An Efficient Hierarchical Scheme for Locating Highly Mobile Users

Evaggelia Pitoura and Ioannis Fudos
Department of Computer Science, University of Ioannina
GR 45110 Ioannina, Greece
email: {pitoura,fudos}@cs.uoi.gr

Abstract

To accommodate the increase in user population in future personal communication systems, hierarchical architectures of location databases have been proposed. In this paper, a scheme based on forwarding pointers is presented, that reduces the cost of the overall network and database traffic generated by frequent location updates in such hierarchical architectures. To reduce the number of forwarding pointers that need to be traversed to locate a user, auxiliary caching techniques are presented. Various conditions for purging the chain of forwarding pointers and updating the database are also introduced. Special care is given so that the scheme correctly supports the concurrent execution of updates and lookups. The applicability of the scheme and the performance of the caching techniques are demonstrated through a number of experiments for a range of call to mobility ratios and for a variety of moving and calling behaviors.

1 Introduction

Advances in wireless telecommunications and in the development of portable computers have provided for the emergence of mobile computing. In mobile computing, users are not attached to a fixed geographical location, instead their point of attachment to the network changes as they relocate from one support environment to another. Besides mobility in wireless networks, other forms of moving objects are also emerging. Software mobile agents, that roam the Internet searching for information, provide a popular such example.

Since delivering any message to mobile objects requires locating them first, deriving efficient strategies for tracking moving users is central to mobile computing. In particular, for future Personal Communication Service (PCS) systems, with high user populations and numerous customer services, such signaling and database traffic for locating users is expected to increase dramatically [15]. To locate users, distributed location databases are deployed that maintain the current location of mobile users. To accommodate the increased traffic associated with locating moving users, hierarchical distributed database architectures are built [15, 1, 5, 14]. In such architectures, each leaf database covers a specific geographical region and maintains location information for all users currently residing in that region. Location databases at internal nodes of the hierarchy contain information about the location of all users registered in areas covered by the databases at their children nodes.

In this paper, we consider the problem of locating moving users using such hierarchical tree-structured location databases. In particular, to reduce the cost of moves, instead of updating all location databases involved in the hierarchy, only databases up to a specific level of the tree are updated and a forwarding pointer is appropriately set at some lower level database. However, if forwarding pointers are never deleted, then long chains are created, whose traversal results in an excessive increase in the cost of locating users during calls.

We consider two alternative conditions for purging forwarding pointers and for updating the hierarchical database: one based on the maximum number of the forwarding pointers and the other on the distance of the move. In addition, we introduce a number of auxiliary caching techniques that effectively reduce the number of pointers a call has to traverse before locating the callee. Finally, we extend our scheme so that it correctly supports the concurrent execution of move and call operations. The treatment of concurrency is general enough and is applicable in any hierarchical location database. To study the performance of the forwarding scheme along with the caching techniques and the various conditions for purging forwarding chains, we have developed an event-driven simulator. We have performed a number of experiments for a range of call to mobility ratios, for users with different mobility and calling behavior, and for a real-word number of cells. Since whether the database or the communication cost is the dominating factor follows from various system-dependent parameters, we treat each of the costs separately in our analysis. The results clearly show that under certain assumptions and for small call to mobility ratios, the forwarding scheme coupled with appropriate auxiliary caching techniques on a per user basis can reduce both the overall database load and the communication traffic by 60% to 20% depending on the call to mobility ratio.

The rest of this paper is organized as follows. In Section 2, we introduce the forwarding scheme and various strategies for improving its performance. In Section 3, we extend the proposed protocol to handle correctness issues that arise from the concurrent execution of call and move operations. In Section 4, we present our models for the call and mobility behavior of PCS users, briefly describe our cost model,

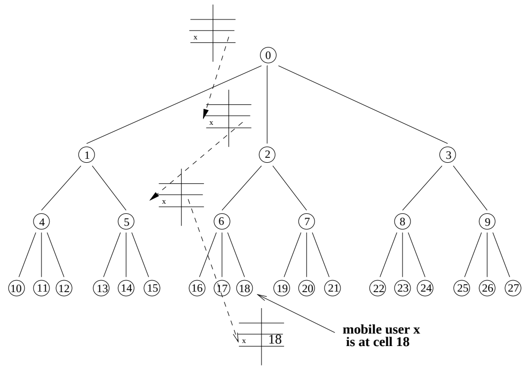

Figure 1: Hierarchical Location Schema.

and report performance results. In Section 5, we compare our work with related research and in Section 6, we offer conclusions.

