, caching is performed on a per-user basis: the cache maintains the address of the last called users. Another approach is to to apply a static form of caching, e.g., cache the addresses of a certain group of users or certain parts of the network where the users' call to mobility ratios ($CMRs$) are known to be high on average.

Caching techniques can also be deployed to exploit locality of calls in tree-structured hierarchical architectures. Recall that in hierarchical architectures, when a call is placed from zone i to user x located at zone j, the search procedure traverses the tree upwards from i to $LCA(i, j)$ and then downwards to j. We also consider an acknowledgment message that

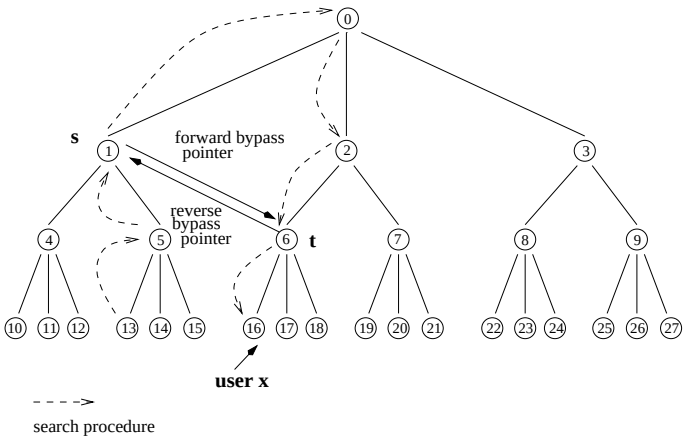

Figure 5: Caching in Hierarchical Location Schemes. For simplicity, the acknowledgment message is not shown; it follows the reverse route of the search procedure.

returns from j to i. To support caching, during the return path, a pair of bypass pointers, called forward and reverse, is created [17]. A *forward bypass* pointer is an entry at an ancestor of i, say s, that points to an ancestor of j say t; the *reverse bypass* pointer is from t to s. During the next call from zone i to user x, the search message traverses the tree upwards until s is reached. Then, the message travels to database t either via $LCA(i, j)$ or via a shorter route if such a route is available in the underlying network. Similarly, the acknowledgment message can bypass all intermediate pointers on the path from t to s.

For example, let a call be placed from zone 13 to user x at zone 16 (Figure 5). A forward bypass pointer is set at node 1 pointing to node 6; the reverse bypass pointer is from 6 to 1. During the next call from zone 13 to user x, the search message traverses the tree from node 13 up to node 1 and then at node 6, either through $LCA(1, 6)$, that is node 0, or via a shorter path. In any case, no queries are posed to databases at nodes 0 and 2.

The level of nodes s and t where the bypass pointers are set varies. In *simple caching*, s and t are both leaf nodes, while in *level caching*, s and t are nodes belonging to any level and possibly each to a different one (as in the previous example). Placing a bypass pointer at a high-level node s, makes this entry available to all calls originated from zones at s's subtree. However, calls must traverse a longer path to reach s. Placing the pointer to point at a high-level node t, increases the cost of lookup, since to locate a user, a longer path from t to the leaf node must be followed. On the other hand, the cache entry remains valid as long as the user moves inside t's subtree. An adaptive scheme can be considered to set the levels of s and t dynamically.

As in the two-tier location scheme, there are many possible variations for performing

cache invalidations [17]. In lazy caching, the move operation remains unchanged, since cache entries are updated only when a cache miss is signaled. In eager caching, cache entries are updated at each move operation. Specifically, consider a move operation from zone i to zone j, where a registration/deregistration message propagates from j via $LCA(j,i)$ to i. During this procedure, the bypass pointers which are no longer valid are deleted. These pointers include any forward bypass pointers found during the upward traversal of the registration message, and any reverse or bypass pointers found during the downward traversal of the deregistration message [17].

Preliminary performance results are reported in [17]. The analysis is based on a quantity called *Regional Call-to-Mobility Ratio (RCMR)* defined for a user x with respect to tree nodes s and t as the average number of calls from the subtree rooted at s to user x, while user x is in the subtree rooted at t. It is shown, that under certain assumptions, for users with $RCMR > 5$, caching can result in up to a 30% reduction in the cost of both calls and moves, when considering only the number of database operations.

Caching in the case of storing the exact location at internal nodes, as opposed to pointers to lower level databases, can also be deployed in many ways again ranging from simple to level caching. In simple caching, the current location of the user is cached only at leaf nodes. In level caching, the current location of a user is cached at *all* nodes up to a given level.

Caching is orthogonal to partitions. In fact, in [40, 41] caching is used in conjunction with partitions. In particular, instead of caching the current location of the callee, the location of its representative is cached. For example, assume that partitions are defined as in Figure 4 and user x is at node 14. Let a call be placed for user x. Instead of caching location 14 (or a pointer to it), location 1, e.g., the representative of the current partition, is cached. This significantly reduces the cost of cache updates, since a cache entry becomes obsolete only when a user moves outside the current partition.

6 Replication

To reduce the lookup cost, the location of specific users may be replicated at selected sites. Replication reduces the look-up cost, since it increases the probability of finding the location of the callee locally as opposed via issuing a high latency remote look-up. On the other hand, the update cost incurred increases considerably, since replicas must be maintained consistent every time the user moves.

In general, the location of a user i should be replicated at a zone j, only if the replication

is *judicious*, that is the savings due to replication exceed the update cost incurred. As in the case of caching, the benefits depend on the $LCMR$. Intuitively, if many calls to i originate from zone j, then it makes sense to replicate i at j. However, if i moves frequently, then replica updates incur excessive costs. Let α be the cost savings when a local lookup, i.e., a query of the local VLR, succeeds as opposed to a remote query and β the cost of updating a replica, then a replication of the location of user i at zone j is *judicious* if

$$\alpha * C_{i,j} \geq \beta * U_i \tag{1}$$

where $C_{i,j}$ is the expected number of calls made from zone j to i over a time period T and U_i the number of moves made by i over T.

Besides cost savings, the assignment of replicas to zones must take into account other parameters as well, such as the service capacity of each database and the maximum memory available for storing replicas. The replication sites for each user may be kept at its HLR. Besides location information, other information associated with mobile users may also be replicated [35]. Such information may include service information such as call blocking and call forwarding, as well as QOS requirements such as minimum channel quality or acceptable bandwidth. Unlike location information which is needed at the caller's region, service and QOS information is needed at the location at which the call is received. Approaches similar to those used for replication of location information can be used to replicate service information at sites that are frequently visited by a mobile user in place of sites from which most calls for that user originate.

