A GIS DATABASE FOR TIME-EVOLVING SPATIAL OBJECTS

Dae-Soo Cho, In-Sung Jang, Kyoung-Wook Min, Jong-Hyun Park

LBS Research Team, Telematics Research Division, ETRI, 161 Gajeong-dong, Yuseong-gu, Daejeon, 305-600 Korea - (junest, e4dol2, kwmin92, jhp)@etri.re.kr

KEY WORDS: Spatial Information Sciences, Databases, Modelling, Query, GIS

ABSTRACT:

In this paper, we have designed a data model for moving objects and implemented it. The moving objects are time-evolving spatial objects, that is, their geometries are dynamically changed as time varies. Generally, a GIS database stores and manages the spatial objects, of which geometries are rarely changed. The traditional GIS database, therefore, has a difficulty to manage the moving objects, due to the fact of geometries being frequently changed and all of the history information of moving objects being managed. To manage moving objects efficiently, we have added new data types, such as moving point and moving polygon, to the traditional GIS data type. We have also defined the semantic of underlying operators for those data types. It is expected that the GIS database we have developed makes it feasible to developing a wide range of database applications managing moving objects, such as cars, aircraft, ships, mobile phone user, hurricanes, oil spills in the sea, forest fires, armies, and tribes of people in history.

1. INTRODUCTION

Recently, various types of location-based services have obtained increasingly high attention according to the extensive spread of mobile handset, which is capable of accessing wireless internet, and the development of location determination technology, that is represented by GPS (Global Positioning System). Location-based services are related the moving objects which change their locations through time. Therefore, to provide location-based services efficiently, it is required that an efficient system which could acquire, store, and query the large number of locations. The time-evolving locations of moving objects are not efficiently managed by existing commercial Database Management System (DBMS) as well as Geographic Information System (GIS). The reason is that there is a critical set of capabilities that are needed by moving objects database applications (Wolfson et al., 1998), such as location-based services, and are lacking in existing DBMS and GIS. The needed capabilities are location data model for moving objects, query language for moving objects, location index for moving objects, and so on.

Previous works for moving objects can be classified into two main groups; works related to location data models and query languages (Sistla et al., 1997; Forlizzi et al., 2000; Wolfson et al. 1998; Güting et al., 2000) and works related to indexing locations (Pfoser et al., 2000; Kollios et al., 1999; Nascimento and Silva, 1998; Vazirgiannis et al., 1998; Song and Roussopoulos, 1987). These works, also, can be classified by works for current and future locations (Sistla et al., 1997; Kollios et al., 1999; Wolfson et al. 1998; Song and Roussopoulos, 1987) and works for trajectories (past locations) of moving objects(Pfoser et al., 2000; Forlizzi et al., 2000; Nascimento and Silva, 1998; Vazirgiannis et al., 1998; Güting et al., 2000). Other type of previous works to is related to generate synthetic data (Pfoser and Theodoridis, 2000; Theodoridis et al., 1999; IBM). Location data generator, which is capable of simulating real-world moving objects, are needed because it is not possible to obtain real datasets, either they do not exist or they are not accessible.

The purpose of this paper is to design and implement the overall architecture of a Moving Objects Database (MODB) which is applicable to the real-world applications. We have integrated various kinds of works related to moving objects into the MODB. The rest of the paper is organized as follows: Firstly, we will discuss the overall architecture of MODB. Then, we will explain each of six modules of which the system consists. Finally, we will conclude by giving directions for future work.

2. OVERALL ARCHITECUTRE OF MODB

The Moving Objects Database (MODB) devised in this paper is depicted by Figure 1. It is composed of six modules, Intelligent (location) Acquisition Module (IAM), a Query Processing Module (QPM), a Buffer Management Module (BMM), a Location Indexing Module (LIM), a Location Storage Module (LSM), and an Attribute Storage Module (ASM).

