A Dynamic Location Tracking Strategy for PCS Using Hierarchical Location Databases

Chi-Chih Lee   Hwa-Chuan Lin   SingLing Lee

Department of Computer Science and Information Engineering

National Chung Cheng University, Chiayi 62107,

Taiwan, Republic of China

Abstract

In this paper, we propose a dynamic location tracking strategy for the hierarchical location database architecture in personal communication service systems (PCS). In our strategy, a user's current location is stored at a specific location database, called the local anchor (LA). We focus on studying the problem how to decide the best position for the LA to get a great benefit in location registration. The LA periodically collects the user's mobility and calling patterns and dynamically changes its position according to his/her behavior after a period of time. When a user has a small call-to-mobility ratio (CMR), our strategy efficiently reduces registration costs. Besides, we use caching bypass pointers for call delivery procedures but avoid cache miss which happens in the caching strategy. Thus our strategy has the same performance as the caching strategy in reducing call delivery costs when the CMR is large, but performs much better than the caching strategy when the CMR is small. Our simulation result shows that our strategy produces 30% performance improvement than the basic strategy and 20% performance improvement than the caching strategy.

1 Introduction

1.1 Background

Personal Communication Service (PCS) system provides people to communicate with each other within the network coverage. Location management is a key component in the operation of PCS [2, 14]. It is an important problem to design an efficient location management strategy to reduce database access and signaling traffic for the system. Location management contains two main operations — *location registration* (or called Move) and *call delivery* (or called Find). There are two commonly standards in the current two-level location database architecture: IS-41 [1, 12] and GSM [12, 13]. Two types of databases are used to record the location information of the mobile terminal (MT), they are the Home Location Register (HLR) and the Visitor Location Register (VLR), respectively. The HLR is a centralized database containing user profiles of subscribers. These user profiles record information such as the type of services subscribed, the billing information, the quality-of-service (QoS) requirements, and the current locations of the MTs. There are several VLRs distributed throughout the PCS network, and each one stores replication of the user profiles of the registered subscribers and the information of the MTs currently residing in its associated area. Figure 1 shows the basic architecture of the current PCS network in the two-level archtecture. The PCS service area is divided into cells. All MTs within a cell communicate with a base station through wireless links. Several cells are grouped together to form a contiguous geographical region called registration area (RA). All base stations within a giving RA are conected to a mobile switch center (MSC). The MSC provides switching function for the MTs in their associated area. Each VLR may associate with one or several MSCs. In this paper we assume each VLR associates with one MSC. The MSCs, VLRs, and the HLR are interconnected through the public switch telephone network (PSTN).

Location registration is the procedure that an MT explicitly reports its new location to the network while the MT crosses RA boundies. When an MT enters a new RA, a location registration request is sent to its serving MSC, which relays the location registration request to the MT's HLR. The HLR updates the MT's profile to record the location of the VLR associated with its serving MSC and sends the service profile to the VLR. Furthermore, the

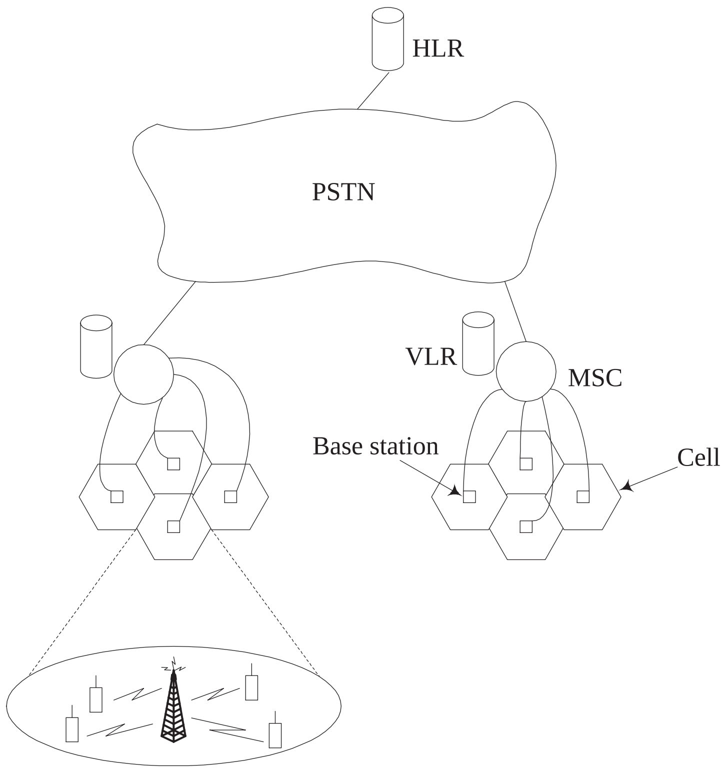


Figure 1: PCS network architecture

HLR sends a deregistration message to the VLR associated with the old RA. The old VLR removes the MT's profile. When a call arrives, the call delivery procedure is used to locate the callee's current location. The caller's MSC sends a query message to the callee's HLR to retrieve the location of his/her VLR. Then the callee's HLR sends a query message to his/her VLR. The VLR relays this request to the serving MSC of the callee. If the callee can receive a call, a Temporary Local Directory Number (TLDN) is associated to the callee and the TLDN is sent to the callee's HLR, which forwards the TLDN to the caller's MSC. The caller then setups a connection.

As the PCS subscribers keep increasing, the requirements of database access and signaling transimission also increase dramatically [2, 14, 17, 18]. The traffic load of the HLR will become system bottleneck. To accommodate the increased traffic for the future PCS systems, it is necessary to use a scalable architecture. And the hierarchical distributed location architecture is a possible solution among different approaches. In this architecture, location databases are interconnected by a Common Channel Signaling (CSS) network. The hierarchical location database architecture greatly reduces network signaling traffic while most calls received by users and most RA crossings are geographically localized [2, 3, 5, 10, 11, 18]. However, a major drawback of the hierarchical location architecture is that the number of database access is more than that of the two-level architecture. A number of auxiliary strategies has been proposed to improve this drawback.

1.2 Previous work

To reduce location registration and call delivery costs further, several auxiliary strategies have been proposed. They are the *user-profile replication*, *pointer-forwarding*, *local anchor*, and *caching* strategies, which are described as follows.

In the user-profile replication strategy [16], a user's profile is replicated at one or several selected locations in order to reduce the call delivery cost. On the other hand, the database update cost will be increased potentially since replicas must be maintained consistently whenever the MT moves. The number of replicas of an MT may be limited by system resource, such as the signaling network and the database capacity. Besides, the replication decision is made by a centralized system, which must collect mobility and calling patterns of all users from time to time. This strategy performs well for those users who receive calls frequently relative to the rate at which they cross RAs, i.e., users with large call-to-mobility ratio (CMR).

In the pointer-forwarding strategy [7, 9, 15], a pointer is created at the old VLR that points to the location of the new VLR instead of updating the HLR when a user moves across his/her RA boundary. When an incoming call arrives, the callee's HLR is queried first to find the first VLR which he/she was registered at, and then the signal follows a chain of forwarding pointers to the his/her current serving VLR. This strategy saves the registration cost while users change RAs frequently. However, the cost of the call delivery procedure will increase due to trace the chain of forwarding pointers whose length is too long. The pointer-forwarding strategy performs good while users with small CMR.

In the local anchor strategy [4], the MT reports its location change to a specific VLR called the local anchor (LA) which points to the location of the MT's current RA. The HLR keeps a pointer to the local anchor. This approach reduces the registration cost. While an incoming call arrives, the caller first queries the callee's HLR to get the location of the local anchor, and then the callee's HLR queries the local anchor to retrieve the location of his/her current serving VLR. This strategy performs well for users with small CMR.

In the caching strategy [8, 10, 11], the callee's location information is stored at the caller's database. The main idea is to reuse the information about the user's location from the previous call to that user. This information consists of *bypass pointers* kept at a database along the path that allows the signaling messages to bypass portions of the path. A pair of bypass pointer will be created between the caller and the called party. They are the *forward bypass pointer* and the *reverse bypass pointer*. The forward bypass pointer is created at the database s which is the ancestor of the caller's RA to reduce the setup time of the call delivery procedure. And the reverse bypass pointer used by the acknowledgement message is created at the database t, the ancestor of the callee's RA. When the CMR is large, clearly, those saving information will be useful and efficient for locating users. When the CMR is small, the caching strategy must pay an additional cost to maintain the validness of the bypass pointers [6].

1.3 Motivation of our work

In this paper, we propose the *dynamic location tracking* strategy to reduce database access and signaling traffic at the same time. This strategy combines the characteristic of the local anchor strategy in registration and that of the caching strategy in call delivery. While a user has a large CMR, the cost of the call delivery will be reduced by using the information of bypass pointers. Besides, the registration cost also can be greatly reduced even when a user has a small CMR. Each user selects a database as the local anchor (LA) to record his/her location information. A user performs the registration procedure only at the current serving VLR and his/her local anchor instead of updating several databases. Besides, each forward bypass pointer records the callee's LA for call delivery. As long as we find the callee's LA, then we can find his/her actual location through querying his/her LA. Hence, our strategy can prevent cache miss when a user moves out of the subtree rooted at the database t. Furthermore, we focus on deciding an appropriate position for each LA in the tree-structured location architecture. Each LA periodically collects a user's mobility and calling patterns, and its position can be dynamically adjusted according to the computing result after a certain time period. When the LA has been adjusted, it implies that several forward bypass pointers which cache this LA need to be updated. If these bypass pointers are updated at once, the update cost will increase dramatically. In order to overcome this problem, we keep a pointer from the old LA to the new LA instead of updating bypass pointers at once. When a call delivery procedure is invoked, the bypass pointer will be updated through call delivery message. First, the reverse bypass pointer will be created at the new LA, then the forward bypass pointer will be updated through the acknowledgement message. In other words, the forward bypass pointer will be updated through the acknowledgement message only when it is used by a user to perform a call delivery procedure. Thus we can avoid that several forward bypass pointers are updated at the same time. Finally, the experiment result shows that our strategy gains a great benefit in reducing database access and signaling traffic by dynamically adjusting bypass pointers with LAs.

