1999

Data Organization Issues for Location-dependent Queries in Mobile Computing

Sunjay Kumar Madria

Bharat Bhargava
Purdue University, bb@cs.purdue.edu

Evaggelia Pitoura

Vijay Kumar

Report Number:
99-038

DATA ORGANIZATION ISSUES FOR LOCATION-DEPENDENT QUERIES IN MOBILE COMPUTING

Sanjay Kumar Madria
Bharat Bhargava
Evaggelia Pitoura
Vijay Kumar

CSD TR #99-038
November 1999

Data Organization Issues for Location-dependent Queries in Mobile Computing[1]

Sanjay Kumar Madria[1], Bharat Bhargava[1] , Evaggelia Pitoura[2] and Vijay Kumar[3]

[1]Department of Computer Science, Purdue University, West Lafayette, IN 47907. {skm,bb}cs.purdue.edu

[2]Computer Science Department, University of Ioannina, GR 45110, Greece, pitoura@cs.uoi.gr

[3]Department of Computer Networking, University of Missouri-Kansas City, MO 64110, kumar@cstp.umkc.edu

Abstract

We consider queries which originate from a mobile unit and whose result depends on the location of the user who initiates the query. Examples of such queries are: find the nearest service provider, what are the traffic conditions on a route. We execute such queries based on location-dependent data involved in their processing. To facilitate location-dependent query processing, we build concept hierarchies [3] based on location. These hierarchies define mapping among different granularities of locations. One such hierarchy is to generate domain knowledge about the cities that belong to a state. The hierarchies are also used as distributed directories to assist in finding the database or relation that contains the values of the location-dependent attribute in a particular location. We extend concept hierarchies to include spatial indexes on the location-dependent attributes. Finally, we discuss how to partition and replicate relations based on the location to process the queries efficiently. We briefly discuss the implementation issues.

1. Introduction

Advances in wireless communication technology make it possible to realize a data processing paradigm that eliminates geographical constraints from data processing activities. In this environment, mobile users (mobile units) are not attached to a fixed location all the time. Instead their point of connection to the network changes as they move. A mobile unit is connected to a server, which manages the data processing activities in a well-defined region. When a unit moves out of one region, it gets connected to the server of a new region. The attachment to different servers is handled in a way that the mobile unit gets continuous service while moving. We envision a ubiquitous data processing system, which gives the impression that the desired data is available in the vicinity of a processing unit and can be accessed at anytime from anywhere.

The data processing mechanism in mobile environment is not fundamentally different from the conventional system [11]. However, the freedom of geographical mobility while processing information gives rise to a number of interesting and challenging problems, which can be categorized into application and system problems. In the application domain, one faces the problem of developing query structure and

[1] This research is partially supported by NSF under grant number 9805693-EIA.

their processing, management of location-dependent data [1], accessing desired data, etc. Other problems concern the consistency, serialization, system recovery and security.

We present research ideas for processing queries that deal with *location-dependent data* [1]. Such queries we refer as location-dependent. The objective is get the right data at different locations for processing a given query. The results returned in response to such queries should satisfy the location constraints with respect to the point of query origin, where the results are received, etc. We propose to build additional capabilities into the existing database systems to handle location-dependent data and queries.

We present some examples to recognize the problems of accessing correct data when point of contact changes. Data may represent SSN of a person, or maiden name, or sales tax of a city. In one representation the mapping of the data value and the object it represents is not subjected to any location constraints. For example, the value of SSN of a person remains the same no matter from which location it is accessed. This is not true in the case of sales tax data. The value of the sales tax depends on the place where sales query is executed. For example, sales tax value of West Lafayette is governed by a different set of criteria than the sales tax of Boston. We can therefore, identify the type of data whose value depends on the set of criteria established by the location and another type of data, which is not subject to the constraints of a location. There is a third type of data that is sensitive to the point of query. We illustrate this data with the following example. Consider a commuter who is travelling in a taxi initiates a query on his laptop to find the nearby hotels in the area of its current location. The answer to this query depends on the location of the origin of the query. Since the commuter is moving he may receive the result at a different location. Thus, the query results should correspond to the location where the result is received or to the point of the origin of the query. The difference in the two correct answers to the query depends on the location and not on the hotel. The answer to the query "find the cheapest hotel" is not affected by the movement. The former depends on the location where as later on the object characteristics.