Intelligent Acquisition Module (IAM): According to the location acquisition policies we are proposed such as static acquisition policy, distance-based acquisition policy, region-based acquisition policy, and predict-based acquisition policy, IAM acquires the current location of moving objects and reports it into the QPM. The policies determine when IAM acquires the location of a moving object and how many threads IAM uses to acquire the locations of all objects. The objective of IAM is as follows. When location based services prevail into the wireless internet applications; we can easily predict that transmission overhead is so heavy to acquire the locations of large subscribers and vehicles between MODB and location server. To solve this problem, MODB must support IAM that lessens transmission overhead as much as possible and guarantees stable system state. So, we have proposed the techniques of minimizing overheads of transmission in acquiring locations of so many moving objects.

Query Processing Module (QPM): First of all, we defined query interfaces to issue user's request. Also, we defined the moving objects model, which is composed of data structures and operations to represent the moving objects. User's request

issued by query interfaces are executed by QPM based on a moving objects model.

Buffer Management Module (BMM): Because of high cost of insert transaction, it is difficult for the LSM to process every requests of location insertion from QPM directly. This is the reason why BMM exists. We designed every requests transferred from QPM into LSM through BMM. BMM do not issue insert request to LSM immediately. Instead, it maintains a memory buffer to gather insertion requests for a specific period, and issue one insertion request to LSM.

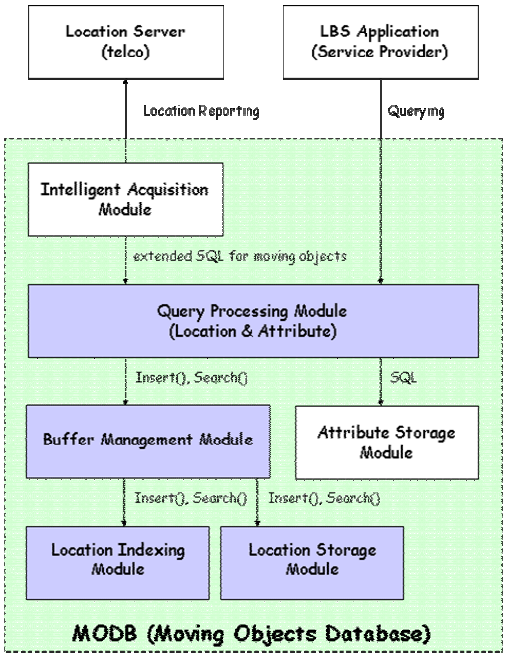

Figure 1. Overall Architecture of Moving Objects Database

Location Indexing Module (LIM): To efficiently search moving objects with some spatio-temporal predicates, MODB should have dedicated indexes for moving objects. LIM could manage several indexes by the Time Segmented Indexing method, which is newly devised in this paper.

Location Storage Module (LSM): Location information that is reported from IAM is permanently stored into location storage managed by LSM through the memory buffer (temporary storage).

Attribute Storage Module (ASM): The traditional database systems, such as MS SQL Server or Oracle, are used to store the attributes of moving objects without any additional efforts. We, therefore, have no comment about ASM especially.

We will give detail explanation about each module except IAM and ASM in the following sections.

3. QUERY PROCESSING MODULE (QPM)

The Query Processing Module should define location data model and location query language for the moving objects. The location data model means data structure and operations for the moving objects. In this paper, we revised the previous works, and newly defined a data model, which is composed of a lot of classes such as MPoint, MLineString, and MPolygon. In this paper, we focus on MPoint.

3.1 Moving Objects Data Model

In this section, we describe class model for moving objects components using UML. This supports time-series operations as well as continuous moving objects operations.

(1) Data Model

Class package for moving objects components (see Figure2) consists of four major components. ETRITime, ETRIGeometry, and ETRIMGeometry component is a set of classes supporting temporal operations, spatial operations, and moving objects operations, respectively. OGISGeometry component which is very similar to ETRIGeometry component is just included in our package design, because we had already implemented it for another project.

