Distributed Real-Time Processing of Range-Monitoring Queries in Heterogeneous Mobile Databases

Wong Cheow Yuen\textsuperscript{1}, Hamidah Ibrahim\textsuperscript{2}, Nur Izura Udzir\textsuperscript{3}

Department of Computer Science
Faculty of Computer Science and Information Technology
Universiti Putra Malaysia, 43400 Serdang, Malaysia
\textsuperscript{1}cheowyuen@gmail.com, \textsuperscript{2}hamidah@fsktm.upm.edu.my, \textsuperscript{3}izura@fsktm.upm.edu.my

Abstract

The emergence of location-aware services calls for new efficient real-time queries processing algorithms. In this paper, we focus specifically on real-time processing of range-monitoring queries. Ying Cai et al. introduced a technique called Monitoring Query Management (MQM) for efficient real-time processing of range-monitoring queries in heterogeneous mobile databases. The technique involves allowing the mobile objects to monitor their movement directly against nearby queries. Mobile objects need to update their locations to the server only when they move out of the assigned resident domain. In this paper, we present a distributed server infrastructure as opposed to the centralized approach in MQM to achieve scalability and robustness. In addition, to add flexibility to MQM technique, we allow mobile objects to adjust their computing capacity to reflect their processing capability at different time.

1. Introduction

Combining the popularity and availability of mobile computing technologies with the functionality of positioning technologies like global positioning system (GPS) enables new environments where virtually all objects of interest can determine their locations. These technologies are the foundation for pervasive location-aware services defined as applications that utilize the location of the user to deliver location-based information as needed. Location-aware services provide the basis for promising applications that virtually span every aspect of our everyday life. The emergence of location-aware services calls for new algorithm to cater for real-time processing of query in heterogeneous mobile database systems.

The type of query considered here is range-monitoring query defined in [1] as: Given a set of user-defined spatial regions, retrieve the mobile objects inside them and provide real-time update as the mobile objects move in and out of these regions. Example of application that will benefit from efficient range query processing is disaster alert system. In UK, SMS is used to alert the public of security threats such as bomb alerts. The public inside the affected area is notified real-time with messages to protect the public and guide them in case of an evacuation plan. In addition, respective department may want to monitor vehicles in some specific section of a route in some period of time to divert some vehicles to alternate routes to alleviate congestion. Another scenario is when a colloquium is about to start in a building conference room, for a half-hour before the talk starts, users with handheld devices can get talk notifications on talk title, speaker and abstract, and directions to the room as well. Besides these, range query offers a brand new advertising channel where companies can target promotions to nearby mobile users with the delivery of location specific advertising and coupons.

To achieve real-time query results, Ying Cai et al. [1] introduced a technique called Monitoring Query Management (MQM). Basically, each mobile object is assigned a resident domain, based on its current location, and is notified of the queries that overlap with the domain. Since an object knows its resident domain and the queries inside it, the object can monitor its spatial relationship with them while it moves. When it detects that it has crossed over some query boundary, the object contacts the server to update the affected query results. In addition, when an object moves out of its resident domain, it needs to report to the server, which will then determine a new resident domain for the object. Furthermore, BP-tree (Binary Partitioning Tree) access method allows efficient query management at the server side.
However, although the mobile objects distributively evaluate the queries, MQM is not completely distributed where all the moving objects must communicate with a single centralized computer, which becomes a bottleneck [2]. To address this problem, in this paper, we propose a distributed server infrastructure, namely Distributed Monitoring-Query Management (DMQM) to dynamically partition the whole domain into a set of service zones and distributively share the workload of processing range-monitoring queries. Besides that, to make MQM technique even more flexible, mobile objects are allowed to dynamically adjust their computing capacity with the server to reflect their true processing capability at the time.

The rest of this paper is organized as follows. Section 2 presents the previous works related to the study. Section 3 explains our proposed distributed server infrastructure, Distributed Monitoring-Query Management (DMQM). The performance study, simulation model and the simulation results are described in Section 4. Conclusions and future work are presented in the last section, 5.

2. Related Work

There is an extensive research that addressed continuous query processing. Mainly, the approaches investigated can be classified into 4 categories:

(1) Result Validation. When a query arrives at the server, the Voronoi diagram is used to efficiently compute the result [3]. In addition to the result, the server sends back to the server the validity time \( T \) of the result, which is the time that the query point will cross the closest boundary of the Voronoi cell of object \( o \). [4] returns a validity region instead of time, that enables mobile objects to determine the validity of previous queries based on their current locations. However, it is difficult to maintain the estimation of valid time or valid region for querying moving objects.