2 The Location Strategy

PCS users are located in system-defined cells, which are bounded geographical areas. A cell is a uniquely identifiable unit. The location problem in PCS systems can be described in terms of two basic operations, a call operation to locate a moving user, and a move operation to update the information about the location of a mobile user when the user enters a new registration area. The basic location strategies proposed in the IS-41 and GSM [9] standards use a two-tier system in which each moving user is associated with a pair of a Home Location Register (HLR) and a Visitor Location Register (VLR). Calls to a given user, first query the VLR in the caller's region, and then if the callee is not found in this region, they query the callee's HLR. Moves require updating the HLR.

2.1 Hierarchical Location Databases

To avoid "long-distance" signaling messages to the HLR, when most calls and moves are geographically localized, a hierarchical directory structure has been proposed [15, 1, 5] and is under study for the European third-generation mobile system called the Universal Mobile Telecommunication System (UMTS) [4]. Hierarchical location schemes extend two-tier schemes by maintaining a hierarchy of location databases, where 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 cell and contains entries for all users registered in this cell. A database at an internal node maintains information about users registered in the set of cells in its subtree. For each mobile user, this information is a pointer to an entry at a lower level database. For example, in Figure 1 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. It is often the case that the only way that two cells can communicate with each other is through the hierarchy; no other physical connection exists among them. For instance, in telephony, the databases may be placed at the telephone switches.

The hierarchical scheme leads to reductions in communication cost when most calls and moves are 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. In addition, there is no need for binding a user to a permanent home location register (HLR), since the user can be located by querying the databases in the hierarchy. On the other hand, the number of database operations caused by call and move operations increase.

Let $LCA(i, j)$ stand for the least common ancestor of nodes i and j and $lca(i, j)$ be the level of the $LCA(i, j)$. Let the level of the leaves be level 0. When user x moves from cell i to cell j, the entries for x in the databases along the path from j to $LCA(i, j)$, and from $LCA(i, j)$ to i must be 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, and an entry for x is added to the databases at nodes 2, 7, and 20. When a caller located at cell i places a call for a user y located at cell 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 cell 21 to user x (Figure 1), 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. We call the above operations *basic move* and *basic call* respectively.

2.2 Forwarding Pointers

To reduce the update cost, a forwarding pointer strategy is deployed. In this approach, 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 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. The level of s and t may vary. 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 2 (left)). 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. Using forwarding pointers may increase the cost of calls, since it may result in a combination of tree links and forwarding pointers traversals.

In the following, we assume forwarding at the leaf level in which case forwarding pointers are set among leaf nodes, but the analysis applies to other types of forwarding as well. In this case, when a user moves from location i to location j, a forwarding pointer is simply set at node i pointing to node j. Calls originating from a cell k to a user x located at cell l proceed initially as in the basic call: go up the tree till the first entry for x is encountered and then down following the entries to a leaf node, say m. However, x may not be actually located at m, instead the entree at m may be a forwarding pointer to another leaf node. Thus, before actually locating a user, a chain of forwarding pointers may need to be traversed.

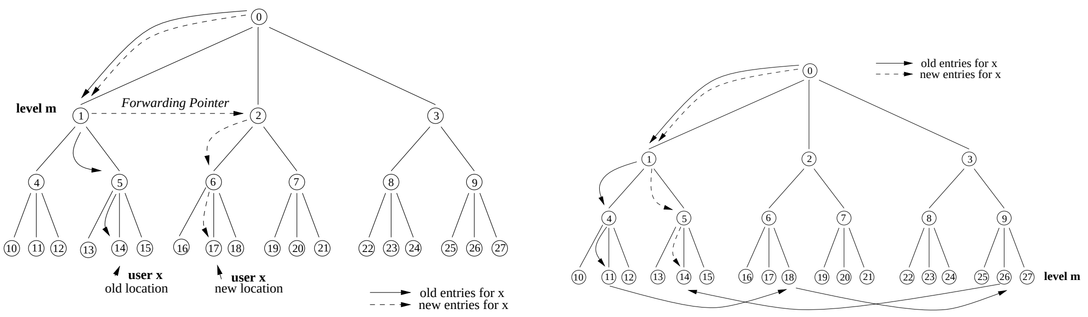

Figure 2: (left) Forwarding Pointers. (right) Full Update of the Directory Entries.

2.3 Caching

We propose auxiliary techniques that reduce the number of forwarding pointers that a call has to traverse before locating the user. The first technique is based on the observation that it should be possible to re-use the information about the user's location from the previous call to that user. To this end, each time a call is set up, after the address of the callee has been resolved, it is saved at the first node of the chain. Thus, any subsequent call to that user can use the cached location to locate the user, instead of traversing a number of forwarding pointers. This is essentially a variation of caching in which instead of caching the address of the callee at the caller [7], the address is cached at the starting node of the chain of forwarding pointers. The cached location can then be used not only by subsequent calls from the same cell, but from calls originated from any cell. Caching does not add to the latency of a call operation, since it can be performed off-line.