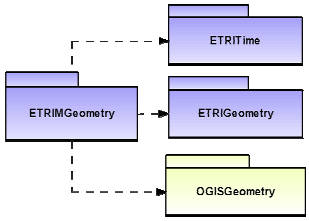

Figure 2. Class Package for Moving Objects Components

As shown in figure 3, ETRITime component has four types of temporal classes. These classes have interfaces such as ITemporal, ITemporalRelation, and ITmporalOperator. Each interface consists of several operations such as figure 4.

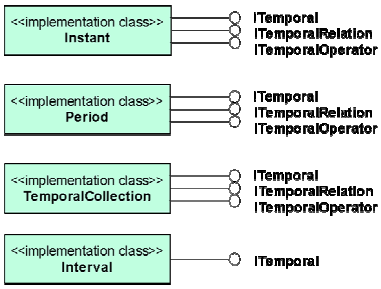

Figure 3. Temporal Classes

ITemporal	ITemporal **Duration**(); ITemporal **Clone**();
ITemporalRelation	bool **Precedes** (ITemporal other); bool **Overlaps** (ITemporal other); bool **Contains** (ITemporal other); Bool **Meets** (ITemporal other); Bool **Equals** (ITemporal other);
ITemporalOperator	bool **Intersection** (ITemporal other, ref object intersection); Bool **Union** (ITemporal other, ref object union); Bool **Difference** (ITemporal other, ref object difference);

Figure 4. Temporal Operations

ETRIGeometry component has several geometry classes such as Point, LineString, Polygon, GeometryCollection, Surface, and et al. UML modeling for these classes is borrowed from that of International Standard of Open GIS Consortium for geographic

information system (Open GIS, 1999). We omit, therefore, the class hierarchy and operations in this paper.

Figure 5 and Figure 6 show the classes for moving objects and the operations of some of major interfaces, respectively.

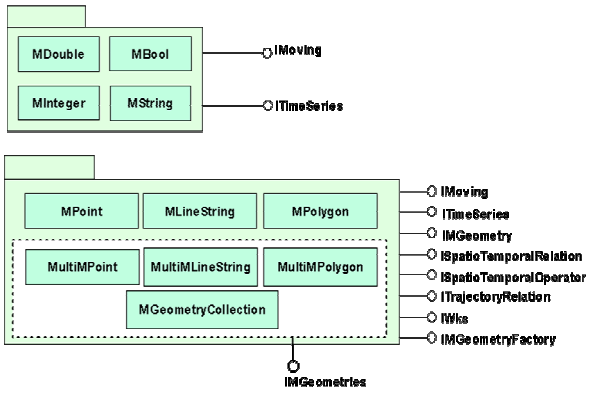

Figure 5. Moving Objects Classes

IMoving	// Interaction function
	object **Snapshot** (Instant instantTime);
	object **Slice** (Instant fromTime, Instant toTime);
	Time[] **SnapshotByValue** (object dataValue);
	Time[] **SnapshotByValue** (int columnIndex, object dataValue);
	Time[] **SliceByValue** (object fromValue, object toValue);
	Time[] **SliceByValue** (int columnIndex, object fromValue, object toValue);
	// projection function
	object **Project** ();
	Time **LifeTime**();

ITimeSeries	int **Count**{ get; } int **Add** (MOInstant instant, object dataValue); int **Add** (TObject instantObject); int **Index**(TObject instantObject); TObject **Min** (); TObject **Min** (int columnIndex); TObject **Max** (); TObject **Max** (int columnIndex); TObject **First** (int k); TObject **Last** (int k); TObject **Next** (TObject currentObject); TObject **Previos** (TObject currentObject); TObject **After** (MOInstant currentTime); TObject **Before** (MOInstant currentTime); // return series object object **SubSequence** (int fromIndex, int toIndex); object **SubSequence** (MOInstant fromTime, MOInstant toTime);
ISpatioTemporalRelation	MBool **Equal** (IMGeometry other); MBool **Touch** (IMGeometry other); MBool **Contain** (IMGeometry other); MBool **Within** (IMGeometry other); MBool **Disjoint** (IMGeometry other); MBool **Cross** (IMGeometry other); MBool **Overlap** (IMGeometry other); MBool **Intersect**(IMGeometry other);
ISpatioTemporalOperator	MDouble **Distance** (IMGeometry other); IMGeometry **Boundary** (); IMGeometry **ConvexHull**(); IMGeometry **Buffer** (double distance); IMGeometry **Union** (IMGeometry other); IMGeometry **Difference** (IMGeometry other); IMGeometry **Intersection** (IMGeometry other); IMGeometry **SymmetricDifference** (IMGeometry other);
IMTrajectoryRelation	MOHRESULT **Enters** (IMGeometry other, ref bool isEnters); MOHRESULT **Leaves** (IMGeometry other, ref bool isLeaves); MOHRESULT **Passes** (IMGeometry other, ref bool isPasses); MOHRESULT **Insides**(IMGeometry other, ref bool isInsides); MOHRESULT **Meets** (IMGeometry other, ref bool isMeets);

