01 Jan 2006

Window Query Processing with Proxy Cache

Gao Xing

John Sustersic

A. R. Hurson
Missouri University of Science and Technology, hurson@mst.edu

Recommended Citation

G. Xing et al., "Window Query Processing with Proxy Cache," *Proceedings of the 7th International Conference on Mobile Data Management (MDM'06) (2006, Nara, Japan)*, Institute of Electrical and Electronics Engineers (IEEE), Jan 2006.
The definitive version is available at https://doi.org/10.1109/MDM.2006.166

Window Query Processing with Proxy Cache *

Xing Gao, John Sustersic, and Ali R. Hurson
Department of Computer Science and Engineering
The Pennsylvania State University
University Park, PA 16802, USA
{xgao, sustersi, hurson}@cse.psu.edu

Abstract

A location dependent query (LDQ) result set is valid only in a specific region called the validity region (VR). While limiting the validity of a particular result set to a given area, the VR may also be used in caching implementations to determine if cached results satisfy semantically equivalent queries. Existing LDQ caching schemes rely on the database servers to provide the VR at a cost of high computational overhead. Alternatively, a LDQ proxy cache, which approximates the VR can be employed, freeing the database servers from the high cost of calculating the VR. A LDQ proxy cache architecture is proposed to compute an estimated validity region (EVR) based on the observed querying history at the proxy server. We present an algorithm - Window_EVR - for the LDQ proxy to compute the EVR for a window query result set. The simulation results show that LDQ proxy caching using the Window_EVR algorithm significantly reduces both the window query response time and the workload at the database servers while maintaining query result set accuracy.

1. Introduction

In a mobile computing system, a mobile client may issue queries with some location restrictions. Such a query is called a location aware query (LAQ) [13]. A subclass of the LAQ is the location dependent query (LDQ), whose result set depends upon the client's current location [1]. For instance, "Find the phone numbers of the McDonald's in New York city" is an LAQ, while "Find the phone numbers of all McDonald's within 10 miles from my current location" is a LDQ. Two most common types of LDQ are the nearest neighbor (NN) queries and range queries. A NN query retrieves the data object satisfying the query that is the closest to the querying location, while a range query retrieves all the data objects within a specific range [8]. A window query is the most important type of range query where the range is an axis-parallel rectangle.

Definition 1: Window query - *A window query, Win_Q(x, y, x_length, y_length), retrieves all objects satisfying the query located in a rectangle region whose center lies on the geographical point (x, y). Edges of the rectangle are x_length and y_length, parallel to x-axis and y-axis, respectively.*

A LDQ result set is dependent on user's current location and is valid only in a specific region (the VR). If the user reissues the same query at a new location, the query must be resubmitted to the database (DB) server. This may lead to unnecessary network traffic and DB server workload if the result set remains valid, i.e. the new issuing location is still within the VR.

VR-aware LDQ caching is one solution to address this problem. Provided with a VR for a cached result set, a VR-aware LDQ cache may determine if the querying location of a LDQ is within the VR of a cached result; this permits some LDQs to be satisfied from the cache. The DB server has full knowledge of the geographical locations of all data, and it can determine the precise VR. Without this information, the mobile clients and proxy servers cannot determine the VR precisely. Consequently, most of the existing LDQ caching solutions rely on the DB server to provide VRs for LDQ result sets. Since computing the VR requires extra storage and processing overhead, DB servers frequently do not provide this service or provide it only when workload permits.

[7] proposed an algorithm to estimate the VR when the DB server does not provide it. In this algorithm, the

* The Office of Naval Research and National Science Foundation under the contracts N00014-02-1-0282 and IIS-0324835 in part have supported this work.

LDQ proxy server caches the most frequently issued LDQs, their result sets, and the corresponding VRs. When the DB server does not provide the VR for a LDQ result set, the LDQ proxy computes an estimated validity region (EVR) based on the observed querying history. The proposed Right-hand algorithm in [7] can determine EVRs for all LDQs with convex VRs, including NN queries.

