Partial Global Indexing for Location-Dependent Query Processing

P

James W. Jayaputera
Monash University, Australia

INTRODUCTION

Mobile computing devices, which are small-size computers, enable mobile users to run their applications to access information via wireless communication anywhere at anytime (Barbara, 1999; Dunham & Kumar, 1998). Applications such as stock or banking activities, weather forecasting, entertainment, and navigation systems have become increasingly popular in recent years (Mobile Computing Horizontal Applications Index, 2006). This implies that people can still access information through their mobile devices without worry about their current location. This situation enables the capability of mobile users to utilize information services easily. However, these mobile computing devices have limited battery power and bandwidth. Hence, speeding up the processing of information access and retrieval is a major challenge to minimize the utilization of battery power and available bandwidth.

In performing the many popular applications mentioned earlier, location-dependent information services is one type of mobile computing capability for providing information based on the current location of mobile users (Baihua, Lee, Xu, & Lee, 2002; Dunham & Kumar, 1998; Waluyo, Srinivasan, & Taniar, 2005). This means that the relaying back of a query result changes according to the current location of the mobile user. If, for example, a mobile user would like to locate the closest restaurant, a list of the closest restaurants depends on the current location of the mobile user. The records in the list change as the location of the mobile user changes.

The objects residing in several servers are partitioned by their locations. This indicates that every object is unique to every server. The objects can be classified into *static* and *dynamic*. An example of a static object is an ATM (automatic teller machine) or building. In contrast, an example of a dynamic object is a moving car or running thief. In this article, we concentrate on static objects.

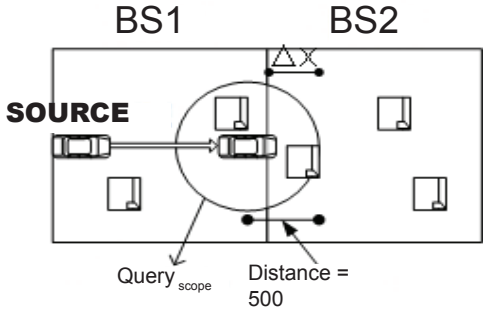
To speed up the searching process, these objects are indexed and the indexes are put into a data structure. There are several popular data structures, such as B+-Tree (Elmasri & Navathe, 2004), Quadtree (Samet, 1984; Tayeb, Ulusoy, & Wolfson, 1998), and R-Tree (Guttman, 1984; Manolopoulos, Nanopoulos, Papadopoulos, & Theodoridis, 2005).

In a mobile computing environment, a specific area is called a *cell* and is served by a base station (BS). In a simple situation, a requested query scope can be differentiated into two types—that is, *within* or *crossing* the current cell. In the former, the access time to process a query within a current cell depends on the queuing and processing times. In the latter, the access time to process a query crossing the current cell is similar to the first case, but the queuing time of each cell varies. Here, the terms *queuing* and *crossing* are used interchangeably.

Figure 1 shows an overview of LDIS in a simple situation. For example, a user sends a *location-dependent query* (LDQ) while driving—that is, it asks for data which can be found within the current location of the user. The requested data is called *location-dependent data* (LDD) (Jayaputera & Taniar, 2005). The LDQ sent is to find some vending machines within 500 meters from the current location.

The query is received by a base station, which forwards the message between the wireless and wired network for a specific area, is called a *cell* (Goodman, 1998). We assume that a BS is connected to a single server. From the figure, the query scope is represented by a circle which shows the search area of the query. Before the server starts the searching process, it needs to know the location of the mobile user from the GPS (global positioning satellite) (Getting, 1993). We assume that the location information taken from the satellite is accurate. Once the recipient's location is received, the searching process begins. In our example, the query scope crosses the area of BS1, which needs to ask for the information from BS2 on behalf of the mobile user. Once BS1 has the requested information, it forwards the information to the mobile user. The correct answer to this query should be the information about vending machines located inside the circle.

Figure 1. Overview of LDIS



In a complex situation, the scope of a requested query might cross a number of cells. The main consideration is that the scope of a requested query in a complex situation clearly differentiates between two types of scopes mentioned earlier: within or crossing the current cell.