Our focus in this paper is to handle location-dependent data and to execute queries processing them. To manipulate this kind of data we build *concept hierarchies* (based on location) and extend them to include spatial indexes on the location-dependent attributes. For efficient query processing, we discuss the horizontal and vertical partitions and replication of relations based on location. We discuss how location-dependent data can be grouped or summarized. The partition tables and summarized relations can be cached at mobile unit so that location-dependent queries can be also processed locally.

The research on predicting, storing and querying the location of mobile objects has been discussed in [4,5,6,7,8]. Location-dependent data considered in this paper are stationary. They do not correspond to moving objects [4,5,6]. In moving object research, objects are constantly moving and their location related data is constantly being updated in the database. We assume here that location information about a mobile host is found with the help of a Global Positioning System (GPS) in conjunction with some of the methods used to predict future locations [5,9]. For example, Omnitracs developed by Qualcomm [10] is a

commercial system used by transportation industry which provides location management by connecting vehicles such as trucks, via satellites, to company databases.

The research questions we addressed in this paper are: (a) How to represent location-dependent data efficiently?, (b) Do we need to represent location differently or could it be represented in the schema?, (c) How to handle database partition and replication in the presence of location-dependent data?, (d) How to create index data using location and manage them for efficient query processing?

The remainder of this paper is structured as follows. In section 2, we discuss mobile database system architecture. Section 3 defines location-dependent data and query processing. We present our motivating example in section 4. We discuss concept hierarchies in section 5. Location-dependent indexing and summarization has been described in section 6. Section 7 presents our discussion on replication and partitioning issues. We conclude in Section 8.

2. Mobile Database System Architecture

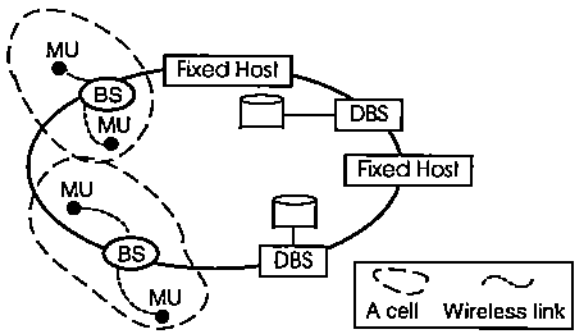

Figure 1. Architecture of MDS

Figure 1 presents a general architecture of the Mobile Database System (MDS). A set of general purpose computers (PC, workstations, etc.) are interconnected through a high speed wired network. These computers are categorized into *Fixed hosts* (FH) and *Base Stations* (BS) or *mobile support stations* (MSS). A number of laptop computers, referred to as *Mobile Hosts* (MH) or *Mobile Units* (MU) are connected to the wired network components only through BSs via wireless channels. MUs are battery powered portable computers, which move around freely in a restricted area, which we refer to as the "geographical mobility domain" (G). In Figure 1, G is the total area covered by all BSs. This size restriction on their mobility is due to the limited bandwidth of wireless communication channels.

To support the mobility of MUs and to exploit frequency reuse, the entire G is divided into smaller areas called cells. Each cell is managed by a particular BS. The mobile discipline requires that a MU must have unrestricted movement within G (inter-cell movement) and must be able to access desired data from

any cell. The process of crossing a cell boundary by a MU and entering into another cell is referred to as a handoff. The entire process of handoff is transparent to a MU and is responsible for maintaining end-to-end data movement connectivity.

A database server (DBS) can either be installed at BSs or can be a part of FHs or can be a separate from BS or FH. We make the following assumptions about the functionality of DBSs and MUs:

- A DBS is an independent data processing node on the wired network. There are more than one DBS on a mobile database system.
- A DBS communicates with a MU only through its BS.
- A DBS can be regarded as a multidatabase node and the database is not fully redundant.
- Queries may originate at a DBS or they may come from MUs through BSs.
- Each MU caches some portion of the database and has some database functionality. Thus, it can update its cache, can process a part of a query, or can send its updates to a DBS through BS. We assume that some caching scheme is in place, which maintains cache consistency.