Computing EVRs for window queries involves more complexity than for NN queries. First, a NN query has only one result, while a window query may have zero, one, or multiple results. Second, the VR for a NN query result is always convex [15], while the VR for a window query result set could be concave. Analyzing the VRs for window query result sets reveals several important common characteristics. Based on these characteristics, we propose the Window_EVR algorithm to determine EVRs for window query result sets.

The rest of this paper is organized as follows. Section 2 reviews the existing work related to LDQ caching and window query processing. We briefly describe the LDQ proxy in section 3. Section 4 presents the basis for and the description of the Window_EVR algorithm. Our simulations of the proposed Window_EVR algorithm are discussed and simulation results are analyzed in section 5. Finally, section 6 concludes this paper and presents future research directions.

2. Related work

The idea of queries with location constraints was originally introduced in [9], and has been discussed in many other works [1, 4, 5, 6, 13]. Naturally, mobile users are likely to query information related to his or her current position. This class of queries was termed the location dependent query (LDQ) in [1]. [13] distinguished LDQs from other queries with location constraints: a query whose result depends on certain location attributes is a location aware query (LAQ), while a LDQ is a query whose result depends on user's current location.

Inspired by the semantic caching [3, 10], [12] proposed a modified semantic caching scheme for location dependent results. Taking validity information into the consideration, [16] presented algorithms for cache invalidation and cache replacement strategies.

There are several algorithms for the DB server to determine the VR for a NN query result. [15] built the static Voronoi Diagram (VD) to index all data objects for NN queries. The Voronoi cell (VC) of the result object is the corresponding VR. The VD, however, is expensive to maintain due to database updates, and it is also inapplicable for the k nearest neighbor (k-NN) query when k is unknown. Even when k is known, an order-k VD is very expensive in terms of computational and storage overhead [17]. Alternatively, [17] proposed algorithms to calculate the VR for NN queries during the run time. It avoids the large storage overhead but introduces extra computing and I/O cost.

Unlike a NN query result set that always has at most one result, a window query result set may contain multiple result objects. One of the most recent algorithms to calculate the VR for window query result sets was proposed in [17]. In this algorithm, the VR is determined at the run time. After retrieving all the result objects, the DB server needs to execute extra window queries in order to determine the VR for the result set.

3. LDQ proxy with Window_EVR algorithm

3.1. LDQ proxy structure

We propose to implement a LDQ proxy at the mobile base station (BS). The LDQ proxy consists of three components: a semantic LDQ cache, a LDQ cache manager, and a LDQ filter (see figure 1). Three essential fields in a semantic LDQ cache entry are LDQ_id, LDQ_result, and LDQ_vr. LDQ_id uniquely identifies a LDQ. LDQ_result is the cached LDQ result set. The LDQ_vr is the corresponding VR for the LDQ result. The LDQ cache manager is responsible for cache maintenance and cache protocols. The LDQ filter determines whether or not a LDQ can be answered by a cached result set.

Definition 2: Semantic LDQ equivalence - *Two location dependent queries (LDQ) are semantically equivalent if and only if they produce same results at any given location when processing the same databases.*

The DB servers in our mobile system model provide a menu of query templates with certain options and variables. The mobile client generates LDQs with query templates and specific variable values. For example, NN("hotel") searches the nearest hotel, and Window("hotel", 5, 10) returns all hotels within a rectangle 5*10 mile^2 axis-parallel window centered at the current position. Consequently, two LDQs are semantically equivalent if and only if they are generated by the same query template and same variable values.

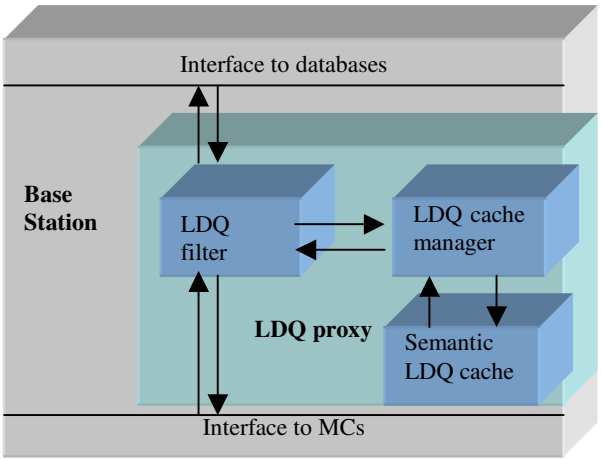

Figure 1. LDQ proxy cache configuration

Theorem 1: Two semantically equivalent LDQs have the same VR for all result sets.

