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
cssamara@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, moving objects, spatio-temporal databases

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 [17, 26, 50] 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 services. In such systems, the signaling and database traffic for locating mobile users is expected to increase dramatically [67]. 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 [66, 1] is a popular

* *To appear in IEEE Transactions on Knowledge and Data Engineering*

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 [68], 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 [8].

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 review approaches to the problems of deferring updates of location databases and saving imprecise location information. In Section 10, we focus on issues related to concurrency and fault-tolerance and in Section 11 on issues related to answering complex queries about the location of moving objects. We conclude in Section 12 by summarizing.

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.

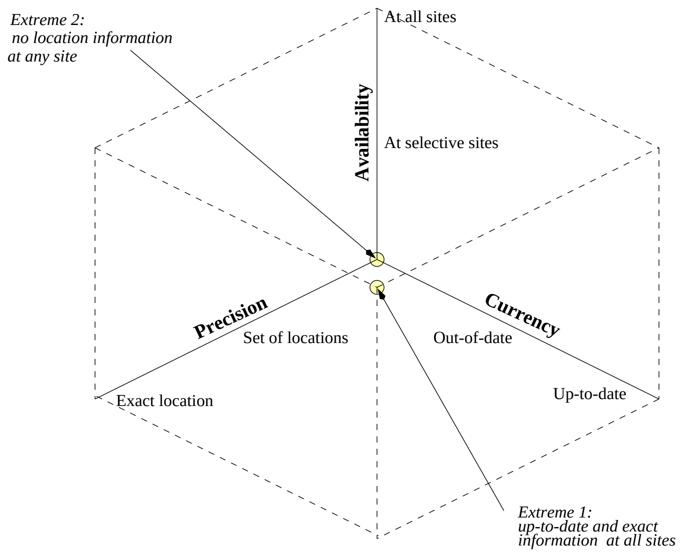

Figure 1: Approaches to Saving Location Information

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 lookups 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 databases 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 [26]). 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 with satellites that will provide the bridge between these different islands of population density.

PCSs involve two types of mobility: terminal and personal mobility [43]. *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 [52]. 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 [67]. 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) [40] 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 [61], 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

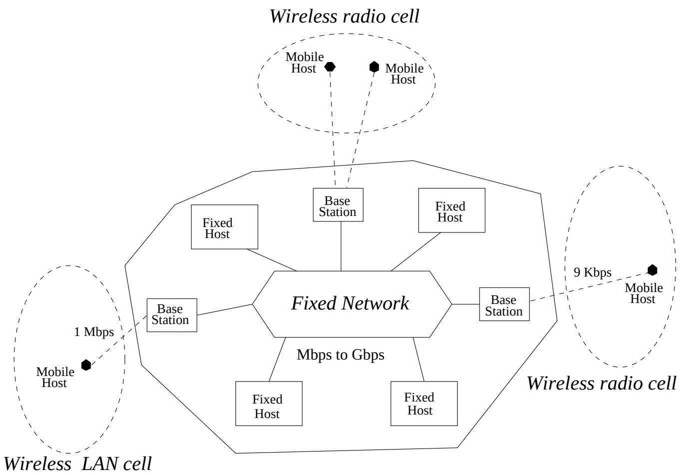

Figure 2: Wireless Computing Architecture

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) [25]. Other applications may involve updating environmental parameters and selecting locally available computing resources (e.g., nearest printer) [44]. 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) [19, 18]. 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 [69]. 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 two most common approaches are a two-tier scheme in which the current location of each moving user is saved at two network locations and a tree-structured distributed database in which space is hierarchically decomposed in sub-regions. We also describe a graph-theoretic approach that employs regional directories. Finally, we refer briefly to a centralized database approach.

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 network location (zone) pre-specified for each user. It 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 [43].

At the Internet networking level, mobile IP [47] is a modification to wireline IP that allows users to continue to receive messages independently of their point of attachment to the Internet. Mobile IP is designed within the IETF (Internet Engineering Task Force) and is outlined in a number of Request for Comments (RFCs) [28]. Wireline IP assumes that the network address of a node uniquely identifies the node's point of attachment to the Internet. Thus, a node must be located on the network indicated by its IP address to receive messages destined to it. To remedy this, in mobile IP, there are two IP addresses associated with each mobile node. One address, known as the *home address* of the node, is used to identify the node and is treated administratively just like a permanent IP address. When away from its home network, a *care-of-address* is associated with the mobile node and reflects the mobile node's current point of attachment. The care-of-address is either the address of a foreign agent which is a router on the visited network that provides services to the mobile node or a co-located address which is an address temporarily acquired by the mobile node. When a mobile node is away of its home, it registers its care-of-address with its home address. Then to deliver any messages, the home agent tunnels them to the care-of-address.

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 where sites are geographically widely dispersed. 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 an 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 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 to 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

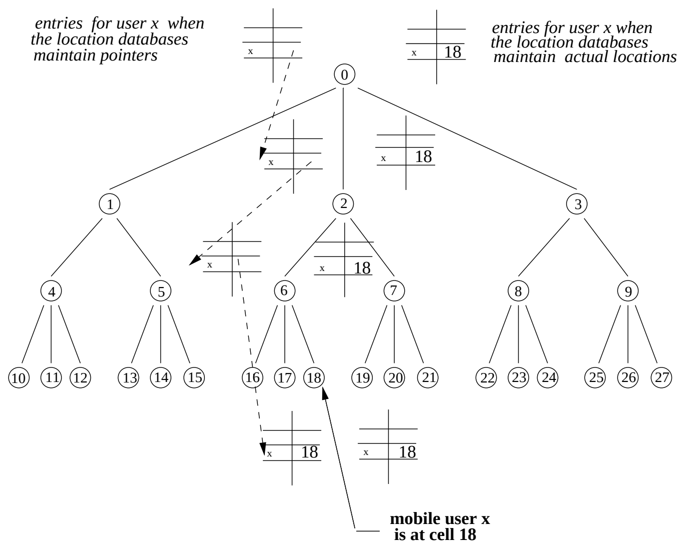

Figure 3: Hierarchical Location Schema. Location databases' entries at the left are pointers at lower level databases, while location databases' entries at the right are actual locations.

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 located at node 18 (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 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.

Proposals for hierarchical versions have also been made within the context of Mobile IP [47]. In this case, the foreign agents are arranged hierarchically in the regional topology. Each ancestral foreign agent considers the mobile node to be register at the foreign node just below it in the

(+)	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

hierarchy. A hierarchical arrangement of location entries is also possible in ATM networks [65].

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 [67].

