Knowl Inf Syst (2005) 8: 414–437
DOI 10.1007/s10115-005-0196-7

Knowledge and Information Systems

REGULAR PAPER

Manish Gupta · Manghui Tu · Latifur Khan · Farokh Bastani · I-Ling Yen

A study of the model and algorithms for handling location-dependent continuous queries

Received: 30 July 2003 / Revised: 29 November 2004 / Accepted: 23 January 2005 /
Published online: 20 April 2005

Abstract Advances in wireless and mobile computing environments allow a mobile user to access a wide range of applications. For example, mobile users may want to retrieve data about unfamiliar places or local life styles related to their location. These queries are called location-dependent queries. Furthermore, a mobile user may be interested in getting the query results repeatedly, which is called location-dependent continuous querying. This continuous query emanating from a mobile user may retrieve information from a single-zone (single-ZQ) or from multiple neighbouring zones (multiple-ZQ). We consider the problem of handling location- dependent continuous queries with the main emphasis on reducing communication costs and making sure that the user gets correct current-query result. The key contributions of this paper include: (1) Proposing a hierarchical database framework (tree architecture and supporting continuous query algorithm) for handling location-dependent continuous queries. (2) Analysing the flexibility of this framework for handling queries related to single-ZQ or multiple-ZQ and propose intelligent selective placement of location-dependent databases. (3) Proposing an intelligent selective replication algorithm to facilitate time- and space-efficient processing of location-dependent continuous queries retrieving single-ZQ information. (4) Demonstrating, using simulation, the significance of our intelligent selective placement and selective replication model in terms of communication cost and storage constraints, considering various types of queries.

Keywords Mobile computing · Location-dependent queries · Location-dependent continuous querying · Multiple-ZQ · Single-ZQ · Hierarchical database framework

M. Gupta · M. Tu · L. Khan (✉) · F. Bastani · I.-L. Yen
Department of Computer Science, University of Texas at Dallas, Richardson, TX 75080, USA
E-mail: lkhan@utdallas.edu

1 Introduction

In a mobile computing environment, mobile users are located in system-defined zones that correspond to bounded geographical areas. Advances in wireless and mobile computing environments allow a mobile user to retrieve data about unfamiliar places or local life styles related to their current zone.

The location of a mobile user plays a significant role in the data retrieval universe [22, 27]. A mobile user may want to retrieve data about unfamiliar places or local life styles related to his current location. The data, the value of which is determined by the location to which it is related, is called *location- dependent data* (*LDD*) [5, 18–20]. Accordingly, we define the *fixed LDD-based user location- dependent query* (*FULDQ*) as a query issued by a *mobile user*, where the LDD involved is *fixed*. For example, a mobile user, while driving a car, might be interested in finding nearest hotels and issues a query: "List all hotels within 1 mile radius of my current location." Besides this, a user may ask, "How much sales tax did I pay?" This query is fully location independent. Also, a user may ask, "What is the room rate at Denver's Holiday Inn?." This query retrieves LDD, but is not a FULDQ because it is not dependent on the user's current location. Furthermore, in the query "Find all neighbouring cars within 100 feet of my car," the LDDs involved in the query are not fixed and are not considered in this paper.

Frequently, a mobile user may be interested in getting LDD repeatedly [7]. For example, the user may ask, "List all hotels within 1-km radius of my current location in intervals of 15 minutes." Because the user is moving, with the progression of time, the results of the query must be updated to reflect the user's changing location. We define a *fixed LDD- based user location-dependent continuous query* (*FULDCQ*) as a continuous FULDQ, which gets re-activated after every fixed time interval.

A FULDQ or a FULDCQ may be related to a single-ZQ or to multiple-ZQ. For example, if the user requests a list of all hotels within a 1-mile radius of his current location in intervals of 15 min, then the LDD may correspond to the user's current base station or zone only, assuming that a single base station covers a 1-mile radius. But if the user request is to list all hotels within a 3-mile radius of his current location in intervals of 15 min, the LDD may correspond to multiple neighbouring zones, extrapolating from the above and assuming that a 3-mile radius is now covered by more than one base station.

In this paper, we propose a framework and a novel approach for handling FULDCQ in a mobile environment. This approach relies on a tree structure that is flexible enough to handle queries of either single-ZQ or multiple-ZQ. The primary contributions in this paper are:

1. Proposing a hierarchical framework for handling location-dependent continuous queries. The system maintains two types of databases, user location database (ULDB) that stores mobile users' current location information and location-dependent database (LDDB) that stores LDD for specific base stations (zones), maintained at some nodes in the tree. We propose a hybrid scheme for storing users' location information in ULDB and propose a continuous-query algorithm that can efficiently handle FULDCQ in our hierarchical framework by reducing communication costs.

2. Analysing the flexibility of our framework for handling FULDCQ related to single-ZQ or multiple-ZQ. For handling multiple-ZQ (excluding all-zone queries that access information related to all zones), selective placement (SP) of more than one LDDB at some intermediate level nodes in the tree yields the best performance. For handling single-ZQ, base station placement (BP) with multiple LDDB at the base stations is the optimum LDDB-placement strategy. This intelligent selective LDDB placement strives to reduce network message hops (NMH, i.e. communication cost).
3. Proposing a selective replication algorithm in the context of FULDCQ of type single-ZQ that is complementary to [15]. Our replication algorithm is based on the users' query pattern. If a particular type of single-ZQ frequently emanates from a particular zone, then we only replicate its corresponding LDD in that zone. This selective replication will serve to reduce NMH and, at the same time, the degree of replication would be limited by the storage constraint.
4. Demonstrate using simulation that, with queries of multiple-ZQ or mixed types, intelligent selective LDDB placement incurs less communication cost than base station placement (BP) or root placement (RP) (both BP and RP will be defined later). And when the queries are only of single-ZQ types, intelligent selective LDD replication will incur less communication cost for different degrees of replication and storage constraints than having no replication.

The remainder of this paper is organised as follows: Sect. 2 gives the background about the basic architectures for managing mobile users' current location and some of the assumptions that we made about the problem and the network chosen. Section 3 presents our proposed framework for handling FULDCQ. Section 4 presents selective replication in detail. Section 5 describes the simulation of our system, including details about the performance of various approaches. Section 6 briefly describes some related work. Finally, Sect. 7 presents the conclusion and future work.

2 Background

The location of a mobile user plays a significant role in processing FULDCQ. The basic database architectures for storing the location of the mobile users are distributed and centralised databases. The two most common approaches for distributed databases are a two-tier scheme [3, 21] and a hierarchical scheme [4, 17, 23, 31].

In two-tier schemes, a home database, termed home location register (*HLR*), is associated with each mobile user. The HLR is located at the zone prespecified for each user. It maintains the current location of the user as a part of the users' profile. The visitor location registers (*VLRs*) are maintained at each zone. The VLR at a zone stores copies of the profiles of a user not at their home location and currently located inside that zone. One major drawback of the two-tier approach is that it does not scale well with highly distributed systems where sites or zones are geographically widely dispersed. To track a mobile user, the possibly distant home location must be contacted first. Similarly, even a move to a nearby location must be registered at a potentially distant home location.

Hierarchical location schemes extend two-tier schemes by maintaining a hierarchy of location databases. In this hierarchy, a location database at a higher level contains location information for users located at levels below it. Usually, the hierarchy is tree structured. A location database at a leaf serves a single zone (base station) and contains entries of all users registered in this zone. A database at an internal node maintains information about users registered in the set of zones in its subtree. When hierarchical location databases are used, there is no need for binding a user to a specific HLR. The user can be tracked by querying the databases in the hierarchy. In the worst case, an entry for the user will be found in the database at the root. Thus, the hierarchical scheme leads to a reduction in communication cost when most moves of the users are geographically localised. 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 neighbourhood are accessed.

