

# Processing Location-Dependent Queries in a Multi-cell Wireless Environment

Baihua Zheng      Dik Lun Lee  
Department of Computer Science  
HongKong University of Science and Technology  
Clear Water Bay, Kowloon  
Hong Kong, P. R. China  
{baihua, dlee}@cs.ust.hk

## ABSTRACT

We develop several methods for scheduling location-dependent queries when clients cross cell boundaries in a multi-cell wireless environment. Our study is based on a common scenario where data objects are stationary while clients, which issue the queries, are mobile. For query processing, we use Voronoi Diagrams to construct an index and a semantic cache for improving data reusability. For handoff clients, we propose three scheduling methods, namely, the priority method, the intelligent method, and the hybrid method to improve performance. A simulation is conducted to study the performance of the methods.

**Keywords:** location-dependent query, semantic cache, handoff

## 1. INTRODUCTION

The advance of the wireless network and the popularity of portable devices have fueled the development of mobile computing and made it one of the hottest topics in academia and industry. Among numerous fields of mobile computing, location-dependent queries (LDQs) have obtained much attention in recent years due to the mobility of clients [1]. LDQs are queries whose answers depend on the locations of the clients that issue them. Our current research focuses on indexing, semantic caching, and the corresponding cache invalidation strategies in a multi-cell environment. Therefore, we have to address the handoff problem. In a mobile computing environment, a cell is covered by a base station. Mobile clients use the wireless network to connect to the base station and the objects in the cell. The handoff problem occurs when a client leaves a cell before it receives a response from the server. Since handoff clients would have been waiting in the original cell for a while, the waiting time in the new cell needs to be shortened in order to maintain a reasonable total waiting time for handoff clients. In our previous paper on location-dependent queries [7], we studied ef-

ficient query-processing methods in a single-cell environment and so did not take handoff into account.

According to the mobility of clients and the queried objects, queries can be categorized into three types, namely, mobile clients querying static objects, stationary clients querying moving objects, and mobile clients querying mobile objects. Here *clients* is used to denote mobile computers submitting the queries and *objects* means all the candidates of the answer set. Most, if not all, LDQs can be classified into one of the three types. We can analyze each type separately to define the scenario clearly and simplify the problem. Our work focuses on the first type because, to the best of our knowledge, no previous research has been done on the first type of query. Furthermore, research on the second type has already been done [3, 4], and research on the third type can be simplified if solutions to the first two types have been found.

In the rest of this paper, Section 2 introduces our ideas about processing queries in a single cell environment. Three different scheduling methods are presented in Section 3 to handle the handoff problem. The simulation result is given in Section 4. Finally, the summary and future work are presented in Section 5.

## 2. QUERY PROCESSING IN A SINGLE CELL

From the examples in the previous section, we can learn that nearest-neighbor queries are very useful in many situations, such as cars looking for a gas station, urgent patients looking for the nearest hospital, hungry tourists looking for the nearest restaurant, and strangers looking for a public service, to name but a few. Motivated by these needs, we develop a method for retrieving the nearest services efficiently in a multi-cell mobile environment. One assumption we make in this paper is that the location of a client is known (e.g., from GPS). When a user issues a query, its location, current velocity, and the current time are also submitted to the server.

Our idea is to use Voronoi Diagrams (VD) to index the nearest-neighbor information of the data objects. A VD records information about what is close to what. Thus, it is naturally suitable for nearest neighbor searching. Let  $P = \{p_1, p_2, \dots, p_n\}$  be a set of points in the plane (or in any dimensional space), each of which we call a *site*. Define  $\mathcal{V}(p_i)$ , the *Voronoi cell* for  $p_i$ , as the set of the points  $q$  in the plane such that  $dist(q, p_i) < dist(q, p_j)$ . That is, the Voronoi cell for  $p_i$  consists of the set of points for which  $p_i$

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright 2001 ACM 0-89791-88-6/97/05 ...\$5.00.