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. Another 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 [64]. 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 [6] is to favor local operations, in that moves to nearby locations or searches for nearby 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, δ}
    /* Update directory RD_i */
        for all sites u in Write(v)
            update entry
    add a forwarding pointer at RD_{i+1}
```

3.4 A Centralized DBMS

The architectural alternatives presented so far are distributed, in that the locations of moving objects are stored in different network sites. For some applications, it is feasible to use a centralized approach in which the locations of all moving objects are stored in a single centralized Database Management System (DBMS). Such applications include for example a trucking company's database, a database representing the location of taxi-cabs or in the context of military applications, a database that keeps track of the position of all moving objects in a battlefield. In this case, all location queries and updates are directed to the central DBMS. Using an existing spatial DBMS is not sufficient, since existing DBMS do not handle well continuously changing data, such as the location of moving objects. Thus, most current research in this area [71] deals with extending spatial databases with such capabilities.

4 Placement of Databases

Maintaining location information at all nodes in the hierarchy results in cost-effective lookups. However, it increases the number of databases that must be 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 traverses the tree from node j up to the lowest level 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 [7]. 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.

Next we consider three alternative architectures: maintaining location information at selective internal nodes so that some performance metric is optimized, a dynamic hierarchical database architecture, and partitions.

4.1 Optimization

The placement of location databases in the hierarchy 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 combination 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 [5]. The objective there is to minimize the number of updates and accesses per unit of 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 Dynamic Hierarchical Database Architecture

The dynamic hierarchical scheme proposed in [24] extends the two-tier scheme by introducing a new level of databases called *directory registers* (DRs). Each DR covers a number of location zones. Its primary function is to periodically compute and store the location configuration for the mobile units located in zones under its service. There are three types of location addresses that can be stored at a DR. In particular, besides maintaining the *local* address of all mobile units located in its coverage, each DR also maintains for selected mobile units either a *direct remote* address to their current location or an *indirect remote* address to their current serving DR. For each particular mobile unit, the selection of the set of DRs that maintain direct remote addresses or indirect remote addresses for it is periodically determined based on the mobility and call arrival patterns of the unit. The HLR may either store the current zone or the current DR of a mobile unit, again depending on the mobility and call arrival patterns of the unit. In the cases that it is more cost effective not to set up any remote addresses, the scheme reduces to the original two-tier scheme. In contrast to the two-tier and the hierarchical architectures where the strategy for the distribution of location information is the same for all mobile units, in this scheme, the distribution strategy is dynamically adjusted for each mobile unit.

4.3 Partitions

To avoid maintaining location entries at all levels of the hierarchy, and at the same time reduce the search cost, *partitions* are deployed [7]. 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 [7]. 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 [12]. A single partition, called local region, is defined by including all nodes between which the user often moves. The

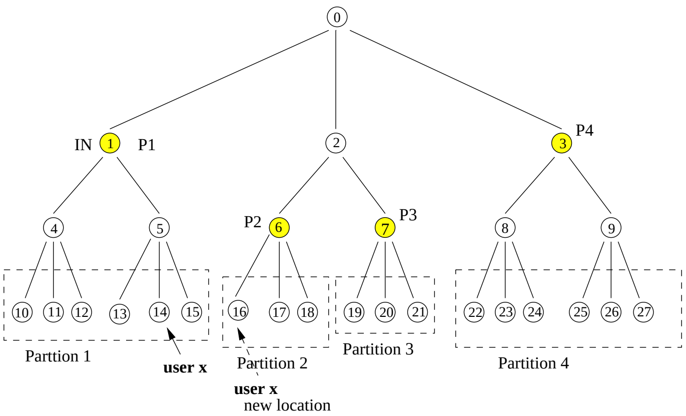

Figure 4: Partitions

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 [32]. 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 this type of caching, the locations of the cache entries for a user's location must be centrally known in order for the updates to be initiated. This leads to scalability problems as well as making the scheme susceptible to fault tolerance problems. 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 [32, 21]. 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 [40] 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 [32]. 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 nearby. In practice, it is expected that $LCMR_T > 7$ [32].

Another approach to cache invalidation, suggested in [38], is to consider cache entries obsolete after a certain time period. To determine when a particular cache should be cleared, a threshold T is used. T it dynamically adapted to the current call and mobility patterns such that the overall network traffic is reduced.

When the cache size is limited, 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 mobile IP, *route optimization* [47] provides a means for any node to maintain a binding cache containing the care-of-address of one or more mobile nodes. Such cache entries are used by the sender to tunnel any messages directly to the care-of-address indicated in its cache. Each entry in the binding cache has an associated lifetime that is specified when the entry is created. The entry is to be deleted from the cache after the expiration of this time period. A lazy procedure is also used to update out-of-date cache entries.

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 apply a static form of caching, e.g., by caching 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

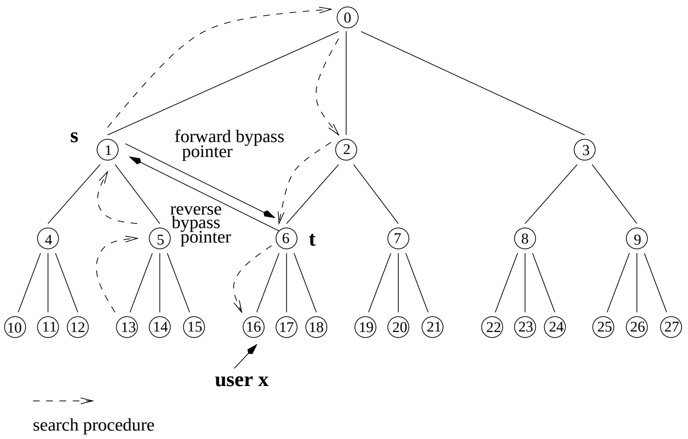

Figure 5: Caching in Hierarchical Location Schemes. For simplicity, the acknowledgment message is not shown; it follows the reverse route of the search procedure.

downwards to j. We also consider an acknowledgment message that returns from j to i. To support caching, during the return path, a pair of bypass pointers, called forward and reverse, is created [30]. 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 to 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 [30]. 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 [30].

Preliminary performance results are reported in [30]. 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 [63, 64] 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 lookup cost, since it increases the probability of finding the location of the callee locally as opposed to issuing a high latency remote lookup. 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

$$\alpha * C_{i,j} \geq \beta * U_i \qquad \text{Inequality} \qquad (1)$$

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.

In addition to cost, the assignment of replicas to zones must take into account other parameters, 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 [56]. 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 lookup costs. If the information replicated is coarse then it needs to be updated less frequently in the expense of a higher lookup resolution cost.

Choosing the network sites at which to maintain replicas of the current location of a mobile user resembles the file allocation [15] and the database allocation [46] 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 lookup/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 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 [57] 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

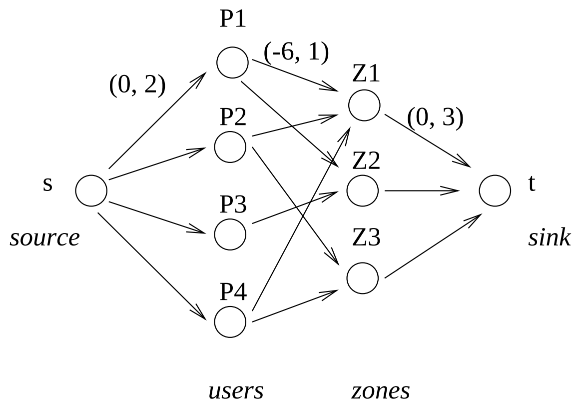

Figure 6: Example of a Flow Network