In our MDS architecture, the *location-dependent* query arrives at MU and are processed with the help of MU, BS and DBS.

3. Location-Dependent Data

A location is a geographical area. Location may be expressed with various granularities. For instance it may be specified as a longitude-latitude pair, a city, a country or a region covered by a cell or a group of cells when referring to the cellular architecture in wireless communications. A data item is location-dependent when it takes on different values based on the location. For example, sale-tax rate is location-dependent while the authors of a book are not.

Definition 1. Location-dependent data - data whose value varies with location.

A location-dependent data item may have some value **a** in region **A** and some other value **b** in another region **B** at the same time. Both values are correct in their respective regions and represent the same data object. The value **b** may be related to value **a** by some functional mapping which may depend on factors such as the distance between the two regions or the two values may be independent. We assume that values in a location remain the same unless explicitly updated. We have not considered items whose value changes continuously with time.

Definition 2. Location attribute – an attribute whose domain consists of locations.

An example of a location attribute is the Address attribute in the relation shown in Figure 1. A location attribute may be *explicit* in the relation. The location attribute may be explicitly specified in the database scheme. However, there are cases in which there is no explicit location attribute, instead the

location attribute is *implicitly* determined from the context. For example an implicit location attribute may be the location at which the database is stored. Such implicit attributes may be included as metadata or as part of an auxiliary schema.

Definition 3. Location-dependent attribute – an attribute whose value is associated with a location.

For example, both the Per day charges and the Tax attribute in the database instance in Figure 1 change based on the location attribute Location-address. A location-dependent attribute does not necessarily have different values at all different locations.

Definition 4 (a). Location-dependent query (type A) - A query Q is called location-dependent if its result depends on the location at which the query is originated.

Definition 4(b). Location-dependent query (type B) – A query Q which process location-dependent data is also called location-dependent query. Location information may be in the form of longitude and latitude or in some other spatial parameter. The location information may be implicit or explicit in the query.

The query "*How many people with salary>1000000 living at present in Kansas City*?" can be regarded as location-dependent because it must access the population of Kansas City. On the other hand "*List the SSN of people who are older than 50 years*", is not a location-dependent query even if the database containing this information is located in Kansas City.

We propose approaches for processing location-dependent data and queries under the following possible cases:

- Location information is explicit in the query. For example, *"What is the humidity and temperature in Kansas City today?"* The location Kansas City may be presented in the *where* clause of SQL-syntax like "*where location = "Kansas City"*. The query must use Kansas City weather data and the result will be location specific. It is possible that the mobile unit could be in a different region when it receives the answer. Still the information is correct and can be used.
- Location attribute is not explicit in the query, and query is evaluated for the location where it originated. The system can identify the implicit location by using longitude and latitude information, which can be available through GPS. There could be some other ways of identifying the location information such as the use of location directory or prediction with the help of the users past behavior. However, the problem with this answer is that the location information is not deterministic. Thus the validity of this result cannot be accepted at a location with different longitude and latitude values.
- Location is not explicit and the query is evaluated such that the results correspond to the current location where the user receives them. For example, suppose a salesman driving on a highway initiates

a query "*Which is the nearest hotel?*" on his laptop. If the query is processed related to the current location of the salesman, then there may never be a precise answer, because by the time the distance is computed and relayed to the salesman, he might have changed his longitude and latitude. The value will be approximate and could be acceptable in many cases. This processing mode, however, will require continuous monitoring of the mobile unit to identify its next location where the query result will be dispatched.

- Location is not explicit and the query is evaluated at the time the query arrives at the database server. In this case, there are two issues. First, the query results reflect the current position of the user at the time query is evaluated. Second, the results reflect the current location associated with the location-dependent data. The first case is similar to the previous case; the query is evaluated with respect to the future location where the results will be received. However, the second case is different where the location information reflects the current state of the database. For example, the query "what is the destination of taxi with registration number "CJ 213" will be evaluated with respect to its destination location at the time query is executed. However, when the results arrived, the target destination may change for that taxi in the database.

In all the cases, the results returned to the user with respect to reference location (location used in the evaluation of the query) may be different but consistent and correct.