A similar caching strategy can be deployed along with move operations. In particular, during each move, the new location of the user can be cached at the first node of the chain. However, in contrast to caching at calls, the entry cached at a move is used only if the next operation involving the user is a call operation. Intuitively, if the call-to-mobility ratio (CMR) is very low, continuously updating the cached entry at each and every move operation, (which corresponds to having in effect a chain of one pointer) adds to the overall traffic without necessarily reducing the latency of some call operation. Thus, instead of caching at each move, caching is done selectively.

The condition for caching at a move is based on an estimation of the probability that the next operation will be a call operation. This is approximated by comparing the number of consequent moves and the call-to-mobility ratio. This requires a counter be associated with each mobile unit. The counter is incremented when the unit changes location and is set to zero when the unit receives a call. It also assumes that an estimation of the CMR is available. For this purpose, techniques such as those proposed in [7] can be utilized. Let $number\_of\_moves$ be the value of the counter and $estimated\_cmr$ be the value of the current CMR, then a simple heuristic is to cache when $number\_of\_moves \geq estimated\_cmr$. Again, caching can be performed off-line without adding up to the move set-up time.

2.4 Full Updates

Besides caching, in order for the hierarchical directory scheme to maintain the property of efficiently supporting local operations, the directory entries must be occasionally fully updated. Otherwise, the first node of the chain, i.e., the node at which each call is initially directed by following the tree database entries, may end up being far away from the user's actual location. Fully updating the data structure includes (a) deleting all intermediate forwarding pointers in the chain and (b) updating the internal nodes of the tree (Figure 2 (right)).

Regarding full updates, two issues must be addressed. One issue is the correct execution of calls that proceed concurrently with a full update. We consider this issue in Section 3. The other issue refers to the condition for initiating these procedures. We consider two possibilities. The first is to initiate a full update when the number of forwarding pointers exceeds some maximum value. This is reasonable since the size of the chain directly affects the cost of calls since the cost depends on the number of pointers to be traversed. The other is to fully update when the user moves outside the vicinity of its current region, i.e., when locality deteriorates. We capture this by considering the level of the least common ancestor of the previous and the new location. In particular, an update is initiated when the user moves to a location u such that the level of the least common ancestor of u and the previous location is larger than a threshold level. We experimented with different values for both criteria for updating. The results are presented in Section 4.3.

Finally, we note that before setting a forwarding pointer, detecting cycles is necessary to avoid infinite loops during calls. For example, consider chain $11 \rightarrow 18 \rightarrow 26 \rightarrow 14$ and a move made to 18. Carelessly adding a pointer to 18 results in chain $11 \rightarrow 18 \rightarrow 26 \rightarrow 14 \rightarrow 18$. Cycles can be detected by checking upon each move of a user A from j to i, whether an entry for A already exists at i. If so, there is a path from j to i which can then be purged. Thus, the chain of the example becomes $11 \rightarrow 18$.

Finally, although during normal moves, cache entries never become invalid, moves that form cycles or cause the initiation of full updates make cached entries obsolete. Thus, there is an additional overhead associated with such moves, that of updating the entry cached at the first node of the chain, if any.

For an explicit description of the move and call procedures see [10]. The deployment of these techniques along with the forwarding pointers strategy reduces the total database load and communication traffic over the basic hierarchical schema by a considerable factor (see Section 4).

3 Correctness of Concurrent Operation

In any hierarchical location database, each call results in a number of query operations being issued at various nodes of the hierarchical database. Similarly, a full update initiated by a move causes update operations to be executed at several nodes of the database. The discussion so far was based on the assumption that moves and calls arrive sequentially and are handled atomically and isolated one at a time, and thus, there is no interleaving between the queries and updates of the various call and move operations. In this section, we extend the move and call procedures to handle concurrency.

First, when a move from cell i to cell j occurs, a forwarding pointer pointing to j is set at i to prevent from being lost any calls that have been issued prior to the movement and have been directed to the old address i. Then, *a specific order* is imposed on the database operations caused by a full update. In particular, each full update is performed in two phases: an *add* and a *delete* phase. Let a user's x movement from i to j that causes a full update, and let v_0 be the first node in the forwarding pointers chain of x. If there is no such chain, v_0 is i. First, entries at the path from j to $LCA(j, v_0)$ are added in a bottom-up fashion and then the entries at the path from $LCA(j, v_0)$ to v_0 are deleted in a top-down fashion. In a similar manner, forwarding pointers from v_0 to i, if any, are deleted starting from v_0 and moving towards i. We also distinguish between the two phases of a call. During the *upward phase*, a call moves up the tree to find an entry for the callee. Once the first entry for the location of the callee is found, the call moves *downwards* to this location.