Finally, instead of the exact location of a user, more coarse location information, e.g., the user's current partition, may be replicated. The coarseness or *granularity* of location replicas presents location schemes with a trade-off between the update and the look-up costs. If the information replicated is coarse then it needs to be updated less frequently in the expense of a higher look-up resolution cost.

Choosing the network sites at which to maintain replicas of the current location of a mobile user resembles the file allocation [10] and the database allocation [28] problem. These classical problems are concerned with the selection of sites at which to maintain replicas of files or database partitions. The selection of sites is based on the read/write pattern of each file or partition, that is the number of read and write operations issued by each site. In the case of location management, this corresponds to the look-up/update pattern of a user's locations. Most schemes for file or database allocation are static, that is they are based on the assumption that the read/write pattern does not change.

We describe next four per-user replication schemes. The first one takes into account resource restrictions and is centralized, whereas the second one does not place any such global restrictions and thus is distributed. The first two algorithms are for two-tier schemes, while the third one is applicable to tree-structured hierarchical architectures. The last algorithm is not developed specifically for location management but treats the problem of dynamic data allocation in its general form. It is a distributed algorithm that considers no global restrictions. It is applicable to any architecture, but it is proven to be optimal for tree-structured hierarchical schemes.

6.1 Per User Profile Replication

The objective of the per user profile approach [36] is to minimize the total cost of moves and calls, while maintaining constraints on the maximum number r_i of replicas per user P_i and on the maximum number p_j of replicas stored in the database at zone Z_j. Let M be the number of users and N be the number of zones. A replication assignment of a user's profile P_i to a set of zones $R(P_i)$ is found, such that the system cost expressed as the sum: $\sum_{i=1}^{N} \sum_{j=1, Z_j \in R(P_i)}^{M} \beta * U_i - \alpha * C_{i,j}$ is minimized and any given constraints on the maximum number of replicas per database at each zone and on the maximum number of replicas per user are maintained.

To this end, a flow network F is constructed as follows. The vertices of the graph correspond to users P_i and zones Z_j. There are two special vertices, a source vertex s, and a sink vertex t. A pair (c, p) of a cost, c, and a capacity, p, attribute is associated with each edge. An edge is added from s to all P_i with $(c, p) = (0, r_i)$ and from all Z_j to t with $(0, p_j)$. An edge from P_i to Z_j with $(c, p) = (\beta * U_i - \alpha * C_{i,j}, 1)$ is added only if it is judicious to replicate P_i at Z_j, i.e., if Inequality (1) holds. Then, computing a minimum-cost (min-cost) maximum-flow (max-flow) on F finds the requested assignment.

In Figure 6, a simple flow network of a system with four mobile users and 3 zones is depicted. The capacity attribute 2 on edge (s, P_1) indicates that P_1's profile can be replicated in at most two zones. The capacity attribute 3 on edge (Z_1, t) indicates that the database at zone Z_1's can store at most three replicas. Finally, in the pair $(-6, 1)$ on edge (Z_1, P_1), the cost attribute -6 indicates that replicating P_1's profile in zone Z_1 will yield a net cost saving of six over not replicating, while the capacity attribute 1 indicates that P_1 should be replicated at most once in Z_1.

Adaptation of the replica assignment to changing calling and mobility patterns is also discussed in [35]. Let $F_{\tau_{new}}$ represent the flow network solution for a new calling and

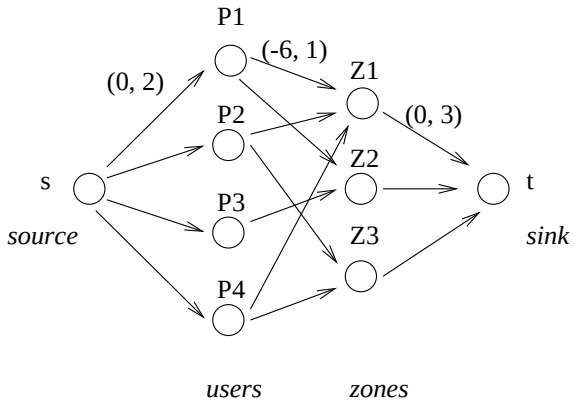

Figure 6: Example of a Flow Network

mobility pattern τ_{new} and $F_{\tau_{old}}$ represent the flow network solution for the previous pattern τ_{old}. An algorithm is presented that incrementally computes the min-cost max-flow of $F_{\tau_{new}}$ given the min-cost max-flow of $F_{\tau_{old}}$. A desired property of the replica assignment algorithm is to keep the cost of evolution from $F_{\tau_{old}}$ to $F_{\tau_{new}}$ low by avoiding radical changes in the replication plan. To this end, two approaches are proposed: (1) a tempered min-cost max-flow, that factors in the cost of replica reassignments when augmenting paths, and (2) a minimum mean cycle canceling algorithm, that augments flow along cycles with the minimum mean cost, where the cost expresses the number of replica reassignments.

6.2 Working Set Replication

The working set method [31] relies on the observation that each user communicates frequently with a small number of sources, called its working set, thus it makes sense to maintain copies of its location at the members of this set. The approach is similar to the per-user replication except from the fact that no constraints are placed on the database storage capacity or the number of replicas per user. Consequently, the decision to provide the information of the location of a mobile unit P_i at a zone Z_j can be made independently at each unit P_i.

Specifically, Inequality (1) is evaluated locally at the mobile unit each time at least one of the quantities involved in the inequality changes. This happen: (a) each time a call is set up and (b) when the mobile unit moves. In the former case, the inequality is evaluated only if the caller's site is not a member of the working set of the callee. If the inequality is found to hold, the caller's site becomes a member of the set. In the later case, the inequality is re-evaluated for all members of the working set, and the members for which the inequality no longer holds are dropped off the set. This way the scheme adapts to the current call

and mobility pattern. Note that in case (a) all four terms of Inequality (1) need to be recomputed, while in case (b) only the number of moves (U_i) needs to be re-evaluated.

Simulation studies in [31] show that, as expected, when the call to mobility ratio (CMR) value is low the scheme performs like a scheme without replication. When the CMR value is high, the scheme behaves like a static scheme in which the working set for a user is fixed. It is also shown that the performance of this adaptive scheme is not primarily affected by the number of units in the working set but rather by the CMR of each individual unit.