In the centralised approach, the locations of all mobile users are stored in a single centralised database management system (*DBMS*). All location queries and updates are directed to the central DBMS. Using an existing DBMS is not sufficient because existing DBMS does not handle continuously changing data well, such as the location of the moving objects. Also, while processing FULDCQ, the centralised approach will not scale up well when the number of mobile users increases. We thus believe that a tree architecture [13, 17, 31] is the more appropriate architecture for processing FULDCQ. This is because location databases in a tree architecture are physically structured based on location. Also, the mobile user can be tracked efficiently by querying in a tree.

3 Framework

We propose a hierarchical database framework (tree architecture and supporting CQ algorithm) for handling location-dependent continuous queries that provide the user with correct current query results, not any outdated results. We also present the basic assumptions made about our tree framework and the problem. We also analyse the flexibility of this framework for handling queries related to single-ZQ or multiple-ZQ.

Our proposed architecture for FULDCQ is based on a tree structure [16], as shown in Fig. 1. The geographical coverage area is partitioned into wireless zones. Each zone is serviced by a base station (BS), which is represented by a leaf node in the tree. In Fig. 1, the leaf nodes 7, 8, 9, 10, 11 and 12 are the base stations.

In the tree, we maintain two types of databases. One is the users' location database (ULDB) that stores a mobile users' current location information. The other is the location-dependent database (LDDB) that stores LDD for specific base stations.

Within the tree, LDDBs are maintained only at selected nodes (called LDDB nodes), but ULDBs are maintained at every node. The LDDB at an internal node maintains LDD for all base stations within its subtree. For example, in Fig. 1, the LDDB nodes are 1 and 2. Node 1 will contain LDD for BS 7, 8 and 9; node 2 will contain LDD for BS 10, 11 and 12. Also, in Fig. 1, ULDBs are placed at each node of the tree. The ULDB at a leaf node serves a single base station and contains

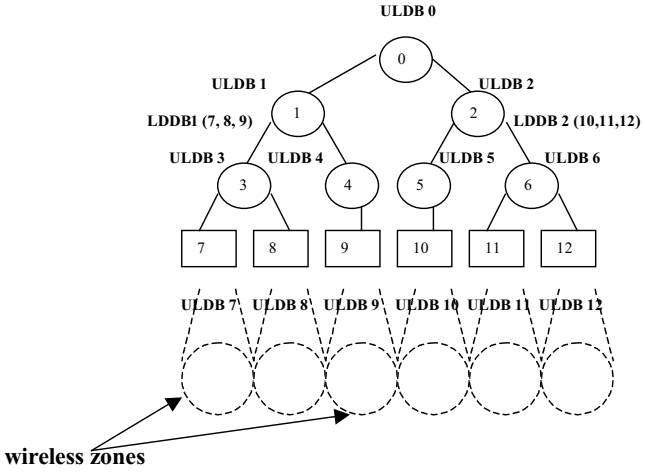

Fig. 1 A sample tree architecture.

location information for all users currently in this base station. The ULDB at an internal node maintains location information about all users currently in the set of the base stations in its subtree.

3.1 Assumptions

Some of the assumptions that we make about our tree framework and the problem are discussed below. The assumptions about our tree framework are as follows:

- Databases at the leaf nodes (base stations) and internal nodes have storage and query processing capabilities, but leaf node storage is relatively limited.
- Databases are usually interconnected through links of the intelligent signaling network [1, 31].
- All nodes maintain information about each BS's LDDB node, e.g. if BS x's LDDB node is y, then LDD of BS x can be found in node y and all nodes know about this information.
- All the leaf nodes (base stations) might not be geographically located in a linear order from left to right as shown in Fig. 1, but could be geographically located in a nonlinear fashion. At the same time, the base stations within the network only correspond to local regions, like nearby cities, nearby counties and not regions that are geographically far apart, like different countries or even different states. Thus, a mobile user is able to move between any two base stations in the network.

Assumptions about the nature of FULDCQ we are handling are as follows:

- We consider FULDQ types where the user requesting a query is moving and the objects related to the query are fixed.
- We consider continuous queries where the answer is automatically updated by the system as opposed to instantaneous queries, for which only a one-time answer is obtained.

- We consider all the queries as equally important and do not give any preferences to a particular query.

3.2 Database updates

We consider LDDB to be a static database, which can only be queried, and rarely needs to be updated because we consider only those types of the *FULDQ*s where the objects involved are fixed and not moving. Thus, the database containing LDD for a particular base station is always fixed. For example, information about local restaurants in a particular base station is usually fixed. On the other hand, ULDB is considered to be a dynamic database, which has to be continuously updated. This is because the database stores the users' current location information.

The location information at ULDB for each mobile user is either a pointer to an entry at a lower level ULDB or the user's actual current location. In order to keep track of a mobile user, his or her location entry is deleted at some ULDB, updated at some ULDB, and inserted at some ULDB, as discussed in Sect. 3.3. For example in Fig. 1, consider all ULDBs maintain users' actual locations. Now a move by a user x from base station 7 to base station 9 causes deletion of his location information from ULDB 7 and ULDB 3, update of his information at ULDB 1 and ULDB 0, and insertion of his information at ULDB 4 and ULDB 9.

3.3 Hybrid scheme for storing information in ULDB

In order to keep track of the mobile user, we store its location information in specific ULDBs within the tree. For storing mobile user location information in ULDB, two alternative approaches can be used [3, 12, 23]. In one approach, all ULDBs maintain the actual location of each user. In this approach, a move by a user x from zone i to j causes an update of all its ULDB entries along the paths from j to the root and from the root to i. In this approach, the update cost is quite high, but the lookup cost is comparatively low. In another approach, a pointer is maintained for x's location that points to lower level ULDBs. In this approach, a move by user x from zone i to j causes update of all its ULDB entries along the paths from j to the $LCA(i, j)$ (i.e. the least common ancestor of nodes i and j) and from the $LCA(i, j)$ to i. The advantage of maintaining this type of location information is that the overall update cost is low, but the lookup cost is quite high. For finding x's actual location, we need to continually follow the chain of forwarding pointers until we reach the actual location.

We use a hybrid scheme that utilises both types of approach. Thus, the ULDB at a certain node where LDDB resides (called LDDB nodes) maintains the users' actual locations. The rationale behind this is to process FULDCQ at LDDB nodes effectively. The ULDB at nodes that are above the LDDB nodes maintain a pointer to the lower level ULDBs. The ULDB at nodes that are below the LDDB nodes maintain the actual location of the users. For example, in Fig. 2, consider LDDB nodes 1 and 2 and a mobile user x who is currently in BS 12. Now, there will be an entry in ULDB 0 for user x pointing to the entry for user x in ULDB 2. This entry for user x in ULDB 2 will contain the actual current location of user x, i.e.

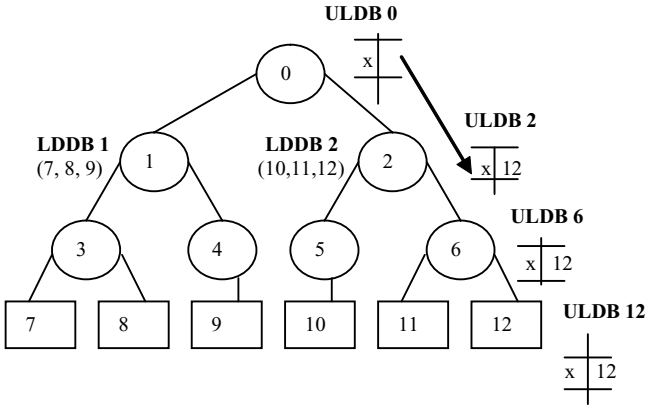

Fig. 2 Location information at ULDBs with LDDB nodes 1 and 2.

location 12. In addition, ULDB 6 and ULDB 12 will contain the actual location of the user *x*, i.e. location 12.