We serialize full updates caused by move operations as follows: a move starts a full update only after the updates caused by the previous full update have been completed. To enforce the serialization of full updates, a full update locks for updates the new address j. A message is sent to unlock j, when the last node is deleted. Only then, can a new full update operation be initiated.

Besides updates, we must also ensure that any call concurrent with a full update will succeed in locating the user. Specifically, we will first show that:

Lemma 1 *During the upward phase, any call placed by a user at cell c and being concurrently executed with a move, say from location i to location j, will be directed either to the old location, i, or the new location j.*

Proof. To distinguish among the possible relative positions of c, i, and j, we use the following two properties of the least common ancestor that hold for any tree nodes x, y, and z: (a) $lca(x, y) = lca(x, z) \Rightarrow lca(y, z) \leq lca(x, z)$, and (b) $lca(x, y) > lca(x, z) \Rightarrow lca(y, z) = lca(x, y)$. Case(i): $lca(j, i) < lca(i, c)$, that is the caller is equidistant from the old, i, and the new, j, locations. Then, during its upward phase, the call encounters the first entry for x at $LCA(j, c)$ ($= LCA(i, c)$). Then, at node $LCA(j, i)$, the call moves towards j, if the add phase has reached the node, and towards i, otherwise. Case (ii): $lca(j, i) > lca(j, c)$, that is the caller is closer to the new location. The call proceeds upwards to node $LCA(j, c)$. If the add phase has reached $LCA(j, c)$, then the call is directed to j. Otherwise, it continues moving upwards till it either reaches a newly added entry or it reaches $LCA(c, i)$, whichever comes first. In the former case, the caller is directed to j and in the later case to i. Case(iii): $lca(j, i) = lca(j, c)$, that is the caller is closer to the old location. If the delete phase has reached node $LCA(i, c)$, the call moves up to $LCA(j, c)$ and then towards j. Otherwise, it moves towards i. □

Now, we must show that the call during its downward phase will succeed in locating the callee. There is a racing issue. To illustrate it, assume the concurrent execution of a full update for user x being in its delete phase and a concurrent call to user x being in its downward phase. Say that at some time, the call queries node m at level k and follows the pointer to the corresponding level $k-1$ node, say node l. Immediately afterwards, the move deletes the entry at level k. Then, traveling faster than the call, the move arrives at node l and deletes the entry for x. When the call arrives at node l, it founds no entry for x.

To treat the problem we associate with each entry for x in each location database at level k a counter, called pending calls counter (PCC_k). The PCC counts the number of calls that queried this entry while moving downwards to locate x or following the chain of forwarding pointers towards x's current location. An additional counter at each node i, called $local_i$, counts the number of calls whose downward phase started at node i, i.e., calls that found the first entry for the callee at node i. Note, that $PCC_{k+1} + local_{k+1} - PCC_k$ is equal to the number of call operations traveling from level $k+1$ to level k.

During the delete phase of a full update operation, an entry at a level k database is deleted only if $PCC_k \geq PCC_{k+1} + local_{k+1}$. Otherwise, the delete operation is delayed till the calls searching for x arrive at the level $k+1$ database.

Lemma 2 *During the downward phase, a call moves correctly to the location found in the upward phase. Thus, it arrives at j either directly or indirectly through i by following pointers.*

Proof. Let a call c be at level k during its downward phase after having found an entry for x. If the entry for x directs the call to the new location j, then the call will succeed in finding an entry at level $k-1$ since in the add phase the entries are included in a bottom-up fashion. If the entry directs the call to the old location i, the call will succeed finding an entry at level $k-1$ since an entry is deleted only when all pending calls from previous levels have been serviced as indicated by the PCC. Similar claims hold for following the forward chain, if any. □

Thus, all calls will eventually succeed in reaching either i or j. In effect, the algorithm in the first case makes a call appear as if it has occurred before the move, that is, it pushes the call backward, or makes it appear as if it has occurred after the move, that is it pushes the call forward. Figure 3 summarizes the extended move and call procedures.

4 Performance Evaluation

We assume a hierarchy of location databases appropriately placed at the nodes of a given communication network. To allow for maximum flexibility in the design of the location management scheme, we consider hierarchies with a variable number of levels. The region covered by each leaf database corresponds to a unique physical address.