6.3 Replication in Hierarchical Architectures

In hierarchical architectures, the location of a mobile user may be selectively replicated at additional sites in the hierarchy. These sites are not necessarily leaf nodes. Specifically, as in the replication schemes for two-tier architectures, the location of a user is replicated at a node only if the cost of replication does not exceed the cost of non replication. However, in a hierarchical location database scheme, if a high $LCMR$ value is the basic criterion of selecting replication sites, databases at higher levels tend to be selected as replication sites over databases at lower levels, since they process higher $LCMR$ values. Recall that the $LCMR$ for an internal node is the sum of the $LCMR$s of its children. This results in excessive update activities at higher-level databases.

Below we describe a replication algorithm, call *HiPer* proposed in [20]. HiPer's objective is to minimize communication cost. HiPer replicates the location of user i in database j if it is judicious, that is, if the benefits of replication exceeds its costs. It turns out, that it is never judicious to replicate i at j if $LCMR_{i,j} < R_{min}$, while it is always judicious to replicate, if $LCMR_{i,j} \geq R_{max}$. If $R_{min} \leq LCMR_{i,j} < R_{max}$, then whether replication should be performed or not depends on the database topology. The constraints taken into consideration are the maximum number N_{max} of replicas per user and a cap L on the maximum level at which locations may be replicated. An off line algorithm to compute the sites of replication for each user i proceeds in two phases. In the first phase, in a bottom-up traversal, it allocates replicas of i at all databases with $LCMR_{i,j} \geq R_{max}$ as long as the number of allocated replicas n does not exceed N_{max}. In the second phase, if $n \leq N_{max}$, the algorithm allocates the remaining replicas to databases below level L with the largest non negative $LCMR_{i,j} - R_{max}$ in a top-down fashion.

6.4 The ADR Algorithm

The Adaptive Data Replication (ADR) algorithm [46] presents a solution to the general problem of determining an optimal (in terms of communication cost) set of replication sites for an object in a distributed system, when the object's read-write pattern changes dynamically. We will describe the ADR algorithm for the case of tree-structure architectures. Let R be the current replication set of object x, i.e., the sites at which x is replicated currently. According to the ADR algorithm, the replication set R of each object x is updated periodically at a time period T. Specifically, at the end of the time period T, specific sites of the network perform three tests, namely the expansion, the contraction and the switch test described next. First, we introduce related terminology. A site i is an $\overline{R}$-*neighbor*, if it belongs to R but has a neighbor site that does not belong to R. If site R is not a singleton set, a site i is an R-$fringe$ site, if it is a leaf at a subgraph induced by R.

The *expansion test* is performed by each $\overline{R}$-*neighbor* site i. Site i invites each of its neighbor j not in R to join R, if the number of reads that i received from j during the last period is greater the number of writes that i received during the same period from i itself or from a neighbor other than j. The *contraction* test is executed by each R-$fringe$ site i. Site i requests permission from its neighbor site j in R to exit R, if the number of writes that i received from j during the last time period is greater than the number of reads that i received during this period. If site i is both an $\overline{R}$-*neighbor* and an $R - fridge$, it executes the expansion test first, and if the test fails (i.e., no site joins R), then it executes the contraction test. Finally, the *switch* test is executed, when R is a singleton test and the expansion test that the single site i in R has executed fails. Site i asks a neighbor site n to be the new singleton site, if the number of requests received by i from n during the last time period is larger than the number of all other requests received by i during the same period.

The ADR algorithm is shown to be convergent-optimal in the following sense. Starting at any replication scheme, the algorithm converges to the replication scheme that is optimal to the current read-write pattern. The convergence occurs within a number of time periods that is bounded by the diameter of the network.

7 Forwarding Pointers

When the number of moves that a user makes is large relative to the number of calls it receives, it may be beneficial not to update all database entries holding the user's location,

each time the user moves. The application of this optimization in two-tier architectures means that the entry in x's HLR is not updated, each time a mobile unit x moves to a new location [18]. Instead, at the VLR at x's previous location, a forwarding pointer is set up to point to the VLR in the new location. Now, calls to a given user will first query the user's HLR to determine the first VLR at which the user was registered, and then follow a chain of forwarding pointers to the user's current VLR. To bound the time taken by the lookup procedure, the length of the chain of forwarding pointers is allowed to grow up to a maximum value of K. An implicit pointer compression also takes place, when loops are formed as users revisit the same areas. Since the approach is applied on a per-user basis, the increase in the cost of call operations affects only the specific user. The router optimization extensions to IEFT Mobile IP protocol include pointer forwarding in conjunction with lazy caching [21].

The pointer forwarding strategy as opposed to replication is useful for those users who receive calls infrequently relative to the rate at which they relocate. Clearly, the benefits of forwarding depend also upon the cost of setting up and traversing pointers relative to the costs of updating the HLR. An analytical estimation of the benefits of forwarding is given in [18]. It is shown that under certain assumptions and if pointer chains are kept short ($K < 5$), forwarding can reduce the total network cost by 20%-60% for users with call to mobility ratio below 0.5.

To reduce the update cost, forwarding pointer strategies may be also deployed in the case of hierarchical architectures. In a hierarchical location scheme, when a mobile user x moves from zone i to zone j, entries for x are created in all databases on the path from j to $LCA(j,i)$, while the entries for x on the path from $LCA(j,i)$ to i are deleted. Using forwarding pointers, instead of updating all databases on the path from j through $LCA(j,i)$ to i, only the databases up to a level m are updated. In addition, a forwarding pointer is set from node s to node t, where s is the ancestor of i at level m, and t is the ancestor of j at level m (Figure 7). As in caching, the level of s and t varies. In *simple forwarding*, s and t are leaf nodes, while in *level forwarding*, s and t can be nodes at any level. A subsequent caller reaches x through a combination of database lookups and forwarding pointer traversals.