4. Location-dependent Query Processing - Motivating Example

Consider the following relational schema: Hotels (Hotel-name, Location-address, Per-day charges, Tax, Distance (from airport)). The relation is shown in Figure 1.

Hotel-name	Location-address	Per day charges	Tax (%)	Rebate(%) (Oct.- March)	Distance (from- airport)
Holiday-Inn	San Jose, CA	110	5	10	5
Holiday-Inn	Palo Alto, CA	110	5	10	10
Holiday-Inn	Berkeley, CA	110	5	10	30
Holiday-Inn	Dallas, TX	100	7	0	40
Ramada-Inn	San Jose, CA	110	5	10	30
Ramada-Inn	Houston, TX	100	7	0	40

Figure 1 : Hotels

In the relation in Figure 1, there are two location-dependent attributes: Per day charges and Tax. They both depend on the location attribute Location-address. They depend indirectly on the Distance (from-airport) location attribute, since the distance from the airport can be computed from the Location-

address attribute. Since the location attributes Location-address and Distance as part of the scheme and so they are explicit location attributes.

We consider SQL-type location-dependent queries that include a location attribute in the where clause as well as queries that do not include such an attribute. Consider the following query:

Find the hotels in the area of California. This query can be translated into the following SQL-syntax

Q1 Select *Hotel-name, Location-address* from *Hotels* where *Location-address* like *"CA";*

This is an example of a query that includes a location-attribute namely the Location-address attribute. This query will return all tuples that contain "CA" in the Location-address in the relation Hotel in Figure 1. However, treating this as a location-dependent query, we expect to retrieve results depending on the location of the mobile user who posed the query. Assume that the user is willing to drive only k miles form its current position. Such user preferences may be a part of a user profile that maintains personalized information for each user. For example, lets assume that the user issues the query from the airport in CA, k = 12 and that the reference location is the location of the user when posing the query. Then, the first two tuples are returned to the user. If instead we consider as reference location the location of the user when it receives the result of its query, then different tuples may be returned since the user may have moved a few miles away form the airport.

In query Q1, one of the location attributes Location-address is part of the query whereas the other location attribute Distance does not appear in the query. In order to process such a query, the database needs another layer which can map the query with the help of the user movement information captured for example by a GPS system to another query like Q2.

Q2 Select *Hotel-name, Location-address* from *Hotels* where *Location-address* like *"CA"* and distance from airport <= 12;

Another realistic query may not include a location attribute but queries for all hotels close to the user: *Find the hotel addresses in my area.* The equivalent query can be represented in SQL-syntax as follows:

Q3 Select *Hotel-name, Location-address* from Hotels where *location-attribute* = ?;

The database system must complete this query with information about the location of the user before executing it.

Another issue arises if different notations are used for describing locations. Consider that the database does not contain the exact location of state such as CA, but suburbs such as Palo Alto or Berkeley. In that case, to execute the query Q1, the database system must first find which locations are in the CA state and then use the distance from the point of the origin of the query to find the exact location of the hotel.

Yet another example of the issues arising due to differences in describing location is the following query:

Find the hotels in Palo Alto area. The equivalent query in SQL-syntax can be as follows:

Q4 Select *Hotel-name, Location-address* from Hotels where Location-address ="Palo Alto";

If query Q4 is executed using the relation **Hotels**, it will not return any hotels, as it does not match any of the tuples.

Yet another possibility is that the location attribute Address contains only CA. In this case, tuples in the relation Hotels can be grouped and replaced by one single tuple containing CA in the Location-address field (see Figure 3).

5. Concept Hierarchies

Concept hierarchies define a sequence of mappings from a set of lower-level concepts to their higher level correspondences [3,12,13] resulting in a hierarchy of concepts. In other words, concept hierarchies provide a set of predefined hierarchical relationships that generalize lower layer (i.e., primitive data) information to high layer. For example, a set {tennis, rugby, hockey, football} can be generalized as "sports" at a high level concept. A concept hierarchy can be defined on one or on a set of attribute domains, for example the Location-address attribute in Figure 1.