The rest of this paper is organized as follows. In Section 2, we describes the dynamic location tracking strategy and the appropriate position deciding scheme. Section 3 briefly discuss analysis of different strategies. Simulation result shows in Section 4. Finally, Section 5 concludes the thesis.

2 Dynamic Location Tracking Strategy

Our strategy is able to make database access and signaling traffic reduced in the hierarchical location database architecture for PCS mobility management. Our objects are as follows :

- Reduce the cost of location registration operation.
- Reduce the cost of call delivery.
- Dynamically adjust the position of the LA according to a user's behavior to make bypass pointers used more efficiently.

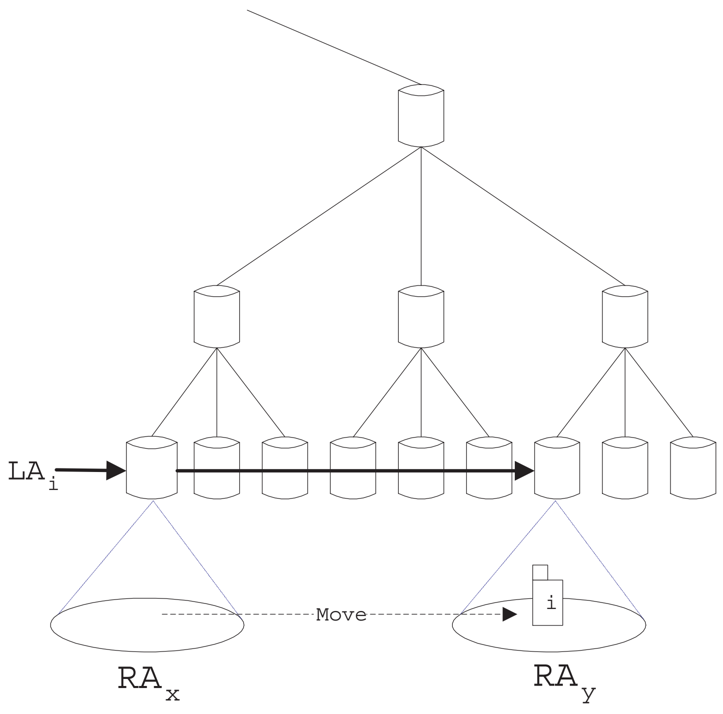


Figure 2: An example of the local anchor strategy

Our strategy makes use of the local anchor to track users' location information, and the cost of updating databases for location registration can be reduced. Whenever an MT moves to a new RA, it sends a registration message to the new RA and its current serving LA. Furthermore, an anchoring pointer is created at the MT's LA. This approach only needs to have two database registrations, hence the registration cost can be reduced efficiently. Figure 2 shows an example of the local anchor strategy. While the MT i moves from the RA_x to the RA_y, the LA_i keeps an anchoring pointer which points to the location RA_y in its database. Generally speaking, the local anchor can be differentiated into *static* and *dynamic* ones. For the static LA, the LA doesn't change its position until an incoming call arrives. For the dynamic LA, the LA changes its location dynamically. However, the dynamic LA performs much better than the static LA according to the previous research[4]. The reason why the static LA does not have good performance is that a user may have to register to his/her LA located at a far region. Thus the registration cost will increase potentially. In this paper, we will focus on studying the strategy how to dynamically adjust the position of the LA.

Furthermore, this strategy is also similar to the caching strategy which reuses the user's location information from the previous call to that user. For users who receive calls frequently, the cost of a call delivery procedure can be saved efficiently. However, the performance of the caching strategy depends upon the probability whether the cached information is valid or not. It should be noted that the cached information becomes invalid when users move out of the subtree rooted at the database t. The database t only stores location information of its child nodes. When a call request arrives the database t, the query message is propagated downward to find the called user. If the called user is not found under the database t, the cache miss happens. In order to avoid the cache miss, we view the location of the database t as the local anchor of called users, hence the called user can be found certainly at the database t. Besides, it is important to decide an appropriate position for the database t, which can make bypass pointers used efficiently. We adjust the position of the database t to the best position t' according to the user's mobility and calling patterns.

The dynamic location tracking strategy for PCS includes two main operations, called *Intelligent local anchor (ILA)Move* and *ILA Find*.

2.1 Intelligent LA Move scheme

When a user moves to a new RA, the *ILAMove* procedure is initialized. The user's LA will record the location of the RA where the user resides. We assume that there is a small memory in the mobile terminal that records which location database is the user's current serving LA. Whenever the user moves across RA boundaries, the registration message is propagated to this LA. Before the LA changes its position from t to t', the user still registers to the LA at t. We denote the old RA of MT i as RA_{old} and the new RA of MT i as RA_{new}. Let $LA_i^{current}$ be the current LA of MT i and LA_i^{new} is the new LA of MT i. Let $LA_i^{adjusted}$ be the adjusted LA. We describe the *ILAMove* as follows:

Case 1: The LA serves a new user who has never performed the dynamic adjusting LA procedure, it collects his/her mobility patterns to decide the best position of the LA.

Case 1.1: The MT i moves from RA_{old} to RA_{new}. It does not move out of the subtree rooted at its serving LA, $LA_i^{current}$.

1. The MT i deregisters at RA_{old} and registers at RA_{new}.
2. The registration message is propagated to $LA_i^{current}$, and $LA_i^{current}$ stores the move record in its event table for the MT i.
3. The acknowledgement message returns from $LA_i^{current}$ to RA_{new}.

Case 1.2: The MT i moves from RA_{old} to RA_{new}. It moves out of the subtree rooted at is serving LA, $LA_i^{current}$ (Figure 3).

1. The MT i deregisters at RA_{old} and registers at RA_{new}.
2. The registration message is propagated to the least common ancestor (LCA) of these two RAs, and the LCA becomes the MT's new LA, LA_i^{new}. LA_i^{new} stores the move record for the MT i.
3. The LA movement procedure is invoked. First, records of $LA_i^{current}$ will be copied to LA_i^{new}. Secord, the system will check whether $LA_i^{current}$ exits reverse bypass pointers for the MT i. If it does exist, a LA movement pointer that points to the location of LA_i^{new} will be crated at $LA_i^{current}$.
4. The acknowledgement message returns from the LA_i^{new} to RA_{new}.

Case 2: The LA at t, $LA_i^{current}$, has performed the dynamic adjusting LA procedure to change its position to t'. Whenever the MT i crosses RA boundaries, it keeps registering to the adjusted LA at t', $LA_i^{adjusted}$ (Figure 3).

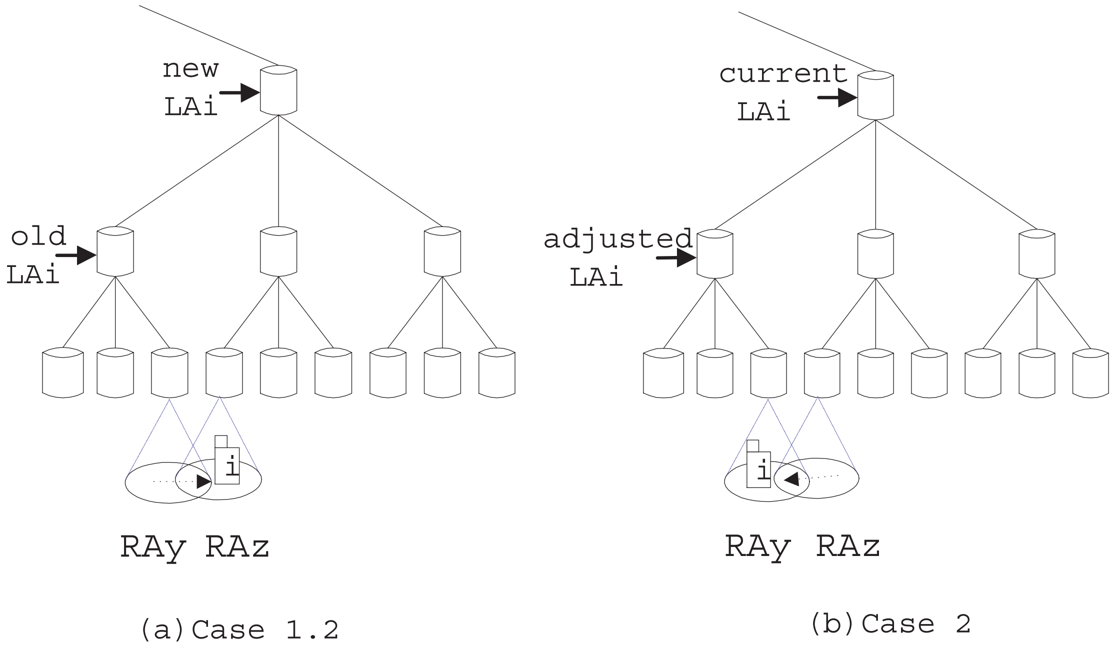


Figure 3: The procedure of ILAMove

1. The deregistration and registration messages are sent to RA_{old} and RA_{new} respectively.
2. The registration message is propagated to $LA_i^{adjusted}$ of the MT i, and $LA_i^{adjusted}$ stores the move record for the MT i.
3. The acknowledgement message returns from the $LA_i^{adjusted}$ to the RA_{new}, and the registration procedure is complete.