3.4 Placement of LDDB

Different strategies can be used to place LDDB in the tree. We discuss three of these strategies and analyse their performance for query types single-ZQ or multiple-ZQ.

3.4.1 LDDB placed at the root node (RP)

A single LDDB can be placed at the root node. In Fig. 2, a single LDDB placed at root node 0 will contain LDD for all the base stations (7–12). RP incurs maximum communication cost when the query type is single-ZQ. For multiple-ZQ, RP may still incur higher communication cost than other strategies because the query will be pushed from the leaf node up toward the root node and the query results will traverse a longer path back to the leaf node (see Sects. 3.6 and 5). But we need to take into consideration the number of zones the queries cover. In an extreme case, if all the queries access information related to all the zones in the system, then RP becomes the best LDDB-placement strategy.

3.4.2 LDDB placed at the base stations (BP)

LDDB can be placed at every base station. Thus, LDD of each base station is stored at its respective LDDB. In Fig. 2, LDD for base station 7 is stored only at node 7. This placement strategy incurs a minimum lookup latency and communication cost when query type is only single-ZQ because it will be handled by the leaf node itself. But this strategy will incur high communication cost when the query type is multiple-ZQ or mixed (see Sects. 3.6 and 5). However, this approach is not realistic due to storage limitations at the base stations and also because, in a real environment, base stations should not be burdened with data-management

activities [23]. This is because, during performance evaluation of our proposed framework, other than communication cost, storage cost will also play an important role because, in the near future, mobile users might be interested in LDD, which include multimedia information, as a combination of text, graphics, animation, images, sound, speech and still or moving video. This will happen due to the increase of network bandwidth in mobile environments. For example, Universal Mobile Telecommunications System and its new technology will allow increased transmission rates up to 10 Mbps by 2010. It is important to note that this multimedia data demands huge storage capacity. For example, 3000 colour pictures require 3 Gb storage ($640 \times 480 \times 24$ bits $= 1$ Mb/picture). Furthermore, with the current state of the art, base station or cell does not facilitate this bulk storage to store data but may provide a limited filtering mechanism with limited buffering at the packet level.

3.4.3 LDDB placed at selected nodes (SP)

Here, LDDBs are placed at selected nodes, leaf or nonleaf nodes or a combination of both. This approach is more realistic. If a base station has a storage limitation, an ancestor node will contain the missing data. In Fig. 2, if LDDB are placed at nodes 1 and 2, then the LDD of base stations 7, 8, 9 are stored at the LDDB of node 1, the LDD of base stations 10, 11 and 12 are stored at the LDDB of node 2. Detailed discussion on placement policies is given in Sect. 3.6.

3.5 FULDCQ algorithm

The LDDB at an internal node maintains the LDD for all base stations within its subtree. We define the valid scope of an LDDB node as the set of base stations within the subtree of this LDDB node. Whenever a mobile user x, currently in the *BS D* (as shown in Fig. 3), issues a continuous query (*CQ*), then *D*'s LDDB node (let it be node *B*) becomes x's current LDDB node (CLDDBN). CQ is then forwarded to CLDDBN B. The LDDB of CLDDBN B generates a continuous query result (CQR) for the user x at BS D. CLDDBN B after generating CQR checks whether x is still within its scope by checking x's location in its ULDB. When x changes zone, the new zone may or may not be within the scope of x's previous CLDDBN B. The two possible cases are shown in Fig. 3.

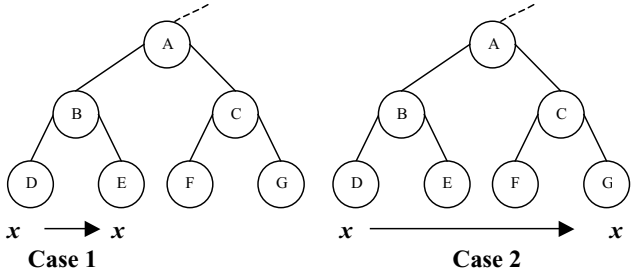

Fig. 3 A subtree to represent valid scope of a mobile user x.

Table 1 Symbols for FULDCQ algorithm.

Symbols	Description
CQ	Continuous Query.
CQR	Continuous Query Result.
XCBS	x's Current Base station.
CLDDBN	x's Current *LDDB* node.
PLDDBN	x's Previous *LDDB* node.
QBS	*LDDB* querying BS.

Case 1 The user x remains within the valid scope of its CLDDBN B. In this case, CLDDBN B checks whether x's current *BS* (*XCBS*) is still the same as its LDDB queried BS. If yes, then CLDDBN B forwards the CQR to x. If not, then the LDDB of CLDDBN B regenerates CQR for the user x at base station XCBS.

Case 2 The user x moves out of the scope of its *PLDDBN B* (CLDDBN B becomes x's previous LDDB node, i.e. PLDDBN). In this case, a user x moves out of the scope of its PLDDBN B. PLDDBN B then redirects the *CQ* to its parent node. This redirection of *CQ* to the parent node continues till x's location is found in ULDB. *CQ* is then redirected to its descendent nodes using x's entry in the ULDB till *CQ* reaches x's current LDDB node (let it be node C of Fig. 3), i.e. following the chain of forwarding pointers for x's location (see Sect. 3.3). The LDDB of CLDDBN C generates CQR for the user x at base station XCBS.

Based on user x's movement across zones, the processing of *CQ* continues till x's current LDDB node finds that XCBS is the same as its LDDB queried *BS* (as seen in Case 1).

The FULDCQ algorithm, which supports our proposed tree architecture for FULDCQ, is shown in Fig. 4. The symbols used in the FULDCQ algorithm are summarised in Table 1. Lines (10–17) of the algorithm represent *Case 1*, while lines (19–28) of the algorithm represent *Case 2*.

3.6 Flexibility of hierarchical framework

A FULDCQ issued by a mobile user x may be related to a single-ZQ or may be related to multiple-ZQ. For example, if the user requests a list of all hotels within a 1-mile radius of his current location in intervals of 15 min, then the LDD may be related to the user's current base station only, assuming that a single base station will cover a 1-mile radius. But if the user's query requests a list of all hotels within a 3-mile radius of his or her current location in intervals of 15 min, the LDD may be related to multiple neighbouring zones. It is likely that coverage of a 3-mile radius will entail the use of multiple base stations.

Our hierarchical framework is flexible enough to handle FULDCQ related to single-ZQ or multiple-ZQ. The framework also facilitates better placement strategies because multiple LDDBs can be placed at any level in the hierarchy, depending on the query types.

```
        /* A mobile user x currently at XCBS sends CQ */
/* XCBS forwards CQ to its CLDDBN */
/* repeat
        forward CQ to its parent- node
   until parent-node = CLDDBN */
Initially QBS ← XCBS
repeat
        CLDDBN generates CQR for QBS
        /* If user x is still within the valid scope of CLDDBN */
        if (x's entry found in (ULDB of CLDDBN))
                if (QBS = XCBS)
                        /* Forward CQR to child-node till we reach XCBS */
                        repeat
                                forward CQR to its child-node
                        until child-node = XCBS
                        XCBS sends CQR to x
                else    QBS ← XCBS
        /* If user x moves out of the valid scope of PLDDBN */
        else
                /* Redirect CQ to its parent-node till user x 's location found */
                repeat
                        redirect CQ to its parent-node
                until x's entry found or 'Root'
                /* Redirect CQ to its child-node till we reach CLDDBN */
                repeat
                        redirect CQ to its child-node
                until child-node = CLDDBN
                QBS ← XCBS
until specified time-period expires
```

Fig. 4 Algorithm for continuous query.