5.1 Location-based Concept Hierarchies

Concept hierarchies are in general data or application specific since they define mapping rules between different levels of concepts. The mapping of a concept hierarchy or some portion of it may also be provided explicitly by a knowledge engineer or a domain expert. Many different concept hierarchies can be constructed based on different view points or users preferences. However, usually, a common concept hierarchy can be associated with an attribute. In most database system implementations, it would be possible for a set of relatively stable and standard concept hierarchies to be made available as a common reference by all the databases. In [3], an automatic and dynamic generation of concept hierarchies is given.

The location based concept hierarchies is delimited as follows:

- Only set-valued domains of location attribute are considered. However, we believe that the method is appropriate for continuous domains, and therefore suitable for numeric location data.
- It is assumed that set-valued domains of attributes are simple structure-valued domains containing only discrete location values.
- The relational databases are summarized by using the location attribute's values at higher or lower level concepts. Concept hierarchy can have any level of depth based on granularities of location.
- There is no common region among two different locations. Thus, concept hierarchies are represented as trees.

Concept hierarchies can be generated among location attribute domains for the following:

- Within a domain itself, there may exist a concept hierarchy among the values in the location attribute. For example, a domain may be successively refined into more specialized location domain values. An attribute may take its value either from the specialized (leaf) values or from higher level descriptions.
- A concept hierarchy can be defined among location attributes using domains of attributes of a relation. For example, a concept hierarchy can relate attributes «City1» and «City2» with "State" at the root (since a state can have many cities). .
- There may exist a concept hierarchy among domains of location attributes of different relations. For example, consider attributes "city" and "state" which may appear in two different relations. In that case, we can define a concept hierarchy across these two relations where "state" appears at higher level and "city" at lower level.

5.2. Constructing Concept Hierarchies

Concept hierarchies are constructed for each of the location attributes across all relations based on the domain values of those location attributes. The granularity of the concepts in the hierarchy may vary from that of the values of the domain of each individual attribute. For instance even if the domain of an attribute is a set of street addresses, the hierarchy may include only streets. Domain experts can be consulted to ensure that the hierarchies are complete and correct.

In general, concept hierarchies of domain values of location attributes are updated infrequently. Since they are constructed based on domain values, there is no need to incrementally update such hierarchies when:

- new values are inserted into a relation, or
- tuples are deleted from a relation.

Concept hierarchies need to be updated when attribute domains change or there is a need to include location attributes of new databases. For example, the introduction of a database that describes locations in Europe necessitates that the concept hierarchy of Figure 2 is extended to include locations in Europe. Note that one may construct concept hierarchies involving only instances of domain values that occur in a given relation instance. In this case, the insertion or deletion of tuples may cause the update of the concept hierarchy.

6. Location-based Indexing and Summarization

Consider the concept hierarchy (CH) for each state in USA with the root as USA.

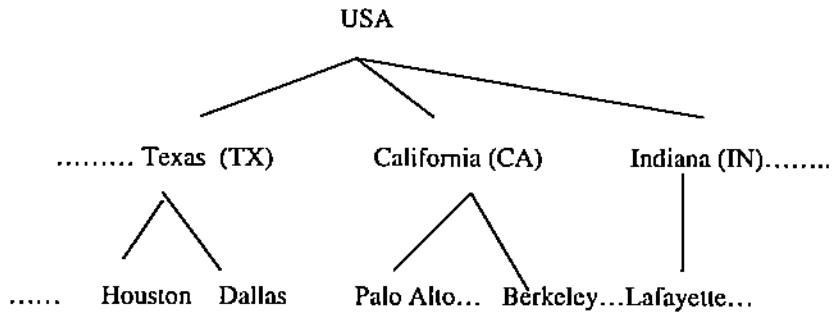

Figure 2 : Concept Hierarchy

Concept hierarchies are used to translate between different granularities used to represent locations. For instance, the concept hierarchy in Figure 2 can be used to determine that Palo Alto is a city in CA. Concept hierarchies can also be used in conjunction with spatial indexes and for summarizing location information.

6. 1 Building Indexes

Concept hierarchies can be used as directories for direct processing using the appropriate database that includes values based on the reference location of a query. For instance, when a user issues a query similar to Q3 while the user is in California, the concept hierarchy can be used to direct to the Hotel relation that includes hotels in the California region.

