Processing of Location-Dependent Continuous Queries on Real-Time Spatial Data: The View from RETINA

Dick Hung[1], Kam-Yiu Lam[1], Edward Chan[1] and Krithi Ramamritham[2]

Department of Computer Science[1]
City University of Hong Kong
85 Tat Chee Avenue, Kowloon
HONG KONG

Department of Computer Science and Engineering[2]
Indian Institute of Technology Bombay
Mumbai, INDIA

Abstract

In this paper, using RETINA, a real-time navigation system, as an example, we study the important design issues underlying the processing of location-dependent continuous queries, especially those requiring access to data describing the current status of a dynamic environment and possessing spatial properties. To minimize the probability of missing the arrival deadline associated with each navigation request, we use a *time-stamp with prediction* scheme to model the traffic data and replicated *dynamic directed graphs* to organize the traffic data required for path searching and path calculation. Correctness of the best path calculations and scalability of the system are improved through an *adaptive Push or Pull (APoP)* scheme to monitor the best path and traffic data in navigation.

Keywords: real-time data management, temporal consistency, mobile computing

1. Introduction

Owing to advances in mobile communication and database technologies, various innovative mobile computing applications are emerging. One of the important mobile applications is to support queries from mobile clients on real-time data items. In this paper, we introduce our system, called ***RETINA***, which is an abbreviation of ***RE****al-****T****ime* ***T****raff****I****c* ***NA****vigation System*. It is a real-time navigation system designed for critical services, i.e., ambulances and fire services. The basic function provided by the system is to calculate the *best path* based on the connections and the current traffic conditions of the roads for mobile clients to go from their current positions to their destinations. While a mobile client is following the suggested path to its destination, the path information is closely monitored. It is assumed that each navigation request is associated with a deadline, called *arrival deadline*, on its arrival time to the destination. Meeting the arrival deadline is one of the prime requirements of the system.

In this paper, we report our views from the design and development of RETINA. Our focuses are to *minimize* the probability of missing the arrival deadlines of requests and at the same time to minimize the processing overhead in data monitoring with an attempt to improve the *scalability* of the system. In particular, we concentrate on the following three issues:

(1) Efficient management of traffic data;
(2) Minimizing the probability of missing arrival deadlines of navigation requests; and
(3) Minimizing the overhead for monitoring the best path.

We have formulated the navigation requests as *location-dependent continuous queries (LDCQs)* [5] and have explored the temporal spatial properties of the traffic data items in data management and monitoring the best path such that the processing requirements of the requests can be met and the processing overhead can be minimized. We propose a replicated directed graph approach for searching the best path in which the servers cache traffic data from other servers. To maintain cached traffic data, we have adopted the *Push* and *Pull* techniques [3,4] for providing traffic data items for execution of the navigation requests [1] and for cached data management. In our design, the best path to a destination is calculated based on the roads connecting the starting location and the destination, and the *real-time* traffic conditions of the roads. In the calculation of the road traffic, future traffic conditions are predicted and used [2], both to minimize the impact of the changes in traffic conditions during the traveling and to increase the probability of satisfying the timing requirement of the system.

2. The Problems

Navigation requests are usually submitted as *continuous queries*. A continuous query stays in the system for a period of time until its arrival deadline is reached. During this period, it will be evaluated continuously until its arrival deadline or until its termination conditions is satisfied. In addition, navigation requests are *location-dependent* since the best path for a mobile client to the destination depends on its current location. To generate correct results, it is important to manage the real-time location of the mobile client, which initiates the request. Its position has to be *mutually consistent* to the execution of its navigation request. However, the network bandwidth of a mobile network is usually very limited and network disconnection is common. Tracking locations of moving objects accurately could incur a heavy workload on the mobile network and at the server for location update processing.

To calculate the best path, the system needs access to a (large) set of traffic data, which record the current traffic conditions of the roads, to perform path-searching computation on the traffic data. The computation could be quite complex and incur a heavy workload overhead if the computation is performed frequently. Road traffic data are *temporal spatial data*. Each traffic data item describes the current traffic condition of a particular road segment in the system. Its value can be highly dynamic and change rapidly with time. Different road segments may have different distances from the current position of the requesting client. To the queries, it is important to provide the latest versions of the data items for their executions. Accessing out-dated (stale) data items may seriously affect the usefulness of the query results and increase the probability of missing the arrival deadlines of the navigation requests. However, maintaining data freshness could incur a heavy processing overhead on the system. Therefore, how to improve the scalability of the system and at the same time to maintain data freshness for

query execution is an important tradeoff in the design of the systems. The management of traffic data for correct execution of navigation requests becomes an even more complex problem if the data items are distributed at several servers. The use of multiple servers is important to the scalability and fault tolerance of the system.