(2) Result Caching. The main idea revolves around using previously cached results to start the search for new results of \( k \)-nearest neighbor queries [5], [6] and range queries [7]. \( K \)-nearest neighbor shares some similarity with DMQM in the sense that given an object, \( k \)-nearest neighbor tries to retrieve \( k \)-nearest neighboring objects while DMQM tries to find \( n \) query rectangles near to the object’s current location. Existing \( k \)-nearest neighbor algorithms, however, were developed for retrieving objects as points in space and cannot be applied directly on spatial rectangular data or in this case, query rectangles. MobiEyes proposed in [7] is similar to DMQM in that MobiEyes exploits the object computational capabilities in order to reduce the processing load of the server. MobiEyes, however, relies on a centralized server infrastructure whereas DMQM is completely distributed.

(3) Result Prediction. If the trajectory of the query movement is known in prior, the query result can be computed in advance. Once the trajectory changes, the query will be reevaluated. In [8], the query result is computed using computational geometry for stationary objects while the concern of DMQM is on moving objects. It is proposed in [9] an algorithm for \( k \)-nearest and reverse \( k \)-nearest neighbor queries on the current and anticipate future positions of points moving continuously in the plane. Again, \( k \)-nearest neighbor algorithm retrieves point data instead of spatial rectangular data.

(4) Incremental Evaluation. To achieve incremental evaluation, only the updates of the previously reported query result are computed. Scalable Incremental Hash-Based Algorithm (SINA) [11] introduced two types of updates, namely positive and negative updates to indicate whether a certain object should be added to or removed from the previously reported result, respectively. SINA, however, applies a centralized approach as opposed to DMQM. Shared Execution Algorithm for Continuous \( K \)-Nearest Neighbor SEA-CNN [10] entails that only queries whose results are affected by the motions of objects or queries are reevaluated. SEA-CNN shares the traits of \( k \)-nearest neighbor meaning it is developed for point data which is unsuitable for DMQM as mentioned above.

3. Distributed Monitoring-Query Management (DMQM)

The idea of DMQM is to introduce a distributed server infrastructure of MQM, partitioning the entire database domain into service zones, each controlled by a server and cooperatively processing the requests of range-monitoring queries. In addition, we let the mobile objects adjust their processing capability instead of fixing it at their initialization as in MQM. The detailed designs of these two features are as follows.

3.1. Service Zone

We leverage the design of BP-tree, which partitions the whole database domain \( D \) into a set of
Figure 1. An example of a domain with 2 servers

subdomains, to locate subdomains to be assigned a server. The subdomain $d$ handled by a server is termed as service zone $s$. The domain partitioning is based on binary partition approach, i.e., the domain is split into two equal-sized subdomains. The domain is split either vertically or horizontally decided by comparing the dimensions of the domain and splitting the longer dimension. When a new server arrives, we search the BP-tree starting from the root. If the domain contains only one server at the time, we assign the left subdomain and right subdomain of root domain to the existing server and new server each, in no particular order. Figure 1 shows an example of the domain containing 2 servers. $s_1$ denotes service zone 1 and $s_2$ denotes service zone 2, each controlled by a server. If there is more than one server when a new server arrives, we compare the existing service zones and split the subdomain containing more monitoring regions as to balance the workload between the servers. Then the new server and the existing server in the split subdomain will take the left and right subdomains each.

After a server is assigned a service zone, we pass to the server all the monitoring regions inside the service zone and the corresponding part of BP-tree with the subdomain acting as the root. Mobile objects register with the new server if their current location is controlled by a server. When a server leaves the system, we need to ensure that the corresponding service zone is taken over by another server. The departing server explicitly hands over its repository of mobile objects and monitoring regions to its neighbor whose service zone can be merged with the departing server’s zone to produce a valid single service zone while the BP-tree of the two neighboring zones are merged with the merged subdomain acting as the root.

3.2. Processing Capability Adjustment of Mobile Object

The following notations are used in the discussion of these components:
- $myID$: the unique identifier of the mobile object;
- $myPos$: the current position of the mobile object;
- $myCapacity$: the maximum number of monitoring regions acceptable to the mobile object.

If the object needs to change its $myCapacity$ to reflect its processing capability at the time, e.g. the object requires more CPU cycles and/or memory for other tasks with higher priorities, it can request from the server a smaller/higher resident domain using a smaller/higher $myCapacity$ by sending the server a message:

$RequestResidentDomain(myID, myPos, myCapacity)$.