Take, for example, user x located at node 14 that moves to node 17 (Figure 7). Let level $m = 2$. A new entry for x is created in the databases at nodes 17, 6 and 2, the entries for x in the databases at nodes 14 and 5 are deleted, and a pointer is set at x's entry in the database at node 1 pointing to the entry of x in the database at node 2. The entry for x at node 0 is not updated. When a user, say at zone 23, calls x, the search message traverses

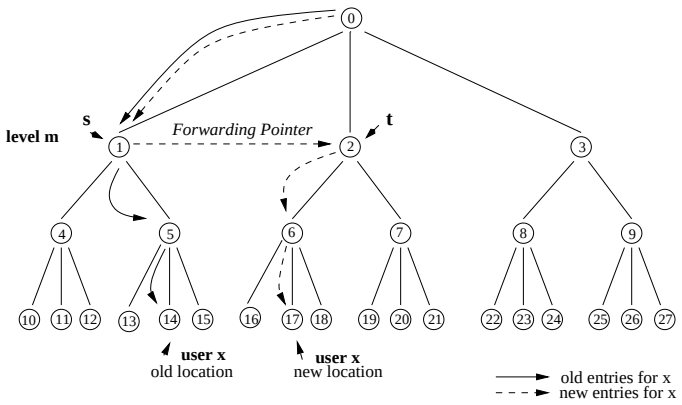

Figure 7: Forwarding Pointers Example (entries are pointers to lower level databases)

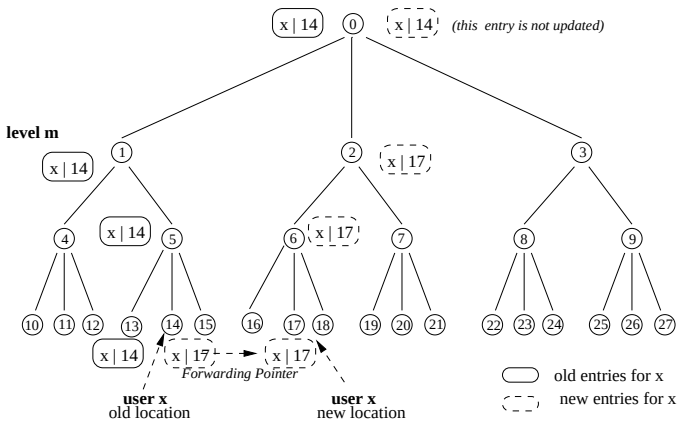

Figure 8: Forwarding Pointers Example (entries are exact addresses)

the tree from node 23 up to the root node 0 where the first entry for x is found, then goes down to 1, follows the forwarding pointer to 2, and traverses downwards the path from 2 to 17. On the other hand, a call placed by a user at 15, results in a shorter route: it goes up to 1, then to 2, and follows the path downwards to 17.

Forwarding techniques can be deployed also for the case in which entries at the internal nodes are actual addresses, rather than pointers to the corresponding entries in lower level databases. The example above is repeated in Figure 8 for this case. Entries for x are updated up to level $m = 2$, and a forwarding pointer at leaf node 14 is set to redirect calls to the new location 17.

An analysis of a forwarding method in a hierarchical location scheme in which entries are actual addresses is presented in [23]. Besides forwarding, the scheme also supports caching: leaf caching (i.e., caching the address of the callee only at the zone of the caller) that is called *jump updates* and level caching (i.e., caching the address of the callee nodes on all

nodes in the search path) that is called *path compression*. All combinations of forwarding (no forwarding (NF), simple forwarding (SF) and level forwarding (LF)) and of caching (jump updates (JU), path compression updates (PC) and no caching (NU)) are considered. Preliminary simulation results are presented for two types of environments: (a) arbitrary moves and calls and (b) short moves and locality of calls (i.e., most calls are received from a specific set of callers). The aggregate cost of search and update is considered, and the cost metric is the number of messages. The simulation showed the combination LF–PC to outperform all other combinations in both environments except of the case of high communication and low mobility and of low mobility and high communication. In these cases, in the second environment, the combination LF–JU performed better due to locality of calls. A per-user adaptive scheme was suggested to choose between the LF–PC and LF–JU combinations based on the call and mobility characteristics. To determine those characteristics, for each mobile unit a sequence is maintained of all moves made and calls received. This sequence determines the degree of mobility of the host (low or high) and whether it has a large number of frequent callers.

Obsolete entries in databases at levels higher than m (e.g., the entry at node 0 in Figures 7 and 8) may be updated after a successful lookup. Another possibility for updates is for each node to send a location update message to the location servers on its path to the root during off-peak hours.

To avoid the creation of long chains of forwarding pointers, some form of pointer reduction is necessary. There are many variations of pointer reduction depending on the type of the reduction and on when it is initiated [29]. For simplicity, we describe these variations for simple forwarding. In *simple purge*, after a successful look-up, a direct pointer is added from the first node of the chain to the current location of the user, while all intermediate forwarding pointers are deleted. This results in a chain of length one. Take for example, chain $11 \rightarrow 18 \rightarrow 26 \rightarrow 14$ that resulted from user x moving from node 11, to nodes 18, 26, and 14, in that order. In simple purging, the entry at 11 is made to point directly to 14 and the entries at nodes 18 and 26 are deleted. Thus, the chain $11 \rightarrow 14$ is produced. In *complete purge*, the entry in the first node of the chain is also deleted, producing a chain of zero length. This involves the deletion of all entries in internal databases on the path from the first node to the LCA of the first node and the current location, and the addition of entries in internal databases on the path from the LCA to the current location. For instance, for the chain $11 \rightarrow 18 \rightarrow 26 \rightarrow 14$, besides the entries for x at 18, and 26, the entry for x at 11 is also deleted. Then, the entries in higher-level databases leading to 11 are

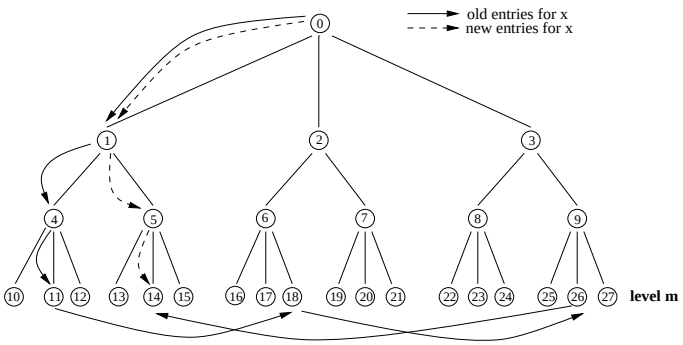

Figure 9: Example of Pointer Purging

also deleted. In particular, the entry for x at 4 is deleted and entries are set at nodes 1, 5, and 14 leading to 14, the new location (see Figure 9). Purging of forwarding pointers, either simple or complete, can occur at calls or moves. *Purging at calls* is initiated when, after a successful lookup, the first node of the chain is reached. *Purging at moves* is initiated when a system-defined maximum on the length of the chain is reached. Alternatively, forwarding pointers may be purged periodically.