Proof by contradiction: Assume the VRs for two semantically equivalent LDQs are different. There is at least one location where they produce different result sets. According to definition 1, they should produce the same result set at any given location. We have reached a contradiction. The assumption must be false and two semantically equivalent LDQs share the same VR for all result sets.

Based on definition 2 and theorem 1, we now can determine when a LDQ can be answered by a cached result set, i.e., if there is a semantic LDQ cache hit.

Definition 3: Semantic LDQ cache hit - *A LDQ can be answered by a LDQ result set in the semantic LDQ cache if the following conditions are met:*

1. *The incoming LDQ and the cached LDQ are semantically equivalent.*
2. *The incoming LDQ is issued within the validity region of the cached result set.*
3. *The cache entry is valid.*

3.2. Window query processing at LDQ proxy

The following procedure shows how the LDQ proxy cache processes a widow query.

Step 1: Look up in the LDQ cache

Upon receiving a window query, the LDQ proxy looks up the satisfying result set in the cache. If there is an entry for a semantically equivalent query and the querying position is in the corresponding VR, there is a semantic LDQ cache hit, then go to step 5. Otherwise, continue to step 2.

Step 2: Forward the window query to the DB server

If there is no semantic LDQ cache hit, LDQ proxy will forward the window query to the DB server and request a VR. After receiving query result, if a VR is provided along with the result set, continue to step 3. Otherwise, go to step 4.

Step 3: Update proxy cache with VR

If there is no cache entry for the LDQ associated with this result set, a new entry with window query description, result set, and the VR will be inserted into the proxy cache. If there is an existing entry for the cached result set, it must be true that the cache entry has only an EVR, instead of a VR, for the result set. (Otherwise, there should be a semantic LDQ cache hit and the query should have been resolved.) The LDQ proxy will update the entry by replacing the EVR with the VR, then go to step 5.

Step 4: Update the proxy cache without VR

If there is no cache entry for the LDQ associated with this result set, the Window_EVR algorithm (to be described in section 4) will generate an EVR based on the semantically equivalent queries in the querying history. A new entry with window query description, result set, and the EVR will be inserted into the proxy cache. If there is a cache entry with the query result set associated with an EVR, LDQ proxy re-computes the EVR using the Window_EVR algorithm. The cache entry is updated with the newly generated EVR.

Step 5: Return the result set to the mobile client

Return the window query result to the mobile client. If the mobile client has requested validity information, LDQ proxy will return the associated VR or EVR to the mobile clients.

4. Window_EVR algorithm

4.1. VRs of window query result sets

In order to define the VR, we need to introduce the term 'Minkowski region (MR)'. The MR of an object is a rectangle identical to the query window whose geometric center lies on the corresponding object. If the querying position is within the MR of an object *a*, object *a* will be a result object. Otherwise, object *a* will not be in the window query result set.

The VR of a window query result set is the area that is within the MRs of all result objects, and outside the MRs of all non-result objects. The intersection of the MRs of all result objects is called inner validity region (IVR). It is the maximum area where the result objects remain in the result set. To prevent non-result objects from the result set, the MRs for all non-result objects must be removed from the IVR, yielding the VR [17], which is a convex or concave polygon.

4.2. Characteristics of window query VRs

We find several important characteristics of the VRs for window query result sets. As the edges of the querying window are parallel to x-axis or y-axis, the edges of the IVR and the MRs for all result objects are also parallel with x-axis or y-axis. Consequently, the IVR is a rectangle, i.e., a convex polygon with 4 right angles (90º). Removal of the MR of a non-result object from IVR may introduce a 270º angle into the VR. Therefore, a VR may be convex or concave. Since an IVR is bounded by the length and width of a MR, it is impossible to have two adjacent 270º angles. From above analyses, we find the following characteristics of the VR for a window query result set.