4.1 Cost Estimation

We differentiate between the database and the communication cost. We consider as database cost the total number of database operations (queries or updates). For the communication cost, we count the total number of links traversed by the messages involved. Assuming that two nodes u and v communicate with each other by traversing the spanning tree connecting them, let $span(u, v)$ be the number of links

```
Move: Full Update

Full update caused by user x moving from node i to node j

<node i>   set forward pointer to j
<node k>   starting with k := j , lock j for full updates
           while no entry for x
                 add entry for x at node k                 (add phase)
                 k := parent(k)
           l := k  (node where entry was found)
<node l>   down := entry(l, x)
           update entry(l, x)
            v := local_l + PCC_l
<node m>   starting with m := down
           while m is not a leaf
                 o = m
                 m =entry(m, x)
                 if PCC_o < v then WAIT
                      else delete entry for x              (delete phase)
            v := local_o + PCC_o
           head := m  (node where x is in or the first node
                        of forward pointers chain)
<node n>   starting with n := head
           while n different than j
                 o = n
                 n =forward_pointer(n, x)
                 if PCC_o < v then WAIT
                      else delete entry for x
            v := local_o + PCC_o
<node j>   unlock j
```

```
Call

Call from cell i to a user x at cell j

<node k>   starting with k := i
           while no entry for x
                 k := parent(k)                      (going upwards)
           lca := k  (node where entry was found)
           local_k ++

<node l>   starting with l := entry(l, x)
           while x has an entry at node l and l is not a leaf
                 PCC_l ++
                 l := entry(l, x)                    (going downwards)
           head := l  (node where x is in or the first node
                        of forward pointers chain)
<node m>   m := head
           while the entry of x at m is a forward pointer
                 PCC_m ++
                 m := forward_pointer(m, x)          (traversing the forward
           j := m (node where x is in)                        pointers chain)

<node j>   send acknowledge message to i
           (off-line)   update cache at the first node of the chain
```

Figure 3: Move and call procedures extended for handling concurrency.

in the path connecting them in this spanning tree. We assume that $span(u, v) = r\, 2\, lca(u, v)$, that is, it costs r times less, (i.e., there are r less links in the path) for nodes u and v to communicate directly than to communicate through their least common ancestor in the tree. Parameter r allows the modeling of different communication infrastructures, for example when the only way for two nodes to communicate is through the links of the tree, then $r = 1$. For a detailed derivation of the formulas used to calculate the costs the reader in referred to [10].

4.2 Calling and Mobility Model

We assume that, for each user, calls and moves occur independently. The inter-arrival times between two calls follow an exponential distribution, with parameter the mean inter-arrival time between two calls, t_c. The inter-arrival times between two moves follow another exponential distribution, with parameter the mean inter-arrival time between two moves, t_m. The ratio of the number of calls over the number of moves called *Call to Mobility Ratio* (CMR) is then $CMR = \frac{t_m}{t_c}$.

The source of a call event is selected using one of the following distributions:

Arbitrary Calls: A call may be placed from any cell with equal probability $\frac{1}{n}$, where n is the number of different cells. We use a discrete uniform distribution to select one from n cells.

Set of Frequent Callers: Each user receives most of its calls from a specific set of locations. This corresponds to a real-life situation in which a user is frequently called by a set of other users or groups of users, e.g., friends, family, business associates or regular customers. We model a set of frequent callers with a discrete bimodal distribution, which distributes a P_f probability uniformly over a set of specific locations and a $1 - P_f$ probability uniformly over all other locations. So, a call has $\frac{P_f}{n_f}$ probability to be placed by a specific frequent calling location, and $\frac{1-P_f}{n-n_f}$ to be placed by another location, where n_f is the number of frequent calling locations.

The destination of a move event is selected via one of the following distributions:

CMR	Call to Mobility Ratio for the mobile user	Varies
Database Cost	Total number of databbase (query and update) operations	Output
Communication Cost	Weighted sum of tree and forwarding pointer traversals (weighted by distance measured by the level of the lca of the two locations)	Output
nf	Number of frequent caller locations	15
Pf	Percentage of calls from frequent callers (Set of Frequent Callers Model)	85 %
Pn	Probability of moving to an adjacent location (Frequent Moves to Neighbor Cells Model)	95 %
r	Models the relative cost of following a forwarding pointer vs a tree link	1/3
Topology Parameters		
Height		8
Node Outdegree		4
n	Number of leaf nodes	~ 65000

Table 1: Parameters of the Experiments.

Arbitrary Moves: A user may move to any location, except from its current one, with the same probability. We use a uniform distribution as in the case of arbitrary calls, however the probability that the user remains in the same location after a move is 0.

Frequent Moves to Neighbor Cells on a Grid: Since users usually move to nearby locations, we model such a situation in which distant moves are unlikely to happen and short moves to neighbor locations are most likely to happen. We assume that the user moves on a grid of cells and that the user may move with a high probability (P_N) to a neighbor cell on the grid while there is a small probability $(1 - P_N)$ that the user will jump to some other non-neighbor cell. The movement to neighbor cells corresponds to an active user physically moving across cells. On the other hand, the rare movement to non-neighbor cells, corresponds to a user turning off its mobile host and turning it on again in some arbitrary cell. To model this distribution, we use a discrete bimodal probability distribution similar to that used for the set of frequent callers.