Figure 6. Moving Objects Operations

A characteristic of operations included in ITimeSeries interfaces is the target of operation. It is not location information of moving points at any time operated but real location data stored in database. For example, the *First(int k)* operation returns *k*-th location information from first value of moving point, and the *After(MOInstance currentTime*) operation returns similarly the nearest location information after *currentTime* in parameter.

The operations of IMoving interface are different from those of ITimeSeries interface. IMoving interface provides functions of calculating location information at all time. For example, the *Snapshot(Instant instantTime)* returns location information of moving object in *instantTime*, and *Slice(Instant fromTime, Instant toTime)* which has period parameter returns moving object between given time period. The *Project()* returns value objects projected by time.

3.2 Moving Object Query Language

We also revised the SQL syntax to support moving object described in section 3.1. We add some data type to present moving object data model (see Figure 7).

Basic Data Type	MOID
Temporal Data Type	Instant, Period, TInterval
Spatial Data Type	Point, Line, Polygon, Rectangle, MultPoint, MultiLine, MultiPolygon, GeoCollection
Moving Data Type (Moving Value Type)	MInteger, MDouble, MBool
Moving Data Type (MGeometry Type)	MPoint, MLine, MPolygon, MRectangle, MultiMPoint, MultiMLine, MulitMPolygon, MGeoCollection

Figure 7. Data Types for Moving Object Query Language

In this section, we just describe some DML statements by examples. Base tables for all of the queries in this section are assumed as follows.

CelluarPhoneUser (id *MOID*, position *MPoint*)
Car (id *MOID*, poistion *MPoint*, type *string*)
People (id *MOID*, location *MPoint*, name *string*)
Region (id *MOID*, area *MPolygon*, name *string*)

(1) Snapshot Queries

In this section, the snapshot queries for moving objects are described. The snapshot query is usually used to search the locations of specific time.

Example 1: Find the current location of cellular phone user 1001 at time *t*.
Select *Project* (*Snapshot* (position, *t*))
From CelluarPhoneUser
Where id = 1001;

Example 2: Find the information on the moving object at point (x,y) at time *t*.
Select *
From CelluarPhoneUser
Where *Equals* (*Snapshot* (position, *t*), *Point* (10, 20)) = *TRUE*;

Example 3: Find 911 cars in Daejeon now.
Select id
From Car, Region
Where *Contains* (*Snapshot* (Car.position, *NOW*), Region.area) = TRUE *AND* Region.name = 'Daejoen';

Example 4: Find taxis within 1km from point(*x*, *y*) now.
Select id
From Car, Region
Where *Withins* (*Snapshot* (position, *NOW*), *Buffer* (*MPoint* (*NOW*, *x*, *y*), 1000)) = *TRUE AND* Car.type = 'Taxis';

Example 5: Find the closest *k* delivery trucks to the point(*x*, *y*) now.
Select id
From Car

Where *Nearest* (*Snapshot* (position, *NOW*), *Point* (*x*, *y*), *k*) = *TRUE*

(2) Slice Queries

In this section, we show the example of the slice queries for moving objects. The slice query requires the period parameter as follows.