3.6.1 Selective placement for multiple-ZQ

In our architecture, each LDDB node maintains LDD for all base stations within its subtree. This suggests that, for multiple-ZQ, the selective placement (*SP*) of more than one LDDB at some intermediate level in a tree may be a better LDDB-placement strategy. This is because the number of network hop traversals for messages, called network message hops (*NMH*), a key cost in message transmission time, will be reduced. On the other hand, for BP and RP, NMH will be substantially high. This is because, in the BP case, the base station receiving a request will not only retrieve results from its own LDDB but will also need to send requests to neighbouring base stations. And in the RP case, the requesting base station will send requests all the way to the root node with results being sent back from the root to the requesting base station. For example, in Fig. 1, a user at base station 7 issues a continuous query asking for a list of all hotels within a 3-mile radius of his or her current location in intervals of 15 min. Let us assume that the area within a 3-mile radius of the users current location will be covered by base stations 7, 8 and 9. With BP, we observe that NMH is equal to 12. This is because base

station 7 sends a request to base station 8 and gets back a reply. Thus, a total of 4 NMH (7 to 3, 3 to 8, 8 to 3 and 3 to 7) accumulates. Similarly, if base station 7 sends a request to base station 9 and gets back the reply, the total NMH is 8. Thus, the total NMH for BP 7 is 12 (4 + 8). With RP, we observe that NMH is equal to 6. This is because base station 7 sends a request to root node 0 and gets back a reply. Thus, a total of 6 NMH (7 to 3, 3 to 1, 1 to 0, 0 to 1, 1 to 3 and 3 to 7) accumulates. With selective placement of LDDB at node 1 for base stations 7, 8 and 9, we observe that NMH is equal to 4. This is because base station 7 sends a request to node 1 and gets back a reply from it. Thus, a total of 4 NMH (7 to 3, 3 to 1, 1 to 3 and 3 to 7) accumulates. Therefore, NMH is lowest for SP.

In general, the SP strategy can incur mimimum NMH when the query type is multiple-ZQ. The hierarchy level at which the LDDB should be placed depends on the zones the queries access. For example, if all the queries are multiple-ZQ and involve only two zones, then placing the LDDB at the nodes one level above the leaf nodes incurs minimum NMH. Furthermore, if all the queries access all the zones in the system, then placing the LDDB at the root node incurs minimal NMH (discussed in Sect. 3.4.1). Because the actual query types can be highly mixed, we use experimental data to determine the best placements for specific access patterns.

3.6.2 Base station placement for single-ZQ

For queries of type single-ZQ, the placement of an LDDB at each base station is the best LDDB-placement strategy to reduce NMH, assuming base stations can handle queries and have enough storage. This is because, while handling single-ZQ in BP, the query does not require being pushed to upper level nodes and the query results are available at the BS itself.

Thus, it is obvious that, for multiple-ZQ type queries, SP at an intermediate node is the better choice to store LDD compared with RP and BP. On the other hand, for query type only single-ZQ type queries, BP is the best choice to store LDD. However, BP placement is not a realistic approach (see Sect. 3.4.2).

4 Replication

Due to storage limitations of the base stations and because base stations should never be burdened with data-management activities [23], LDDB placement at the base stations is not a realistic strategy. Hence, we propose a selective replication algorithm in the context of FULDCQ of type single-ZQ, which depends on the user's query pattern.

If a query of single-ZQ type is frequently asked from a particular base station, it would be judicious to replicate corresponding LDD at that base station (if possible) or at some intermediate node near that base station. On the other hand, if multiple-ZQ type is frequently asked from a particular base station, there are several possible alternatives. One alternative is replicating relevant LDD corresponding to multiple base stations at that base station. This alternative is not a practical one because base stations cannot have enough storage for storing LDD that corresponds to multiple base stations and, indeed, storage requirements will be higher as compared with storing LDD that corresponds to a single

base station (i.e. single-ZQ). The other alternative would be selective placement at an intermediate node (discussed in Sect. 3.6.1), which is a more realistic approach. Therefore, in this paper, replication is considered only for single-ZQ type queries.

Replication reduces network message hops because it increases the probability of finding the LDD locally [6, 9, 24, 25]. Replication of LDD at selected nodes does not increase the update cost assuming it is always fixed for a particular base station as discussed in Sect. 3.2.

4.1 Replication based on user query pattern

In the tree architecture, the LDD can be replicated both at the leaf node (i.e. base station) as well as the internal nodes of the hierarchy. If a particular type of query Q_i is mostly requested from base station j, then the LDD corresponding to query Q_i should be replicated locally at base station j. For analysing LDD replication, we define the following parameters:

Let N_{ij} be the number of FULDCQs of type Q_i requested from base station j over a given time period, T. Let $S_{i,j}$ be the total size (in megabytes) of the LDD corresponding to query Q_i for base station j. The local query-to-size ratio, $\mathrm{LQSR}_{i,j}$, for leaf node j is the ratio of $N_{i,j}$ to $S_{i,j}$, i.e.

$$\mathrm{LQSR}_{i,j} = N_{i,j}/S_{i,j} \tag{1}$$

For internal node j

$$\mathrm{LQSR}_{i,j} = \sum_k N_{i,k} \Big/ \sum_k S_{i,k}, \tag{2}$$

where k is a child of j. That is, local query-to-size ratio for a query Q_i and an internal node j is the ratio of the total number of FULDCQs of type Q_i requested from any of the base stations of j's subtree to the total size of the LDD of query Q_i for all of the base stations of j's subtree.

4.2 Parameters in replication

We define four parameters: N, $C_{\min}$, $C_{\max}$ and L, where N is the maximum number of replicas of LDD for each query type (excluding the LDD at the LDDB node); $C_{\min}$ and $C_{\max}$ together determine when a node may be selected for an LDD replica, and L denotes the node level at which the LDDB is placed inside the tree. Within the tree, leaf nodes are at level 0, and level increases as we move toward the root node.

4.2.1 Number of replicas (N) for each query

To satisfy the storage constraint, the number of replicas of LDD needs to be bounded. For this, we introduce the parameter N, which will be the same for each query and it can be calculated, based on the system's average storage capacity for each query. Let S_{avg} be the average size of the LDD per query type. For m base

stations and n query types,

$$S_{\text{avg}} = \sum_{i=1}^{n} \left(\sum_{j=1}^{m} S_{i,j}/m \right) /n. \tag{3}$$

Let $S_{\max}$ be the maximum memory available for storing replicas for each query, which is given to the system. Then the maximum number of replicas per query, N, will be calculated by

$$N = S_{\max}/S_{\text{avg}}. \tag{4}$$

4.2.2 Replication selectivity

The LDD corresponding to query Q_i should not be replicated at site j if the cost of replication exceeds the benefit of replication. Based on the cost model to minimise communication cost, we compute the local query-to-size ratio, $\text{LQSR}_{i,j}$. If $\text{LQSR}_{i,j}$ is smaller than some minimum threshold $C_{\min}$, then never replicate the LDD for query Q_i at database j. If $\text{LQSR}_{i,j}$ exceeds some maximum threshold $C_{\max}$, then always choose to replicate the LDD for query Q_i at database j, if constraints on N and L are satisfied. If $C_{\min} \leq \text{LQSR}_{i,j} \leq C_{\max}$, then the replication of the corresponding LDD will depend on the replication algorithm discussed in Sect. 4.3.