Another important concern in improving the scalability of the system is how to minimize the overhead for calculating the best path. It could be very expensive and the cost depends on the number of roads in the systems, and how the roads are arranged and connected. In order to ensure that a client can meet its arrival deadline, the system has to monitor the traffic data. Close monitoring of traffic data therefore could be very expensive especially if the update rates of the traffic data are high and the data are distributed.

The navigation system is for critical services and its prime performance objective is to minimize the probability of missing the arrival deadline. If it is expected that the arrival deadline will be missed, it is important to minimize the lateness and to inform the client of the situation as earlier as possible. To achieve the above objectives, the system needs to know what is the *"best path"* to the client's destination. However, the definition of the meaning of the "best path" is not trivial in this kind of *temporal spatial problem*. As a client takes time to its destination, the best path at current time may not be the best one if the objective is to minimize the probability of missing the arrival deadline. Consider the example shown in Figure 1. At time t_1, client C is at edge p_1 and is following the calculated best path (p_1, p_2, p_3 and p_4) to its destination, D. At time t_2, C is at p_2 and the calculated best path to p_4 has been changed to (p_2, p_1, p_3, p_4), i.e., due to an increase in traffic congestion between p_2 and p_3. Therefore, it has to go back to p_1. However, when it reaches p_1 at t_3, the best path may change back to (p_1, p_2, p_3 and p_4). We can see that client C is moving back and forth between p_1 and p_2 and has wasted a lot of time. Compared to the move between the two paths, another choice for the client is follow the original suggested path (p_1, p_2, p_3 and p_4) at time t_1. Although it may not be the best path at t_2, it is the best path at time t_3. The cause of the problem is that in the calculation of the best path, it only considers the current traffic data and does not consider what will be the future traffic data.

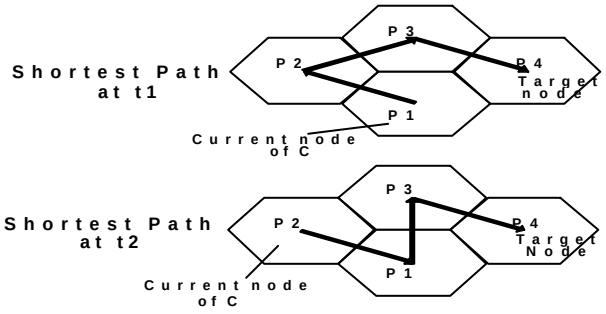

Figure 1. Problem in the Calculation of the Best Path

Although it is impossible to have the exact values of future traffic data, accuracy in prediction can be improved by *predicting* the future values of traffic data. Since the predicted value of a data item has a higher probability to be the value at the time when the client is on the edge than the current value of the data item, the chance of going back and forth could be lower.

3. System Model and Architecture

The system is supported on a cellular mobile network. In each cell, there is a traffic server for managing the traffic data belonging to the cell and for serving the clients within the cell. The traffic servers are connected together by a high-speed network and the traffic server maintains a database to records the connections of the roads in the whole service area. The roads are divided into road segments. A traffic sensor is installed at the control point of each road segment to measure the traffic condition of the road segment. The measured traffic data values are sent to the server of their cells as traffic updates through a reliable wired network. The mobile clients may generate navigation requests to the traffic server of their current cells while they are moving. It is assumed that each client is carrying a positioning device and knows its current location. If a mobile client moves into another cell, the traffic server of the new cell will take up the job to serve the client and monitor the best path.

The traffic server at a cell is responsible for managing the traffic data of its cells and for serving the clients within the cell. Figure 2 shows the architecture of a traffic server. The data manager is responsible for managing the database. It consists of two components: cache manager and version manager. The database consists of two types of data items: real-time and static data items. Each server maintains the same set of static data which are the information about the road information and connection information of the roads in the whole system. The real-time data items are state-based information and are used to record the traffic conditions of the road segments. Updates for the traffic data of its cells are received from road sensors. Real-time data of the traffic conditions of the road segments of other cells are cached from other cells and are managed by the cache manager. In order to reduce the data access delay, the data items are downloaded into the cache buffer in the main memory from stable storage at system startup. In the traffic server, the update stream handler receives traffic updates from the sensors installed on the roads. The update stream handler puts the new updates into the update buffer from which the data manager will install the updates into the cache buffer. The version manager is responsible for maintaining coherence amongst the different versions in the various servers. The client service manager is responsible for serving navigation requests from mobile clients.

