Adapting Relational Database Engine to Accommodate Moving Objects

Beng Chin Ooi Zhiyong Huang Dan Lin Hua Lu Linhao Xu

School of Computing
National University of Singapore, Singapore
{ooibc, huangzy, lindan, luhua, xulh}@comp.nus.edu.sg

Abstract

Past few years have witnessed considerable research efforts on moving objects databases. One main design criteria of the algorithms and data structures is cost effective integration into an existing DBMS. In this work, we extend an existing RDBMS, MySQL, to include a new indexing mechanism and query processing strategies with minimal alteration to existing codes, i.e., our implementation does not infiltrate into the MySQL core.

1 Introduction

Past few years have witnessed considerable research efforts on moving objects databases. These previous works so far have been emphasizing on different aspects including data modelling, novel indexing, efficient query processing and etc. Whereas from a practical perspective, a real system supporting moving objects storage and retrieval needs to take into account all these aspects and to integrate them in a systematic way. There are two approaches to implementing an MOD (Moving Objects Databases) system. One, a specialized system could be developed from scratch. Such a system is lean and efficient. However, it may offer less functionalities that are required in conventional business processing, and such approach is time consuming and costly. Two, an existing DBMS is extended to provide the required capability for handling spatio-temporal data and processing. Such an approach is less costly and its existing core database engine can continue to serve the conventional business processing, and complement the MOD applications. This is in line with the approach being taken in the commercial DBMS products where different cartridges are built on top of the data server for different applications. However, such an approach may be tricky as any major alteration to the data server may cause many of its core components to be affected. Further, it may not yield the best performance since riding on top of already bulky system offers less room for optimization.

In this paper, we will discuss our implementation for managing moving objects on top of a popular relational database system MySQL, namely SpADE system (*Sp*atio-temporal *A*utonomic *D*atabase *E*ngine for managing moving objects). Figure 1 illustrates an example of moving objects applications. In this example,

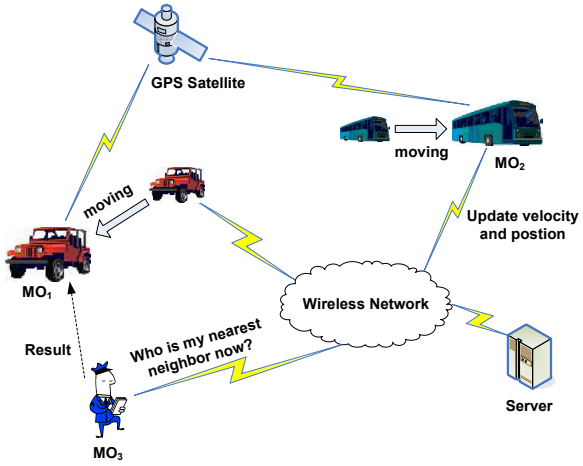


Figure 1. An Example of Moving Objects Applications

non-static entities like vehicles and pedestrians are abstracted as moving objects. They obtain positioning information with GPS (Global Positioning System) receivers installed, and are able to communicate via wireless network with the server, sending queries to and receiving results from it. The server is responsible for managing moving object information and processing queries from mobile users. By employing the industry

standard JDBC for the data access, our server can also support providing services for other application interfaces such as the Web.

Our proposal implements the B^x-tree [1] as a stored procedure in existing DBMS without touching the code. Although we have the source code of the MySQL, we keep the modification to the minimal. In our implementation, moving object data is transformed and stored directly on MySQL, and queries are transformed into standard SQL statements which are efficiently processed in the relational engine. Most importantly, all these are achieved neatly and independently without infiltrating into the MySQL core.

The remainder of this paper is organized as follows. Section 2 presents the system architecture with MySQL as the underlying relational engine. Section 3 concretizes the integration of the B^x-tree to the MySQL. Finally, Section 4 concludes the paper.

2 System Overview

This section gives an overview of the SpADE system that employs the traditional client-server model. Figure 2 demonstrates the system architecture.

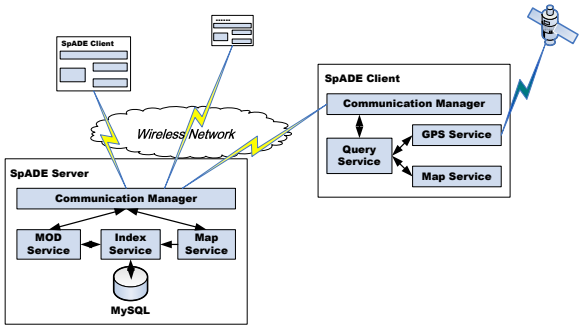


Figure 2. System Architecture

2.1 The SpADE Client

In the SpADE system, each client has a mobile computing device (e.g., PDA or high-end mobile phone) that is equipped with the SpADE client software and a GPS receiver. Thus, all these moving objects are able to report their latest velocity and position information to the SpADE server. Specifically, a SpADE client includes the following four components:

- ***GPS Service Module*** is responsible for receiving and parsing the GPS information. Therefore, each moving object can know its own latest velocity and position information.
- ***Map Service Module*** is responsible for displaying moving objects on a city map, listening to all events from the user, and downloading the latest city map from the SpADE server.
- ***Query Service Module*** is in charge of constructing the spatial-temporal queries according to the user's input or pen event on the map.
- ***Communication Manager*** listens to all network messages occurring between the client and the server and then dispatches them to the corresponding service module for further processing.