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 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.

In such a centralized approach, generating, distributing and applying to all sites, a particular replica assignment decision is a time consuming, computational intensive, and bandwidth demanding process. Thus, computing and applying a new replication plan is very expensive, and a graceful adaptation of the replica assignment to changing calling and mobility patterns is very important [56]. Let $F_{\tau_{new}}$ represent the flow network solution for a new calling and 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 [51] 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 [51] 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, in addition to leaf nodes, the location of a mobile user may be selectively replicated at internal nodes of the hierarchy. As in the replication schemes for two-tier architectures, the location of a user should be 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 determining factor for selecting replication sites, then the databases at higher levels will tend to be selected as replication sites over databases at lower levels, since they possess much higher $LCMR$ values. In particular, if a database at level j is selected, all it ancestors are selected as well. Recall that the $LCMR$ for an internal node is the sum of the $LCMR$s of its children. Such a selection would result in excessive update activities at higher-level databases. To compensate, replication algorithms for hierarchical databases must also set some maximum level of the hierarchy at which to replicate.

HiPer proposed in [33] is a family of location management techniques with four parameters: N_{max}, S_{min}, S_{max} and L, where N_{max} determines the maximum number of replicas per user, S_{min} and S_{max} together determine when a node may be selected as a replication site, and L determines the maximum level of the hierarchy at which replicas can be placed. The location of user i is not replicated at j if $LCMR_{i,j}$ is smaller than S_{min}, while it is replicated if $LCMR_{i,j}$ exceeds S_{max}.

If $S_{min} \leq LCMR_{i,j} < S_{max}$, then whether replication should be performed or not depends on a number of constraints placed by the database topology. The constraints taken into account by HiPer are N_{max} and L. 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 $LCMR_{i,j} \geq S_{max}$ 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 $LCMR_{i,j} - S_{max}$ in a top-down fashion.

The optimal values S_{min}^{opt} and S_{max}^{opt} for S_{min} and S_{max} are determined based on whether replication is judicious, that is, if the benefits of replication exceeds its costs. It is shown that $S_{min}^{opt} = b_u/(b_l(2E[LCA] - l_j))$ and $S_{max}^{opt} = b_u/b_l$, where b_u is the network cost of each update message, b_l is the network cost of a lookup message for adjacent nodes in the hierarchy, l_j is the level of node j and $E[LCA]$ is the expected number of sites visited before a replica is found.

6.4 The ADR Algorithm

The Adaptive Data Replication (ADR) algorithm [70] 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-structured architectures. The tree represents a physical or logical communication structure. Two sites are neighbor sites if they are connected through a tree edge. Let R be the current replication set of object x, i.e., the sites at which x is replicated currently. A read of object x is performed from the closest replica in R, while a write of x updates all replicas in R.

Metaphorically, the replication set R forms a variable-size amoeba that stays connected at all times and constantly moves towards the center of the read-write activity. The ADR algorithm updates the replication set R of each object x periodically at a time period T. The replication set expands as the read activity increases and contracts as the write activity increases. 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 below. 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 i is not a singleton set, 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-$fringe$, 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 too expensive, to update all database entries holding the user's location, each time the user moves. Instead, entries may be selectively updated and calls directed to the current location of a user through the deployment of forwarding pointers.

7.1 Two-Tier Architectures

In two-tier architectures, if the mobility of a mobile unit is high while it is located far way from its HLR, an excessive amount of messages is transmitted between the serving VLR and the HLR. Thus, to reduce the communication overhead as well as the query load at the HLR, the entry in x's HLR is not updated, each time the mobile unit x moves to a new location [31]. 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 IETF Mobile IP protocol include pointer forwarding in conjunction with lazy caching [34].

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 [31]. 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.

A method for dynamically determining whether to update the HLR or not is proposed in the local anchoring scheme [23] where a pointer chain of length at most one is maintained. For each mobile unit, a VLR close to it is selected as its local anchor (LA). In some cases, the LA may be the same as its serving VLR. Otherwise, the LA maintains a forwarding pointer to the current VLR of the mobile unit. For each mobile unit, the HLR maintains its serving LA. To locate a mobile unit, the HLR is queried first and then the associated LA is contacted. If the LA happens to be the serving VLR, no further querying is necessary, else the forwarding pointer is used to locate the mobile unit. Since after a call delivery the HLR knows the current location of a mobile unit, the HLR is always updated after a call to record the current VLR. Depending on whether the HLR is updated upon a move. two schemes are proposed: static and dynamic local anchoring. In static local anchoring, the HLR is never updated at a move. In dynamic local anchoring, the serving VLR becomes the new LA if this will result in lower expected costs.

7.2 Hierarchical Architectures

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 can vary. 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 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 also be deployed for hierarchical architectures in which the entries of 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.

Such an architecture with internal nodes storing actual addresses rather than tree pointers is considered in [37] where a performance analysis of forwarding is presented. Besides forwarding, the scheme in [37] 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 on all nodes on 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

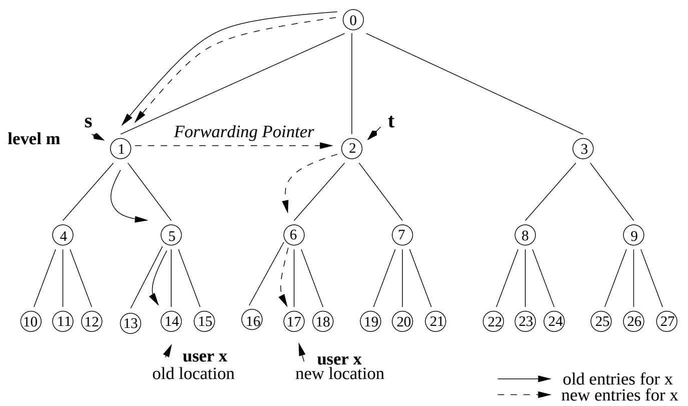

Figure 7: Forwarding Pointers Example (entries are pointers to lower level databases)

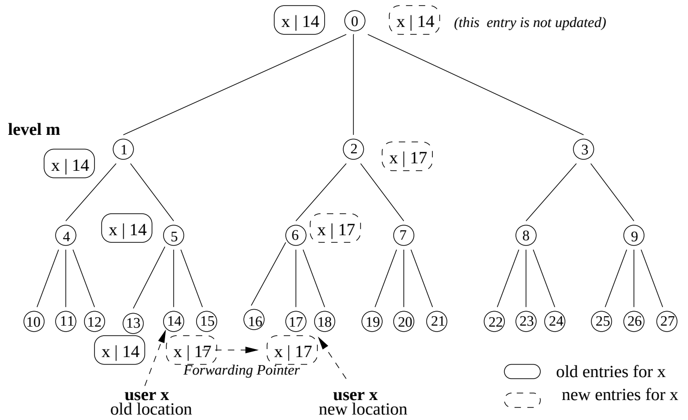

Figure 8: Forwarding Pointers Example (entries are exact addresses)

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 stability of calls (i.e., most calls are received from a specific set of callers). The aggregate cost of search and update is considered. The cost metric is the number of messages for each operation.