In particular, we propose extending concept hierarchies to include information about the location of the database/relations that include location attributes. We build indexes for each location attribute in a relation. Such indexes may be spatial indexes, such as R^+ trees. Nodes in the concept hierarchy point to the appropriate node of the spatial index. For instance, assume that we build a spatial index for the Address-Location attribute of the Hotel relation. In this case, the California node of the CH will include an entry (Hotel, Pointer) where Pointer points to the node of the spatial index on the Address-location attribute whose subnodes cover hotels in California.

In the case of a location-dependent query, the location of a moving user is commonly given as a pair of (latitude, longitude) values. This pair is mapped to the appropriate leaf of the concept hierarchy. In particular:

(a) If the reference location is the location of the user at the time the query is posed, then before sending the query for evaluation, the location of the user is estimated.

(b) If the reference location is the location of the user at the time the query arrives, then the estimation of the position of the user is initiated at the site at the static network that receives the query.

(c) If the reference location is the location of the user when the result arrives at its MH, then the estimation of the location is again performed at the static site. However, estimating the location is now an involved procedure since this is a future location. This computation must also account for the time to transmit the result.

Once the location of the user is determined, this location is mapped to a leaf in the CH. For example, if the location is inside John Hopkins University, the location is mapped to the leaf "Berkeley". Then, we move up the CH tree until we find an entry for the requested relation. If the requested relation is the Hotel relation, then we end up at the node with label CA. This node points to the appropriate entries in the Hotel relation.

6.2 Location-based Summarization

Another advantage of concept hierarchies is that the database relations can be condensed or summarized [13] using the location attributes. For example, suppose that the address field in the relation in Figure 1 contains no state such as CA and TX. This relation is summarized as follows using the CH in Figure 2. The Distance attribute now basically represents the aggregate operator maximum. It is applied here to the set of locations contained in the column and computes a new distance.

Hotel-name	Location-address	Per day charges	Tax (%)	Rebate(%) (Oct.-March)	Distance (from air-port)
Holiday-inn	CA	110	5	10	30
Holiday-Inn	TX	110	7	0	20
Ramada-Inn	CA	110	5	10	40
Ramada-Inn	TX	110	7	0	30

Figure 3 : Hotel 1: Summarized Relation Hotels

The above relation (Figure 3) can be summarized as given below (Figure 4) using an aggregation operator like max for numeric values and conceptual summarization for location. Another possibility is taking an aggregation operator like minimum. The error in the query can be generated if we know the maximum or minimum values for example of Per day Charges irrespective of locations then one can calculate the approximate answers [13]. For example, consider that the mobile user has received a response to his earlier query about the Per day charges in CA area as 100 and tax as 5% for Holiday-Inn. Now he has received in response to the same query Per day charges as 110 max. and tax as 7% max in another region A consisting of CA and TX using the relation shown in Figure 4. For simplicity, we assume that the query does not take into account rebates. With the help of earlier answer, he can approximately calculate that the error in per day charges between two locations CA and TX is (110 + 7%) - (100 +5%). This will be the error in the positive side. That is, user might have to pay minimum 0 dollars to maximum 12 dollars. If we

use the aggregation operator min then error can be calculated on the negative side. User knows in this case he has to pay minimum amount as 100+5% and his earlier answer can be of no help. Thus, it is better to use aggregation operator max.

Hotel-name	Location-address	Per day charges Max.	Tax (%) Max.	Rebate(%) (Oct.-March) Max.	Distance (from airport in miles) Max.
Holiday-Inn	Region A	110	7	10	30
Ramada-Inn	Region A	110	7	10	40

Figure 4. Hotel2 : Summarized relation Hotel1

By performing one more level of summarization and aggregation, we get the following relation as shown in Figure 5. Note that if the query is to find the Per day charges in Holiday-Inn, then the query in SQL-syntax will look like as follows:

Select Per day charges from Hotel2 where Location-address = "Region A".

The above query gives approximate answers that are valid in a much larger region A.