Whenever we want to adjust the database t to t', we may encounter a problem that several forward bypass pointers need to be updated at the same time. The database update overhead will be increased potentially. Therefore, we setup a LA pointer which points to the new LA instead of updating these forward reverse bypass pointers at once.

2.2 Intelligent LA Find scheme

The *ILAFind* procedure is invoked when a calling MT sends a call request. The *ILAFind* can be divided into four cases, each of them is described as follows. We denote the RA_{caller} as the caller's RA, the RA_{callee} as the callee's RA, and the LA_{callee} as the callee's LA.

Case 1: The caller and the callee are at the same RA.

1. The call request is sent to the RA_{callee}, and it is queried directly to find the callee.
2. In this case, no bypass pointers will be created.

Case 2: No bypass pointer exists between the caller and the callee, and the RA_{caller} is under the coverage of the LA_{callee}.

1. The call request is sent to the LA_{callee}, and then the call request is forwarded to the RA_{callee} through the LA_{callee}.
2. The acknowledgement message returns from the RA_{callee} to the RA_{caller} through the LA_{callee}.
3. No Bypass pointers will be created for the callee in this case.

Case 3: No bypass pointer exists between the caller and the callee, and the caller does not reside under the coverage of the LA_{callee}.

1. The call request is sent to the LA_{callee}, and then the call request is forwarded to the RA_{callee} through the LA_{callee}. Besides, the LA_{callee} stores the call record for the callee.
2. The acknowledgement message returns from the RA_{callee} to the RA_{caller} through the LA_{callee}.
3. A pair of bypass pointers is created between the RA_{caller} and the LA_{callee}.

Case 4: A pair of bypass pointers exists between the caller and the callee, and the caller does not reside under the coverage of the LA_{callee}.

Case 4.1: The position of the LA_{callee} has not been moved.

1. The forward bypass pointer is found at the RA_{caller}, then the call request is send to the LA_{callee} through this forward bypass pointer.
2. The LA_{callee} forwards the call to the RA_{callee}, and it stores the call record for the callee. If the LA_{callee} is just the RA_{callee}, then the LA_{callee} is queried directly to find the callee.

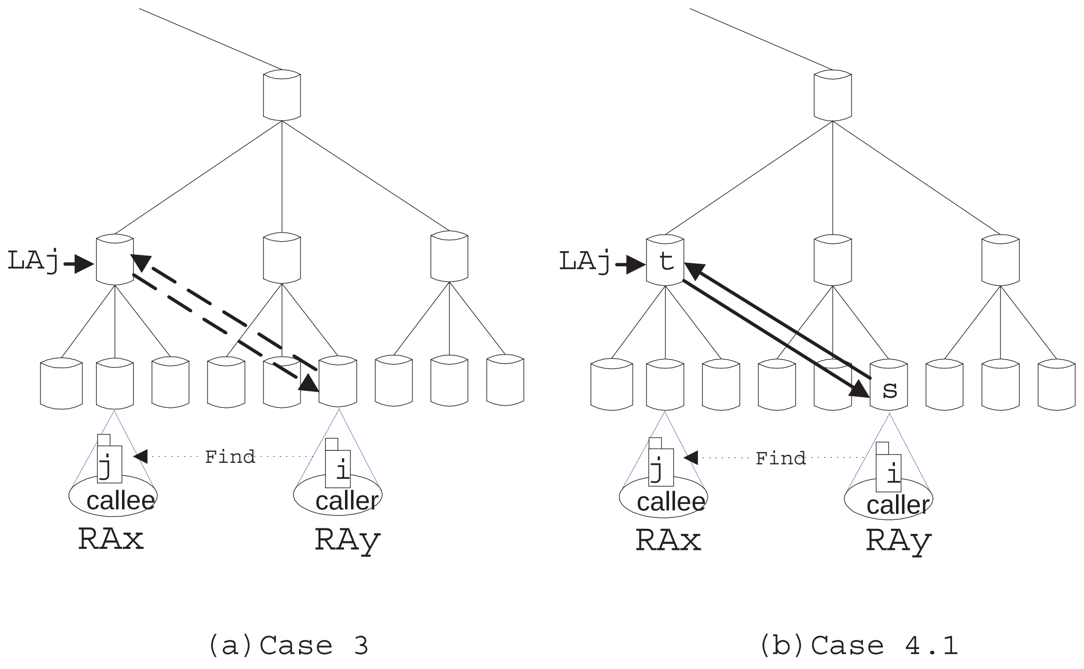


Figure 4: The procedure of ILAFind

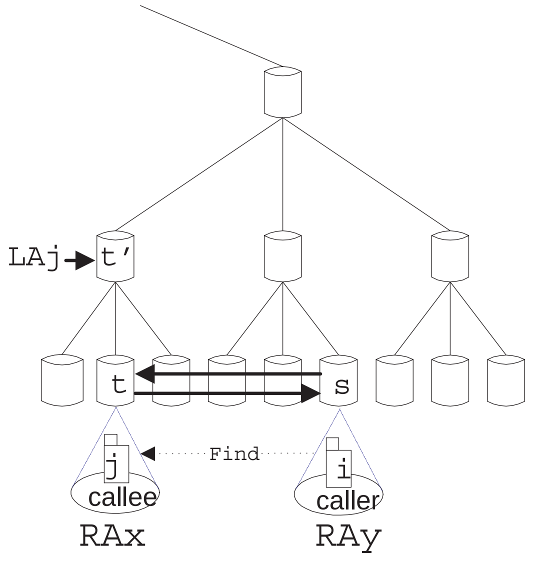
3. The acknowledgement message returns from the RA_{callee} to the RA_{caller} through the reverse bypass pointer which is stored at the LA_{callee}.

Case 4.2: The position of the LA_{callee} has been moved from t to t', and t is the leaf node.

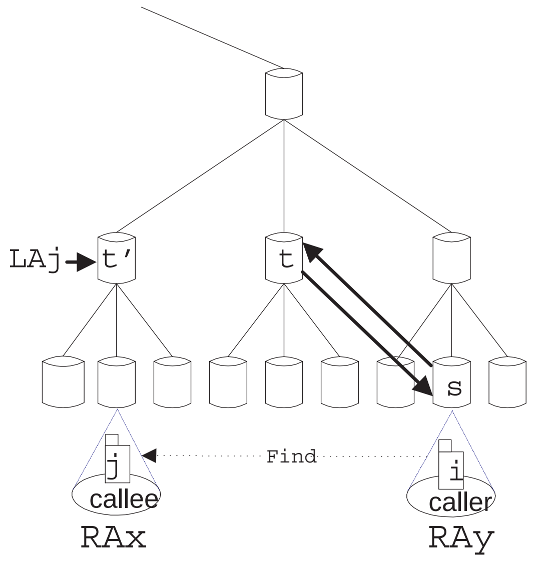
1. The call request is send from the RA_{caller} to the old LA_{callee} t through this forward bypass pointer.
2. The old LA_{callee} is queried to find the callee. It also gets the location of the adjusted LA_{calee} t', and the reverse bypass pointer stored at t will be moved to t'.
3. The acknowledgement message returns from the RA_{callee} to the RA_{caller} through the reverse bypass pointer stored at t'. And the call record is stored at t'. During the return path, the forward bypass pointer will be updated to cache t' at the RA_{caller}.

Case 4.3: The position of the LA_{callee} has been moved from t to t', and t is not the leaf node.

1. The call request is send from the RA_{caller} to the old LA_{callee} t through the forward bypass pointer.
2. The old LA_{callee} is queried to get the location of the adjusted LA_{calee} t',



(c)Case 4.2



(d)Case 4.3

Figure 5: The procedure of ILAFind

and the reverse bypass pointer which is stored at t will be moved to t'.

3. When the call arrives t', LA_{callee} stores the call record and forwards the call to the RA_{callee}.
4. The acknowledgement message returns from the RA_{callee} to the RA_{caller} through the reverse bypass pointer stored at t'. During the return path, the forward bypass pointer will be updated to cache t' at the RA_{caller}.