For the type (a) environment, the simulation showed the combination SF–PC to outperform all other combinations. The strategies using either NF or LF incurred a high cost of updates at each move. The SF–NU combination suffered due to the very high search costs. Finally, the SF–JU did not perform well as the cached entries were not used very frequently since the calls were arbitrary. For the type (b) environment, SF–PC performed better as well except from the cases of high communication and low mobility and low communication and high mobility. In these cases, the combination SF–JU performed better because jump updates were more effective in reducing the search cost, since there was a specific set of callers. A per-user adaptive scheme was suggested to choose between the SF–PC and SF–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. To reduce the number of forwarding pointers, a variation of caching is proposed in [49]. After a call to user x, the actual location of the user is cached at the first node of the chain. Thus, any subsequent calls to x directed to the first node of the chain use this cache entry to directly access the current location of x, bypassing the forwarding pointer chain. Besides, this form of caching that reduces the number of forwarding pointers that need to be traversed to locate a user, the database hierarchy must also be updated to avoid excessive look-up costs. Besides deleting forwarding pointers, this also involves the deletion of all entries in internal databases on the path from the first node, i, of the chain to the LCA of i and the current location, j, and the addition of entries in internal databases on the path from the LCA to j. 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. The entries for x at nodes 11, 18, and 26 are deleted. Then, the entries in higher-level databases leading to 11 are 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). Two conditions for initiating updates are proposed and evaluated based on setting a threshold either on the number of forwarding pointers or on the maximum distance between the first node of the chain and the current location.

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 [35] is an

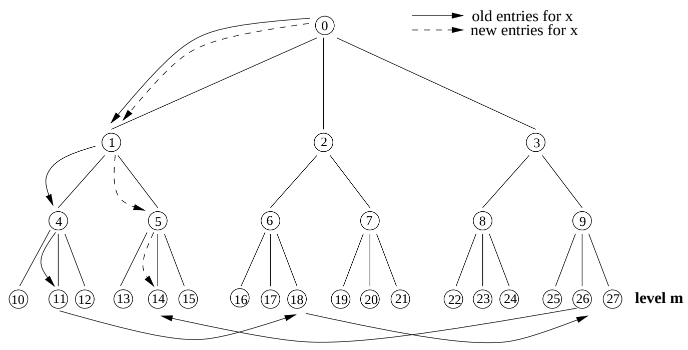

Figure 9: Example of Pointer Purging

object-based system in which objects can move within the system. SSP chains [55] 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 each user originate from the same set of locations, for example, each user may receive most calls from a specific set of friends, family and business associates. Stability of moves refers to the fact that users tend to move inside a specific set of regions. For instance, they 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. Locality refers to the fact that local operations are common. In particular, in the case of calls, a user frequently receives calls from nearby places, while in the case of moves, the user moves to neighbor locations more often than to remote ones.

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. These techniques 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, [54, 4]), 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

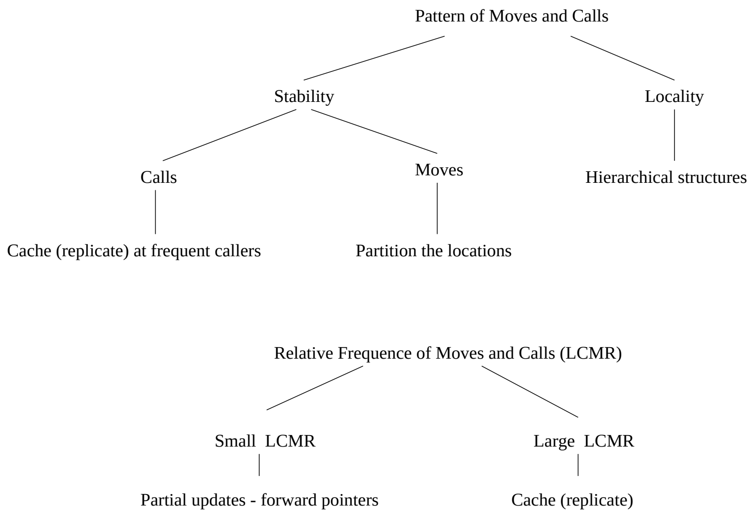

Figure 10: Techniques along the Dimensions of Locality, Stability, and CMR (call to mobility ratio).

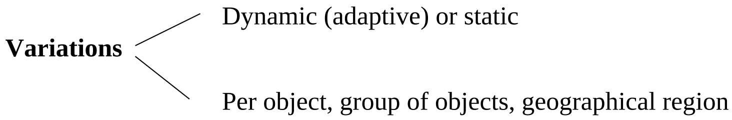

Figure 11: Further Taxonomy of Location Techniques.

temporal changes in the movement and calling patterns and their relative frequency 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 [32]. 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. One problem with the running average algorithm is that estimations are taken from the entire past history of the user's movement and thus the algorithm may not be sufficiently dynamic to adequately reflect the recent history of the user's behavior. When the distribution of the incoming call process or

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.

the user movement process changes, a variation of this procedure, called the *reset-K algorithm*, gives more accurate estimations. With reset-K, running averages are estimated every K incoming calls. 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. Finally, traces of actual moving users can be used (for example, the Stanford University Mobile Activity TRAces (SUMATRA)[59]).

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.

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 Precision and Currency of Location Information

The focus of the previous sections was on efficiently storing, updating and retrieving information about the location of moving objects. However, in some cases, to reduce the update cost, the stored information may not be precise in that it may cover more than one zone (cell). Then, to actually locate the user, after retrieving its stored location, a search is necessary inside all zones covered by the stored location. Another possibility is that the stored location is not kept current, that is, it is not updated after each move. In this case, the cost of actually locating the user includes also the

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.

cost of finding the current location based on the stored one.

9.1 Granularity of Location Information

The granularity of location information differs with respect to how many location zones it covers. In the cellular architecture, this translates to how many and which cells are covered by each registration area. Then to locate a user all cells in the area are polled; a process called paging. There is a trade-off in defining the granularity of a registration area. If it covers a small number of cells, the cost of updates is large, while if it covers a large number of cells, then the cost of searching increases.

Defining the shape and size of each registration area is formulated as a combinatorial optimization problem in [3]. The objective is to minimize the location update cost subject to a constraint on the search cost to locate the user inside the registration area. Since, it turns out that the rectangular shapes are a good approximation to the optimum registration area shapes, the optimization problem is also stated for rectangular registration areas. The optimal registration area is calculated for each particular mobile unit or for each particular class of mobile units based on their respective mobility and call arrival patterns. In [72], the optimal registration area size is calculated for a mesh cell configuration with square shaped cells given the costs of location updates and of searching inside a registration area. Each registration area consists of $k \times k$ cells arranged in a square and the value of k is selected on a per-user basis. The work in [72] uses a different model of mobility from the work in [3].

9.2 Frequency of Updates

So far, we have assumed that the stored information about the location of a moving object is updated each and every time the user moves. However, to reduce the update cost, the stored location information may be updated less frequently. Three strategies for initiating location updates are proposed in [10]: the time-based strategy, the movement-based strategy and the distance-based strategy. In the *time-based* update strategy, the stored location for each mobile user is updated periodically every T units of time. In the *movement-based* update strategy, the stored location is updated after the user has performed a predefined number of movements across zones boundaries. Finally, in the *distance-based* update strategy, the stored location is updated when the distance of the stored location from the actual location of the user exceeds a predefined value D. Analytical performance results show that the distance-based update approach outperforms the other approaches in most cases. However, distance based approaches are more difficult to implement since they require knowing and computing a distance function.

A different approach to signaling location updates is presented in [9]. A subset of all cells is selected and designated as reporting cells. The location of a mobile user is updated only when it enters a reporting cell. The search to locate a mobile user is restricted to all cells that are in the vicinity of the reporting center to which the user last reported. For an arbitrary cellular topology, finding an optimal set of reporting cells is shown to be an NP-complete problem. Thus, optimal and near optimal solutions are advanced for special cases such as for the common topology of hexagonal cells. The reporting cells strategy is static in the sense that the set of reporting cells is fixed. It is also global since the set of reporting cells is the same for all mobile users.