Forwarding pointer techniques find applications in mobile software systems, to maintain references to mobile objects, such as in the Emerald System and in SSP chains. Emerald [22] is an object-based system in which objects can move within the system. SSP chains [34] are chains of forwarding pointers for transparently migrating object references between processes in distributed computing. The SSP-chain short-cutting technique is similar to the simple update at calls method.

8 Taxonomy of Location Management Techniques

The techniques proposed in the previous sections are based on exploiting knowledge about the calling and moving behavior of mobile objects. Basically, two characteristics are considered: *stability* of calls and moves and *locality* of moves and calls. Stability in the case of calls means that most calls for a user originate from the same set of locations. Stability of moves refers to the fact that users tend to move inside specific regions. Locality refers to the fact that the cost of a lookup or update operation increases with the distance traveled. Local operations such as moves to neighbor locations or calls from near-by places are common and should cost less than remote operations.

Several more specific types of movement and calling behaviors have been identified in the literature. For example, users usually move to nearby locations more often than to

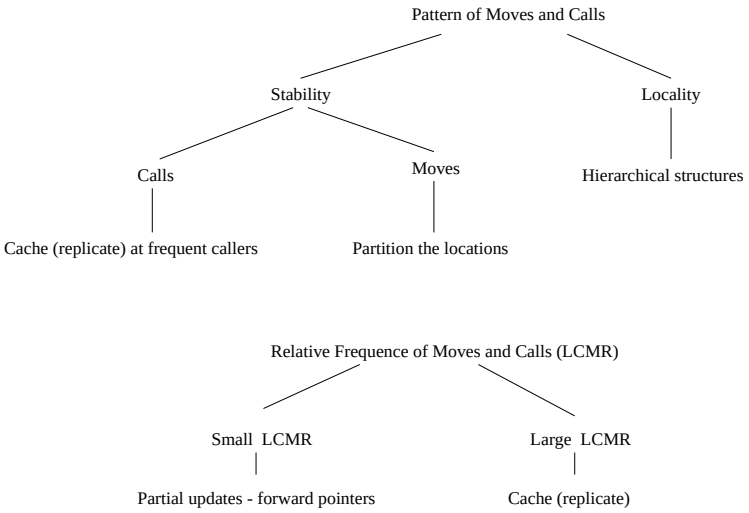

Figure 10: Techniques along the Dimensions of Locality, Stability, and CMR (call to mobility ratio).

remote locations, or follow a certain mobility pattern that is the same over short periods of time. For example, users may follow a daily routine, e.g., drive from their home to their office, visit a predetermined number of customers, return to their office, and then back to their home. This pattern can change but remains fixed for short periods of time. Another possible moving behavior is that there is an epicenter (e.g., home location) of movement. Similarly, various calling types are possible. For instance, there may be time locality of calls, or each user may receive most of its call for a specific set of locations, e.g., friends, family, business associates. Another determinant factor in designing location techniques is the relative frequency of calls and moves expressed in the form of some call to mobility ratio. In general, techniques tend to decrease the cost of either the move or call operation in the expense of the other. Thus, the call to mobility ratio determines the efficacy of the technique. Figure 10 summarizes the various techniques that exploit locality, stability and the call to mobility ratio. This technique are orthogonal; they can be combined with each other.

Besides developing techniques for the efficient storage of location information, the advancement of models of movement can be used in guiding the search for the current location of a mobile object (see for example, [33, 3]), when the stored information about its location is not current or precise. For instance, potential locations may be searched in descending order of the probability of the user being there.

An important parameter of any calling and movement model is time. The models should capture temporal changes in the movement and calling patterns and their relative frequency

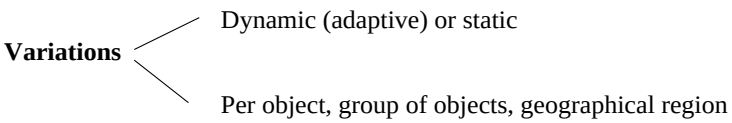

Figure 11: Further Taxonomy of Location Techniques.

as they appear during the day, the week or even the year. For instance, the traffic volume in weekends is different than that during a workday. Thus, dynamic adaptation to the current pattern and ratio is a desirable characteristic of location techniques. Another issue is the basis on which each location technique is employed. For instance, a specific location technique may be employed on a per user basis. Alternatively, the technique may be adopted for all PCS users or for a group of users based either on their geographical location (i.e, all users in a specific region), on their mobility and calling characteristics (i.e., all users that receive a large number of calls) or a combination of both. Figure 11 summarizes these two dimensions of location techniques.

Table 2 and 3 summarizes correspondingly the variations of the two-tier and hierarchical location scheme and their properties.

Since the performance of most location techniques depends on the call to mobility ratio (CMR), in order for the system to adapt to the most appropriate technique based on the current CMR, dynamically estimating the current value of the CMR is a central issue. One approach to estimating $CMRs$ is to calculate running estimates of CMRs on a per user basis. Two such strategies are proposed in [19]. The *running average algorithm* maintains for every user the running counts of the number of incoming calls and the number of times that the user changes location. When the distribution of the incoming call process or the user movement process changes, a variation of this procedure, called the *reset-K algorithm*, gives more accurate estimations. Another approach is to maintain information about the CMR, for instance in the HLR, and download it during off-peak hours. Analytical estimations of the CMR are also possible. For instance, if the coming call stream to a user is consider a Poisson process with arrival rate λ and the time a user resides in a region has a general distribution with mean $1/\mu$, then $LCMR = \lambda/\mu$. Finally, traces of actual moving users can be used (for example, the Stanford University Mobile Activity TRAces (SUMATRA)[38].

Finally, another parameter that affects the deployment of a location strategy is the topology of network sites, how they are populated and their geographical connectivity. How the strategy scales with the number of mobile objects, location operation and geographical distribution is also an important consideration.

Method	Variations		Applicable when:
Caching When x is called by y, cache x's location at y's zone	*Eager caching:*	Cache update overhead occurs at moves	Large LCMR Call Stability
	Lazy caching:	Cache update overhead occurs at calls	
Replication Selectively replicate x's address at the zones from which it receives the most calls	*Per-user Profile Replication:*	Additional constraints are set on the number of replicas per site and on the number of replicas per user	Large LCMR Call Stability
	Working Set:	Adaptive and distributed: the replication sites are computed dynamically by each mobile host locally	
Forwarding Pointers When x moves, add a forwarding pointer from its old to its new address	Restrict the length of the chain of forwarding pointers		Small LCMR

Table 2: Summary of Enhancements to the Basic Two-Tier Scheme. $LCMR$ stands for the Local Call to Mobility Ratio.