2.3 The Appropriate Position Deciding Scheme

Each location database that serves as the LA has to record its serving user's calling and mobility patterns. Its primary function is to collect these information, and to compute the most appropriate location for the LA. By adjusting the position of the LA, it not only greatly reduces the registration cost, but also reduces the call delivery cost efficiently. We use a user event table to store a user's mobility and calling patterns. They consists of registration events and call events. Whenever a user crosses RA boundaries or a caller makes a call through bypass pointers, these events will be saved in this event table. Table 1 shows an example of a user's event table. In order to explain the appropriate position deciding scheme, we define some parameters as follows:

R_{event}:	The amount of the registration events
C_{event}:	The amount of the call events
CLA_i:	The set of MT_i's candidate LAs
$cost({CLA_i}^j)$:	The cost of the j-th candidate LA of MT_i
RA_i:	The set of RA where MT_i resides
${RA_i}^k$:	The k-th RA where MT_i resides
$d(s, {CLA_i}^j)$:	The distance from s to the candidate ${LA_i}^j$
ρ_s:	The registration rate of MT_i in s, and $s \in RA_i$
γ_s:	The call rate of MT_i in s, and s $\in RA_i$

$$cost({CLA_i}^j) = \tfrac{R_{event}}{R_{event}+C_{event}} \cdot 2 \sum_{s \in RA_i} \rho_s \cdot d(s, {CLA_i}^j)$$

$$+\tfrac{C_{event}}{R_{event}+C_{event}} \cdot 2 \sum_{s \in RA_i} \gamma_s \cdot [d(s, {CLA_i}^j) + d(s', {CLA_i}^j)]$$

$$\text{where } s' = LCA(RA_i, {CLA_i}^j)$$

Figure 6: The appropriate position analysis function

The best LA position deciding algorithm

Input: A set of RA where MT_i resides.

Output: The most appropriate position for MT_i's LA.

Step 1: Select the location databases as the candidate LAs for evaluating which one is better. The candidate LAs contain the databases which locate between RA_i and the LCA of RA_i.

Step 2: Gather statistics of the registration rate and the call rate for each RA of RA_i.

Step 3: For each candidate LA, compute its link costs of registration and call, respectively.

Step 4: Execute the appropriate position analysis function. If all candidate LAs finish computing, then go to Step 5, else go to Step 3.

Step 5: Return the position of the candidate LA with the minimum cost, t' , to be MT_i's new serving LA. Adjust MT_i's LA to this new LA t'.

We use the best LA position deciding algorithm to compute the appropriate position t' for a user's LA according to his/her mobility and calling patterns. The registration rate and the call rate of each RA are known through a user's event table. They will be useful to decide the best position for a user's LA. First, we select several location databases as the candidate LAs. These candidate LAs contain the databases which locate between the RAs where the user resides and the LCA of these RAs. After these candidate LAs have been choosed, we compute the cost of each candidate LA. The cost can be divided into registration and call costs. The cost of the candidate LA is computed by the appropriate position analysis function (Figure 6). When the costs of all candidate LAs have been computed, the most appropriate LA is with the minimum cost. Thus we will adjust the user's LA to the best position.

RA id	Reg. events	Call events
RA_{555}	1	2
RA_{556}	2	1
RA_{557}	2	3
RA_{558}	4	2
RA_{559}	11	19
RA_{560}	12	9
RA_{560}	4	0

Table 1: An example of the user event table

RA id	Reg. rate	Call rate
RA_{555}	0.0278	0.0556
RA_{556}	0.0556	0.0278
RA_{557}	0.0556	0.0833
RA_{558}	0.1111	0.0556
RA_{559}	0.3056	0.5278
RA_{560}	0.3333	0.25
RA_{561}	0.1111	0

Table 2: The registration rate and the call rate of the event table

We give an example to explain our appropriate position deciding scheme. Table 1 shows a user's event table of his/her current LA_{61}, and it records call and registration events of each RA respectively under this subtree (Figure 7). We can know the registration rate and the call rate of the event table (Table 2). Each database which locates between these registration areas where the user resides and the LCA of these registration areas will be computed for deciding the most appropriate LA for the user.

In this example, $DB_{61}, DB_{184}, DB_{185}, DB_{185}$, and $DB_{555} \sim DB_{561}$ will be computed respectively, and we have to compute registration and call costs for each DB. For example, DB_{555} is computed as follows:

DB_{555} :

Reg. cost $= 2 * 4 * 0.9722 = 7.778$

Call cost $= 2 * 2 * 0.0556 + 2 * 6 * 0.9444 = 11.556$

Total cost $= 7.778 + 11.556 = 19.333$

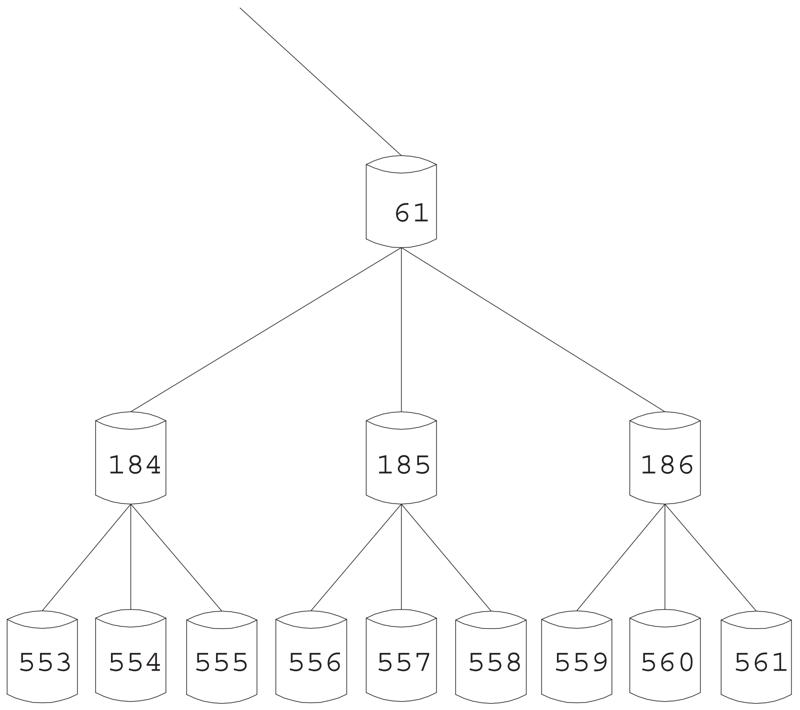


Figure 7: An example of the position deciding scheme

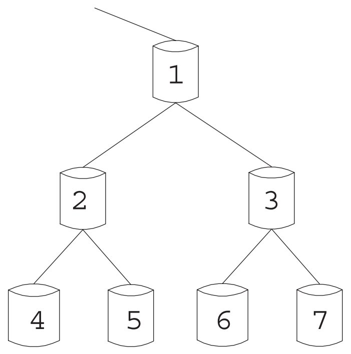
$$\text{Weighted cost} = 0.5 * 7.778 + 0.5 * 11.556 = 9.667$$

The *total cost* means that we sum up the registration and call costs directly, and the *weighted cost* means that we assign different weight values to registration and call costs according to the user's registration and call events. In this example, we can choose DB_{186} as the most appropriate LA for the user according to the computing result(Table 3).

Database id	Reg. cost	Call cost	Total cost	Weighted cost
DB_{61}	4	4	8	4
DB_{184}	5.889	7.778	13.667	6.883
DB_{185}	5.111	7.333	12.444	6.222
DB_{186}	3	4.889	7.889	3.944
DB_{555}	7.778	11.556	19.333	9.667
DB_{556}	6.889	11.222	18.111	9.056
DB_{557}	6.889	11	17.889	8.944
DB_{558}	6.667	11.111	17.778	8.889
DB_{559}	3.778	6.778	10.556	5.278
DB_{560}	3.667	7.889	11.556	5.778
DB_{561}	4.556	8.889	13.444	6.722

Table 3: The result of computing the best position

Generally speaking, we can decide the most appropriate LA according to the total cost of each DB (Table 3). However, we should consider the relation between registration and call events. In some cases, the most appropriate LA will not be decided clearly if it is only decided according to the total cost. Figure 8 is a special example, and we are not sure which one is the most appropriate LA. We describe this situation as follows.



RA id	Reg. events	Call events
RA_5	60	9
RA_6	40	1

Figure 8: An example of the position decideing scheme

In this case, we cannot decide the best position of the LA according to the total cost. We are not sure whether DB_5 is better than DB_1 and DB_2, because their total costs are the same (see Table 4). However, we can observe that the number of registration events is

Database id	Reg. cost	Call cost	Total cost	Weighted cost
DB_1	4	4	8	4
DB_2	3.6	4.4	8	3.673
DB_3	4.4	7.6	12	4.691
DB_5	3.2	4.8	8	3.345
DB_6	4.8	11.2	16	5.382

Table 4: The result of computing the best position

more than the number of call events. Hence, the weight value of registration cost should be more expensive than that of call cost. Because the registration cost of DB_5 is smaller than those of DB_1 and DB_2, we can still easily decide that DB_5 is the most appropriate LA than the others. In other words, it can greatly reduce registration cost to select DB_5 as the LA; because DB_5 is close to the region where the user frequently registers. This result leads to the conclusion that we should decide the most suitable position for the LA by the weighted cost.

So far, we have seen how to decide the most appropriate position of the LA for a user through our computing scheme. The LA periodically collects the user's mobility and calling patterns to decide whether the LA should be adjusted its position. We execute the appropriate position deciding scheme off-line, and it won't increase the system load dramatically. We assume each location database has the computing function for adjusting LA dynamically. Besides, the LA does not necessary proceed computing everyday, it depends on the user's behavior. If the user's behavior changes frequently, the system trends to collect and to compute everyday. If the user rarely changes his/her behavior, then the system will collect this user's information of a week and compute the most suitable LA for him/her. Therefore, the user's LA dynamically changes according to his/her behavior.