A timer-based approach to location updates is developed in [52]. A timeout parameter T_m is defined as the maximum amount of time to wait before updating the stored location given that the last stored location was m. The set of the time-out parameters T_m can be calculated by the system and communicated to the mobile users as necessary or calculated by the user directly.

A distance-based update strategy is taken by the DOMINO (Databases fOr MovINg Objects) project [58, 69]. In particular, a set of distance-based update strategies, called dead-reckoning policies [45], are proposed that update the database location whenever the distance between the current location and the stored location exceeds a given threshold h. A cost model is developed to estimate the threshold h. The model takes into account the deviation and uncertainty in the estimation of the moving object's position as well as the communication cost of a location update. The deviation of a moving object at a particular time is the distance between the actual location of object x and the location of x stored in the database, e.g, one mile. The uncertainty of a moving object x is the size of the area in which the object x can possibly be, e.g., a circle with radius one mile. Both uncertainty and deviation have a cost or penalty in terms of incorrect decision making which is proportional to the size of the uncertainty and deviation respectively. In the *speed dead-reckoning policy*, the threshold is fixed for each mobile object. In the *adaptive dead reckoning policy*, the threshold h is computed anew after each update so that it minimizes the cost until the next update. The *disconnection detecting dead-reckoning policy* considers the case in which for some reason the object is unable to generate updates. To avoid explicitly contacting the object, the threshold h is continuously decreasing as the tine interval from the last updates increases.

9.3 Search Procedures

When the registration area covers a number of possible locations or the stored location is not current, besides retrieving the stored location of the user, additional searching is necessary. The search procedure first identifies the set of potential locations and then queries the locations in the set. The set of potential locations depends on the update policy and the granularity of the stored information. For instance, in the case of a distance-based strategy, all possible locations are in distance smaller that D from the stored location.

Depending on whether we set any constraints on the delay or on the maximum number of locations that are polled before locating the mobile user, a search is called constrained or unconstrained. The straightforward approach, also known as the "blanket polling" strategy is to query all potential location simultaneously. For the unconstrained case, it is shown in [53] that given a probability distribution on user location, the search strategy that minimizes the expected number of locations polled is to query each location sequentially in order of decreasing probability. It is also shown that this strategy substantially reduces the mean number of polling requests over the blanket approach even after moderate constraints are imposed. The results are extended in [73] for the case where mobile units are allowed to move during the search procedure. It is shown that the optimal strategy is to search the conditionally most likely locations after each polling failure.

In [41], a distance-based update strategy is adopted. An iterative algorithm is proposed based on dynamic programming for generating the optimal threshold distance D. Locations are searched in a shortest-distance-first order such that locations closest to the location where the last location update occurred are queried first. This an unconstrained search; the delay to locate an mobile user is proportional to the distance traveled since the last location update.

In [22], constrained searching is considered for a distance-based update strategy. The delay to locate a user is constrained to be smaller than or equal to a predefined maximum value. When a call arrives, the residing area of a mobile user is partitioned into a number of subareas. These subareas are then searched sequentially. The search in each subareas is by blanket polling, that is all locations in the subarea are simultaneously polled. By limiting the number of subareas to a given value m, the time to locate a mobile user is smaller than or equal to the time required for the m polling operations.

10 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.

10.1 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 [6] and for an application to tree-structure architectures to [49].

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 [11]. 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.

10.2 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.

10.2.1 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 [39], 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 [39]. 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.

10.2.2 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 [39]. 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 [39]. 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.

11 Querying Location Databases

Besides the efficient support of location lookups and updates, a challenging issue is the management of more advanced location queries. Examples of such queries include finding the nearest service when the service or the user is mobile (which is a form of a nearest-neighbor query), or identifying the route with the best traffic condition (which requires applying an aggregation operator to estimate the number of moving users in each route). Another application is sending a message to all users within a specified geographical area for example to perform geographically targeted advertising [27]. Location queries may be imposed by either static or mobile users. In the case in which a single centralized DBMS is used to store the location of all moving objects, most research proposals follow the approach of building additional capabilities for handling moving objects on top of existing DBMSs. There is not much work on providing advanced query capabilities in distributed architectures. However, there is some very recent work on querying network directories that may be applicable to location directories as well.

11.1 Issues

A number of issues render processing location queries different from query processing in traditional database systems in both the centralized and the distributed case:

- The data values representing the location of mobile users are continuously changing.
- Besides a spatial dimension, querying location data has also a *temporal* dimension, thus an important issue is how to express and answer spatio-temporal queries, for instance queries of the following form: what is the location of moving object x at time t.
- There are interesting queries that refer to *future* time, for example: "find all objects that will enter a specified region in the next hour". The answer to such queries is only tentative, that is it should be considered correct according to what is currently known.
- 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 uncertainty, since to control the volume of location updates, the stored information about the location of a mobile object may be imprecise or out-of-date. Furthermore, in a variety of location queries, knowing the exact location of some users may not be necessary. Interesting question are:

– how to model and quantify imprecision in query answering, and

– besides retrieving the stored locations, what is the optimal way to search to acquire the exact locations.

- The protocols for placing, replicating, caching and updating location data must be re-designed to efficiently handle advanced queries in addition to workloads based on look-up and move operations.