1. The VR is a convex or concave polygon.
2. The edges of the VR are parallel to x-or y-axis.
3. All interior angles of the polygon are either 90º or 270º.
4. There are no adjacent 270º angles.

4.3. Window_EVR algorithm

The LDQ proxy does not have direct access to the databases maintained at the DB servers. It has only a partial knowledge about the database via its querying history recording the recent querying activities. The content of a query event includes 1) the LDQ description, 2) the querying location, and 3) the query result set. If several querying events return the same result set for a window query, all these querying positions are within the VR of this LDQ result set.

The LDQ proxy uses the Window_EVR algorithm to generate the EVR for a window query result set based on the querying history. The goal is to estimate the largest region that is guaranteed to be within the corresponding VR. The Window_EVR algorithm generates EVR in 3 steps:

Step 1: Estimate the IVR

The algorithm determines the estimated inner validity region (EIVR) based on the query history from the set of semantically equivalent LDQs sharing the same result set. This EIVR is taken as the minimal bounding rectangle (MBR) that covers all querying locations from which semantically equivalent LDQs generate the same result set.

Step 2: Pessimistically estimate MRs of non-result objects

Each pair of querying locations indicates two bounds on the proximity of non-result objects. Pessimistically assuming that the querying locations are precisely on the boundary of the VR, these two bounds imply one of two worst-case MRs of a data object that is not part of the result set. By themselves, it is impossible to disprove the existence of either non-result MR. However, the existence of other querying locations known to reside in the VR permit many of these possible MRs to be eliminated.

Step 3: Obtain the EVR

The EVR is obtained by pessimistically estimating MRs from potential non-result objects and eliminating them with known valid LDQ querying locations. LDQs issued from within the EVR provide no new information regarding the VR; however, LDQs issued outside the EVR but from within the VR do. These LDQs expand the EVR by eliminating estimated MRs from non-result objects as before, replacing them with less-pessimistic ones while still guaranteeing that the EVR is completely contained within the VR. This methodology results in a monotonically increasing EVR that approached the VR asymptotically.

A running example of 4 querying events demonstrates the Window_EVR algorithm. The outer polygon represents the precise VR that is unavailable to the LDQ proxy. The EIVR is the bounding rectangle of the known querying locations. The fading regions represent the worst-case MRs of the non-result data objects.

The EIVR after the first querying event contains only one point $<x_1, y_1>$. The lack of further knowledge requires the pessimistic approximation of non-result MRs - $\{(x<>x_1)\&(y<>y_1)\}$ (see figure 2). As a result, the EVR is only one point.

After the second querying event, the EIVR is the rectangle $\{<x_2, y_2>, <x_1, y_1>\}$, and non-result MRs are described by $\{(x<x_1)\&(y>y_2)$ or $(x>x_2)\&(y<y_1)$ shown as two fading shadow regions (see figure 3). The EVR contains only two points.

The EIVR after the third query is the rectangle $\{<x_2, y_3>, <x_3, y_1>\}$, and non-result MRs described by $\{(x<x_1)\&(y>y_2)$ or $(x<x_3)\&(y<y_2)$ or $(x>x_1)\&(y>y_3)\}$ (see figure 4). So far, the EVR consists of only the querying locations of the observed LDQs.

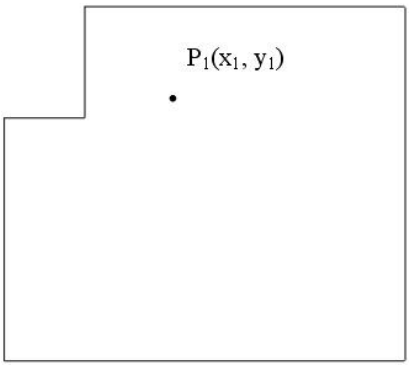

Figure 2. EVR after first query

After the forth query, however, the EIVR becomes {<x_2, y_3>, <x_4, y_1>}, and pessimistically estimated non-result MRs are described by {($x<x_1$)&($y>y_2$) or ($x<x_3$)&($y<y_2$) or ($x>x_3$)&($y<y_4$) or ($x>x_1$)&($y>y_4$)}. The current EVR is a rectangle {<x1, y2>, <x3, y4>}, illustrated as the white area in figure 5.