Example 6: Find the location of cellular phone user 1001 between time *t1* and *t2*.
Select *Project* (*Slice* (location, *t1*, *t2*))
From People
Where id = 1001;

Example 7: Find people who were within 1km from point (*x*, *y*) between time *t1* and *t2*.
Select id
From People
Where *Withins* (*Slice* (location, *t1*, *t2*), *Buffer* (*MPoint* (*t1*, *t2*, *x*, *y*), 1000)) = *TRUE*;

Example 8: Find k people who were closest to the point (*x*, *y*) time *t1* and *t2*.
Select id
From People
Where *Nearest* (*Slice* (location, *t1*, *t2*), *Point* (*x*, *y*), 1) = *TRUE*;

(3) Trajectory Query

Trajectory query include the operations such as *Enter()*, *Leave()*, *Passes()*, *Insides()*, *Meets()*.

Example 9: Find people who were (at least once) in Daejeon between time *t1* and *t2*.
Select id
From People, Region
Where *Passes* (*Slice* (location, *t1*, *t2*), Region.area) = *TRUE AND* Region.name = 'Daejeon';

4. BUFFER MANAGEMENT MODULE (BMM)

The BMM plays an important role in enhancing the performance of the query of location insertion. Locations of moving objects are permanently stored into various types of database systems by LSM. In this environment, it is difficult for LSM to process every insertion request from QPM, because the cost of insertion transaction is very high in database system and insertion requests from QPM occurs very frequently. Therefore, buffering of insertion request and batch processing are very effective.

Another role of BMM is that when QPM issues a search request BMM searches moving objects in the memory buffer, transfer the request into LSM, and then bind the results from the memory buffer and those from permanent storage.

4.1 MORow Object

BMM stores locations of a moving object into a MORow object for insertion request from QPM. There is one-to-one relationship between a moving object and a MORow object. Therefore, a MORow object take a responsibility for buffering locations of a corresponding moving object. MORow object depicted in Figure 8 is composed of *MOID* (Moving Object IDentifier), *Length* (the number of locations stored in it), *MBR* (Minimum Bounding Rectangle of the locations), *From* (time that first location in it is acquired) and *To* (time that last location in it is acquired). The *MaxLocation* means the maximum number of locations stored in a MORow object. If the *Length* of a MORow Object is equal to the MaxLocation, then all of the locations in the MORow object are transferred to LSM in order to store permanently.

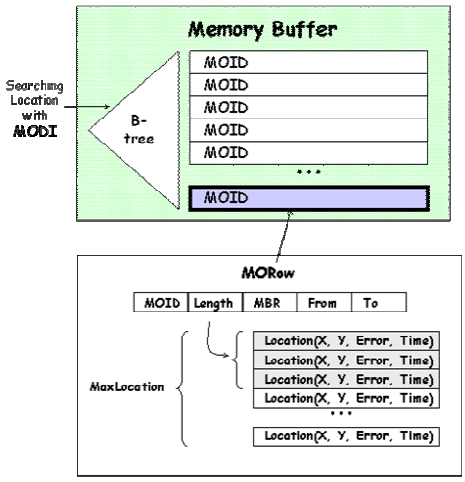

Figure 8. Structure of Memory Buffer

4.2 Memory Buffer

The overall structure of memory buffer is composed of a set of MORow objects. Each MORow object represents a trajectory of a moving object from From time to To time. As the figure indicates, there is a B-tree for indexing MOIDs of MORow objects. The BMM, therefore, finds efficiently a corresponding MORow object by using MOID.

5. LOCATION INDEXING MODULE (LIM)

According to the previous works (Pfoser et al., 2000; Kollios et al., 1999; Nascimento and Silva, 1998; Vazirgiannis et al., 1998; Song and Roussopoulos, 1987), there are three kinds of location indexes for moving objects.

Current Location Indexes: The indexes of this type take only current locations of continuously moving objects into consideration. And current locations are also used for anticipating future locations of moving objects. These indexes should have capabilities to process frequently updates of numerous moving objects.