CMR	Optimal Height		Optimal Length	
	database cost	comm cost	database cost	comm cost
0.01	8	8	20	20
0.05	8	8	13	12
0.1	8	7	11	8
0.2	8	7	8	5
0.3	8	7	5	4
0.4	8	6	5	3
0.5	8	6	3	2
0.6	8	4	3	2
0.8	8	4	3	2
1.0	8	4	2	1

(a)

CMR	Optimal Height		Optimal Length	
	database cost	comm cost	database cost	comm cost
0.01	8	8	20	20
0.05	8	8	20	10
0.1	8	8	14	6
0.2	8	8	13	6
0.3	8	8	9	4
0.4	8	8	8	3
0.5	8	8	7	3
0.6	8	8	6	2
0.8	8	8	5	2
1.0	8	8	5	2

(b)

Table 2: Optimal values for purging in terms of the overall database and communication cost (a) for nearby-moves and (b) for arbitrary moves.

4.3 Experiments and Results

We have developed an event-driven simulator to evaluate the performance of the location strategies. An event is either a move or a call event. The simulator software has been developed in C++ and runs on a SUN 10 workstation. The purpose of the set of experiments is to:

- determine the optimal condition for initiating full updates,
- demonstrate the benefits of caching,
- provide a comparison of the forwarding schema with the basic schema,
- illustrate optimizations of the forwarding schema for frequent callers.

We run the experiments for a wide range of call to mobility ratios and for a total of about 6000 move and call events per user. We have experimented with hierarchies of different height and width. The results show that the relative performance of the schemes under consideration remains the same. The results presented are for a tree of height 8 and out-degree 4. The experiments were performed multiple times and statistical means were derived for all estimates. Since the relative size of the communication and the database cost depends on many factors such as the load of the network and the size of the databases, we consider the two costs separately. The value of r affects the communication cost of the forwarding scheme which increases with r. For the experiments, we set the communication factor r equal to 1/3. This is a realistic assumption, because in practice, even when traversing a forwarding pointer between two nodes means following the tree path through their LCA, it costs less, since there is no need for establishing connections and querying at each tree node on the path. Table 1 summarizes the parameters for the experiments.

Experiment 1: Condition for Full Updates

The first experiment aims at determining an appropriate condition for initiating full updates. We considered both conditions set in Section 2, i.e., (a) the condition based on the distance of the move, as determined by the level (height) of the least common ancestor of the new and old location of the move operation, and (b) the condition based on the length of the chain of forwarding pointers. In particular, we are looking for an optimal value for the level and length beyond which to fully update so that the overall database and communication cost is minimized. Table 2 (a) shows the optimal such values for nearby moves, and 2 (b) for uniform moves. As expected, this value directly depends on the CMR, since frequent updates decrease the cost of calls while increasing the cost of moves. Also, the optimal values are slightly higher for uniform moves. This is because basic moves cost more in the uniform model of moves than in the near-by model of moves and thus the savings from forwarding are correspondingly higher.

Experiment 2: Benefits of Caching

We have performed a set of experiments to verify the benefits of caching. For this set of experiments, the condition for fully updating was based on the height, and the optimal values were taken from Experiment 1. Caching adds no overhead to the latency of move or call operations, since it can be performed off-line. Figure 4 demonstrates the effectiveness of caching for near-by moves. For uniform moves the gain is smaller but still significant (see [10]). The mean and maximum chain lengths refer to the mean and maximum number of pointers a call has to follow before locating the callee. As shown, caching proves very effective in minimizing both the mean and the maximum values of the chain. In addition, caching reduces the call set-up time without adding to the overall database and communication cost as demonstrated by Figure 5 for near-by moves. The same holds for uniform moves (see [10]).

Experiment 3: Forward vs Basic

We have performed a number of experiments to study the effect of the forwarding scheme in decreasing the overall database load and communication traffic. Depending on the CMR both the overall database traffic and the communication cost are decreased by a factor of 52% to a factor of 5% for near-by moves (Figure 6(left)). For uniform calls, the corresponding values ranges from around 18% to around 7% (see [10]).

The price for this improvement is an increase of the call set up time. This is shown in Figure 6(right) for near-by moves. For uniform moves the increase is almost constant and around 2% for the database cost and 5% for the communication cost [10].