3 Cost Analysis

In this section, we derive the cost function of the *basic*, *eager caching*, and *dynamic location tracking* strategies to analyze the registration and call delivery costs. We assume that s and t are location databases with forward and reverse bypass pointers for *callee* j. Let RA_x and RA_y be the RAs where *caller* i and *callee* j reside respectively, and RA_z is the RA which *callee* j moves to. We assume the level of the leaf node is zero. Suppose b_1 be the levels from s to $LCA(RA_x, RA_y)$ and b_2 be the levels from t to $LCA(RA_x, RA_y)$. And q is the levels from RA_y to RA_z. The levels from RA_x (or RA_y) to $LCA(RA_x, RA_y)$ is r. The user's LA will be dynamically adjusted its position from t to t' according to our appropriate position deciding scheme. Assume that the levels from t' to $LCA(RA_x, RA_y)$ is ℓ.

The total cost of a location management strategy depends on Move cost and Find cost. Our cost analysis considers communication and database access as the basic measurement of costs. In order to estimate the total cost clearly, we define the cost parameters as follows:

Table 5: Define the cost parameters

n_1	Average number of move operations.
n_2	Average number of find operations.
$T(t)$	The coverage area of *database* t.
γ	Average probability of moving out the subtree rooted at *database* t between two consecutive calls.
$\alpha(i)$	The probability of i moves between two consecutive calls.
$\beta(t, i)$	The probability of the user moves out of *database* t given there are i moves between two calls.
R	The cost of reading a database.
U	The cost of updating a database.
C	The cost of traversing a signal link.

3.1 Analytical Model

We select m-way search tree as the architecture of hierarchical location database for our analytical model [11]. First, we discuss the probability of the user moving out of the subtree rooted at *database t* between two consecutive calls. Between two calls arrive, we assume the user moves i times. Therefore, we can describe the probability as follows:

$$\begin{aligned}\gamma &= \sum_i \text{P[a user moves out } t \text{ and there are } i \text{ moves between two calls]}\\ &= \sum_i \gamma(t,i) = \sum_i \alpha(i)\beta(t,i)\end{aligned}$$

$\alpha(i)$ denotes the probability of a user who moves i times between two consecutive calls. The probability $\beta(t,i)$ denotes a user moving out of the subtree rooted at *database t*. Let h be the level of *database t*, and the RAs under *database t* are numbered from 1 to m^h. Hence,

$$\beta(t,i) = \sum_{j=1}^{m^h} \text{P[the user starts in } j \text{]}\beta'(t,i,j)$$

$\beta'(t,i,j)$ denotes the probability of the user moving out of *database t* in i steps given that he/she starts in *RA j*. Assume that the probability of the user's moving to the right is u, and the probability of the user's moving to the left is v. Thus we can get the following equations.

$$\beta'(t,i,j) = \beta'(t,i-1,j+1)u + \beta'(t,i-1,j-1)v$$
$$\text{for} \quad 2 \le j \le m^h - 1 \ , \ 1 < i$$

$$\beta'(t,i,1) = \beta'(t,i-1,2)u \qquad \text{, for} \quad 1 < i$$

$$\beta'(t,i,m^h) = \beta'(t,i-1,m^h-1)v \qquad \text{,for} \quad 1 < i$$

The boundary conditions are described as follows:

$$\beta'(t,1,1) = v \ , \qquad \beta'(t,1,m^h) = u$$
$$\beta'(t,1,j) = 0 \ , \text{ for} \qquad 2 \le j \le m^h - 1$$

Finally, we get the value, γ, and we use it to evaluate the costs of different strategies later.

3.2 Basic Strategy

First, we analyze the costs of the *basic* strategy. As we know, there are two main operations in the *basic* strategy: *BasicMove* and *BasicFind*. They are involved in the registration and call delivery procedures respectively.

- The cost of *BasicMove*

  When an MT moves form RA_y to RA_z, the registration and deregistration messages propagate from RA_y to RA_z through $LCA(RA_y, RA_z)$. The databases under $LCA(RA_y,RA_z)$ are queried and updated during the registration and deregistration operations. The communication signal traverses from RA_y to RA_z through $LCA(RA_y, RA_z)$, and it returns from RA_z to RA_y. Assume that MC_{basic} is the cost of a $BasicMOVE$ operation, then

$$MC_{basic} = 2(2q+1)R + (2q+1)U + 4qC$$

- The cost of $BasicFind$

  When *caller i* at RA_x calls *callee j* at RA_y, the query message is propagated from RA_x to RA_y through $LCA(RA_x, RA_y)$. The acknowledgement message returns form RA_y to RA_x. Let FC_{basic} be the cost of a $BasicFind$ operation, then

$$FC_{basic} = 2(2r+1)R + 4rC$$

  Thus the total cost for *callee j* in the *basic* strategy is defined as follows:

$$TC_{basic} = n_1 \cdot MC_{basic} + n_2 \cdot FC_{basic}$$

3.3 Eager Caching Strategy

The two main operations in the *eager caching* strategy are called *EagerMove* and *CacheFind*. They are involved in the registration and call delivery procedures respectively.

- The cost of *EagerMove*

  EagerMove is similar to *BasicMove*, but it pays an additional cost to delete invalid bypass pointers. We assume that there is a pair of bypass pointers constructed at the ancestors of RA_x and RA_y for *callee j*. When an MT moves from RA_y to RA_z, it leaves the area of the coverage of *database t*. The forward bypass pointer of *database s* becomes invalid, and it will be deleted by *EagerMove*. In order to delete invalid bypass pointers, *EagerMove* deletes records in *database s* and *t*. The communication signal for deleting invalid bypass pointers propagates from RA_y to RA_x through $LCA(RA_x, RA_y)$. Thus the additional cost for deleting a pair of bypass pointers for *callee j* is $2U + 2(b_1 + b_2)C$. We should notice that the more forward bypass pointers cache the location of *database t*, the more costs need to pay to delete invalid bypass pointers when a callee leaves *database t*. Assume that the maximum height of the tree is H. Let $\overline{M}$ be the average cost of a *BasicMove* operation. $M'(t, i)$ denotes the cost of i *EagerMove* operations between two consecutive calls given a bypass pointer directed at *database t*. MC_{cache} denotes the average cost of a *EagerMove* operation, hence we can obtain it as follows:

$$\overline{M} = \sum_{q=1}^{H} M(q)\tfrac{m^{H-q}}{m^{H-1}}$$

$$M'(t, i) = [(\overline{M} + 2U + 2(b_1 + b_2)C)\gamma(t, i)]i + [\overline{M}(1 - \gamma(t, i))]i$$

$$MC_{cache} = \sum_{i=0}^{\infty} M'(t, i)\alpha(i)$$

- The cost of *CacheFind*

  The difference between *CacheFind* and *BasicFind* is that *CacheFind* locates a callee by using bypass pointers. If no bypass pointers exist between the caller and the called party. A pair of bypass pointers will be constructed when the first time the caller calls the callee. To setup bypass pointers, *CacheFind* updates records in *database s* and t. The cost for constructing a pair of bypass pointer is $2U$. If there is a pair of bypass pointers between *caller i* and *callee j*, the cost of reading databases from s to t can be saved. The saving cost of reading databases through bypass pointers is

$2(b_1 + b_2 - 1)R$.

When the communication shortcut is available between *database* s and t, the cost of communication between s and t can also be saved. The saving cost of communication is $2(b_1 + b_2)C$. In this paper, we assume that communication shortcuts are unavailable in the underlying signaling network. Let FC_{cache} be the average cost of a *CacheFind* operation, and it can be described as follows:

$$FC_{cache} = (FC_{basic} + 2U)\gamma + (FC_{basic} - 2(b_1 + b_2 - 1)R)(1 - \gamma)$$

Finally, the total cost for *callee* j in the *eager caching* strategy is defined as follows:

$$TC_{cache} = n_1 \cdot MC_{cache} + n_2 \cdot FC_{cache}$$

3.4 Dynamic Location Tracking Strategy

In the *dynamic location tracking* strategy, a pair of bypass pointers is created between the caller and the called party. The forward bypass pointer for the callee caches the location of the callee's LA, and the reverse bypass pointer which caches the location of the caller's RA is created at the callee's LA. The two main operations in this strategy are called *ILAMove* and *ILAFind*. They are involved in the registration and call delivery procedures respectively.