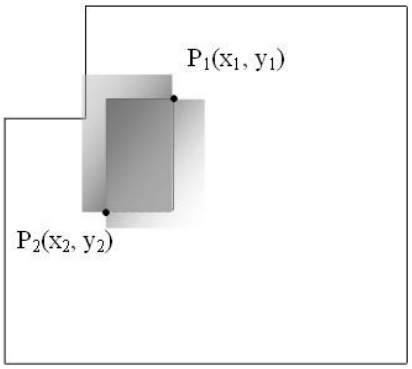

Figure 3. EVR after second query

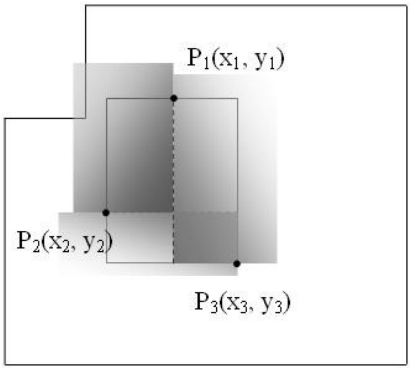

Figure 4. EVR after third query

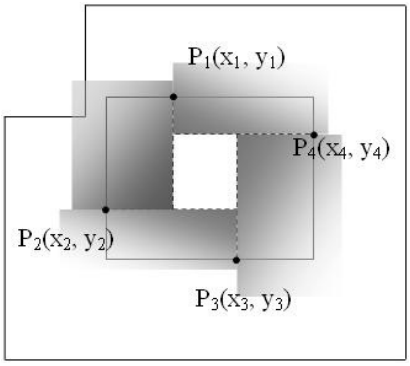

Figure 5. EVR after fourth query

4.4. EVR containment guarantee

Theorem 2: The Window_EVR algorithm generates an EVR that is completely contained with the VR.

Proof: By definition the EVR is a sub region of the IVR. By pessimistically assuming that all observed querying locations occur at the very boundary of the VR and considering all possible non-result MRs that are not eliminated by the existence of querying locations known to be valid, the algorithm guarantees that at no time the EVR exceeds the boundary of the VR. Therefore the EVR is guaranteed to be a sub region in the VR.

5. Evaluation and analysis

The Window_EVR algorithm is evaluated using a simulator constructed in [2]. The experiments simulate two systems with similar environment setup. The only difference is that the proxy server in one system, named EVR system, employs Window_EVR algorithm and implements LDQ cache. The other system, named NO_EVR system, implements the LDQ cache and relies solely on the DB server to provide VRs for query result sets. The simulation results show that when the DB server does not always provide the VR for window query, the LDQ proxy and the Window_EVR algorithm greatly reduce both the number of LDQs sent to the DB server and the average window query response time.

5.1. Simulation model

The experiments employed the dataset of a region centered on State College, Pennsylvania, and including the Pennsylvania State University. The region is bounded by a 12,000m*12,000m square (a longitude span of 0.1425 degree and latitude span of 0.1078 degree). Based on demographic information, we modeled the simulated area into 4 regions: centre, west, south, and northeast. The city has a population of 50,000, and the population density in each region follows a normal distribution. The regional centers (μ_x, μ_y), standard deviations σ, and total population distributions (α) during working and non-working hours are shown in table 1.

Table 1. Region model and parameters

Region	Center (μ_x, μ_y)	σ	Population 9am~5pm	Population 5pm-9am
Centre	6000,6000	σ_0=100	α_0=0.4	α_0=0.25
West	2000,7000	σ_1=60	α_1=0.2	α_1=0.25
South	6000,3000	σ_2=60	α_2=0.2	α_2=0.25
Northeast	10000,8000	σ_3=100	α_3=0.2	α_3=0.25

The population is also categorized into 4 groups: G_0 (those are less than 18 years old), G_1 (college students), G_2 (middle age workers), and G_3 (senior

residents). Each group has different moving pattern, and table 2 lists the population distributions β (based on the real demographical data of the simulated region), moving probabilities P_{Move}, and speed patterns for each population category.