In response to this request, the server searches the BP-tree to determine a new resident domain for the mobile object. The server then broadcasts the message $SetResidentDomain(oid, d, l)$, where $d$ and $l$ denote the new resident domain of the object $oid$ and the list of monitoring regions inside $d$, respectively. When the object receives this message, it sets its new resident domain and monitoring regions accordingly.
Table 1. Parameters used for the simulation studies

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>DEFAULT</th>
<th>RANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain space</td>
<td>[0…100K, 0…100K]</td>
<td>N/A</td>
</tr>
<tr>
<td>Number of mobile objects</td>
<td>500</td>
<td>100 – 1,000</td>
</tr>
<tr>
<td>Number of monitoring regions</td>
<td>50,000</td>
<td>10,000 – 100,000</td>
</tr>
<tr>
<td>Skew of mobile processing capability</td>
<td>0.5</td>
<td>0.1 – 1.0</td>
</tr>
<tr>
<td>Mobile processing capability (queries)</td>
<td>N/A</td>
<td>50 – 500</td>
</tr>
<tr>
<td>Sizes of monitoring queries</td>
<td>N/A</td>
<td>10x10 – 100x100</td>
</tr>
<tr>
<td>Velocities of mobile objects</td>
<td>N/A</td>
<td>0 - 20</td>
</tr>
<tr>
<td>Velocity skew factor</td>
<td>0.7</td>
<td>N/A</td>
</tr>
<tr>
<td>Simulation time</td>
<td>10,000</td>
<td>N/A</td>
</tr>
</tbody>
</table>

4. Performance Study

We carried out simulation to compare the performance between MQM and DMQM:

- **MQM**: Monitoring Query Management technique proposed by Ying Cai et al. [1] which allows mobile objects to monitor their movement directly against their nearby queries to achieve real-time processing of range-monitoring queries.

- **DMQM**: Distributed Monitoring Query Management technique proposed in this paper which is a distributed server version of MQM. Since we are also interested in how the overall system performance is improved as the number of servers increases, we implemented 4 versions of DMQM:
  - **DMQM 1**: DMQM with 1 server which is basically the same as MQM which also operates on a single server except that under DMQM, an additional feature is added where processing capacity of mobile objects can be changed.
  - **DMQM 2**: DMQM with workload distributed among 2 servers.
  - **DMQM 3**: DMQM with 3 servers.
  - **DMQM 4**: DMQM with 4 servers.

The performance metrics selected for this study are the same as performance metrics of MQM which consist of: (1) **Server processing cost** measured as the total number of index-tree nodes accessed in order to process requests from the mobile objects, (2) **Server communication cost** measured as the total number of messages transmitted from the server to the mobile objects, and (3) **Mobile communication cost** measured as the total number of messages sent by the mobile objects to the server. The server processing cost and server communication cost are good indicators of whether the server can become bottleneck, measuring system scalability while mobile communication cost is a good indicator of power consumption.

4.1. Simulation Model

The simulation program is written using Microsoft C. For each simulation run, we

1. Generated range-monitoring queries with sizes ranging from 10 x 10 to 100 x 100, following a uniform distribution,
2. Generated mobile objects with computing capability ranging from 50 to 500 monitoring regions and velocity falling in between 0 and 20 per time unit, following a zipf distribution,
3. Placed them randomly over a rectangular database domain of [0…100K, 0…100K],
4. Partitioned the domain into service zones as needed,
5. Set the initial moving directions of mobile objects randomly,
6. Moved mobile objects linearly until any of the mobile objects reach any one of the four boundaries of the database domain, in which case the direction is reflected and continues to move at the same speed,
7. Adjusted the computing capability of mobile objects randomly to higher or lower capacity at some random simulation time units, and
8. Repeated Step 6 and Step 7, and ended at 10,000 simulation time units after which point, the factor of the performance gap between MQM and DMQM became rather stabilized.

We assume that the velocity of each mobile object is constant throughout the simulation. Table 1 summarizes the parameters used for the simulation studies.
Figure 2. Scalability with regard to the number of monitoring queries. (a) Server communication cost. (b) Server processing cost. (c) Mobile communication cost. (d) Categorized mobile communication costs.