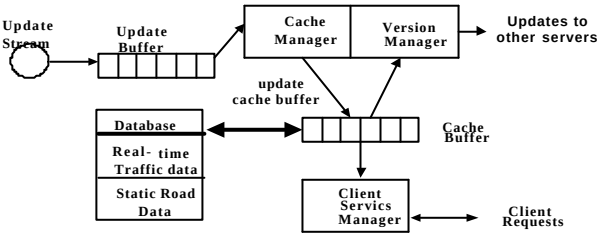

Figure 2. The Navigation Server

4. Management of Traffic Data and Traffic Value Prediction

In this section, we will discuss the details of the various schemes designed to manage the traffic data to support the navigation function efficiently at each traffic server. We propose a *time-stamp with prediction* scheme for modeling the traffic data and a replicated *dynamic directed graphs* scheme to organize the traffic data for best path searching.

4.1 Traffic Data Modeling

For each traffic data item x, three copies are maintained in the cache buffer: current value (x_c), previous value (x_p) and last database updated value (x_d). The current value, x_c, is the value from the last update. Previous value, x_p, is the value

before the current data value. Last database update value, x_d, is the current value of the data item in the database. Whenever a new update arrives, the current value will be copied into the previous value, and the new value from the update will be copied into the current value. Associated with each value is a time-stamp, *ts(x)*. In addition to the three data versions, a prediction function *pf(x)* is maintained for each data item *x*. It calculates the trend of a data item *x* to describe how the value of *x* changes compared to previous values. In our design, we use an *exponentially weighted moving average* approach for calculating the data trend:

$f_{x,n} = (f_{x,n} + \alpha f_{x,n-1} + \alpha^2 f_{x,n-2} + \ldots. + \alpha^{n-1} f_{x,1}) / S$

where S is the sum of the weights $(1 + \alpha + \alpha^2 + \ldots. + \alpha^{n-1})$

$f_{x,n} = (x_{p,n} - x_{p,n-1}) / (ts(x_{p,n}) - ts(x_{p,n-1}))$, where α is a tuning parameter with a value smaller than 1.

Periodically, an update transaction is executed to save the current values of the data items, x_c, at the buffer (into the database). The update period is dynamically defined and is based on the difference of the data values.

4.2 Traffic Graph

Each server maintains a directed graph at its cache buffer. The graph is defined according to how the road segments are connected in the service area. The length of an edge is using the current value of the data item corresponding to the road segment. It is a dynamic directed graph since the length of an edge in the graph is not fixed. New values from the traffic sensors are periodically sent to their servers to update the values of the traffic data corresponding to the road segments. For the control points, which are maintained by other servers, new traffic data values are cached from the other servers as shown in Figure 3. Therefore, an important issue is to maintain validity of cached data so that temporal consistency of the data items can be ensured for best path calculation.

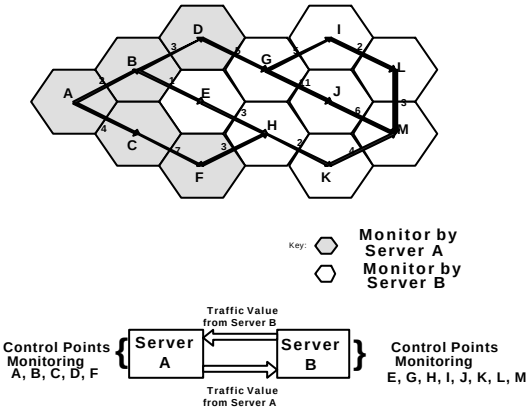

Figure 3. Management of Traffic Graphs

4.3 Traffic Graph Value Prediction and Path Searching

When a server receives a navigation request from a mobile client, it performs a best path search on the traffic graph using a shortest path-searching algorithm to find the best path to the destination from its current position in the shortest time. Each request is associated with a *current location, client ID* and *arrival deadline*. To improve the correctness of the results, we use predicted traffic values for calculating the length of the edges in the traffic graph.