Hotel-name	Location-address	Per day charges Max.	Tax (%) Max.	Rebate(%) (Oct.-March) Max.	Distance (from airport in miles) Max.
Hotels	Region A	110	7	10	40

Figure 4. Hotel3: Summarized relation Hotel2

7. Replication and Partitioning

In mobile computing, replication of data can improve availability and allow a mobile host to operate even when disconnected for the fixed network. However, it incurs overheads since all replicas must be kept up-to-date. In the case of location-dependent data, our replication scheme is innovative as it involves a hierarchical replication of database.

7.1 Data Partitioning based on Location

Since the tuples returned by a location-dependent query depend on the value of their location attribute, to speed-up query processing, a relation may be partitioned based on the value of the location attribute. This partition may be physical or logical. In the case of a logical partition, the spatial index is used to define the

tuples in the partition. A MH can store the part of the database that corresponds to its location. By doing so, queries can be answered locally on the MH.

The relation Hotels can be partitioned shown in Figure 1 based on locations. We can get two partitions one for CA and another for TX. The partition (relation) for CA is as follows.

Hotel-name	Location-address	Per day charges	Tax	Distance
Holiday-Inn	San Jose , CA	110	5	5
Holiday –Inn	Palo Alto, CA	110	5	10
Ramada-Inn	San Jose, CA	110	5	30
Holiday-Inn	Berkeley, CA	110	5	30

Figure 5. Hotel 4: Partition relation Hotels

The attributes such as Per day charges, Tax can be removed since they repeat in values. Thus, we get the following relations shown below in Figures 6 and 7. The original relation can be obtained by taking the cross product of the relations shown in Figure 6 and 7. Note that Figure 5 corresponds to horizontal partition where as Figures 6 and 7 are with respect to vertical partitions of relation Hotels.

Hotel-name	Location-address	Distance
Holiday-Inn	San Jose , CA	5
Holiday –Inn	Palo Alto, CA	10
Ramada-Inn	San Jose, CA	30
Holiday-Inn	Berkeley, CA	30

Figure 6. Vertical partition 1: Hotels

Per Day Charges	Tax
110	5

Figure 7. Vertical partition 2: Hotels

The Location-address can be removed as well and the location information can be augmented either along with hotel name or using the concept hierarchies as shown in Figure 2 since we know that this relation is for locations in CA. Thus, we can again summarize the partition shown in Figure 6 as discussed before in section 6.2.

7.2 Hierarchical Replication based on Location

Location dependency provides for an innovative replication scheme. We view a database as having multiple layers in which case we fully replicate *base data* at the lowest level. Base data is location-independent data. Replication of base data can be done at all the servers and at all locations. Replication of base data is similar to distributed replication; that is base data should have consistent data values at all times. At layer 0, we only replicate base data; data that do not depend upon location. At the next higher level, we have data that are the same for that set of locations at that level and so on. Thus, we can replicate data in a hierarchical fashion.

Let us assume that Per day charges are same in all Holiday Inns hotels in CA region as in Figure 1. We also assume here that relation contains data about hotels in CA and TX only. We consider Hotel name and Per day charges as base relation and attach the CA and TX as location information with this relation. At next higher level, we create a relation with Tax, Address (location), Winter rebate and Distance (maximum) as attributes.

Per day charges	Hotel names
110	Holiday-Inn
100	Ramada-Inn

Figure 8. Level 0 – Base relation - location (CA, TX)

Tax(%)	Hotel address	Winter rebate(%)	Distance (maximum)
5	CA	10	30
7	TX	0	40

Figure 9. Level 1 – relation at level 1- location CA

In Figure 1, Per day charges remain the same within the CA area, so there is no need to repeat this information for all the locations in CA, but it can be attached only to CA so that it will be valid for all the locations in CA. We can also replace the tuples that belong to different regions in CA by a single tuple with address CA as shown in Figure 9. This strategy avoids repeating information at all locations. However, note that some information may need to be repeated in case we need to join two relations linked to two different locations. We do not address this issue further in this paper.