2.2 The SpADE Server

In general, the SpADE server is responsible for storing the velocity and position data of all moving objects and processing spatial-temporal queries issued by clients. Specifically, the SpADE server comprises the following five modules:

- ***Communication Manager*** receives query messages or GPS data updating requests from the clients, dispatches them to other modules, and returns query results to the client user.
- ***Map Service Module*** maintains the map data displayed at the client side and notifies all clients to update their map if a new map is provided.
- ***MOD (Moving Object Data) Service Module*** is used to define the motion pattern of the moving objects for different moving objects with different motion patterns. A distinguishing feature of the SpADE system is that it allows users to plug their preferred motion patterns to the server for precisely predicting the future position of the moving objects. This makes the SpADE system very flexible in terms of customizing the most suitable motion pattern for a particular moving object.
- ***Index Service Module*** is responsible for constructing and maintaining the indices used for processing spatial-temporal queries. Similar to the MOD service module, a user can add a particular index structure into the index service module for the motion pattern customized by the user. In our implementation, we adopt the B^x-tree.
- ***MySQL Database*** is used to store index and query the velocity and position data of all moving objects. It contains three types of tables, namely *user table*, *moving object table* and *static object table*. The user table stores all users registered into the SpADE system, which is used to evaluate the validation of moving clients; the moving object table stores moving object information (e.g. positions and velocities); and the static object table stores the road network information as well as buildings in the city.

3 System Implementation

In this section, we describe the implementation of the B^x-tree on top of the MySQL. We first briefly review the B^x-tree, and then discuss the implementation issues occurring in the indexing and querying phases respectively.

3.1 Data Modelling and the B^x-tree

A moving object is modelled by $O = (\overrightarrow{x}, \overrightarrow{v})$, a position and a velocity respectively. Every moving object updates its information to the data server when it feels necessary at time t_u.

The B$^+$-tree based B^x-tree employs space-filling curves to linearize the locations of moving objects. Attractive space-filling curves such as the Peano curve (or Z-curve), which we use in our system, preserve proximity, meaning that points close in the multidimensional space tend to be close in the one-dimensional space obtained by the curve [2]. Then, the B^x-tree effectively partitions the index by placing entries in partitions based on their update time. Updates in the same phase are mapped to the same so-called *label timestamp*. For an object with label timestamp t_{lab}, its position at t_{lab} is computed according to its position and velocity at t_u. This future position is then transformed to a 1-dimensional value by applying the space-filling curve. Finally, an object O updated at t_u is represented by a value $B^x value(O, t_u)$:

$$B^x value(O, t_u) = [index\_partition]_2 \oplus [x\_rep]_2 \quad (1)$$

where *index_partition* is an index partition determined by the update time, *x_rep* is obtained using a space-filling curve, $[x]_2$ denotes the binary value of x, and $\oplus$ denotes concatenation. The computation of the two values can be found in [1].

3.2 Implementation Issues

3.2.1 Relational Table Definition

First, we need to define for moving objects a proper relational table that can be indexed by the B$^+$-tree so that we can exploit its power when manipulating moving object information. For this purpose, we add a field B^x_Value (computed by equation 1) into the table of moving object and make it as the primary key indexed by the B$^+$-tree. A simple relation scheme is demonstrated in Table 3.2.1 with concise explanations. Extra fields can also be defined according to the practical needs in a specific application.

In our system, there are two such tables. One is to store the latest update information of the moving objects, called the *current information table*. Note that this table is indexed by the B^x_Value obtained according to the B^x-tree rationale. The other table simply keeps all the historical information, called the *historical information table*.

Field	Type	Note
id	integer	Object identifier
$B^x\_Value^*$	long	Value of formula 1
pos_x	double	Longitude of last update
pos_y	double	Latitude of last update
vel_x	double	Longitude speed of last update
vel_y	double	Latitude speed of last update
time	long	Update time

Table 1. Moving Object Relation Scheme

3.2.2 Processing Spatial-Temporal Queries

The SpADE system supports various types of queries such as historical (current/predictive) range and k nearest neighbor queries. The typical process of executing a spatial-temporal query will take several steps involving different system components, as shown in Figure 3.