We now try to optimise the values of $C_{\min}$ and $C_{\max}$ in order to choose potential sites of replication in a tree-structured database, which could minimise NMH communication cost. Let b_l be the communication cost/delay of a lookup message that traverses one hop in the path from a leaf node j to the node hosting the LDDB for node j. Let bs be the communication cost of storing one megabyte of LDD. During replication, two possible cases may arise:

1. If Q_i's LDD is replicated at database j, then communication cost incurred due to the replica at j is $b_s \times S_{i,j}$.
2. If Q_i's profile is not replicated at database j, then communication cost incurred is only due to the lookup delay cost. The minimum lookup cost is when the LDD of query Q_i is available at the next higher level in the hierarchy, and this cost is $b_l \times N_{i,j}$. The worst lookup cost is the case when the LDD of query Q_i is only available at the LDDB node in the hierarchy, where the communication cost is $b_l \times L \times N_{i,j}$.

Replicating Q_i's profile at j always incurs less communication cost than not replicating if

$$\begin{aligned} b_l \times N_{i,j} &\geq b_s \times S_{i,j} \\ \Rightarrow N_{i,j}/S_{i,j} &\geq b_s/b_l \\ \Rightarrow \text{LQSR}_{i,j} &\geq b_s/b_l. \\ \text{Thus, we choose } C_{\max}^{\text{opt}} &= b_s/b_l \end{aligned} \tag{5}$$

$$\begin{aligned} b_l \times L \times N_{i,j} &\leq b_s \times S_{i,j} \\ \Rightarrow \text{LQSR}_{i,j} &\leq b_s/(b_l \times L). \\ \text{Thus, we choose } C_{\min}^{\text{opt}} &= b_s/(b_l \times L). \end{aligned} \tag{6}$$

4.2.3 Maximum level of replication

The maximum level of replication should be lower than level L at which the LDDB node is placed. This is because the LDDB node can process all continuous queries. Therefore, LDD replicated at any node at a level higher than L will not help to reduce communication cost.

4.3 Selection of nodes for replication

The replica allocation algorithm is performed over our proposed database model with estimated local query-to-size ratio (LQSR) information. We proceed in two phases.

For each query Q_i, replica allocation algorithm proceeds in two phases, as shown in Fig. 5a and b. In the first phase, it replicates the relevant LDD to all nodes j with $\mathrm{LQSR}_{i,j} \geq C_{\max}$. This replication is performed as long as the number of allocated replicas of LDD $n \leq N$. The replication in the first phase is done in

Phase 1

$n = 1$, $R(Q_i) \leftarrow \phi$
Compute $LQSR_{i,j}$ for each node j at level 0.
For each level k of hierarchy from 0 to L
For each $j \in D(K)$
  If $LQSR_{i,j} \geq C_{max}$
    $R(Q_i) \leftarrow R(Q_i) \cup \{j\}$
    $LQSR_{i,\,parent(j)} = (N_{i,\,parent(j)} - N_{i,j}) / (S_{i,\,parent(j)} - S_{i,j})$
    $n = n + 1$
If $n = N$ then exit

(a)

Phase 2

$T(Q_i) \leftarrow D(L)$
For each $j \in T(Q_i)$
  If $LQSR_{i,j} \leq C_{min}$
    $T(Q_i) \leftarrow T(Q_i) - \{j\}$
  Else $T(Q_i) \leftarrow T(Q_i) \cup children(j)$
$T(Q_i) = T(Q_i) - R(Q_i)$
Select into $R(Q_i)$ $N - n$ databases from $T(Q_i)$ with largest $LQSR$ value

(b)

Fig. 5 Phase 1 and phase 2 of the replication algorithm.

bottom-up traversal from level 0 to level L so that replicas are as close to the leaf-level nodes as possible to reduce the lookup latency assuming that leaf-level nodes have storage and query-processing capacity. As soon as node j is assigned a replica of LDD for query Q_i, j's parent's LQSR value for Q_i is updated in the following way:

$$\text{LQSR}_{i,\,\text{parent}(j)} = \big(N_{i,\,\text{parent}(j)} - N_{i,j}\big)\big/\big(S_{i,\,\text{parent}(j)} - S_{i,j}\big).$$

This is because query Q_i requested from nodes below j will be processed by node j.

If $n \leq N$ at the end of Phase 1, in Phase 2, the nodes below level L and with the largest nonnegative $\text{LQSR}_{i,j} - C_{\max}$ are chosen as the nodes for replication. The replication in Phase 2 proceeds in a top-down fashion (from level L). This way, we maximise the coverage area of replication. The Phase 1 and Phase 2 algorithms are shown in Fig. 5a and b.

Let $D(K)$ be the set of databases at level K. Let $R(Q_i)$ be the set of databases that should contain replicas of LDD corresponding to query Q_i. Let $T(Q_i)$ be the temporary set of databases used in the second phase for query Q_i.

5 Performance evaluation

We conducted a simulation experiment to obtain the NMH of the SP technique with respect to the other two placement techniques (RP and BP) when queries are single-ZQ and multiple-ZQ types. We also obtained NMH and average memory requirements for selective replication for different degrees of replication, when queries are only of single-ZQ type.

5.1 Simulation model

We performed a simulation on the tree architecture as discussed in Sect. 3. We developed a simulation program in Java to generate a realistic user query profile with most of the FULDCQs requested between the 6th and 20th hour of a 24-hour period as established by [15].

The assumptions underlying the design of the simulation model include the following:

- Most of the FULDCQs requested by mobile users are between the 6th and 20th hour of a 24-hour period because most users use their mobile phones during working hours.
- Mobile users' FULDCQs exhibit some temporal and spatial locality. In other words, a user tends to ask the same query over a period of time (e.g. finding hotels as he is driving), and different users tend to query the same objects (e.g. location attraction in the city, nearest restaurants), i.e. users follow certain query patterns.
- Mobile users make random moves based on an exponential distribution to simulate realistic user movement [15].
- During selective replication, we considered replication selectivity $C_{\min} = C_{\min}^{\text{opt}}$, and $C_{\max} = C_{\max}^{\text{opt}}$. We set $b_s/b_l = 1$, assuming that the communication cost

of storing one megabyte of LDD and communication cost/delay of a lookup message are equal.

For generating the user query profile, with m queries of FULDCQs types and n base stations, we created a dynamic input matrix, M. Each element $M[i, j]$ represents the number of FULDCQs of type Q_i requested from base station j over a given time period, T. We divided the 24-hour period into 12 time zones with every time zone being 2 hours. For each of these time zones, we recorded the input matrix M. Thus, we had 12 such input matrices, i.e. M^k for $k = 1, 2, \ldots, 12$. To determine NMH, we considered Q_i of either type single-ZQ or multiple-ZQ. Furthermore, while obtaining NMH and storage cost for selective replication, we considered Q_i of single-ZQ type only.

For calculating NMH of SP technique versus the other two placement techniques (RP and BP), where the queries are of single-ZQ or multiple-ZQ type, we defined a set NZ_j (using deterministic distribution), where each element of the set corresponds to the neighbouring zones covered by the query Q_j of type multiple-ZQ. For a given set NZ_j, we computed NMH for the three placement techniques: RP, BP and SP.

While determining the storage cost for selective replication for different degree of replications, when queries are only of single-ZQ type, we compute an input matrix, S, where each element $S[i, j]$ represents the total size of the LDD corresponding to query Q_i for base station j.

For calculating the total number of FULDCQs of type Q_i requested by mobile users from base station j over a given 24-hour period, we need to add up all the 12 input matrices M^k for $k = 1, 2, \ldots, 12$. The output matrix T represents this. Each element $T[i, j]$ is represented as:

Using output matrix T and input matrix S, we calculated the $\text{LQSR}_{i,j}$ over a given 24-hour period. The output matrix L represents this. Each element $L[i, j]$ is represented as $L[i, j] = T[i, j] / S[i, j]$. For internal nodes j, $\text{LQSR}_{i,j}$ is calculated using Eq. (2).