Table 2. Group characteristic

	G_0	G_1	G_2	G_3
Population	β_0=0.1	β_1=0.5	β_2=0.25	β_3=0.15
P_{Move}	μ_{0a} =9, μ_{0b}=16, σ_0=1.5	μ_{1a} =10, μ_{1b} =17, σ_1=3	μ_{2a} =9, μ_{2b}=17, σ_2=2	μ_3 =11, σ_3=3
$V_{Walking}$ (mph)	μ_0=2.5, σ_0=0.5	μ_1=2.5, σ_1=0.3	μ_2=2.0, σ_2=0.3	μ_3=1.0, σ_3=0.3
V_{Local} (mph)	μ_0=25, σ_0=5	μ_1=25, σ_1=5	μ_2=25, σ_2=5	μ_3=25, σ_3=5
$V_{Highway}$ (mph)	μ_0=40, σ_0=5	μ_1=40, σ_1=5	μ_2=40, σ_2=5	μ_3=40, σ_3=5
Trips/day	λ_0= 1.5	λ_1= 3	λ_2= 2	λ_3= 0.7
Hops/trip	Λ_0= 0.8	λ_1= 0.8	λ_2= 0.8	λ_3= 0.8

The walking speed for each group follows normal distribution with corresponding (μ, σ). All four groups share the same driving patterns on local roads or highways. The number of trips per day and number of hops per trips follow the Poisson distribution with corresponding λ. While the probability to start a movement for group G_3 follows a normal distribution, G_0, G_1, and G_2 are modeled with a composite normal distribution to best capture the mobility pattern of mobile users (see equation 1 and 2).

Equation 1:

$$P_{move\_g0,1,2}=0.495*\frac{e^{\frac{-(t-\mu_a)^2}{2\sigma^2}}}{\sigma\sqrt{2\pi}}+0.495*\frac{e^{\frac{-(t-\mu_b)^2}{2\sigma^2}}}{\sigma\sqrt{2\pi}}+0.01$$

Equation 2:

$$P_{move\_g3}=0.99*\frac{e^{\frac{-(t-\mu_3)^2}{2\sigma_3^2}}}{\sigma_3\sqrt{2\pi}}+0.01$$

The simulator simulated one DB server and one proxy server serving the simulated region. A newly developed geographical information systems, e.g., [11], can process up to 200 queries per second, so we model the mean database service rate as 0.01 seconds and exponentially distributed. According to [17], computing the VR doubles the number of database I/O accesses required for processing a query, therefore we modeled the database service rate VR by an exponential distribution with a mean of 0.02 seconds.

The database contains about 700 data objects in 20 categories for local businesses and community locations within the simulated region. The querying probability for object categories and data within a category is modeled as a Zipf [14] distribution.

The querying window for each query may be one of three different sizes: small (600m*600m), medium (1000m*1000m), and large (2000m*2000m). Similar to most web search engines, there is a limit in the number of result objects returned in response to a query. This simulation returns up to 10 result objects. Both proxy cache and client caches employ the least recently used (LRU) cache replacement policy. The lookup latency at client cache and proxy cache are both 0.0001 seconds. It is hard to precisely model the overhead of Window_EVR in generating and update EVR. Considering potentially sheer database size at the DB server and the limited querying history (500 querying events) at the proxy, we model the Window_EVR algorithm overhead, in searching and computation, to be 10% of the query processing delay at the DB server, i.e., 0.001 seconds.

Wireless transmission delay between mobile clients and the BS is determined by the bandwidth and package size. The latency between the mobile client and the BS depends on the message size and the available bandwidth. The latency between the BS and the DB server depends on the fixed network bandwidth and traffic patterns. Table 3 gives the value of these and other parameters from which our simulation is modeled.