- The cost of *ILAMove*

  Whenever *callee j* crosses RA boundaries, he/she reports the current location to his/her LA (*database t*). The LA keeps *callee j*'s location information; thus the bypass pointers for *callee j* are still valid. When *callee j* moves from RA_y to RA_z, the registration message is propagated from RA_z until it reaches $LCA(RA_y, RA_z)$. The acknowledgement message is returned from $LCA(RA_y, RA_z)$. Hence, the communication cost of $ILAMove$ is $2qC$, and it saves $\frac{1}{2}$ communication cost than *BasicMove* or *EagerMove*. $ILAMove$ just updates 3 databases, RA_y, RA_z, and $LCA(RA_y, RA_z)$. The cost of updating databases is 3U, which saves $(2q-2)U$ than *BasicMove* or *EagerMove*. Futhermore, there is a *LA address mapping table* stored at each RA. So, we get the location of the LA by querying this table of the RA. Besides, the cost of querying databases becomes 4R, which saves $(4q-2)R$ than *BasicMove* or *CacheMove*. Let $PM(q)$ be the cost of a *ILAMove* operation without leaving *database t*.

  $$PM(q) = MC_{basic} - (4q-2)R - (2q-2)U - 2qC) = 4R + 3U + 2qC$$

  If *callee j* leaves the subtree rooted at *database t*, then the *LA movement* procedure is invoked by *ILAMove*. And the new serving LA of *callee j* is $LCA(RA_y, RA_z)$. Besides, *callee j*'s profile is moved from the old LA to $LCA(RA_y, RA_z)$, and a *LA movement* pointer kept at the old LA points to the new serving LA. The bypass pointers for *callee j* are still stored at this old LA. In this situation, *ILAMove* pays an additional cost for changing the location of the LA, but saves the cost for updating bypass pointers at once. This additional cost for changing the LA is $2R + 2C + U$. Assume that the maximum height of the tree is H. Let $\overline{PM}$ be the average cost of a *ILAMove* operation without leaving its LA. $M''(t,i)$ denotes the cost of i *ILAMove* operations between two consecutive calls given a bypass pointer directed at *database t*. Then, we can obtain the average cost of a *ILAMove* operation denoted MC_{ILA}.

$$\overline{PM} = \sum_{q=1}^{H} PM(q)\frac{m^{H-q}}{m^{H-1}}$$

$$M''(t,i) = [(\overline{PM} + (2R + 2C + U))\gamma(t,i)]i + [\overline{PM}(1 - \gamma(t,i))]i$$

$$MC_{ILA} = \sum_{i=0}^{\infty} M''(t,i)\alpha(i)$$

- The cost of $ILAFind$

  If *callee* j does not leave *database* t, there is no need to update the location information of the forward bypass pointer stored at *database* s. When *caller* i at RA_x makes a call to *caller* j at RA_y through bypass pointers, the cost of reading databases between s and t can be saved. The saving cost of reading databases between s and t is $2(b_1 + b_2 - 1)R$. Besides, the cost of reading databases from t to RA_y can also be saved through the anchoring pointer. Hence, the saving cost of reading databases between t and RA_y is $2(r - b_2 - 1)R$.

  If *caller* i calls *callee* j and *callee* j has left *databse* t, it implies that the bypass pointers for *callee* j cache the location of his/her old LA. In this circumstance, the cache information needs to be updated. When the call request arrives *database* t, the call request needs to be forwarded to his/her current serving LA through the *LA movement* pointer stored at the old LA. By querying this old LA, the reverse bypass pointer for *callee* j can be acquired. This reverse bypass pointer should be moved to *callee* j's current serving LA. Hence, the bypass pointers for *callee* j will be updated at his/her current LA and *database* s through the return path of the acknowledgement message. Let FC_{ILA} be the average cost of a $ILAFind$ operation.

$$FC_{ILA} = (FC_{basic} + (3U + C) - (b_1 + b_2 - 1)R - (b_1 + b_2 - 2)R - 2(r - b_2 - 2)R)\gamma$$
$$+ (FC_{basic} - 2(b_1 + b_2 - 1)R - 2(b_1 + r - 2)R)(1 - \gamma)$$

Before deciding the most appropriate position for *callee* j's LA, the total cost for *callee* j in the *dynamic location tracking* strategy is defined as follows:

$$TC_{DLT} = n_1 \cdot MC_{ILA} + n_2 \cdot FC_{ILA}$$

Finally, we formula the cost of *callee* j after deciding the most appropriate location for his/her LA. Assume that *callee* j's LA has been adjusted from t to t'. Before next time the LA invokes a dynamic adjusting procedure, the postion of this LA won't be changed, even *callee* j leaves *database* t'. Let γ' denote the probability that the user leaves $T(t')$ between two consecutive calls. $M''(t', i)$ denotes the cost of i $ILAMove$ operations between two consecutive calls given a bypass pointer directed at *database* t'. We can clearly obtain MC'_{ILA} which denotes the average cost of a $ILAMove$ operation when *callee* j's LA has been dynamically adjusted from t to t':

$$\overline{PM} = \sum_{q=1}^{H} PM(q) \frac{m^{H-q}}{m^{H-1}}$$

$$M''(t', i) = [(\overline{PM} + 4C)\gamma'(t', i)]i + [\overline{PM}(1 - \gamma'(t', i))]i$$

$$MC'_{ILA} = \sum_{i=0}^{\infty} M''(t', i)\alpha(i)$$

Furthermore, Let FC'_{ILA} be the average of a $ILAFind$ operation when *callee* j's LA has been dynamically adjusted from t to t', which is described as follows:

$$\begin{aligned} FC'_{ILA} = {} & (FC_{basic} + 4C - 2(b_1 + \ell - 1)R - 2(r - \ell - 1)R)\gamma' \\ & + (FC_{basic} - 2(b_1 + \ell - 1)R - 2(r - \ell - 1)R)(1 - \gamma') \end{aligned}$$

Therefore, after deciding the most appropriate position for *callee* j's LA, the total cost for *callee* j in the *dynamic location tracking* strategy is defined as follows:

$$TC'_{DLT} = n_1 \cdot MC'_{ILA} + n_2 \cdot FC'_{ILA}$$

4 Simulation Results

4.1 Simulation Environment

In our simulation, m-way search tree is selected as the architecture of hierarchical location databases. Each node of the tree implies that the location database stores users' profiles. A database at a higher level stores users' profiles when users locate at the levels below it. Location databases are interconnected by the signal links of the network. Besides, Leaf nodes are location databases of the RAs. The user can store the records for his/her friends in the friend table of the mobile terminal. When the system starts, each user gets an event per hour, move or find. If he/she gets a find event, it implies that someone makes a call to him/her. 85% of a user's calls comes from his/her friends and 15% of those comes from other users in the system. When he gets a move event, he chooses a direction, left or right, and then moves to the neighbor RA.

We define system parameters listed in Table 6 for our simulation environment. We use different levels of the tree structure, different cost value, and different measure days to compare the performance between different strategies: the *basic*, *eager caching*, and *dynamic location tracking* (DLT) strategies.

Table 6: System parameters values

Parameter	Value	Comment
L	6, 7	Maximum height of the tree structure
M	4, 3	The degree of the tree structure
DB	1365, 1099	The number of location databases
MT	100,000	The number of mobile terminals
FT	5	The number of a user's friends
CMR	$1 \sim 20$	Call-to-Mobility Ratio
D	3, 5	Measure days
U	2, 0.4, 2	The cost of updating a database
R	1, 0.2, 1	The cost of reading a database
C	1, 1, 0.2	The cost of traversing a signal link

4.2 Experiment Results

We use different parameters such as tree level, cost value, and measure day to observe the effect of the simulation. Besides, for the *dynamic location tracking* strategy, we assume that the system performs computing the appropriate LA for users every three and five days.

First, we use different levels of the tree structure to compare their performance (See Figure 9 and Figure10), our strategy performs better than the *basic* and *eager caching* strategies. In this case, we can see that the *eager caching* performs worst when the CMR is smaller than 6. When the CMR is small, it implies that users may move to neighbor RAs frequently. When the user moves, the *eager caching* has to pay an extra cost to delete invalid bypass pointers. When the CMR is large, it implies that the user receives call frequently rather than moves. The *eager caching* has better performance than the *basic* because of using the bypass pointers to save the call setup time. Our strategy has the same performance in different tree structure. When the CMR is small, our strategy is better than the *eager caching* and *basic*, because it saves the cost of location registration. Our strategy uses the local anchor to keep the user's location information.