Figure 3. Scalability with regard to the number of mobile objects. (a) Server communication cost. (b) Server processing cost. (c) Mobile communication cost. (d) Categorized mobile communication costs.
4.2. Simulation Results

Similar to MQM, we are interested in the scalability and robustness of the proposed technique DMQM. Therefore, we study how the performance metrics are affected by the number of monitoring queries, the number of mobile objects, and the skew factor of mobile computing capability. In addition, we examine how the metrics will perform with increasing number of servers as stated previously. We report and discuss the results as follows.

4.2.1 Scalability with Regard to the Number of Monitoring Queries. In this study, we generated 500 mobile objects and set the skew factor of mobile computing capability to be 0.5. We increased the number of monitoring queries from 10,000 to 100,000. The performance results are plotted in Figure 2. They show that DMQM 1 performs worse than the MQM approach while DMQM 4 performs the best followed by DMQM 3 and DMQM 2, respectively. Under DMQM, mobile objects are allowed to adjust their computing capacity to reflect their processing capability at different times. Each time an object changes its computing capacity; it needs to make a request for new resident domain. Upon receiving the request, the server needs to search the BP-tree to find a new resident domain. These result in high communication and processing costs. This condition does not apply to MQM hence DMQM 1 performs worse than MQM although both are similar in the sense that both apply a centralized or single server infrastructure. DMQM 2 performs much better than MQM since the workload is distributed among two servers. The performance steadily increases as the number of server increases as showed in DMQM 3 and DMQM 4, respectively. There are two types of messages sent by mobile objects: UpdateQueryResult and RequestResidentDomain. The former message is sent when an object crosses a query boundary while the later one is sent when an object moves out of its current resident domain. As explained earlier, DMQM 1 sent out more RequestResidentDomain messages than MQM. However, both techniques generate the same number of UpdateQueryResult messages, as showed in Figure 2d, because an object remains in the same position when requesting for new resident domain thus eliminating the chance of moving in or out of a query boundary.
4.2.2 Scalability with Regard to the Number of Mobile Objects. In this study, we generated 50,000 monitoring queries and set the skew factor of mobile processing capability to be 0.5. The number of mobile objects is varied from 100 to 1,000. The simulation results are plotted in Figure 3. As the number of mobile objects increases, all five approaches incur higher communication and server processing costs. DMQM 4 performs the best followed by DMQM 3, DMQM 2 and MQM while DMQM 1 performs the worst. Again, DMQM 1 performs the worst is due to the fact that mobile objects send out more requests for new resident domain as they adjust their processing capacity. This performance study again shows that distributing the workload among several servers can significantly reduce the communication and server processing costs even when allowing mobile objects to adjust their processing capacity. We can see that both communication and server processing costs steadily decreases as more servers are added. This shows that DMQM is highly scalable in terms of supporting more mobile objects.

4.2.3 Effect of the Skew of Mobile Computing Capability. This study evaluated how the performances of the techniques are affected by the skew of mobile computing capability. We generated 500 mobile objects and 50,000 monitoring queries. The skew of mobile computing capability is varied from 0.1 to 1.0, where a higher skew means a higher average of mobile processing capability. The simulation results are plotted in Figure 4. All five approaches incur less communication and server processing costs as the skew factor increases as mobile objects become more and more capable on average in loading monitoring queries. This study confirms again that distributing the workload among several servers can effectively reduce both communication and server processing costs.

5. Conclusion

With the growing popularity of location-aware services, it spurred a great research interest in real-time query processing. In this paper, we took the challenge of providing Distributed Monitoring Query Management (DMQM) technique for real-time processing of range-monitoring queries. DMQM added two new features to the existing Monitoring Query Management (MQM) technique: (1) Distributed server infrastructure is applied and (2) Mobile objects are allowed to adjust their computing capacity. Following are the advantages of the proposed technique:

- **Flexibility**: Mobile objects can adjust their processing capability to tailor to their needs.
- **Scalability**: DMQM is able to support location-based services with a large number of mobile objects and monitoring queries without burdening server with overwhelming workload.
- **Reliability**: DMQM is more reliable in the sense that there is no single point of failure.

At the moment, DMQM only processes stationary queries on moving objects. For our future work, we plan to include processing moving queries on moving objects. We would like to introduce a single algorithm with single access structure that can handle a wide variety of queries in a location-aware server thus eliminating the need to maintain different algorithm and access structure to accommodate different type of queries, which will degrade the performance of the location-aware server. Furthermore, since the processing capability of mobile objects is adjusted randomly as well as the frequency of the adjustment, we will investigate the performance of the proposed schemes using different distributions.

References