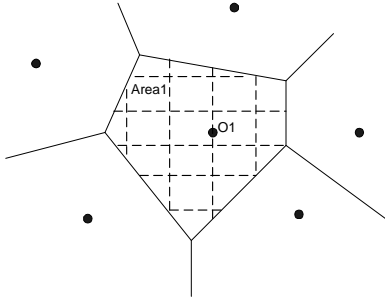


Figure 1: VD used in nearest neighbor searching

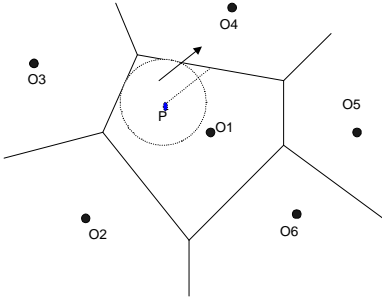


Figure 2: Semantic Caches in Voronoi Diagrams

is the unique nearest neighbor to  $q$ :  $\mathcal{V}(p_i) = \{q | \text{dist}(q, p_i) < \text{dist}(q, p_j), \forall j \neq i\}$ . Figure 1 is an example, in which  $O_1$  is the nearest neighbor of any points in the shaded area  $Area_1$ .

Since  $VD$  is a traditional data structure in computational geometry, efficient structures to store the  $VD$  and point location methods to locate the point are available. Only a brief description has been included in this paper since it has been described in detail in our previous paper [7]. One thing different from the previous paper [7] is the construction of the  $VD$  in a multi-cell environment. If each cell only constructs the corresponding  $VD$  according to the sites within its cell, the  $VD$  may return incorrect results to the clients near the boundary of the cell when the real nearest neighbor of the clients is outside of the cell's coverage. In order to prevent this from happening, the base station needs to communicate with the base stations in neighboring cells to obtain the sites which are just outside its boundary.

For the scenario of finding the nearest restaurant, the restaurants are the fixed sites and the mobile clients are the points needing to be located. Once the mobile client is located in one area, the unique fixed site of this region is the answer. An advantage of this type of query is that all the services are fixed and only change occasionally. Furthermore, the location information of the services is available before query submission, so some preprocessing can be performed in advance. Rather than storing the answer to the current query in its local cache, we use a semantic cache which stores not only the frequently used data but also the semantic description of the data [2, 5]. In our scenario, the semantic information includes not only the description but also the invalidation information about the answer. The *Maximal Valid Circle (MVC)* is a circle centered at the client's current location with a maximal radius to keep the whole circle within the *voronoi cell* of this site. Figure 2 depicts an example where the dash line represents the trajectory of the

client  $p$  and the circle around  $p$  is the *MVC*. Here we deploy two different schemes to compute the radius of the *MVC* and to provide prediction of the next nearest neighbor.

- According to the current velocity and the length of the trajectory within the region of this site, we can predict the time when the client will cross this region and also locate the next nearest site. As shown in Figure 2, the length of the dash line divided by the client's current speed is the valid time interval of this answer  $O_1$ . After this interval, the next site,  $O_4$ , will be the correct answer. As such, an answer like  $\langle O_1, 2, O_4 \rangle$  will be transmitted to the client, where two seconds is the valid time interval. One drawback is that any update to the client's velocity within the predicted valid period will make the prediction invalid. A way of preventing any false prediction is to resubmit the query when the client changes velocity, but this will obviously incur a high overhead.
- The other way is to use the maximal speed of the client rather than the current velocity. Here, the radius of the *MVC* is used because the client needs to travel this distance at least to leave the current voronoi cell. This distance divided by the client's maximal speed is the time interval during which the current answer is guaranteed to be valid. Using this scheme, changes to the client's speed need not be considered, but the price is no prediction since we use the maximal speed rather than the client's current velocity.

In summary, the following steps will be taken by both the client and the server to answer a query. First the client checks its local cache, only when a cache miss happens would the client submit the query to the server. The query will also include the location and current velocity of the client and the timestamp. Then, based on the  $VD$ , the server locates the voronoi cell containing the client and also computes the *MVC*'s radius. When the client receives the results from the server, it inserts the center and radius of the *MVC* and the respected service object in its cache as a cached record, and finishes the query. Here, we adopt the first scheme to provide prediction. If the second scheme that can guarantee the valid duration is employed, the maximal speed rather than the current velocity of the client is used.

### 3. HANDOFF IN A MULTI-CELL ENVIRONMENT

In a single-cell environment, the above steps can handle the query perfectly, but in a multi-cell environment, the problem of handoff has to be considered. How to provide fast service to the handoff clients is the main consideration of our work in this paper. We propose several scheduling strategies by which the server can schedule the queries from the clients. We assume that the client can detect the time that it departs the current cell and enters another cell and then inform the server of this transition. In order to make a clear comparison among different schemes, we use *cross number* to denote the number of clients that have not received the response from the server before leaving the current cell, and *recross number* to identify the number of handoff clients that still have not received service before leaving the new cell.