Now, for each query Q_i, we ran the replica allocation algorithm proceeding in two phases 1 and 2 (see Sect. 4.3) based on the LQSR information obtained earlier and replicating each query Q_i at the appropriate node j.

5.2 Experimental observation

Based on the assumptions underlying the design of the simulation model, we are proposing the following hypotheses:

Hypothesis 1 With queries of only multiple-ZQ types or mixed types, SP technique incurs less communication cost than other techniques BP or RP.
Hypothesis 2 With queries of only single-ZQ types, intelligent selective LDD-replication incurs less communication cost than having no replication.
Hypothesis 3 As the degree of replication increases, the communication cost decreases but the storage cost increases.

These hypotheses are guaranteed to hold in conditions matching those stated in the simulation model and are validated in the following experimental result section.

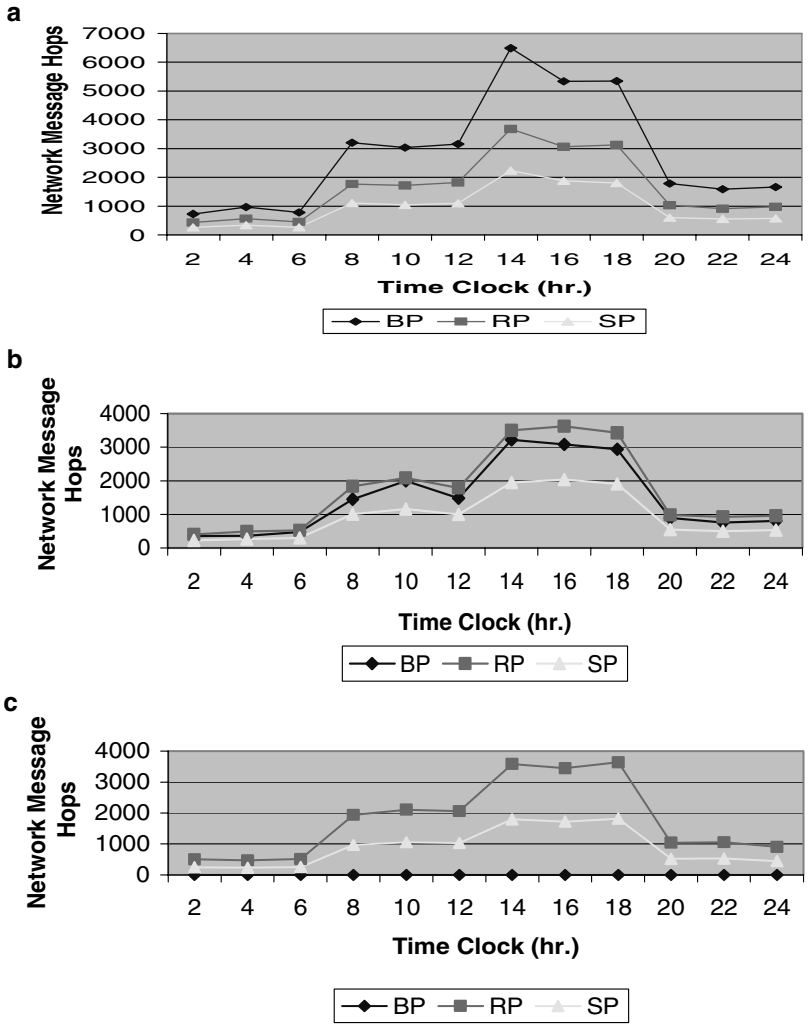

Fig. 6 Network message hops with 100%, **a** 50%, **b** 0%, **c** multiple-ZQ.

5.3 Experimental results

In Fig. 6, we compare the NMH for the three placement techniques RP, BP and SP. Distribution of query multiple-ZQ (single-ZQ) types is 100% (0%), 50% (50%) and 0% (100%) in Fig. 6a, b and c, respectively. In Fig. 6a and b, as we can see, RP and BP incur the higher NMH in a 24-hour duration while SP incurs the least NMH in this period. This confirms the superiority of selective placement (SP) technique over the other two placement techniques when the queries are mixed or only multiple-ZQ types, thus validating *Hypothesis 1*.

In Fig. 6c, when queries are only single-ZQ types, BP incurs minimum NMH and SP will be in the middle. This is because, for the BP case, the query result

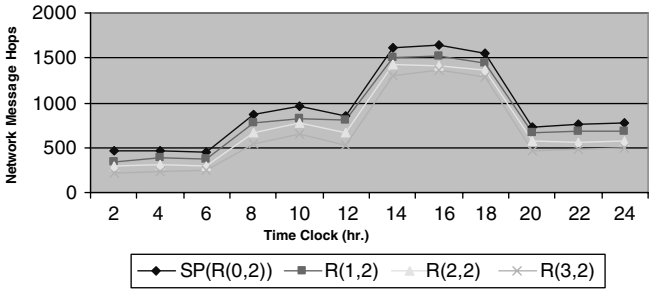

Fig. 7 NMH for different degrees of replication with no multiple-ZQ.

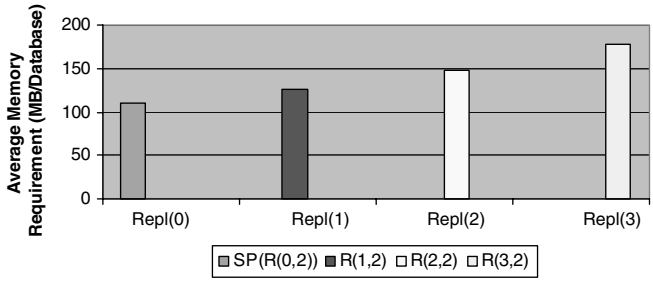

Fig. 8 Storage costs for different degrees of replication with no multiple-ZQ.

will be retrieved from the corresponding BP's LDDB. Thus, in BP for single-ZQ types, NMH will always be zero.

In Fig. 7, we only consider queries of single-ZQ type. In this figure, we compare the NMH for different degrees of replication in the context of SP over a 24-hoour duration. We can see that $R_{3,2}$, in which N, the number of replications, is 3, and L, the level of replication, is 2, has least NMH in this period as compared with others such as $R_{0,2}$, $R_{1,2}$ and $R_{2,2}$. We can see that $R_{0,2}$ with no replication incurs the highest NMH over $R_{1,2}$, $R_{2,2}$, and $R_{3,2}$ with selective replication thus validating *Hypothesis 2*.

In Fig. 8, we see the storage requirement for different degrees of replication. From Figs. 7 and 8, we observe that, as the degree of replication increases, the number of NMH decreases, but at the same time, the storage requirement for LDD replicas increases. Thus, $R_{3,2}$ incurs the minimum communication costs but requires the maximum memory, thus validating *Hypothesis 3*.

We observe that, when queries are only multiple-ZQ type or mixed, SP in a tree will be a suitable LDDB-placement strategy because the number of NMH will be reduced. When the queries are only of single-ZQ types, by exploiting different degrees of replication and query patterns, we replicate corresponding LDD in the context of SP to reduce the NMH. With this replication, SP causes the LDD replicas to be placed closer to the leaf nodes, i.e. near the base station, but at the same time the degree of replication in our tree architecture would be limited by the storage constraint.

Table 2 and 3 summarise the results of our simulation experiment. Table 2 presents the results in terms of NMH for various placement strategies with query

Table 2 NMH for different placement strategies.

	5.3.1 Network message hops		
Query distribution	BP	RP	SP
100% multiple-ZQ	High	Moderate	Low
50% multiple-ZQ	Moderate	High	Low
100% single-ZQ	Low	High	Moderate

Table 3 NMH and average memory requirement for SP with various degrees of replication.