Method	Issues/Variations	Appropriate when:
Caching When x at zone i is called by user y at zone j, cache at a node on the path from j to LCA(i, j) a pointer to a node on the path from i to LCA(i, j) to be used by any subsequent call to x from zone j.	Up to which tree level to maintain cache entries When to update cache entries	Large CMR Call Stability
Replication Selectively replicate x's location at internal and/or leaf databases.		Large CMR Call Stability
Forwarding Pointers When x moves from cell i to cell j, instead of updating all databases on the path from i to LCA(i, j) and from LCA(i, j) to j, update all databases up to some level m and add a forwarding pointer at the level m ancestor of i to point to the level m ancestor of j.	When and how to purge the forwarding pointers Setting the level m	Small LCMR
Partitions Divide the locations into sets (partitions) so that the user moves inside a partition frequently and crosses the boundary of a partition rarely. Keep information about the partition in which the user resides instead of its exact location		Move Stability

Table 3: Summary of Proposed Enhancements to Hierarchical Location Schemes.

Location strategies are evaluated based on two criteria, namely, the associated database and network overhead. In terms of database operations, various objectives are set including minimizing (a) the total number of database updates and queries, (b) the database load and size, and (c) the latency of each database operation. In terms of communication, location schemes aim at reducing among others (a) the total number of messages, (b) the number of hops, (c) the distance traveled, (d) the number of bytes generated, and (e) the sum of the traffic on each link or over all links.

9 Consistency and Recovery

The focus of this section is on consistency and recovery issues for location databases. Moves and calls are issued asynchronously and concurrently. Since each of them results in number of database operations, concurrency control is required to ensure correctness of the execution of these operations. In the case of a location database failure, database recovery is also required. We discuss recovery in the context of two-tier location schemes. Approaches to handling recovery in hierarchical schemes and their enhancements is an interesting, but less studied, research problem.

9.1 Concurrency Control

Since call and move operations arrive concurrently and asynchronously, concurrency control issues arise. If no special treatment is provided for concurrency, a call may read obsolete location data and fail to track the callee. In this case, the call is lost and is reissued anew. This simple method does not provide any upper bound on the number of tries a call has to make before locating a moving user.

Concurrency issues get more involved in hierarchical location schemes. In such schemes, a lookup operation results in a sequence of query operations issued at location databases at various levels in the hierarchy. Similarly, a move operation causes a sequence of update operations to be executed on various location databases. The underlying assumption so far was that moves and calls arrive sequentially and they are handled one at a time. Thus, it was assumed that there is no interleaving between the queries and the updates of the various call and move operations. This is a reasonable assumption only if all network and database operations are performed in negligible time. There are various approaches to the problem. For instance, setting at the old address a forwarding pointer to the new location is necessary to ensure that calls that were issued prior to the movement and thus arrive at

the old address will not be lost. If a transactional approach is adopted, traditional database concurrency control techniques are used to enforce that each call and move operation is executed as a transaction, i.e., an isolated unit. This approach is highly impractical, since, for instance, acquiring locks at all distributed databases involved in a call or move operation causes prohibitive delays.

A more practical approach is based on imposing a specific order on the way updates are performed. In particular, upon a move operation from i to j, first entries at the path from j to $LCA(i, j)$ are added in a bottom-up fashion and then the entries at the path from the $LCA(i, j)$ to i are deleted in a top-down fashion. Special care must be given so that during the delete phase of a move operation, an entry at a level $k-1$ database is deleted only after servicing all lookups for higher-level databases. For an application of this approach to the regional matching method refer to [5].

When a replication scheme is used, there is a need for deploying coherency control protocols, to maintain consistent replicas every time the user moves. Coherency control is a well-studied problem in transaction management [8]. However, traditional approaches based on distributed locks or timestamps may be expensive, thus other techniques that ensure a less strict form of replica consistency may be advanced. For example, if there is an HLR or a master copy that is always consistent, i.e, maintains the most up-to-date location, then a lookup can rely on this copy to locate the user when the location at a replica proves to be obsolete. Another approach is to use forwarding pointers at the old location to handle any incoming calls directed there from obsolete replicas.

9.2 Failure Recovery

Database recovery is required after the failure of a location database. In the case of the VLR/HLR either the VLR, the HLR, or both may be periodically checkpointed. If this is the case, after the failure the backup is restored. However, some of the records of the backup may be obsolete.

9.2.1 VLR Failure Restoration

If the VLR is checkpointed, the backup record is recovered and used upon a failure. If the backup is obsolete, then all areas within the VLR must be paged to identify the mobile users currently in the VLR's zone. Thus, the restoration procedure is not improved by the checkpointing process. In [25], the optimal VLR checkpointing interval is derived to balance the checkpointing cost against the paging cost. GSM exercises periodic location

updating: the mobile users periodically establish contact with the network to confirm their location. It is shown that periodic confirmation does not improve the restoration process, if the confirmation frequency is lower than 0.1 times of the portable moving rate [25]. A mechanism is proposed, called *location update on demand*, which eliminates the need for periodic confirmation messages. After a failure, a VLR restoration message is broadcasted to all mobile users in the area associated with the VLR. The mobile users then send a confirmation message. To avoid congesting the base station, each such message is sent within a random period from the receipt of the request.

9.2.2 HLR Failure Restoration

In GSM, the HLR database is periodically checkpointed. After a HLR failure, the database is restored by reloading the backup. If a backup record is obsolete, then when a call delivery arrives, the call is lost. The obsolete data will be updated by either a call origination or a location confirmation from the corresponding mobile user. An estimation of the probability of lost calls can be found in [25]. In IS-41, after a HLR failure, the HLR initiates a recovery procedure by sending an "Unreliable Roamer Data Directive" to all its associated VLRs. The VLRs then remove all records of mobile users associated with that HLR. Later, when a base station detects the presence of a mobile portable within its coverage area and the portable is registered at the local VLR, the VLR sends a registration message to the HLR allowing it to reconstruct its internal structures in an incremental fashion. Before the location is reconstructed, call deliveries to the corresponding mobile user are lost.