8. Implementation Issues

In an implementation, one can store concept hierarchies and corresponding indexes at MUs. These concept hierarchies (CHs) and indexes correspond to the current location of MU. Once the MU moves to different location (note that many cells collectively may define a location), they need to be replaced by

corresponding CHs and associated indexes. The CHs are stored at DBS permanently. Since there are number of DBS, each one can have CHs and indexes corresponding to their location or they may also have replicated CHs and indexes. One advantage of replicating them is that MU can cache the CHs of next location before moving. However, in case there is a change in CHs and indexes at DBS, cache of MU needs to be made consistent. Normally CHs are changed very infrequently since they are generated based on domain knowledge, not on instance level.

We assume here that MUs have some capabilities of processing a query. Once a query arrives at MU, it can reformulate the query with the help of CHs and then use indexes to find the corresponding database (relations). The query is then redirected to the corresponding DBS through BS which executes the query.

Another choice for storing CHs is DBS. Here it is we assumed that MUs have no capabilities of processing a query. Once the query arrives at MU, it is forwarded to DBS which then reformulate and executes the query.

Regarding the databases, each MU (if has database capabilities) can also store the partition tables (as shown in Figures 5, 6 and 7) with respect to its location. It then can execute the query locally. These partitioned tables however needs to be consistent with respect to original table at DBS. However, at the time disconnection, these tables can still be used to provide approximate answers [13].

9. Conclusions

We discuss location-dependent data and query evaluation. The modeling aspect and the data structures needed to implement location-dependent queries are given. The hierarchical replication scheme discussed is different than traditional replication of databases. Our location-dependent data and attribute concept is different than moving objects [4,5,6] where location-attribute is constantly changing (they call it dynamic attribute). We are currently working on building location-based indexing techniques. We plan to generate concept hierarchies based on location information given in the database. In the future, we plan to build location-dependent database query processing system, which can be placed on the top of any database management system.

References

[1] Vijay Kumar and Margaret H. Dunham, "Defining Location Data Dependency, Transaction Mobility, and Commitment. Technical Report 98-CSE-1, Southern Methodist University, February 1998.

[2] Margaret H. Dunham and Vijay Kumar, "Location Dependent Data and its Management in Mobile Databases, *Dexa '98*, Austria 1998.

[3] J. Han, and Y. Fu, Dynamic Generation and Refinement of Concept Hierarchies for Knowledge Discovery in Databases, in Proceedings AAAI'94 Workshop on Knowledge Discovery in Databases (KDD'94), Seattle, WA, 157-168. July, 1994.

[4] Ouri Wolfson, Bo Xu, Sam Chamberlain, Liqin Jiang: Moving Objects Databases: Issues and Solutions. SSDBM 1998: 111-122

[5] A. Prasad Sistla, Ouri Wolfson, Sam Chamberlain, Son Dao: Modeling and Querying Moving Objects. ICDE 1997: 422-432 .

[6] George Kollios, Dimitrios Gunopulos and Vassilis J. Tsotras: On Indexing Mobile Objects. PODS 1999: 261-272.

[7] Evaggelia Pitoura and Ioannis Fudos, An Efficient Hierarchical Scheme for Locating Highly Mobile Users. CIKM 1998: 218-225

[8] Jamel Tayeb, Uzgur Ulusoy, Ouri Wolfson: A Quadtree-Based Dynamic Attribute Indexing Method. The Computer Journal 41(3): 185-200(1998)

[9] Ouri Wolfson, Sam Chamberlain, Son Dao, Liqin Jiang, Gisela Mendez: Cost and Imprecision in Modeling the Position of Moving Objects. ICDE 1998: 588-596

[10] OmniTRACS. Communicating without Limits. http://www.qualcomm.com/ProdTech/Omni/prodtech/omnisys.html.

[11] Margaret H. Dunham and A. Helan, "Mobile Computing and Databases: Anything New?", SIGMOD Record, Vol. 24, No. 4, December 1995.

[12] J. F. Roddick, N. G. Craske, and T. J. Richards, Hierarchical and set-valued domains as an approach to summarization and query optimization in databases, Department of Computer Science and Computer Engineering, La Trobe University, July, Technical Report, 12/93, 1993.

[13] S.K. Madria, Mukesh Mohania and J. Roddick, A Query Processing Model for Mobile Computing using Concept Hierarchies and Summary Databases, in proceedings of 5th Intl. Conference on Foundation for Data Organization (FODO'98), Japan, Nov. 1998.