Table 3. Other simulation parameters

Parameters	Value
Simulation length (days)	50
Number of data objects about this city	680
Proxy cache size (query result sets)	100
Client cache size (query result sets)	10
Proxy querying history size	500
Network bandwidth, BS to DB link (Mbps)	1000
Background network (BS to DB) utilization	0.4
Uplink bandwidth (Kbps)	19.2
Downlink bandwidth (Kbps)	144
Window query request size (byte)	32
Average window query results size (byte)	320
Average VR descriptor size (byte)	60
Max number of objects in a result set	10
Average query rate, daytime (hours)	½
Average query rate, night (hours)	1/5

In order to verify the efficiency of the proposed Window_EVR algorithm, the simulation considers window queries only. The performance improvement is measured by two metrics: speed up of LDQ response time and the DB server workload reduction. The speed up of LDQ response time, $S_{response}$, is the difference between the measured response times of the NO_EVR system and the EVR system divided by the response

time of NO_EVR system (see equation 3). The DB server workload reduction, $R_{workload}$, is the difference between the number of LDQs sent to DB servers in NO_EVR and the EVR system divided by the number of LDQs sent to DB servers in NO_EVR system (see equation 4).

Equation 3:

$$S_{response} = \frac{RT_{NO\_EVR} - RT_{EVR}}{RT_{NO\_EVR}}$$

where RT is the average window query response time.

Equation 4:

$$R_{workload} = \frac{NQDB_{NO\_EVR} - NQDB_{EVR}}{NQDB_{NO\_EVR}}$$

where NQDB is the number of window queries sent to the DB server.

5.2. Performance evaluation and analysis

We simulated and compare both systems for different scenarios in which the DB server provides the VR for a LDQ with probabilities {0, 0.2, 0.4, 0.6, 0.8, and 1.0}. Figures 6~8 show the performance improvements realized when the proxy LDQ cache employs the Window_EVR algorithm duing a 50-day simulation. The Window_EVR algorithm results in significant speed up in the LDQ response time while reducing the workload at the DB server, particularly when the DB server never provides VRs.

Figure 6 displays the speed up in window query response time achieved through Window_EVR algorithm. Figure 7 shows the percentage of mobile queries that are processed by the DB server. Derived from these percentages, the curve in figure 8 illustrates the workload reduction at the DB server. In the case that the DB server always provides the VR, the performance of both scenarios are virtually identical. When the VRs are only occasionally available from the DB server, the Window_EVR algorithm in EVR