**Naive Method** If the client enters a new cell before it gets a response from the original server, it resubmits the

same query to the new server. For the server, just before answering a query, it checks if the client is still in its cell. If not, the query is dropped. The advantage of this method is the simplicity of implementation. However, this method makes the waiting time of the handoff clients a little longer and may cause impatient clients to lose their interest in the queries.

**Priority Method** This method gives the handoff clients higher priority than the normal clients. Clients act in a similar manner as they do in the previous method, but the server answers queries from handoff clients first. Therefore, the waiting time of normal clients is lengthened. Consequently, we can expect that the *recross number* is decreased while the *cross number* is increased. This is because the cost of shortening the waiting time of handoff clients is to make normal clients wait longer. The trade-off must be considered. It can be controlled by limiting the percentage of handoff clients that can receive high priority; for example, 50 percent. The optimal value depends on the applications. In a situation where most clients are patient, the *recross number* is important since fairness is taken into consideration. When most clients are impatient, we can assume crossed clients have a higher possibility of losing interest, hence reducing the *cross number* is more reasonable.

**Intelligent Method** When a query is submitted, the client also computes the expected time of leaving the current cell. Given a reasonable threshold, clients which are expected to leave this cell soon would have their queries answered immediately. Other clients are treated as in the *naive method*. The advantage of this method is that some clients near the border would get a quick service. For normal clients, however the waiting time will be longer compared to that of the naive method and handoff clients win no bonus here.

**Hybrid Method** This is a mixture of the above two methods. Actually, the *priority* and *intelligent* methods are two different ways of improving performance. The *priority mode* pays attention to handoff clients, making their waiting time shorter than that of normal clients. This is to compensate the handoff clients which had already waited in the previous cell. The *intelligent mode* is a preventive approach. It services the clients that are most likely to handoff first in order to prevent them from going into the handoff mode. Although the combination of these two schemes is very attractive, the cost of this method is to increase the average waiting time of normal clients.

## 4. PERFORMANCE EVALUATION

Since the implementation environment is multi-cell, we assume that a cell crosses several sites. Otherwise, if there is only one site in one cell, the nearest neighbor problem becomes trivial.

In order to simplify the implementation of the simulation, we use a closed model and assume that nearly all the cells are in a steady state. Once we decide the ratio of the normal clients to handoff clients, the probability that a client is a handoff client can be determined. *HandoffPart* is the ratio of the number of handoff clients to the total number of clients in a cell. In the *priority method*, *PriorityPart* is used to identify the percentage of the handoff clients that are given higher priority. In the *intelligent method*, *UrgentRequest* represents the time threshold that once a client is expected to leave the cell within this time threshold, the an-

swering of its request should be given a higher priority. Table 1 gives a complete description of the parameter setting. Furthermore, in our simulation, all the clients have a semantic cache using *Area*<sup>1</sup> as the cache replacement scheme. The *Area* method replaces the cached data that have the smallest *MVC* when the cache is full.

Parameter	Description	Setting
PriorityPart	percentage of the handoff clients that have higher priority	0.0 – 1.0
HandoffPart	percentage of the total clients that are handoff clients	0.0 – 1.2
UrgentRequest	the time limitation of one request that should be answered quickly	10.0
ServNum	No. of service objects	70
RegionArea	the area of the cell region (unit: m)	$10^3 \times 10^3$
Max_Speed	the maximal speed of the client (unit :m/s)	10
ServiceTime	the time server used to answer one query and also send the request	1.0
Max_ThinkTime	the maximal time duration between one client's receiving the answer to the original query and sending out the next request	20.0
CacheSize	the size of the client's cache	10
ClientNum	the average number of clients in one cell	1000

Table 1: Simulation Model Setting

We use *CSIM* [6] to implement our simulation. A server is implemented as a process. It answers the queries submitted by the clients according to different scheduling schemes. If there are some queries waiting for an answer, the server provides the service. Otherwise it just remains idle. The clients are also implemented as processes. They submit a query and then wait for a response from the server. Before they get the answer, they will not issue a new request. Once a client gets the answer, it will think for a while. The maximal think time is denoted as the parameter *Max\_ThinkTime*. Since we use a closed model in the simulation, the number of clients in one cell is determined. Given *HandoffPart*, every newly-created client has a probability of *HandoffPart* to be a handoff client. If a client crosses the border, it is deleted and a new client within this cell is produced randomly. Details of the construction of *VD* and the data structures can be found in our previous paper [7] and will not be repeated here due to limitation of space. Below, we will show the results of different parameter settings and also discuss the observations we made from the results. The model parameters, unless specified otherwise, are initialized according to

<sup>1</sup>In the previous paper [7], we have developed three different cache replacement schemes. A detailed description and a performance comparison can be found in that paper.