Call latency can be effectively kept short if information about the calling behavior is known before hand. In this case, a simple scheme that caches the location of the user at the callers can reduce the cost. We present results for such a scheme. For the purposes of this experiment, we assumed a pre-specified set of frequent callers, uniformly chosen among all cells. In practice, either the user explicitly specifies this set, or the set is derived by observing the frequency of receiving calls for each mobile user.

We have extended both the basic and the forwarding schemes with a simple caching scheme that replicates the location of the user at the cell of its frequent callers. To locate the user, a frequent caller first uses the location saved at its cache. If this location is outdated, (e.g., the user has in between moved to a new location), a cache miss is signaled. Upon a cache miss, the caller uses the usual procedure to locate the callee. The cache entry is updated upon a cache miss after the user is located using the normal procedure.

Setting forwarding pointers improves the performance of caching since it decreases the cache miss ratio. Figure 7 (left) shows the benefits of forwarding in terms of the overall communication cost, while Figure 7 (right) shows the

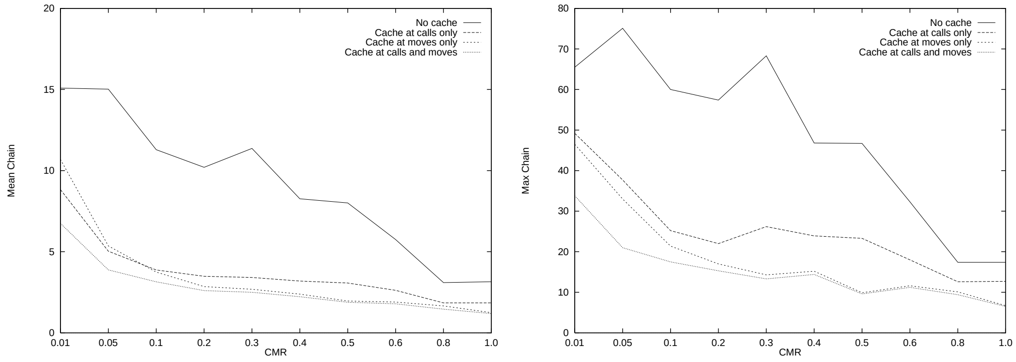

Figure 4: Effectiveness of caching for near-by moves.

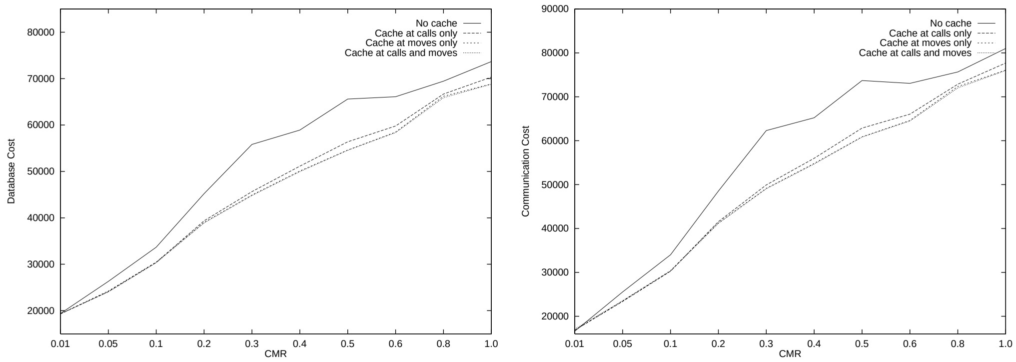

Figure 5: Overall database and communication cost for near-by moves.

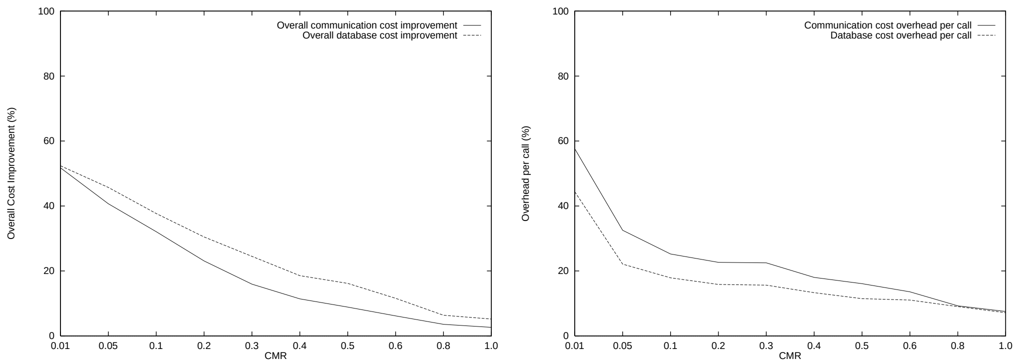

Figure 6: (left) Decrease of the traffic caused by forwarding. (right) Increase of the call set-up time.