A method called *aggressive restoration* is proposed in [25]. Following this method, the HLR restores its data by requesting all the VLRs referenced in its backup copy to provide exact location information of the mobile users. The probability p_U that the HLR fails to request information from a VLR is estimated. An algorithm is also proposed to identify VLRs that are not mentioned in the backup copy. These VLRs are such that there are portables that move in the VLR between the last HLR checkpointing and the HLR failure and do not move out of the VLR before the failure.

10 Querying Location Data

Besides the efficient support of location look-ups and updates, a challenging issue is the management of more advanced queries that involve the location of moving objects. Examples of such queries include finding the nearest service, as well as more involved queries

such as identifying the shortest route with the best traffic conditions. Such location queries may not include location directly, but may require tracking mobile objects indirectly, e.g., queries that involve data produced and located at mobile hosts. Location queries may be imposed by either static or mobile users and may include databases located at both static and mobile sites.

The management of location queries is complicated by the fact that location is a fast and continuously changing data. Furthermore, there are location queries that may have both a spatial dimension, e.g., involve the position of a user and a temporal dimension, e.g., involve time, for example: "find all objects that will enter a specified region in the next hour" or "what is the weather and traffic within 1 mile from a moving user". Location queries may include *transient data*, that is data whose value changes while the queries are being processed, e.g., a moving user asking for nearby hospitals. Another possible type of location queries are *continuous queries*, e.g., a moving car asking for hotels locating within a radius of 5 miles and requesting the answer to the query to be continuously updated. Issues related to continuous queries include when and how often should they be re-evaluated and the possibility of a partial or incremental evaluation.

An issue that complicates further the processing of location queries is the introduction of imprecision. To control the volume of location updates, an approach is to store imprecise information about the location of moving users. This means that either the stored location is not maintained up-to-date, that is it is not updated each time the user moves, or approximate location information is saved, for instance only the partition where the user resides not its exact location. Furthermore, in a variety of location queries, knowing the exact location of some users may not be necessary. Thus, a new problem that arises in query processing is how to derive an optimal execution plan for a location query that will acquire only the missing information necessary to answer it.

Querying location in cases in which the stored location of a user is approximate is discussed in [16]. In this approach, partitions are defined such as zones between which the user relocates very often constitute a partition while zones between which it relocates infrequently belong to different partitions. The system guarantees *bounded ignorance*, in that the actual and stored location of a user are always in the same partition. Only movements among zones belonging to the same partition are considered. Thus, to determine the actual location of a user, searching in the partition of its stored location is sufficient. In this scenario, deriving an optimal execution plan reduces to determining an optimal sequence in which to search inside the partitions of the users involved in the query.

The transient and continuous aspects of location queries are considered in [37]. The position of a moving object is represented as a function of time. Thus, position changes *continuously* with time even without an explicit update through a database operation. A new data model, called MOST, is proposed to incorporate such *dynamic attributes*. MOST enables queries that refer to future values of dynamic attributes, e.g., retrieve all the airplanes that will come within 30 miles in the next 10 minutes. The answer to future queries is tentative, i.e., it should be considered as correct according to what is currently known.

Another approach to reducing the traffic generated by location updates is exploiting the notion of a route [45]. In this approach, objects move on predefined routes. The current position of an object is modeled as the distance from its starting point along a given route.

Indexing the location of moving objects is another important topic. The problem with a straight-forward use of spatial indexing is that the spatial index has to be continuously updated, since objects are continuously moving. Two methods of indexing location are proposed in [37] in the context of dynamic attributes.

11 Conclusions

Managing the location of moving objects is becoming increasingly important as mobility of users, hosts or programs becomes widespread. This paper focuses on data management techniques for locating, i.e., identifying the current location, of mobile objects. The efficiency of techniques for locating mobile objects is critical since the cost of communicating with a mobile object is augmented by the cost of finding its location. Location management techniques use information concerning the location of moving objects stored in location databases in combination with search procedures that exploit knowledge about the objects' previous moving behavior. Various enhancements of these techniques include caching, replication, forwarding pointers and partitioning. The databases for storing the location of mobile objects are distributed in nature and must support very high update rates since the location of objects changes as they move. The support of advanced queries involving the location of moving objects is a promising yet intriguing research topic.

References

[1] Special Issue on Intelligent Agents. *Communications of the ACM*, 37(7), 1994.

[2] Special Issue on Internet-based Agents. *IEEE Internet Computing*, 1(4), 1997.

[3] I. F. Akyildiz and J. S. M. Ho. Dynamic Mobile User Location Update for Wireless PCS Networks. *ACM/Baltzer Wireless Networks Journal*, 1(2), 1995.

[4] V. Anantharam, M. L. Honig, U. Madhow, and V. K. Kei. Optimization of a Database Hierarchy for Mobility Tracking in a Personal Communications Network. *Performance Eavaluation*, 20:287–300, 1994.

[5] B. Awerbuch and D. Peleg. Concurrent Online Tracking of Mobile Users. In *Proceedings of SIGCOMM 91*, pages 221–233, November 1991.

[6] B. R. Badrinath, T. Imielinski, and A. Virmani. Locating Strategies for Personal Communications Networks. In *Proceedings of the 1992 International Conference on Networks for Personal Communications*, 1992.

[7] M. Baentsch, L. Baum, G. Molter, S. Rothkugel, and P. Sturm. Enchancing the Web's Infrastructure: From Caching to Replication. *IEEE Internet Computing*, 1(2):18–27, March 1997.

[8] P. A. Bernstein, V. Hadjilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems.* Addisson-Wesley, 1987.

[9] G. Cho and L. F. Marshall. An Efficient Location and Routing Schema for Mobile Computing Environments. *IEEE Journal on Selected Areas in Communications*, 13(5), June 1995.

[10] L. W. Dowdy and D. V. Foster. Comparative Models of the File Assignment Problem. *ACM Computing Surveys*, 14(2):288–313, June 1982.

[11] G. H. Forman and J. Zahorjan. The Challenges of Mobile Computing. *IEEE Computer*, 27(6):38–47, April 1994.

[12] GPS - Introduction to GPS Applications. www.redsword.com/gps/apps/index.htm.

[13] GPS - USCG Navigation Center GPS Page. www.navcen.uscg.mil/gps/.

[14] H. Harjono, R. Jain, and S. Mohan. Analysis and Simulation of a Cache-Based Auxiliary User Location Strategy for PCS. In *Proceedings of the 1994 International Conference on Networks for Personal Communications*, March 1994.