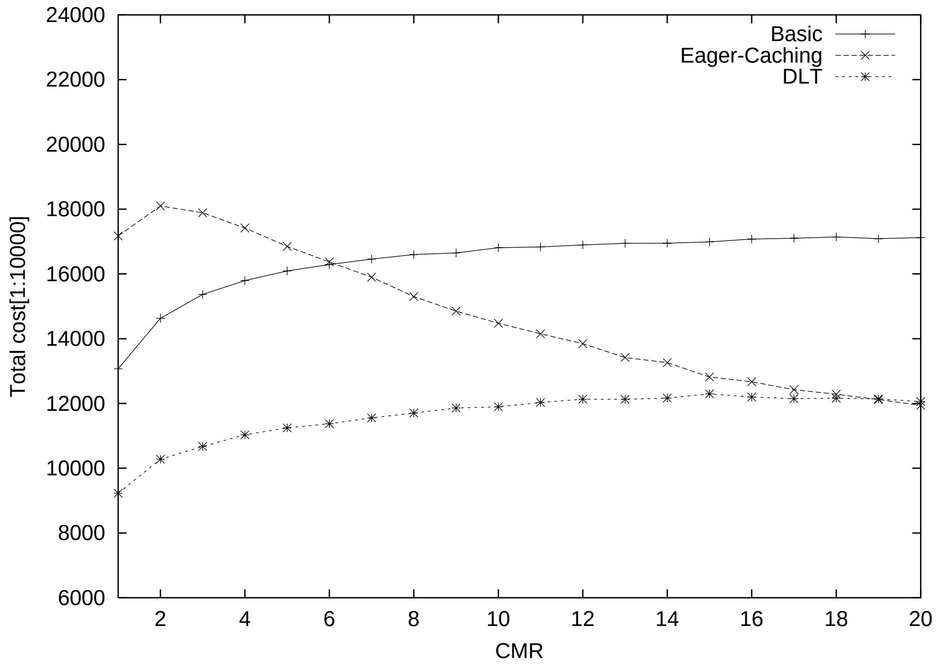


Figure 9: L=6, M=4, MT=100,000, D=3, FT=5, U=2, R=1, C=0.2

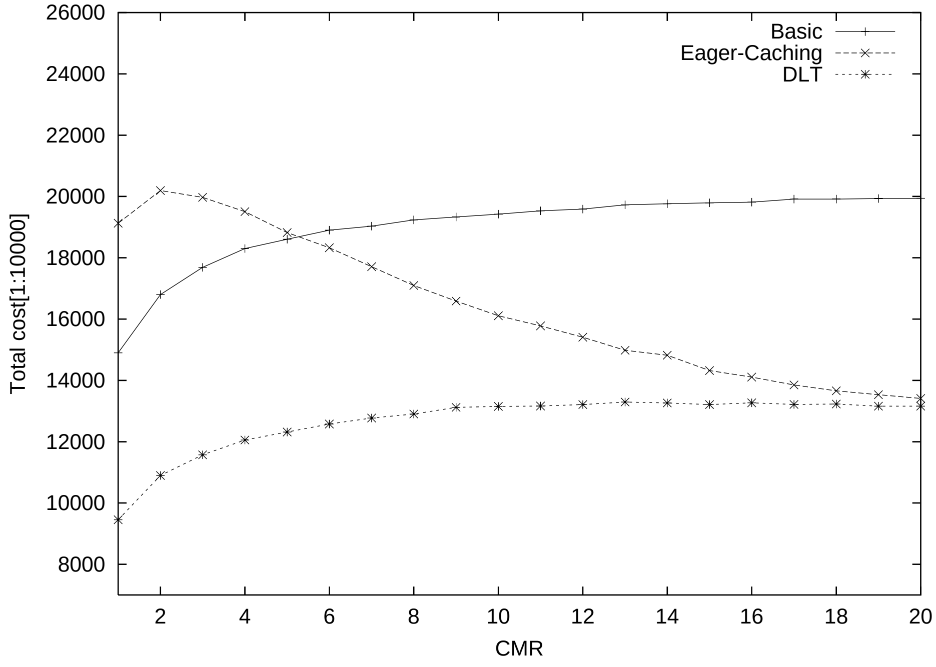


Figure 10: L=7, M=3, MT=100,000, D=3, FT=5, U=2, R=1, C=0.2

When the CMR is large, our strategy still performs well because of the fact that the DLT also uses bypass pointers to efficiently reduce the cost of the call delivery operation.

Furthermore, the DLT uses the appropriate position deciding scheme to evaluate whether the position of the user's LA should be adjusted. When the CMR is small, the LA will adjust its position to be close to the region where the user frequently moves. Thus the cost of location registration can be greatly saved . Besides, the position which the LA is adjusted to is also the region where the user receive calls frequently. When the CMR is large, the find events are much more than move events. The LA at a higher level of the tree is better than that at a lower level of the tree. Hence, the location of the LA is the same as before, it implies that the location of the LA won't be changed.

Figure 10 and Figure 11 shows that the appropriate LA deciding scheme is invoked in three and five days respectively, and users rarely change their behavior. We can see if a user rarely changes his/her behavior, and then his/her LA will keep static according to our appropriate position deciding scheme. The system can extend the period for invoking the dynamic adjust procedure. In Figure 10, Figure 12, and Figure 13, we use different cost value to compare the performance. When the communication cost is more expensive than the cost of accessing databases, the *eager caching* does not perform well because of paying an addiction cost to delete invalid bypass pointers when the user leaves *database t*. If several bypass pointers cache this location t, the cost of deleting invalid bypass pointers will increase dramatically. Our strategy doesn't delete bypass pointers when the user leaves *database t*. The bypass pointers are updated through the acknowledgement messages of the call delivery procedure.

Finally, we compare the difference between the DLT before invoking the appropriate position deciding scheme (DLT_{before}) and the DLT after invoking the appropriate position deciding scheme (DLT_{after}) in Figure 14. When the CMR is small, the performance of DLT_{after} is better than DLT_{before}. We see that the move events do influence the result of deciding which location is suitable for the user's LA. As the CMR increases, DLT_{before} and DLT_{after} will be the same because of the fact that call events are more than move events. When the user's call events are more than move events, the LA won't be adjusted to a lower level of the tree.