Traffic data have spatial properties towards a location-dependent query. If a control point is further away from its current position, the time when the mobile client will reach the control point will become later, and the current traffic conditions reported by the control points will have a higher probability to be different from the traffic condition when the mobile client is passing through the control point. In our proposed prediction scheme, we estimate the future traffic of an edge at the time when a mobile client will be on the edge. The length of an edge, L_i, in the graph is a variable and is calculated as the product of the physical length of the road segment, p_i, and its traffic condition, w_i: $L_i = p_i \times w_i$

The physical length of a road segment is a constant while w_i is a variable. In the search of the best path searching, the value of w_i is obtained from a time varying function, *traffic(i,t)* which returns the traffic condition of road segment *i* at time, *t*. The function, *traffic(i,t)* first calculates the predicted time (t), which is the estimated time when the mobile client will be on the starting point of road segment *i*. It is the summation of the current time and the predicted traveling time from its current position to the start point of the road segment, *i*. Note that the traffic condition returned from a road sensor of a road segment indicates the approximate time for a client to pass through the road segment. The predicted traffic of the road segment *i* at the predicted time (t) will be calculated using a linear extrapolation method where the future predicted value for data item *x* at time *t* is calculated as: $x_t = f_{x,n} \times t + x_c$.

5. Path Monitoring and Re-calculation

While a mobile client is following the suggested path to its destination, the best path has to be closely monitored to ensure that it is still the best path to the destination. The data monitoring problem can be divided into two parts:

(1) To monitor the best path information between the client and the server, which is serving the client, and;
(2) To monitor the values of the traffic data, especially those cached data from other servers, to ensure that they are still temporally consistent for best path calculation.

To resolve the first problem, the system may need to perform re-calculation when there is a change in traffic condition of the roads or the client's movement is different from prediction. In principle, any change in traffic data even for the edges not in the path may affect the best path. The frequency of re-calculation affects the shortest path information to the client. However, on the other hand, it is important to minimize the monitoring overhead since the data values could be highly dynamic and the cost for the performance of the path-searching algorithm could be very high. To resolve the second problem, a traffic server needs to forward new values to refresh cached data items at other servers whenever it receives a new traffic update. The frequency of data forwarding affects the update processing workload and communication overhead for data transmission.

5.1 Adaptive Push or Pull (APoP)

5.1.1 Overview

In this section, we will introduce an adaptive Push or Pull method, called *adaptive Push or Pull (APoP)* for maintaining consistency of cached traffic data. In APoP, a request may be served in *Pull* or *Push* modes. Conditions are defined to determine the switching between *Push* and *Pull*. The design objectives of ***APoP*** are:

(1) To balance the workloads on the mobile network and at the server,
(2) To improve the scalability of the system, and

(3) To meet the data consistency requirements and arrival deadlines of the requests.

In principle, the conditions for switching between *Push* and *Pull* are defined based on several factors: the current system workload, the urgency of the requests, and the size of the set of data items interested by each request. In general, the network overhead of using *Pulling* is higher than using *Pushing* while the processing overhead at the server is higher with *Pushing* than *Pulling*. Normally, *Pushing* can provide a closer monitoring of the best path than *Pulling* at the expense of a higher processing overhead at the server by adjusting the frequency for checking of *pushing* conditions. In addition, unlike conventional pulling schemes, the pulling period in ***APoP*** is assigned by the server. So, if the pulling time has arrived and the server does not receive the pulling request, the server will assume that the client has been disconnected from the network. Since a server may be serving multiple navigation requests, the server should serve the set of requests in a fair manner such that each request receives similar amount of server (defined in terms of service time) from the server. So, if the processing power of the server is *P*, then the average amount of workload to be assigned to serve a request is *P/N* where *N* is the number of concurrent requests in the server. The purpose is to prevent a navigation request from overloading the server and affecting the processing of other requests.

5.1.2 Pull Mode

In *APoP*, initially, a newly submitted request will be served in *Pull* mode, i.e., the client periodically sends a navigation request with its current position to the server. The reason is that the *Pull* approach can provide a higher degree of resiliency to server failure and at the same time it can control the workload at the server and on the network by adjusting the *Pulling* period. When it is not necessary to have a close monitoring of its shortest path, i.e., its arrival deadline is far from the current time, a larger *Pulling* period may be used in order to minimize the workloads at the server and on the mobile network. Each request in *Pull* mode is assigned a *Pulling* period bound, *min_pp* and *max_pp*. The next *Pulling* period is defined within this bound by the server based on:

(1) The predicted workload for serving the request; and
(2) The dynamic properties of the data items.