- Since the number of moving objects may be large, to answer queries efficiently, we would like to avoid examining the location of all objects. Thus, we would like to build an index on the location attribute. The type of index depends on the architecture of the location databases.

A spatio-temporal query language, called FTL, with temporal operators that refer to the future has been proposed in [58]. FTL augments SQL with temporal (e.g., until, late) and spatial (e.g., inside-region) operators.

11.2 Centralized Database Architecture

Querying moving object databases has been discussed in the context of spatio-temporal databases (for a survey on spatio-temporal databases see for example Chapter 7 of [42] and in particular for indexing [62]). Spatio-temporal databases deal with geometries changing over time; that is with spatial objects whose position as well their extent (i.e., the region they cover) changes with time [16]; queries refer to both the past and the future histories of moving objects. Here we focus on continuously moving objects having a zero extent. We focus on an important type of spatial queries called range queries. An example of a range query is "retrieve the objects that are currently inside a given region P". How such queries are processed depends on how the objects are modeled, and how they are stored and indexed.

Modeling. To model the location of moving objects, a new data model, called MOST was introduced in [58]. The novelty of MOST is the concept of a dynamic attribute, i.e., an attribute whose value changes continuously as a function of time without being explicitly updated. Location is modeled as a dynamic attribute. The value of the dynamic attribute depends on time t. Formally, a dynamic attribute A is represented by three subattributes: A.*value*, A.*updatetime* and a A.$function$. A.$function$ is a function of a single variable t that has value 0 at time $t = 0$. At time A.*updatetime* the value of A is A.*value* and until the next update of A, the value of A at time A.*updatetime*+t_0 is given by A.*value* + A.$function(t_0)$, that is it changes with time according to f. An explicit update of the dynamic attribute may update any of its sub-attributes, e.g., update the function sub-attribute.

The above model has been extended for the case in which mobile objects move on pre-specified routes [69]. This is the case for example of airplanes or vehicles moving on a highway. In this case, three sub-attributes: A.*route*, A.*direction* and A.*speed* are used instead of the function attribute. A.*route* is a line spatial object denoting the route the object is moving on, A.*direction* is a binary indicator having value 0 or 1 indicating towards which endpoint of the route the object is moving, and A.*speed* is a linear function indicating the speed of the moving object. The model is also extended to include information about the potential uncertainty and deviation of the stored location [69, 45].

Representing and Indexing Moving Objects. The indexing problem can be best described by decomposing it into two sub-problems [71]. The first problem concerns the geometric representation of the location attributes in multidimensional space. The issues involved are how to define the multidimensional space and how to map the attributes of a moving object into a region (e.g., point, line) in this space. The object's region is not updated continuously but only when the attributes are explicitly updated. The second problem concerns developing an indexing method appropriate for the proposed representation. Existing spatial methods can be used, however, it is still unclear which one is more appropriate for the location distribution of mobile objects and for the specific geometric representation.

First, assume that objects move on an 1-dimensional line, that is the location y of each object is described as a linear function of time, $y(t) = v(t - t_0) + y_0$, where v is the velocity of the object and y_0 the location of the object at time t_0.

Value-Time Representation and Indexing [71, 36]. This method plots the function y representing the way location changes with time. Thus, the horizontal-axis represents time (t) and the vertical-axis represents the value of location (y). An object is mapped to a trajectory that plots the location as a function of time. In fact, the trajectory is not a line but a semi-line starting at point (t_0, y_0). One way to index the lines is to use a spatial access method, for example each line could be approximated by a minimum bounding rectangle which is then indexed using an R-tree or a R*-tree. However, this approach is problematic [36]. First, the corresponding minimum bounding rectangle covers a large portion of the space, whereas the actual space occupied by the line is small, thus leading to extremely large and overlapping rectangles [62]. Second, it cannot represent infinite objects well. Another approach is to decompose the data space into disjoint cells and store with each cell the set of lines it intersects. A drawback of this approach is that each line has many copies. This approach is taken in [60] that uses a quadtree-based index. The infinite time dimension is partitioned into equal-sized time slices and an index is created for each slice. Theoretically, the union of these indexes is the master index of the whole time-value space being indexed. In practice, however, since the storage space is limited, when the period ΔT of an index is over, the index is disposed and the next index is generated. Thus, the index is reconstructed every ΔT time units; ΔT is called the index reconstruction period. An index reconstruction algorithm is also proposed that is optimal in CPU and disk access overheads.

Intercept-Slope [71] or *Dual Space* [36] *Representation and Indexing.* Consider an object whose location as a function of time is $y(t) = a + ut$, a is called the intercept and u is called the slope. Then the representation space is constructed by the horizontal-axis representing the intercept and the vertical-axis representing the slope. Thus, the object is mapped to the point (a, u) in this space.

The query region is transformed into a polygon. A number of indexing techniques are proposed and analyzed in [36].

The problem becomes more difficult if we consider moving objects in the plane. An important case is when objects move in the plane but their movement is restricted on using a given set of routes on the finite terrain. This is called the *1-5 dimensional problem* in [36]. They propose representing each predefined route as a sequence of connected line segments and indexing the positions of these line segments using a standard spatial access method. The full 2-dimensional problem is harder. In this case, in the value-time representation, the trajectories of moving objects are lines in the space. The dual space representation is not directly applicable. One way to get the dual [36] is to project the lines on the (x, t) and (y, t) planes and then take the dual (intercept-slope) representation for the two lines on these planes. Thus, now a line can be represented by a 4-dimensional point. In [48], the case is considered in which the trajectory of moving objects in the plane is obtained by discretely sampling the movement of objects in time and then using linear interpolation between these samples. Each line of the trajectory is then approximated by a minimum bounding box. An extension of the R-tree is proposed that keeps line segments that belong to the same trajectory together, i.e., in the same or neighbor nodes. This work does not address queries that refer to the future.