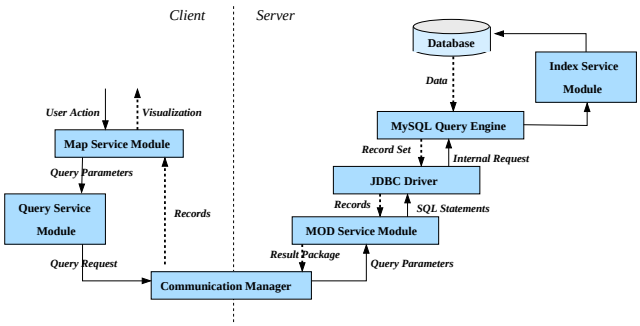
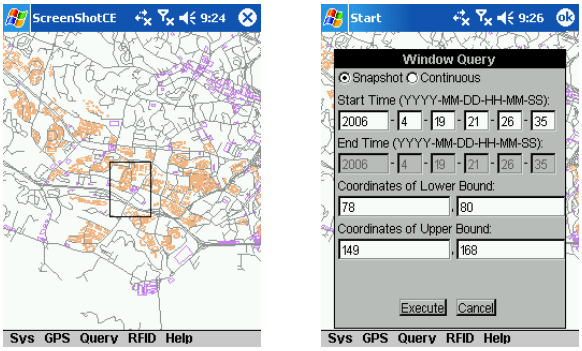


Figure 3. Execution of a Spatial-temporal Query

1. On the client side, a user first selects the query type (e.g. kNN or range query). The map service module then obtains the query parameters by listening to the user's pen event on the map. Figure 4(a) and (b) demonstrate how to form a query by drawing a window on the map.
2. Then, the query service module constructs the user query according to the query parameters and sends the query request to the server via the communication component.
3. Once receiving the query message, the server side communication part extracts query parameters and passes them to the MOD service module.



(a) (b)

Figure 4. An Example of Range Query on PDA

4. The MOD service module composes SQL statements according to the query parameters and the B^x-tree rationale, and passes the SQL statements to the MySQL query engine via the JDBC connection. For a query, an example SQL statement is shown below:

   SELECT $*$ FROM MO
   WHERE ($B^x\_Value \geq sub_0^{st}$ AND $B^x\_Value \leq sub_0^{ed}$)
   OR ...
   OR ($B^x\_Value \geq sub_m^{st}$ AND $B^x\_Value \leq sub_m^{ed}$);

5. Inside the MySQL, the index established will be exploited to retrieve data requested. The data will be returned to MOD service module via the JDBC connection.

6. Along the reversed direction, the data will be sent back to the client with necessary packing/unpacking operations inserting/extracting corresponding controlling bytes.

7. Finally, when the map service module on client side receives the desired moving objects with the velocity and position information, it can visualize the results on the map according to the user's preference specified beforehand.

3.2.3 Updating Spatial-Temporal Data

To query moving objects, the velocity and position of all moving objects must be kept fresh. The SpADE system provides two ways to update a moving object's spatial-temporal information. One is that users are able to set their preferred update frequency to determine how often to update their velocity and position data. When the setting time has expired, the latest GPS information will be automatically reported to the server. The other way is that the client queries the server at a regular time interval to check if the distance between the predicted location and the actual location exceeds the system threshold. If it is the case, the new GPS data will be updated to the server.

Generally, the SpADE system takes four steps to refresh the velocity and position information of all moving objects. In the first step, the client obtains the velocity and position information from the GPS service module and then transfers them to the server. In the second step, according to the identifier of the moving object, the old information is removed from the table and inserted into a history table. In the third step, the index service module calculates the index key of the B^x-tree in terms of the new velocity and position data. Finally, the new index key, velocity and position data are inserted into the MySQL database.

3.3 Performance Studies

The SpADE system resides at an IBM x255 server running Linux with four Intel Xeon MP 3.0GHz/400MHz processors and 18G DDR main memory. To evaluate the system performance, we simulate clients from 100K to 1M and measure the response time of each query and update. The results show that the performance pattern is similar to that reported in [1] (due to the space constraint, we do not include graphs here). This indicates that our SpADE system preserves the properties of the B^x-tree and is ready to support a large number of clients while providing high quality service.

4 Conclusion

In this work, we present our design on extending an existing DBMS to support spatio-temporal query processing. We present the architecture of SpADE. SpADE is based on a client/server architecture, within which the moving object data is stored and managed on the server on top of a relational database system called MySQL. On the server side, we made use of the the B$^+$-tree to implement the B^x-tree for indexing moving objects. We present implementation issues related to both updates and queries. Note that all these have been achieved cleanly without rewriting any main components of MySQL backend. It is obvious that our proposed design could be implemented on any proprietary commercial backends cost effectively.

References

[1] C. S. Jensen, D. Lin, and B. C. Ooi. Query and update efficient B$^+$-tree based indexing of moving objects. In *Proc. VLDB*, pages 768–779, 2004.

[2] B. Moon, H. V. Jagadish, C. Faloutsos, and J. H. Saltz. Analysis of the clustering properties of the hilbert space-filling curve. *IEEE TKDE*, 13(1):124–141, 2001.