[15] T. Imielinksi and B. R. Badrinath. Wireless Mobile Computing: Challenges in Data Management. *Communications of the ACM*, 37(10), October 1994.

[16] T. Imielinski and B. R. Badrinath. Querying in Highly Mobile Distributed Environmnets. In *Proceedings of the 18th International Conference on Very Large Data Bases (VLDB 92)*, 1992.

[17] R. Jain. Reducing Traffic Impacts of PCS Using Hierarchical User Location Databases. In *Proceedings of the IEEE International Conference on Communications*, 1996.

[18] R. Jain and Y-B. Ling. A Auxiliary User Location Strategy Employing Forwarding Pointers to Reduce Network Impacts of PCS. *Wireless Networks*, 1:197–210, 1995.

[19] R. Jain, Y-B. Ling, C. Lo, and S. Mohan. A Caching Startegy to Reduce Network Impacts of PCS. *IEEE Journal on Selected Areas in Communications*, 12(8):1434–44, October 1994.

[20] J. Jannink, D. Lam, N. Shivakumar, J. Widom, and D.C. Cox. Efficient and Flexible Location Management Techniques for Wireless Communication Systems. *ACM/Baltzer Journal of Mobile Networks and Applications*, 3(5):361–374, 1997.

[21] D. B. Johnson and D. A. Maltz. Protocols for Adaptive Wireless and Mobile Networking. *IEEE Personal Communications*, 3(1), 1996.

[22] E. Jul, H. Levy, N. Hutchinson, and A. Black. Fine-Grained Mobility in the Emerald System. *ACM Transactions on Computer Systems*, 8(1):109–133, February 1988.

[23] P. Krishna, N. H. Vaidya, and D. K. Pradhan. Static and Dynamic Location Management in Mobile Wireless Networks. *Journal of Computer Communications (special issue on Mobile Computing)*, 19(4), March 1996.

[24] Y. B. Lin and S. K. DeVries. PCS Network Signaling Using SS7. *IEEE Personal Communications*, June 1995.

[25] Y-B. Ling. Failure Restoration of Mobility Databases for Personal Communication Networks. *Wireless Networks*, 1:367–372, 1995.

[26] S. Mohan and R. Jain. Two User Location Strategies for Personal Communication Services. *IEEE Personal Communications*, 1(1):42–50, 1st Quarter 1994.

[27] B. Clifford Neuman, S. S. Augart, and S. Upasani. Using Prospero to Support Integrated Location-Independent Computing. In *Proceedings USENIX Symposium on Mobile & Location-Independent Computing*, pages 29–34. USENIX, August 1993.

[28] M. T. Ozsu and P. Valduriez. *Principles of Distributed Database Systems.* Prentice Hall, 1991.

[29] E. Pitoura and I. Fudos. An Efficient Hierarchical Scheme for Locating Highly Mobile Users. In *Proceedings of the 7th Iternational Conference on Information and Knoweledge Management (CIKM'98)*, November 1998. To appear.

[30] E. Pitoura and G. Samaras. *Data Management for Mobile Computing.* Kluwer Academic Publishers, 1998.

[31] S. Rajagopalan and B. R. Badrinath. An Adaptive Location Management Strategy for Mobile IP. In *Proceedings of the 1st ACM International Conference on Mobile Cmputing and Networking (Mobicom'95)*, Berkeley, CA, October 1995.

[32] C. Rose. Minimizing the Average Cost of Paging and Registration: A Timer-Based Method. *ACM/Baltzer Wireless Networks Journal*, 2:109–116, 1996.

[33] C. Rose and R. Yates. Location Uncertainty in Mobile Networks: a Theoretical Framework. *IEEE Communications Magazine*, 35(2), 1997.

[34] M. Shapiro, P. Dickman, and D. Plainfosse. SSP Chains: Robust, Distributed References Supporting Acyclic Garbage Collection. Technical Report Technical Report 1799, INRIA, Rocquentcourt, France, November 1992.

[35] N. Shivakumar, J. Jannink, and J. Widom. Per-User Profile Replication in Mobile Environments: Algorithms, Analysis, and Simulation Results. *ACM/Baltzer Journal of Mobile Networks and Applications*, 2(2):129–140, 1997.

[36] N. Shivakumar and J. Widom. User Profile Replication for Faster Location Lookup in Mobile Environments. In *Proceedings of the 1st ACM International Conference on Mobile Cmputing and Networking (Mobicom'95)*, 161-169, October 1995.

[37] A. P. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Modeling and Querying Moving Objects. In *Proceedings of the 13th International Conference on Data Engineering (ICDE 97)*, 1997.

[38] Stanford Pleiades Research Group. Stanford University Mobile Activity TRAces (SUMATRA). www-db.stanford.edu/sumatra.

[39] F. Teraoka, Y. Yokote, and M. Tokoro. A Network Architecture Providing Host Migration Transparency. In *Proceedings of the ACM SIGCOMM Symposium on Communications, Architectures and Protocols*, pages 209–220, September 1991.

[40] M. van Steen, F. J. Hauck, G. Ballintijin, and A. S. Tanenbaum. Algorithmic Design of the Globe Wide-Area Location Service. Technical Report IR–440, Faculty of Mathematics and Computer Science, Vrije University, December 1997.

[41] M. van Steen, F. J. Hauck, P. Homburg, and A. S. Tanenbaum. Locating Objects in Wide-Area Systems. *IEEE Communications Magazine*, pages 2–7, January 1998.

[42] J. Vitek and C. Tschudin, editors. *Mobile Object Systems: Towards the Programmable Internet*. Springer Verlag, LNCS 1222, 1997.

[43] J. Z. Wang. A Fully Distributed Location Registration Strategy for Universal Personal Communication Systems. *IEEE Journal on Selected Areas in Communications*, 11(6):850–860, August 1993.

[44] M. Weiser. Some Computer Science Issues in Ubiquitous Computing. *Communications of the ACM*, 36(7):75–84, July 1993.

[45] O. Wolfson, S. Chamberlain, S. Dao, L. Jiang, and G. Mendez. Cost and Imprecision in Modeling the Position of Moving Objects. In *Proceedings of the 14th International Conference on Data Engineering (ICDE 98)*, 1998.

[46] O. Wolfson, S. Jajodia, and Y. Huang. An Adaptive Data Replication Algorithm. *ACM Transactions on Database Systems*, 22(2):255–314, June 1997.