Uncertainty in Query Processing. Since the stored location of a moving object may deviate from its actual current location, there is some uncertainty in answering a query. Depending on the bound on the uncertainty of the stored location, it should be possible to calculate a bound on the uncertainty of the answer. The DOMINO project offers two approaches, a qualitative and a quantitative one. In the *qualitative approach* [58, 71], two kinds of semantics, namely the may and must semantics, are incorporated. Under the may semantics, the answer to a range query is the set of all objects that are possibly inside the query polygon P, i.e., the objects whose uncertainty interval intersects P. Under the must semantics, the answer is the set of all objects that are definitely inside P, i.e., the objects whose uncertainty interval are entirely inside P. In the *quantitative approach* [45], the answer is a set of objects each of which is associated with a probability that the object is inside P.

To support such semantics, indexing should be extended. How to extend the value-time representation for the 1-dimensional case to support the may-must semantics is considered in [71]. In this representation, two lines are plotted for each object, one represents the maximum distance from y_0 and the other the minimum distance from y_0. Thus, at time t the value of location is an interval, the uncertainty interval, instead of a point. In this case, instead of being represented by a line or trajectory, an object is represented by a plane (the one between the two lines).

11.3 Distributed Database Architectures

There is not much research in querying distributed location directories. Query processing depends on the type of the architecture, for example in the case of hierarchical architectures, location databases are physically structured based on location. For example, an internal node in the hierarchy, contains location information for all mobile users currently in the geographical area it covers. Thus, it can be viewed as a distributed spatial index.

Location queries in distributed architectures were introduced in [25]. In this approach, the architecture is based on partitions which are sets of locations between which the user relocates very often. A mobile user moves only infrequently to locations that belong to different partitions. The stored location information about a moving object is not its actual location but just the partition to which its actual location belongs. Thus, only movements among partitions generate database updates. The system guarantees *bounded ignorance*, in that the actual and stored location of a user are always in the same partition. To determine the actual location of a user, searching all locations in the partition of its stored location is necessary. Thus, deriving an optimal execution plan for a query involves determining an optimal sequence in which to search inside the partitions involved in the query. A tree-representation of this problem is proposed.

11.4 Service Discovery Protocols

With the wide-spread use of networking and the increasing number of network devises, there is a need for a scalable means to locate services. The location directories we have considered so far associate the name of a mobile object (service) with its location. Many recent approaches consider the problem of finding an appropriate object (service) by specifying a number of desired characteristics for the service.

The Service Location Protocol (SLP) [20] provides a flexible and scalable framework for providing hosts with access to information about the existence, location and configuration of networked services. Traditionally, to locate a service, users provide the name of a network host (which is an alias for its network address) that supports the service. SLP eliminates the need for a user to know the name of a network host. Rather, the user supplies the desired type of service along with a set of attributes which describe the service. Based on this description, the SLP resolves the network address of the service for the user. Client applications are modeled as user agents and services are advertised by service agents. The user agent issues a service request on behalf of the client application specifying the characteristics of the service. The user agent receives a service reply specifying the location of all services in the network with the requested characteristics. The user agent may directly contact the service agents or in larger networks a directory agent. The directory agent functions as a cache. Service agents register the services they advertise in the directory agents. These advertisements must be refreshed or they expire. Services are grouped together using scopes. A scope may indicate a location, administrative grouping, proximity in a network topology or some other category.

Interesting problems related to service location protocols include:

- Modeling services whose location change, e.g., how is the location of a moving service specified,
- storing, caching and replicating directory entries when either the services are mobile and/or the requests originate from mobile clients,

- updating directory entries and refreshing directory caches when the services are mobile and thus their location is fast changing,
- efficient location-aware querying, e.g., finding services based on location attributes, when the client requested the service, or the service is mobile: should the directory be hierarchically structured based on location or should an appropriate spatial index be built on top of it; what is a an appropriate index in this case;
- interoperability: how to relate information available at different layers in the network, e.g., information stored at an HLR, to actually locate a service using a directory service protocol.

Most of the above questions remain open. A hierarchical architecture for service discovery directory is proposed in [14] which is based on the use of a hash-based index. Finally, there has been some very recent research in incorporating database techniques in manipulating network directories including developing a data model and a declarative language for network directories [29] and semantic caching of directory entries [13]. Extending this work for the case of directories that include the location of moving objects is an interesting problem.

12 Conclusions

Managing the location of moving objects is becoming increasingly important as mobility of users, devices and 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 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] A. Abutaleb and V. O. K. Li. Location Update Optimization in Personal Communication Systems. *ACM/Baltzer Wireless Networks Journal*, 3:205–216, 1997.

[4] 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.

[5] 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 Evaluation*, 20:287–300, 1994.

[6] B. Awerbuch and D. Peleg. Online Tracking of Mobile Users. *Journal of the ACM*, 42(5), 1995.

[7] 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.

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

[9] A. Bar-Noy and I. Kessler. Tracking Mobile Users in Wireless Communications Networks. *IEEE Transactions on Information Theory*, 39:1877–1886, 1993.

[10] A. Bar-Noy, I. Kessler, and M. Sidi. Mobile Users: To Update or not to Update? *ACM/Baltzer Wireless Networks Journal*, 1(2), 1995.

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

[12] 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.

[13] S. Cluet, O. Kapitskaia, and D. Srivastava. Using LDAP Directory Caches. In *Proceedings of the PODS Conference*, 1999.

[14] S. E. Czerwinski, B. Y. Zhao, T. D. Hodes, A. D. Joseph, and R. H. Katz. An Architecture for a Secure Service Discovery Service. In *Proceedings of the 5th ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'99)*, 1999.

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

[16] M. Erwig, R. H. Goting, M. Schneider, and M. Vazirgiannis. Spatio-Temporal Data Types: An Approach to Modeling and Querying Moving Objects in Databases. *GeoInformatica*, 3(3), 1999.

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

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

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