Each request is assigned a workload index, which indicates the relative cost for performing the shortest path search for the request. The workload index is calculated as a function of the distance of the moving client from its destination, its speed and the workload index of its previous calculation. After defining the shortest path, the set of the control points, which are *close* to the shortest path, are defined as the set of *interested data items* of the request. The set of interested data items are the data items on the shortest path and those within a threshold distance from the shortest path. The changes in the values of the data items in the interested data item set have a higher probability of affecting the shortest path. Thus, if their rate of change is higher, the pulling period should be shorter. The dynamic properties of the data items can be calculated as the mean of the *trends* of the data items, i.e., in $f_{x,n}$ in our data model, of the data items in the interested data set.

In serving a Pulling request, a traffic server may need to generate pulling request to other servers if it has cached data from other servers. The additional pulling not only increases the pulling case but also increases the delay for processing the requests. To resolve the problem, we may define a similarity bound on each data item. If the time-stamp of the current value of a cached data item is close to the current time and its trend is not large, the request for pulling to the server, which maintains the data item, may not be necessary. Even though a new version has been created, its value will be *similar* to the value of the cached data item

5.1.3 Push Mode

5.1.3.1 Pull to Push

When the arrival deadline of a request is approaching and the workload of the database server is within system capacity, the system may switch to serve some of the requests to *Push* mode (*Pull-to-Push*) in order to minimize the mobile communication cost and to provide a closer monitoring of data items for the request. The selection of which requests to be switched to *Push* mode *(Pull-to-Push)* is based on the deadlines and the slacks of the requests. Note that when the deadline of a mobile client is approaching, normally, it should be close to its destination. Therefore, the processing overhead of the path-searching algorithm should be smaller comparing to the case where the deadline is far away. In order to prevent frequent re-calculation due to a highly variable value of an edge, if the predicted arrival time is smaller than the arrival deadline, re-calculation will be performed only when there is a change in traffic value of a path in the shortest path; and the change is an increase in value.

The set of requests which are being served in *Push* mode are maintained in the push list and data structures for maintaining the state of the requests will be created by the system including the set of interested data items of the requests. Similar to the requests being served in *Pull* mode, each request (Q) in Push mode will be defined with a set of interested data items set, ID(Q). The union of the interested data items set of all the requests, which are in Push mode, is called *server interested data item set*. It means the server has to closely monitor the values of this set of data items. The changes in their values have a high probability in changing the best path. Again, the definition of the set of interested data items for a request is based on the shortest path and the defined threshold distance.

When the workload at the server is heavy, some of the requests may be switched back to Pull mode (*Push-to-Pull*). The selection is again based on the deadlines, the slacks, and the update rates of the set of data items required by the requests, which are in the push list. If the total update rate of the interested data items is higher, switching it to pull mode can provide a greater saving in server workload.

5.1.3.2 Minimizing Monitoring Overhead in Push

The main concern in data monitoring using the *Push* mode is how to reduce the number of recalculation. Not all changes in traffic data will affect the shortest path especially those changes are smaller. To minimize the number of recalculation, we may use a *batching* method for performing the recalculation. Since each request will have an estimated arrival time, we can calculate its slack time for meeting the arrival deadline. If the increase in traffic value of those edges on the shortest path is smaller than the slack, it does need to perform re-calculation. Although the best path at the client side may not still be the shortest path, the client should still meet its arrival deadline. On the other hand, if the total decrease in traffic data of those edges, which are in the interested data of a request but are not those edges on the shortest path, is not greater than the slack of the request, the system also does not need to perform re-calculation.

5.2 Client Location Prediction

As explained in Section 1, in processing the navigation request, it is important to maintain consistency between the current location of the client and the execution of the request since it is location-dependent. Uncertainty in location can be large when a client is being served in push mode. When performing the data monitoring, the server estimates the location of a mobile client for the calculation of the shortest path if it is serving the client in push mode. The estimation can be based on the information in the traffic graph. In the traffic graph, the length of a node indicates how long the client will take to complete the edge. In addition, the client will report its location if its location deviates from the estimated position more than a location update threshold. In this way, we can minimize the number of location update for those clients in push mode.