	5.3.2 Network message hops			
Query distribution	SP (R(0, 2))	R (1, 2)	R (2, 2)	R (3, 2)
100% single-ZQ	High	<R (0, 2)	<R (1, 2)	Low
	5.3.3 Average memory requirement			
Query distribution	SP (R(0, 2))	R (1, 2)	R (2, 2)	R (3, 2)
100% single-ZQ	Low	>R (0, 2)	>R (1, 2)	High

types multiple-ZQ and single-ZQ. Table 3 presents the results in terms of NMH and average memory requirement for various degrees of replication in the context of SP with query type single-ZQ.

6 Related work

Most of the research in querying location-dependent information focuses on the data management issues of the mobile users and tracking the location of these users. We present related work in the context of centralised databases domain first and then the distributed databases domain.

In [28, 29], the moving object spatiotemporal (*MOST*) data model is proposed for managing the locations of mobile users and for the prediction of their future locations. According to the MOST model, the attributes of the moving data item can be static, that is, can be changed only when an explicit update is applied or *dynamic*, that is, changes over time according to a certain function. Also in [28, 29], *adaptive dead-reckoning* (*adr*) is proposed in which an update is generated to refresh the location of an object and redefine its location function whenever the deviation of its current location is greater than the last update by a threshold. In [28, 29], the main focus is on the problem of storing location information of mobile users in a centralised database with the goal of processing spatial and temporal queries in an efficient way. In our proposed tree framework, the location information of mobile users is not centralised but distributed across the nodes of the tree. Here each node manages location information of mobile users within the set of the base stations in its subtree. Compared with the centralised approach, our distributed framework will scale up better when the number of mobile users increases.

Seydim et al. [26] propose the location-dependent service manager (*LDSM*) architecture, between the mobile user and the location-dependent services (*LDS*), to service location-dependent applications that access location-dependent data. The proposed middleware does location translation, adjusts the location granularities

and gets query results with the help of LDS. In this approach, LDSM acquires location information of mobile users from location services or a global positioning system (*GPS*)-based solution. In our approach, the LDDB node, which maintains a LDDB and processes FULDCQ for mobile users' within the set of the base stations in its subtree acts as LDSM, which acquires the location information of mobile users from the ULDB present within that node. Unlike centralised LDSM architecture, our framework that places LDDB nodes intelligently at selective places in the tree will scale up better when the number of mobile users increases.

Ilarri et al. [10] propose a solution to the location-dependent query processing problem in which location information of mobile users is distributed across several base stations; each base station manages location information about mobile users under its coverage area. But in this approach, mobile agents are used to support distributed query processing, track interesting mobile users and optimise wireless communication transmissions. In this approach, all the query processing is performed at the base station, which might burden the base station with data-management activities, and also this approach will not scale up well for FULDCQ related to multiple-ZQ. In contrast, in our approach, the LDDB node, which maintains a LDDB and processes FULDCQ for mobile users' within the set of the base stations in its subtree, can be located in the tree either at root node, internal nodes or leaf nodes (base stations). Also, our framework is flexible enough to efficiently handle FULDCQ related to single-ZQ or multiple-ZQ, as discussed in Sect. 3.6.

In [16], we propose a basic framework for handling location-dependent continuous queries but had assumptions that FULDCQ are only of single-ZQ types by ignoring multiple-ZQ types.

7 Conclusion and future work

We have proposed a potentially powerful and novel approach for handling FULDCQ in a mobile environment. This approach relies on a tree architecture that is flexible enough to handle queries of either single-ZQ or multiple-ZQ types. We have discussed various placement strategies to store LDD in the tree structure and showed that selective placement outperformed other strategies for queries of multiple-ZQ or mixed types. In addition, when queries are only of single-ZQ types, by exploiting different degrees of replication and query patterns with specified storage constraint, we replicate corresponding LDD in the context of SP to reduce the network message hops. With this replication, SP causes the LDD replicas to be placed closer to the base station intelligently by considering storage constraint at the same time.

We would like to extend the work in the following directions. First, our experimental results need to be further verified under Real-life scenarios with mobile users' query profiles being obtained from the real mobile network. This is because our experimental results are only guaranteed to be valid in conditions matching those stated in the simulation model. Second, we would also like to consider *FULDQ* types, where the objects related to the query are moving. Third, we would like to include users' mobility pattern [4, 30, 32, 34] and caching techniques [2, 8, 14] in LDD replication. Fourth, we would also like to investigate partitioning techniques to split LDDBs at higher level nodes into small databases at the lower level nodes [30]. Finally, we want to investigate various techniques

to manage ULDB based on the location management techniques as discussed in [33, 35].

Acknowledgements We thank anonymous reviewers for their very useful comments and suggestions. This work was supported in part by a gift from Sun and the National Science Foundation grant NGS-0103709.

References

1. Acharya S, Franklin M, Zdonik S (1997) Balancing push and pull for data broadcast. In: Proceedings of ACM SIGMOD, pp 183–194 Tucson, AZ
2. Alonso R, Barbara D, Molina HG (1990) Data caching issues in an information retrieval system. ACM Trans Database Syst **15**(3), 359–384
3. Badrinath BR, Imielinski T, Virmani A (1992) Locating strategies for personal communications networks. In: Proceedings of the 1992 International Conference on Networks for Personal Communications
4. Badrinath BR, Imielinski T (1992) Replication and mobility. In: Second Workshop on the Management of Replicated Data. IEEE, pp 9–12 Monterey, California, USA
5. Dunham M, Kumar V (1998) Location dependent data and its management in mobile databases. In: Proceedings of the 9th International Workshop on DEXA, pp 414–419 Vienna, Austria
6. Faiz M, Zaslavsky A (1995) Database replica management strategies in multi-database systems with mobile hosts. In: Proceedings of the 6th International Hong Kong Computer Society Database Workshop: Database Reengineering and Interoperatibility. Hong Kong
7. Gok HG, Ulusoy O (2000) Transmission of continuous query results in mobile computing systems. Information Sciences. **125**(1–4) pp. 37–63
8. Harjono H, Jain R, Mohan S (1994) Analysis and simulation of a cache-based auxiliary location strategy for PCS. In: IEEE Conference on Networks and Personal Communications
9. Huang Y, Sistla P, Wolfson O (1994) Data replication for mobile computing. In: Proceedings of 1994 ACM SIGMOD, pp 13–24
10. Ilarri S, Mena E, Illarramendi A (2003) Dealing with continuous location-dependent queries: just-in-time data refreshment. In: Proceedings of the International Conference on Pervasive Computing and Communications (PerCom'03). Dallas, FortWorth, Texas, USA
11. Imielinski T, Badrinath BR (1992) Querying in highly mobile distributed environments. In: Proceedings of the International Conference on Very Large Data Bases (VLDB 92). Vancouver, Canada
12. Imielinksi T, Badrinath BR, Virmani A (1992) Location strategies for personal communication networks. In: IEEE Globecon92 Workshop on Networking for Personal Communication Applications. Orlando, Florida, USA
13. Jain R (1996) Reducing traffic impacts of PCS using hierarchical user location databases. In: Proceedings of IEEE International Conference on Communications. Dallas, Texas, USA
14. Jain R, Lin YB, Lo C, Mohan S (1994) A caching strategy to reduce network impacts of PCS. IEEE J Selected Areas Commun **12**(8), 1434–1444
15. Jannink J, Lam D, Shivakumar N, Widom J, Cox DC (1997) Efficient and flexible location management techniques for wireless communication systems. ACM/Baltzer J Mobile Netw Appl **3**(5), 361–374
16. Khan L, Gupta M (2002) Effective management of location dependent continuous queries in mobile environment. In: Proceedings of DEXA Fifth International Workshop on Mobility in Databases and Distributed Systems (MDDS), pp 705–709. France
17. Krishna P, Vaidya NH, Pradhan DK (1994) Location management in distributed mobile environments. In: Proceedings of Third International Conference on Parallel and Distributed Information Systems, pp 81–88. Austin, Texas
18. Kumar V, Dunham MH (1998) Defining location data dependency, transaction mobility and commitment. Technical Report 98-CSE-1, Southern Methodist University
19. Lin YB, Devries SK (1995) PCS network signaling using SS7. In: IEEE Pers Commun **2**(3), 44–55