[20] E. Guttman, C. Perkins, J. Veizades, and M. Day. Service Location Protocol, Version 2. In *IETF, RFC 2608*, June 1999. ftp://ftp.isi.edu/in-notes/rfc2608.txt.

[21] 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.

[22] J. S. M. Ho and I. F. Akyildiz. A Mobile User Location Update and Paging Mechanism Under Delay Constraints. *ACM/Baltzer Journal of Wireless Networks*, 1(4), 1995.

[23] J. S. M. Ho and I. F. Akyildiz. Local Anchor Scheme for Reducing Signalling Cost in Personal Communication Networks. *IEEE/ACM Transactions on Networking*, 4(5), 1996.

[24] J. S. M. Ho and I. F. Akyildiz. Dynamic Hierarchical Database Architecture for Location Management in PCS Networks. *IEEE/ACM Transactions on Networking*, 5(5), 1997.

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

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

[27] T. Imielinski and J. C. Navas. GPS-Based Geographic Addressing, Routing, and Resource Discovery. *Communications of the ACM*, 42(4), 1999.

[28] IP Routing for Wireless/Mobile Hosts Working Group. RFC Documents. http://www.ietf.org/html.charters/mobileip-charter.html.

[29] H. V. Jagadish, L. V. S. Lakshmanan, T. Milo, D. Srivastava, and D Vista. Querying Network Directories. In *Proceedings of the SIGMOD Conference*, 1999.

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

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

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

[33] 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.

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

[35] 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.

[36] G. Kollios, D. Gunopulos, and V. J. Tsotras. On indexing Mobile Objects. In *Proceedings of the 18th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 1999.

[37] 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.

[38] Y. B. Lin. Determining the User Location for Personal Communications Service Networks. *IEEE Transactions on Vehicular Technology*, 43(3), August 1994.

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

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

[41] U. Madhow, M. L. Honig, and K. Steiglitz. Optimization of Wireless Resources for Personal Communications Mobility Tracking. *IEEE/ACM Transactions on Networking*, 3(6):698–707, 1995.

[42] Y. Manolopoulos, Y. Theodoridis, and V. Tsotras. *Advanced Database Indexing*. Kluwer Academic Publishers, 1999.

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

[44] 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.

[45] A. P. Sistla O. Wolfson, S. Chamberlain, and Y. Yesha. Updating and Querying Databases that Track Mobile Units. *Distributed and Parallel Databases*, 7(3), 1999.

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

[47] C. E. Perkins. *Mobile IP: Design Principles and Practices*. Addison Wesley, 1998.

[48] D. Pfoser, Y. Theodoridis, and C. S. Jensen. Indexing Trajectories of Moving Point Objects. Technical Report Chorochronos Technical Report, CH-99-3, October 1999.

[49] E. Pitoura and I. Fudos. An Efficient Hierarchical Scheme for Locating Highly Mobile Users. In *Proceedings of the 7th International Conference on Information and Knowledge Management (CIKM'98)*, pages 218–225, November 1998.

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

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

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

[53] C. Rose and R. Yates. Minimizing the Average Cost of Paging Under Delay Constraints. *ACM/Baltzer Journal of Wireless Networks*, 1(2), 1995.

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

[55] 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.

[56] 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.

[57] 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 Computing and Networking (Mobicom'95)*, 161-169, October 1995.

[58] 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.

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

[60] J. Tayeb and O. Wolfson O. Ulusoy. A Quadtree-Based Dynamic Attribute Indexing Method. *The Computer Journal*, 41(3), 1998.

[61] 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.

[62] Y. Theodoridis, T. Sellis, T. Papadopoulos, and Y. Manolopoulos. Specification of Efficient Indexing in Spatiotemporal Databases. In *Proceedings of the 10th International Conference on Scientific and Statistical Database Management*, 1998.

[63] M. van Steen, F. J. Hauck, G. Ballintijin, and A. S. Tanenbaum. Algorithmic Design of the Globe Wide-Area Location Service. *The Computer Journal*, 41(5):297–310, 1998.

[64] 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.

[65] M. Veeraraghavan and G. Dommety. Mobile Location Management in ATM Networks. *IEEE Journal on Selected Areas in Communications*, 15(8), 1997.

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

[67] 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.

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

[69] 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.

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

[71] O. Wolfson, B. Xu, S. Chamberlain, and L. Jiang. Moving Objects Databases: Issues and Solutions. In *Proceedings of the 10th International Conference on Scientific and Statistical Database Management*, 1998.

[72] H. Xie, S. Tabbane, and D. Goodman. Dynamic Location Area Management and Performance Analysis. In *Proceedings of the IEEE Vehicular Technology Conference*, 1993.

[73] A. Yener and C. Rose. Highly Mobile Users and Paging: Optimal Polling Strategies. *IEEE Transactions on Vehicular Technology*, 47(4), 1998.

Evaggelia Pitoura received her BSc from the Department of Computer Science and Engineering of the University of Patras, Greece in 1990 and her MSc and PhD in Computer Science from Purdue University in 1993 and 1995 respectively. Since September 1995, she is on the faculty of the Department of Computer Science of the University of Ioannina, Greece. Her main research interests are data management for mobile computing and multidatabases. Her publications include several journal and conference articles and a recently published book on mobile computing. She received the best paper award in the IEEE ICDE 1999 for her work on mobile agents. Evaggelia Pitoura has served on a number of program committees and was program co-chair of the MobiDE workshop held in conjunction with MobiCom 99.

George Samaras received a PhD in computer science from Rensselaer Polytechnic Institute, USA, in 1989. He is currently an Associate Professor at the University of Cyprus. He was previously at IBM Research Triangle Park, USA and taught at the University of North Carolina at Chapel Hill (adjunct Assistant Professor, 1990-93). He served as the lead architect of IBM's distributed commit architecture (1990-94) and co-authored the final publication of the architecture (IBM Book, SC31-8134-00, September 1994). He was member of IBM's wireless division and participated in the design/architecture of IBM's WebExpress, a wireless Web browsing system. He recently (1997) co-authored a book on data management for mobile computing (Kluwer A.P). He has a number of patents relating to transaction processing technology and numerous technical conference and journal publications. His work on utilizing mobile agents for Web database access has received the best paper award of the 1999 IEEE International Conference on Data Engineering (ICDE/99). He has served as proposal evaluator at a national and international level and he is regularly invited by the European Commission to serve as project evaluator and auditor in areas related to mobile computing and mobile agents. He also served on IBM's internal international standards committees for issues related to distributed transaction processing (OSI/TP, XOPEN, OMG). His research interest includes mobile computing, mobile agents, transaction processing, commit protocols and resource recovery, and real-time systems. He is a voting member of the ACM and IEEE Computer Society.