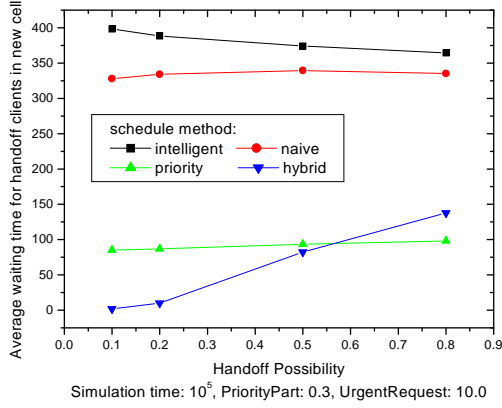


Figure 3: Waiting time of handoff clients

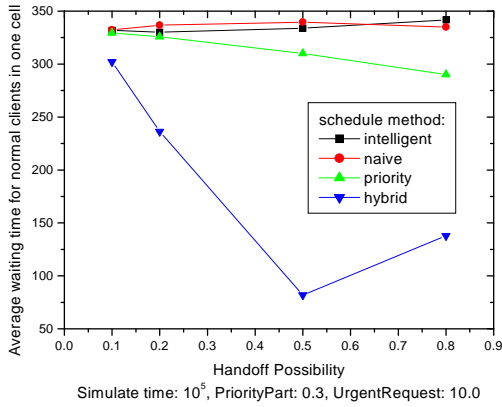


Figure 4: Average waiting time of normal clients

Table 1, with *PriorityPart* equaling 0.3 and *HandoffPart* equaling 0.2.

**A General Comparison of Different Methods** Figure 3 shows the waiting time of the handoff clients in a new cell without considering the waiting time in the original cell. Figure 4 shows the average waiting time of normal clients in a cell. Figures 5 and 6 show the *cross number* and *recross number* compared with the *naive method*. If we take the *naive method* as the baseline, the *priority method* can reduce the waiting time of the handoff clients in the new cell and so decrease the *recross number*. However, when there are numerous handoff clients in one cell, e.g., when *HandoffPart* is 0.8, this method loses its advantage. The reason for this is that when nearly all the clients are in handoff mode, giving them higher priority gains no benefit. The difference resulting from different *PriorityPart* settings will be analyzed later. For the *intelligent method*, the obvious gain is the reduction of the *cross number*. Since it gives a bonus to the urgent clients that will leave the cell soon, the handoff clients are not considered a priority unless they are also urgent clients. Therefore, the waiting time for handoff clients is a little bit longer than in the *naive method*. Combining the advantages of both methods, the *hybrid method* can result in a better performance than either one. It gives priority to both handoff clients and urgent clients, while the price of doing this is the delayed service of normal clients.

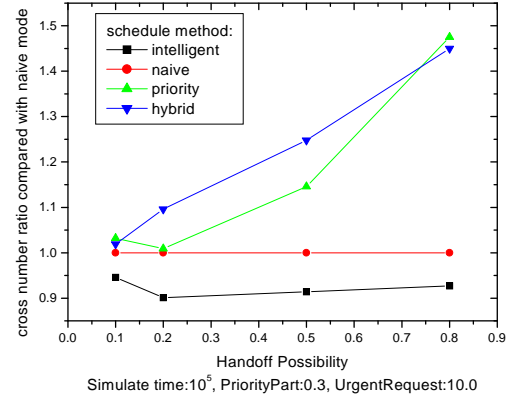


Figure 5: The cross number compared with the naive mode

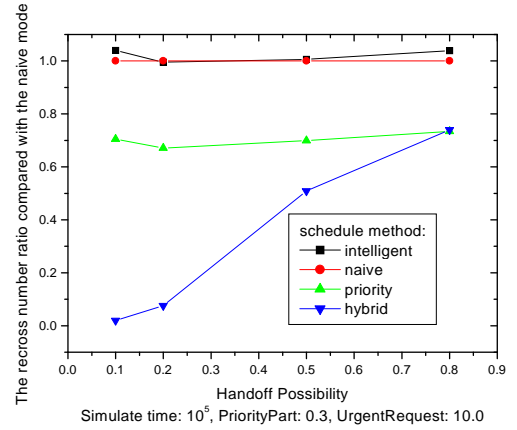


Figure 6: The recross number compared with the naive mode

Therefore, the *cross number* is increased. In a later discussion, we will analyze the different levels of performance caused by various compositions.