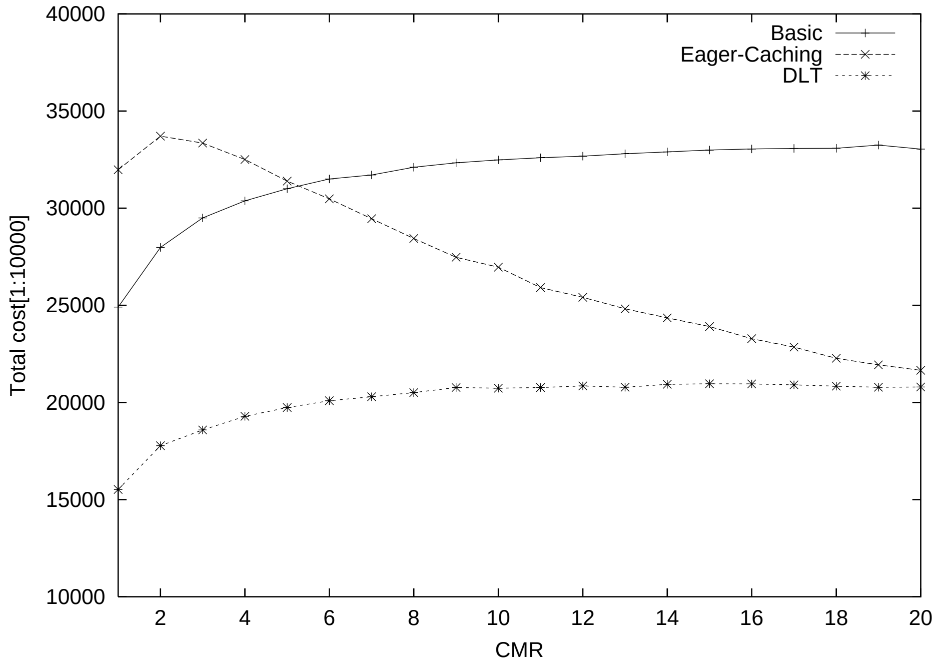


Figure 11: L=7, M=3, MT=100,000, D=5, FT=5, U=2, R=1, C=0.2

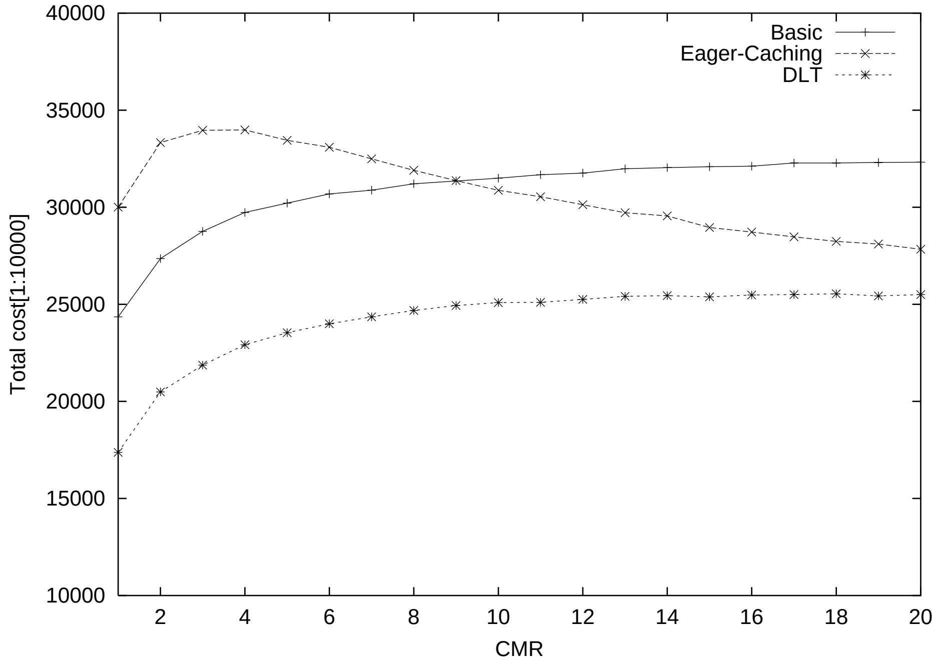


Figure 12: L=7, M=3, MT=100,000, D=3, FT=5, U=2, R=1, C=1

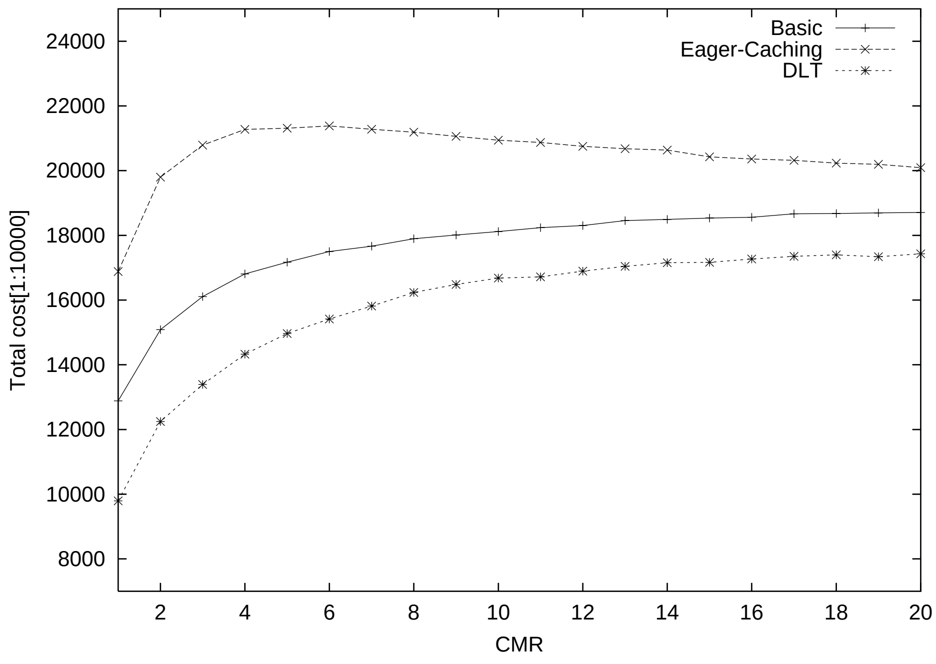


Figure 13: L=7, M=3, MT=100,000, D=3, FT=5, U=0.4, R=0.2, C=1

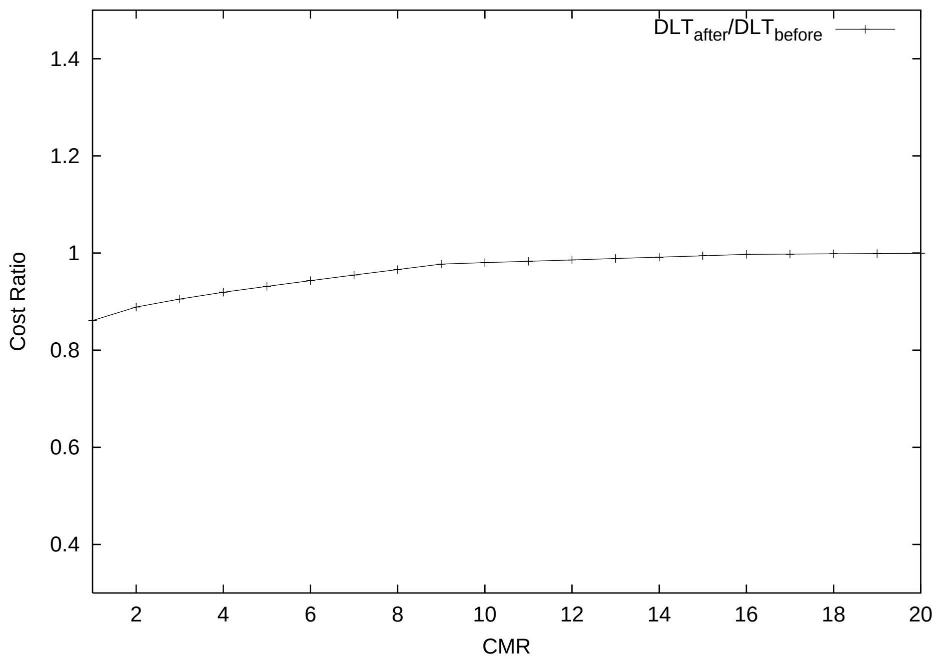


Figure 14: L=7, M=3, MT=100,000, D=3, FT=5, U=0.4, R=0.2, C=1

5 Conclusion

The main contribution of the paper is to propose the *dynamic location tracking* strategy, which dynamically adjusts the position of the local anchor according to the user's behavior. Our strategy is based on the *caching* strategy, but modifies the registration and call delivery procedures to get a great benefit in reducing communication and database access costs. For registration procedures, we uses the LA to track a user's current location, and it can greatly reduce the registration cost. For call delivery procedures, we make the forward bypass pointers cache each user's LA, which can avoid cache miss when the user leaves t. Besides, the appropriate position deciding scheme is invoked in our strategy to decide which position is better for a user's LA according to his/her mobility and calling patterns. Then we adjust his/her LA to the best position. We can get a great benefit in location registration through this dynamic adjusting LA procedure. Besides, while the position of the LA has been changed from t to t', several forward bypass pointers which cache the previous position t may need to be updated at the same time. In our strategy, these bypass pointers will be updated through the acknowledgement messages of the call delivery procedure. In other words, the bypass pointer will be updated through the acknowledgement message only when it is used by a user to perform the call delivery procedure. Thus we do not need to update several bypass pointers at once.

With a small CMR, the frequency of successive move operations is high. Our strategy greatly reduces registration cost by using a LA to keep a user's current location. With a large CMR, the frequency of successive find operations is high. Our strategy still greatly reduces call delivery cost by using bypass pointers to locate mobile users. In our strategy, each location database which serves as a LA must store a user's information and provide the computing function for adjusting LA dynamically. Besides, each user's LA will not locate at the same level; thus all users' profiles are efficiently distributed in the tree structure. We compare the performance of different strategies in the simulation. The results clearly indicate that our strategy has better performance than the *basic* and *caching* strategies.

References

[1] EIA/TIA,"Cellular radio-telecommunicaiton intersystem operations,"Tech. Rep. IS-41 Reversion B, EIA/TIA, 1991.

[2] I.F. Akyildiz, J. McNair, J. Ho, H. Uzunalioglu and W. Wang, "Mobility management in next-generation wireless systems," *Proc. IEEE*, vol. 87 , no. 8 , pp. 1347 -1384, Aug. 1999.

[3] C. Eynard, M. Lenti, A. Lombardo, O. Marengo, and S. Palazzo, "A methodology for the performance evaluation of data query strategies in universal mobile telecommunication systems (UMTS)," *IEEE Journal Selected Areas in Communications*, vol. 13, no. 5, pp. 893-907, June 1995.

[4] J. S. M. Ho and I. F. Akyildiz, "Local anchor schmem for reducing signaling costs in personal communications networks," *IEEE/ACM Trans. on Networking*, vol. 4, no. 5, Oct. 1996,pp. 709-725

[5] J.S.M. Ho and I.F. Akyildiz, "Dynamic hierarchical database architecture for location management in PCS networks," *IEEE/ACM Transactions on Networking*, vol. 5, no. 5, pp. 646-660, Oct. 1997.

[6] H.-C. Lin and S.L. Lee, "A Presetting Location Strategy for Personal Communication Using Hierarchical Location Database," *Proceeding of International Conference on Parallel and Distributed System*, pp. 349-356, 2000.

[7] Y.-B. Lin and W.-N. Tsai, "Location tracking with distributed HLR's and pointer forwarding," *IEEE Transactions on Vehicular Technology*, vol. 47, no. 1, pp. 58-64, Feb. 1998.

[8] R. Jain, Y.-B. Lin, C. Lo, and S. Mohan, "A caching strategy to reduce network impacts of PCS," *IEEE Journal Selected Areas in Communications*, vol. 12, no. 8, pp. 1434-1444, Oct. 1994.

[9] R. Jain and Y.-B. Lin, "An auxiliary user location strategy employing forwarding pointers to reduce network impacts of PCS," *IEEE International Conference in Communications*, vol. 2, pp. 740-744, 1995.

[10] R. Jain, “Reducing traffic impacts of PCS using hierarchical user location databases,” *IEEE International Conference in Communications*, vol. 2, pp. 1153-1157, 1996.

[11] R. Jain and F. Anjum, “Caching in hierarchical user location databases for PCS,” *IEEE International Conference on Personal Wireless Communication*, pp. 496-500, 1999.

[12] S. Mohan and R. Jain, “Two user location strategies for personal communications services,” *IEEE Personal Communications*, vol. 1, no. 1, pp. 42-50, First Quarter 1994.

[13] M. Mouly and M. B. Pautet, ”The GSM system for personal communication services,” M. Mouly, 49 rue Louise Bruneau, Palaiseau, France, 1992.

[14] E. Pitoura, G. Samaras,”Locating objects in mobile computing,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 13 , no. 4 , pp. 571-592, July-Aug. 2001.

[15] K.-L. Sue and C.-C. Tseng “One-step pointer forwarding strategy for location tracking in distributed HLR environment,” *IEEE Journal Selected Areas in Communications*, vol. 15, no. 8, pp. 1455-1466, Oct. 1997.

[16] N. Shivakumar and J. Widom, ”User profile replication for faster location lookup in mobile environments,” in *Proc. ACM MOBICOM’95*, Nov. 1995, pp. 161-169.

[17] S. Tabbane, “Location Management Methods for Third-Generation Mobile System,” *IEEE Personal Communications.*, pp. 72-84, Aug. 1997.

[18] J.Z. Wang, “A fully distributed location registration strategy for universal personal communication systems,” *IEEE Journal Selected Areas in Communications*, vol. 11, no. 6, pp. 850-860, Aug. 1993.