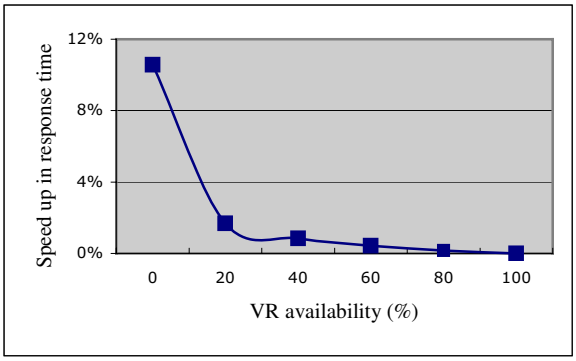

Figure 6. Speed up in LDQ response time

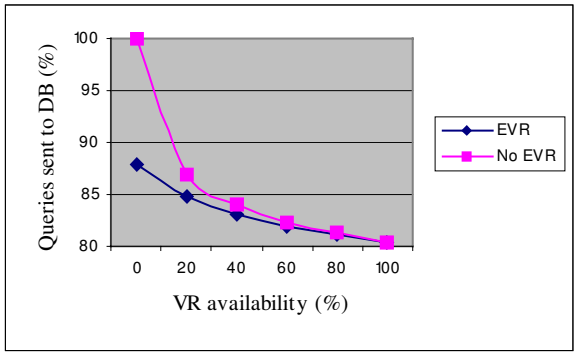

Figure 7. Number of queries sent to the DB server

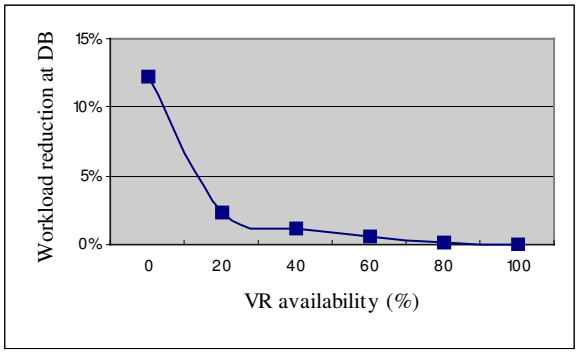

Figure 8. Workload reduction at the DB server

system reduces the average LDQ response time and the number of queries sent to the DB server. In the scenario when the DB server never provides the VR for a LDQ result, Window_EVR algorithm achieves an 11% speedup in LDQ response time while reducing the DB server workload by 12%.

6. Conclusion and future work

We proposed an algorithm to estimate the VR for window queries. The EVR was proven to be a sub-region of the corresponding VR. This algorithm was evaluated using a detailed simulation scenario modeled after a real, modern community and including components that consider actual population demographics, modes of transportation, time-of-day affects and trip-based mobility. The simulation results show that the Window_EVR algorithm permits effective caching of LDQ results without relying on the DB server to provide the VR.

We plan to extend this work to study the cooperative LDQ caching between mobile clients. Furthermore, some mobile users prefer fast response time at a cost of acceptable inaccuracy of the LDQ results or the VRs. We will study Quality of Service (QoS) issues in LDQ cache management and to further improve the system performance.

7. Reference

[1] D. Barbara, "Mobile Computing and Databases - A Survey", *EEE Transactions on Knowledge and Data Engineering*, 11(1), 1999.

[2] http://www.mesquite.com/

[3] S. Dar, M. Franklin, B. Jonsson, D. Srivatava, and M. Tan, "Semantic Data Caching and Replacement", *International Conference on Very Large Data Bases (VLDB)*, 1996.

[4] M. Dunham and A. Helal, "Mobile Computing and Databases: Anything New?" *SIGMOD Record*, Vol. 24, No. 4, 1995.

[5] M. Dunham and V. Kumar, "Location Dependent Data and its Management in Mobile Databases", *Proceedings of the Ninth International Workshop on Database and Expert Systems Applications*, 1998.

[6] G. Forman and J. Zahorjan, "The Challenges of Mobile Computing", *IEEE Computer*, Volume: 27(4), 1994.

[7] X. Gao and A. R. Hurson, "Location Dependent Query Proxy", *ACM Symposium on Applied Computing*, 2005.

[8] R. Guting, "An Introduction to Spatial Database Systems", *Special Issue on Spatial Database Systems of the VLDB Journal*, 3(4), 1994.

[9] T. Imielinski and B. Badrinath, "Querying in Highly Mobile Distributed Environments", *International Conference on Very Large Data Bases (VLDB)*, 1992.

[10] A. Keller and J. Basu, "A Predicate-based Caching Scheme for Client-server Database Architectures", *Very Large Data Bases Journal*, 5(1), 1996.

[11] MetaCarta, products information, http://www.metacarta.com/products/technology.asp

[12] Q. Ren and M. Dunham, "Using Semantic Caching to Manage Location Dependent Data in Mobile Computing", *International Conference on Mobile Computing and Networking (MobiCom)*, 2000.

[13] A. Seydim, M. Dunham, and V. Kumar, "Location Dependent Query Processing", *International Workshop on Data Engineering for Wireless and Mobile Access*, 2001.

[14] G. K. Zipf, "Relative Frequency as a Determinant of Dhonetic Change", *Harvard Studies in Classical Philology*, 15, 1929.

[15] B. Zheng and D. Lee, "Semantic Caching in Location-dependent Query Processing", *Seventh International Symposium on Spatial and Temporal Databases*, 2001.

[16] B. Zheng, J. Xu, and D. Lee, "Cache Invalidation and Replacement Strategies for Location-Dependent Data in Mobile Environments", *IEEE Trans. on Computers, Special Issue on Database Management and Mobile Computing*, 51(10), 2002.

[17] J. Zhang, M. Zhu, D. Papadias, Y. Tao, and D. Lee, "Location-based Spatial Queries", *International Conference on Management of Data (SIGMOD)*, 2003.