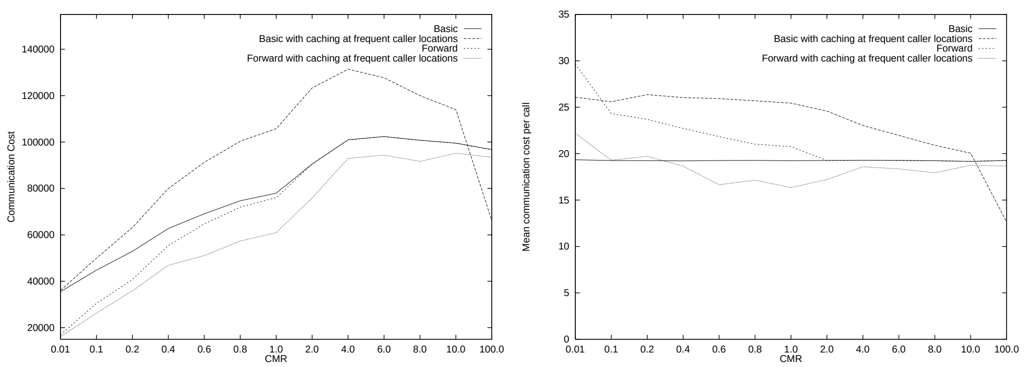

Figure 7: Set of frequent callers: (left) Communication cost. (right) Call set-up time due to communication.

benefit of forwarding in terms of the call set-up time due to communication (similar results are derived for the database cost [10]). Note, that in the case of frequent callers, the forwarding scheme works well even for high values of the CMR.

5 Related Work

Location management has attracted much current research in mobile computing. Various approaches have been proposed for reducing the cost of move and call operations for both hierarchical and non-hierarchical location database architectures [11]. Due to space limitation, please refer to [10] for a more extensive discussion.

Proposals for reducing the cost include caching (see e.g. [5]), replication ((see e.g. [13]) and partitioning (that is, grouping together the cells among which a mobile unit moves frequently, see e.g., [3]). A hierarchical method for location binding in wide-area systems is used in the Globe wide-area location service [14]. Those schemes are orthogonal to the address forwarding strategy and can be used in conduction with it to further reduce database and communication costs.

Address forwarding for a non-hierarchical architecture is discussed in [6] along with an analysis of its benefits for different CMRs. In [2], forwarding is explored in a different framework for a hierarchical, though not tree-structured, location database architecture, called regional matching directory. The treatment is theoretical and based on a worst case order analysis. A non-hierarchical forwarding scheme, SSP chains [12], has also been proposed for transparently migrating object references between processes in distributed computing. The SSP-chain short-cutting technique is somewhat similar to our caching at calls method. In [8], forwarding pointers are used in hierarchical location databases. However, the architectural assumptions are different in that the actual location is saved at each internal database instead of a pointer to the corresponding lower level database. The forwarding pointer is set at different levels in the hierarchy and not necessarily at the lower level database as in our scheme. The emphasis there is on choosing an appropriate level for setting the forwarding pointers and on updating obsolete entries in the hierarchy after a successful call.

6 Summary

Tracking mobile users is central to mobile computing. In this paper, we have studied the use of forwarding pointers in a hierarchical database arrangement. In such schemes, instead of fully updating the hierarchical database upon each move, a forwarding pointer to the new location is set at the previous location of the user. However, the forwarding scheme increases the cost of a call operation, since a number of forwarding pointers may have to be traversed before locating the callee. We have introduced caching techniques that effectively reduce this number as well as conditions for initiating a full update of the database entries. Finally, we have described a synchronization method to control the concurrent execution of call and move operations. Experimental results show that under certain assumptions and for small call to mobility ratio, the proposed forwarding scheme leads in reduction of both the overall database load and the communication traffic by 20% to 60% depending on the call to mobility ratio and the moving behavior.

References

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

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

[3] 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.

[4] C. Eynard, M. Lenti, A. Lombardo, O. Marengo, and S. Palazzo. A Methodology for the Performance Evaluation of Data Query Startegies in Universal Mobile Telecommunication Systems (UMTS). *IEEE Journal on Selected Areas in Communications*, 13(5), June 1995.

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

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

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

[8] 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.

[9] M. Mouly and M. B. Pautet. The GSM System for Mobile Communications. Published by authors, 1992.

[10] E. Pitoura and I. Fudos. Address Forwarding in Hierarchical Location Directories for Mobile PCS Users. Technical Report 6-98, Department of Computer Science, University of Ioannina, 1998. Also electronically available through www, http://www.cs.uoi.gr/~fudos.

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

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

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

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

[15] 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.