This article concentrates on how to reduce the access time to retrieve information about static objects in a complex situation. The aim of this article is not to discuss concurrent search trees with their related problems and solutions, but to investigate how the practice of data/table partitioning in parallel databases can be applied in a mobile computing environment. As an example, the first query would be "Tell me a restaurant within 500 meters" and the second query: "Tell me a restaurant within 400 meters." Two mobile devices, which are located nearby, receive the results of these two queries, and the queries' scopes cross the current cell. In this scenario, the BS of the current cell needs to forward the request to the crossing cells. We assume that each BS has knowledge of its own service area and its neighboring cells. Requesting the same data from neighboring cells is not efficient since the process depends on some factors, such as the processing and waiting times on each neighboring cell. In the LDIS environment, a short access time is an important issue since the mobile user moves to a new location very frequently before receiving an answer from a server. We argue that accessing data locally needs to be improved and requesting data from other neighboring cells should be minimized.

The idea of this article is based on the parallel indexing concept (Taniar & Rahayu, 2002) in which an indexed object residing in a BS is either fully, partially, or not replicated to others BSs. Therefore, every server contains either partial or all indexes of other servers. In our proposed approach, whenever the requested results return from neighboring cells, we append the resulting items to the current cell. This implies that when the next user sends a request, the current cell needs to look up its own index first to verify if the data is in its local storage. If the data is not present, the current server sends a request to the neighboring cells on behalf of the client; otherwise, the current server directly sends the requested query to the client. We have evaluated our proposed approach and showed that the access time can be reduced by a factor of two.

The next section of this article describes some related work. We then describe our proposed work and the simulation model, and we compare the performance of our proposed technique to other techniques. Finally, we conclude the article and suggest future work.

RELATED WORK

Jayaputera and Taniar (2005) have shown the efficiency of using a square as a valid scope, which is an area in which the objects are valid for a certain time. They have shown that a square is the easiest shape to use.

Taniar and Rahayu (2002) discussed global indexing for a parallel database. The purpose of their approach was to have a global index of database servers. Whenever there is an update for an item in one server, the global index needs to be updated in all servers. The updating process for every server will increase the CPU load, where it can affect the performance of processing the LDQ. Therefore, this is not a practical choice to be applied directly for LDIS application.

B+-Tree (Elmasri & Navathe, 2004) has been widely known as one of the data structures for index which contains sub-tree and leaf nodes. A sub-tree is formed by a collection of non-leaf nodes. A non-leaf node contains up to m keys and $m+1$ pointers to the nodes on the next level of the tree hierarchy. All nodes on the left-hand side of the parent node have key values less than or equal to the key value of the parent node. In contrast, the key values of the right-hand side nodes of the parent node are greater than the key values of the parent node. The most bottom nodes are called *leaf* nodes. Each leaf node contains up to m keys where every key has two pointers: to the actual data items and to the right-side neighboring leaf node.

Some researchers (Beckmann, Knegel, Schneider, & Seeger, 1990; Guttman, 1984; Sellis, Roussopoulos, & Faloutsos, 1987; Theodoridis & Sellis, 1994) used R-Tree and its variants to provide an efficient and dynamic index structure for spatial data. Some researchers (Sellis et al., 1987) have applied R-Tree to LDQ processing. R-Tree uses the *minimum bounding rectangle* (MBR) to group the closest objects together into a rectangle where every area has the least enlargement area. This data structure does not have a problem when a query range can be fitted into one rectangle. On the other hand, we need to have multi-way searching if a query range involves some rectangles.

Furthest away replacement (FAR) is one of the cache replacements proposed in Dunham and Kumar (2000). In their approach, they eliminated the data items located furthest away from the user and which will be evicted first since the users no longer need those objects. The other replacement method is using *timestamp,* which is the time at which data items are received by the cell. *Least recently used (LRU)* (Jelenkovic & Radovanovic, 2003) is used to eliminate data items that have the oldest timestamp.

PARTIAL GLOBAL INDEXING FOR OBJECT RETRIEVAL: PROPOSED ALGORITHM

In general, the data items are located in several cells; we assume that every cell has a single server, and users often