20. Madria S, Bhargava BK, Pitoura E, Kumar V (2000) Data organization issues for location-dependent queries in mobile computing. In: Proceedings of ADBIS-DASFAA. Prague, Czech Republic
21. Mohan S, Jain R (1994) Two user location strategies for personal communication services. IEEE Pers Commun **1**(1), 42–50
22. Perkins CE (1998) Mobile IP: Design Principles and Practices. Addison Wesley, Reading, MA
23. Pitoura E, Samaras G (2000) Locating objects in mobile computing. IEEE Trans Knowledge Data Eng **13**(4), 571–592
24. Ratner DH (1998) Roam: a scalable replication system for mobile and distributed computing. Ph.D. Thesis, Computer Science Department, University of California, Los Angeles
25. Ratner D, Reiher P, Popek G (1997) Replication requirements in mobile environments. Technical Report 970021, Department of Computer Science, University of California, Los Angeles
26. Seydim AY, Dunham MH, Kumar V (2001) An architecture for location dependent query processing. In: DEXA Workshop, Munich, Germany, pp. 549–555
27. Sistla AP, Wolfson O (1995) Temporal triggers in active databases. IEEE Trans Knowledge Data Eng **7**(3), 471–486
28. Sistla AP, Wolfson O, Chamberlain S, Dao S (1997) Modeling and querying moving objects. In: Proceedings of the 13th International Conference on Data Engineering (ICDE 97). Birmingham, UK, pp 422–432
29. Sistla AP, Wolfson O, Chamberlain S, Dao S (1998) Querying the uncertain position of moving objects. In: Temporal Database: Research and Practice. Lecture notes in computer science, pp 310–337. Springer, Berlin Heidelberg New York
30. Van Steen M, Hauck FJ, Homburg P, Tanenbaum AS (1998) Locating objects in wide-area systems. IEEE Commun Mag **36**, 104–109
31. Wang JZ (1993) A fully distributed location registration strategy for universal personal communication systems. IEEE J Selected Area Commun **11**(6), 850–860
32. Wang DCC (1994) A survey of number mobility techniques for PCS. In: International Conference on Universal Personal Communications, ICUPC '94, New Orleans, USA, pp 340–344
33. Wolfson O, Chamberlain S, Dao S, Jiang L (1997) Location management in moving objects databases. WOSBIS'97, pp 7–13. Budapest, Hungary
34. Wolfson O, Xu B, Chamberlain S, Jiang L (1998) Moving objects databases: issues and solutions. In: Proceedings of the 10th International Conference on Scientific and Statistical Database Management, Capri, Italy
35. Wolfson O, Sistla P, Chamberlain S, Yesha Y (1999) Updating and querying databases that track mobile units. Distributed Parallel Databases **7**(3), 257–287

Manish Gupta received his B.E. degree in Electrical Engineering from Govindram Sakseria Institute of Technology & Sciences, India, in 1997 and his M.S. degree in Computer Science from University of Texas at Dallas in 2002. He is currently working toward his Ph.D. degree in the Department of Computer Science at University of Texas at Dallas. His current research focuses on AI-based software synthesis and testing. His other research interests include mobile computing, aspect-oriented programming and model checking.

Manghui Tu received a Bachelor degree of Science from Wuhan University, P.R. China, in 1996, and a Master's Degree in Computer Science from the University of Texas at Dallas 2001. He is currently working toward the Ph.D. degree in the Department of Computer Science at the University of Texas at Dallas. Mr. Tu's research interests include distributed systems, wireless communications, mobile computing, and reliability and performance analysis. His Ph.D. research work focuses on the dependent and secure data replication and placement issues in network-centric systems.

Latifur R. Khan has been an Assistant Professor of Computer Science department at University of Texas at Dallas since September 2000. He received his Ph.D. and M.S. degrees in Computer Science from University of Southern California (USC) in August 2000 and December 1996, respectively. He obtained his B.Sc. degree in Computer Science and Engineering from Bangladesh University of Engineering and Technology, Dhaka, Bangladesh, in November of 1993. Professor Khan is currently supported by grants from the National Science Foundation (NSF), Texas Instruments, Alcatel, USA, and has been awarded the Sun Equipment Grant. Dr. Khan has more than 50 articles, book chapters and conference papers focusing in the areas of database systems, multimedia information management and data mining in bio-informatics and intrusion detection. Professor Khan has also served as a referee for database journals, conferences (e.g. IEEE TKDE, KAIS, ADL, VLDB) and he is currently serving as a program committee member for the 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD2005), ACM 14th Conference on Information and Knowledge Management (CIKM 2005), International Conference on Database and Expert Systems Applications DEXA 2005 and International Conference on Cooperative Information Systems (CoopIS 2005), and is program chair of ACM SIGKDD International Workshop on Multimedia Data Mining, 2004.

Farokh Bastani received the B.Tech. degree in Electrical Engineering from the Indian Institute of Technology, Bombay, and the M.S. and Ph.D. degrees in Computer Science from the University of California, Berkeley. He is currently a Professor of Computer Science at the University of Texas at Dallas. Dr. Bastani's research interests include various aspects of the ultrahigh dependable systems, especially automated software synthesis and testing, embedded real-time process-control and telecommunications systems and high-assurance systems engineering.

Dr. Bastani was the Editor-in-Chief of the IEEE Transactions on Knowledge and Data Engineering (IEEE-TKDE). He is currently an emeritus EIC of IEEE-TKDE and is on the editorial board of the International Journal of Artificial Intelligence Tools, the International Journal of Knowledge and Information Systems and the Springer-Verlag series on Knowledge and Information Management. He was the program cochair of the 1997 IEEE Symposium on Reliable Distributed Systems, 1998 IEEE International Symposium on Software Reliability Engineering, 1999 IEEE Knowledge and Data Engineering Workshop, 1999 International Symposium on Autonomous Decentralised Systems, and the program chair of the 1995 IEEE International Conference on Tools with Artificial Intelligence. He has been on the program and steering committees of several conferences and workshops and on the editorial boards of the IEEE Transactions on Software Engineering, IEEE Transactions on Knowledge and Data Engineering and the Oxford University Press High Integrity Systems Journal.

I-Ling Yen received her B.S. degree from Tsing-Hua University, Taiwan, and her M.S. and Ph.D. degrees in Computer Science from the University of Houston. She is currently an Associate Professor of Computer Science at University of Texas at Dallas. Dr. Yen's research interests

include fault-tolerant computing, security systems and algorithms, distributed systems, Internet technologies, E-commerce and self-stabilising systems. She has published over 100 technical papers in these research areas and received many research awards from NSF, DOD, NASA and several industry companies. She has served as Program Committee member for many conferences and Program Chair/Cochair for the IEEE Symposium on Application-Specific Software and System Engineering & Technology, IEEE High Assurance Systems Engineering Symposium, IEEE International Computer Software and Applications Conference, and IEEE International Symposium on Autonomous Decentralized Systems. She has also served as a guest editor for a theme issue of IEEE Computer devoted to high-assurance systems.