6. Implementation and Operations

6.1 Implementation

In the implementation of RETINA, two basic components, database servers and mobile clients, are developed. The server program is implemented in VC^{++} 6.0 on Windows NT and the client program is implemented in both VC^{++} 5.0 for Win CE 3.0 and VC^{++} 5.0 on Windows 2000. The server process consists of multiple flags for different functions including servicing requests from mobile client and for system management. A SQL server is connected to the server program at each site and it maintains a database containing the road connections of a district in Hong Kong. A simulated process is defined at the server to create new traffic data values randomly for the road segments maintained by the system. Each client machine, PC or handheld PC, is pre-loaded with a set of maps for the roads defined in the database servers. The client program consists of multiple flags for displaying a map, getting and displaying traffic data on the map and generating different types of location-dependent requests. In this section we will main focus on the screen and menu design

6.2 Screen Design and Operations

The following screen is designed to setup the connection between server and client, user can input the server address and port number to connect to the navigation server according to that IP address and port number via a wireless network.

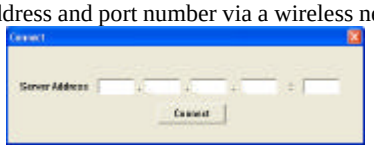

Figure 4. Connect Screen

The following is the main screen of the system. It shows the detailed map of the control points, static object icons and the current location of the mobile client itself.

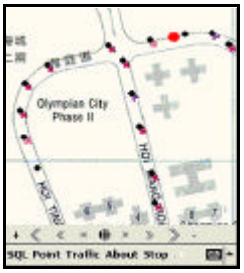

Figure 5. Navigation Screen

In navigation, a moving client may select the destination by click on the control point sensor associate with it. After selecting the destination, the shortest path will be display as the green filled arrows. In navigation mode, only control points along the shortest path will be display and the path will be update, according to the traffic condition and displacement of the mobile client.

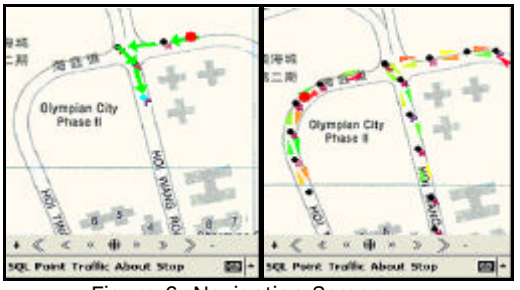

Figure 6. Navigation Screen

The Traffic screen displays the current traffic status on the road. If the traffic is in low density, a green triangle will be display, otherwise the triangle will trended to be red and indicate that the traffic is in high volume on that road segment. The Traffic screen is trigger by the "Traffic" menu item on the menu bar.

7. Conclusions and Future Works

In this paper we presented the design issues involved in the development of our real-time navigation system. Using a time-stamp with prediction scheme to model the traffic data and replicated dynamic directed graphs to organize the traffic data required for path searching and calculation, we hope to minimize deadline misses associated with navigational requests. An adaptive push and pull technique is used to improve the scalability of the monitoring of traffic in our system. We are currently implementing the system on a Windows CE/ Windows 2000 based platform to test the performance of our system and the effectively of using the push and pull schemes for maintaining the temporal consistency of traffic data.

References

[1] O. Ulusoy, "Real-Time Data Management for Mobile Computing", in *Proceedings of International Workshop on Issues and Applications of Database Technology (IADT'98)*, Berlin, Germany, 1998.

[2] A.P.Sistla, O.Wolfson, S.Chamberlain, S. Dao, "Querying the Uncertain Position of Moving Objects", in Temporal Database: Research and Practice, Lecture Notes in Computer Science (Springer Verlag), 1998.

[3] S. Acharya, M. Franklin, S. Zdonik, "Balancing Push and Pull for Data Broadcast", *Proceedings of ACM SIGMOD*, Tucson, Arizona, 1997.

[4] Pavan Deolasee, Amol Katkar, Ankur Panchbudhe, Krithi Ramamritham, Prashant Shenoy "Adaptive Push-Pull: Disseminating Dynamic Web Data", in *Proceedings of the 10th WWW Conference*, Hong Kong, 2001.

[5] M.H. Dunham and V, Kumar, "Location dependent data and its management in mobile databases," in *Proceedings of International Workshop of Database and Expert Systems Applications*, pp. 414-419, 1998.

Acknowledgement: The work described in this paper was partially supported by a grant from the Research Grants Council of Hong Kong SAR, China [Project No. CityU 1076/02E].