**Priority method** Intuitively, increasing *PriorityPart* can reduce the waiting time of handoff clients, while the increase is not monotonous from the results of experiments. When a small number of clients are in a handoff state, the bigger the *PriorityPart*, the shorter the waiting time for handoff clients and the smaller the *recross number*, while the waiting time for normal clients becomes longer. When clients have a high chance of being in a handoff state, the situation is different. For example, when *HandoffPart* is 0.5, the shortest waiting time for handoff clients is achieved when *PriorityPart* is between 0.8 and 0.85, rather than 1.0. Whether there is an analytical model for the optimization problem is an interesting question to be explored. Another problem of this mode is the longer waiting time of normal clients, especially the huge variance of the average waiting time. For instance, when *HandoffPart* is 0.5 and *PriorityPart* is set to 0.8, the average waiting time is about 1615 and the variance is nearly  $4.6 \times 10^6$ . This shows the unsteadiness of the performance that we should avoid in real applications.

**Hybrid method** Generally, this integrated method works

best. Since both *PriorityPart* and *UrgentRequest* can affect the performance, different settings can get different results. From the results, one can find that the performance variation under this method is very obvious, due to the mutual impacts among the parameters. Because no result can outperform all others in all situations, the evaluation of the performance is a problem. One solution is to use a function to evaluate gains of different settings. If we set *average waiting time of normal clients*, *waiting time of handoff clients*, the *recross number*, and the *cross number* as performance criterion, a possible function is as follows:  $P = \alpha \times Wait_{normal} + \beta \times Wait_{handoff} + \gamma \times N_{recross} + \theta \times N_{cross}$ . Since different parameters have various quantities, the ratios to thresholds rather than the real values are reasonable. Here,  $\alpha$ ,  $\beta$ ,  $\gamma$ , and  $\theta$  denote the weight of different performance criteria, depending on the different applications. The appropriate setting is still under investigation.

## 5. CONCLUSION AND FUTURE WORK

In this paper, we have introduced the basic ideas about how to answer *Location-Dependent Queries* in a multi-cell environment. Three scheduling methods have been described for handling handoffs in such an environment. If an application does not pay special attention to handoff clients, the *naive method* can be used. The *priority method* reduces the waiting time for handoff clients in the new cell and the *intelligent method* services urgent clients first in order to prevent them from entering the handoff state, while the *hybrid method* combines the advantages of both methods in order to get better trade-off in performance. Although we have done preliminary work in this area, there is a lot that needs further investigation.

In this paper, we identify the optimization problem as our next direction. Another target is how to set the parameters according to different application requirements. The impacts caused by varying parameters are currently being studied. Also, the impacts of various distributions of service sites and the scalability of the methods with large number of clients need further study.

## 6. REFERENCES

- [1] D. Barbara. Mobile computing and databases-a survey. *IEEE Transactions on Knowledge and Data Engineering*, 11(1), January -February 1999.
- [2] S. Dar, M. J. Franklin, and B. T. Jonsson. Semantic data caching and replacement. In *The 22nd Very Large Data Bases Conference (VLDB'96)*, 1996.
- [3] G. Kollios, D. Gunopulos, and V. J. Tsotras. Nearest neighbor queries in a mobile environment. In *International Workshop on Spatio-Temporal Database Management*, September 1999.
- [4] G. Kollios, D. Gunopulos, and V. J. Tsotras. On indexing mobile objects. In *18th ACM SIGMOD-SIGACT-SIGART Symp. on Principles of Database Systems*, May 1999.
- [5] Q. Ren and M. H. Dunham. Using semantic caching to manage location dependent data in mobile computing. In *The Sixth Annual International Conference on Mobile Computing and Networking (MobiCom'00)*, August 2000.
- [6] H. Schwetman. *Csim User's Guide (version 17)*. MCC Corporation, 1992.
- [7] B. H. Zheng and D. L. Lee. Semantic caching in location-dependent query processing. accepted by the Seventh International Symposium on Spatial and Temporal Databases (SSTD'01), July 2001.