Past Location Indexes for time interval (or time point) queries: The indexes, such as 3DR-tree and HR-tree, have a special purpose of efficient processing of a time interval (or time point) queries for the current and past locations.

Past Location Indexes for trajectory queries: The indexes, such as STR-tree and TB-tree, have a special purpose of efficient processing of a trajectory queries for the past locations.

LIM we implemented in this paper supports all kinds of indexes mentioned above. We implemented Adaptive Quad-tree (show Figure 9) as current location index. This index partitioned

spaces into sub-space, and then maintains a quad-tree for each sub-space.

We also implemented TB-tree, STR-tree, 3DR-tree, and HR-tree as past location indexes. But these trees for indexing past locations of moving objects are not applied in real world applications due to the problems described in following section.

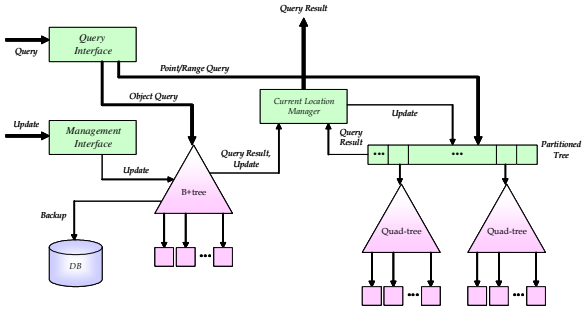

Figure 9. Current Location Indexes (Adaptive Quad-Tree)

5.1 Problems in Whole Indexing Method

The whole indexing method means that a tree is used to index all of the locations (including past locations) of moving objects. Figure 10 shows the whole indexing method. Most of past location indexes are based on R-tree, and therefore, are height-balanced trees.

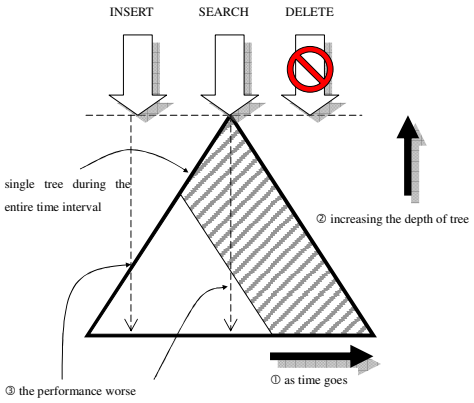

Figure 10. Whole Indexing Method

In whole indexing method, all of the locations of moving objects are managed by single tree during entire time interval. As shown in Figure 10, □ as time goes, □ increasing the depth of tree, makes □ the performance of INSERT and SEARCH operation worse.

Another problem is about the DELETE operation. The DELETE operation of R-tree variables requires the reorganization of the tree, if the number of entries of node N, which contained the deleted entry, is less than the minimum number (m) of entries. Because the reorganization of the tree requires the deletion and re-insertion of m-1 entries, DELETE is costly operation in this environment. Therefore, the deletion of entries during a specific time period, which is probably usable operation of managing the moving objects, is hardly performed.
Architecture of Time Segmented Indexing Method

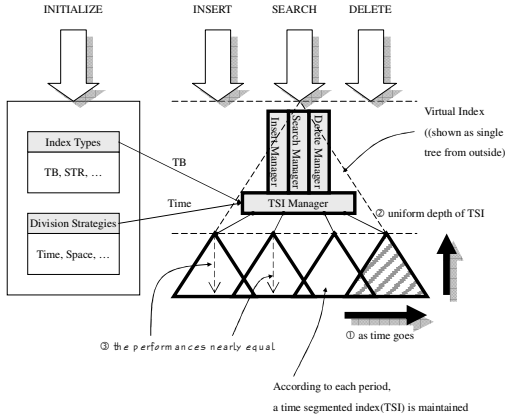

Figure 11. Time Segmented Indexing Method (Time Division Framework)

To solve the above problems, we revised the indexing framework (called Time Division Framework), which could be applicable regardless of the types of past location indexes. Figure 11 shows the overall architecture of time segmented indexing method. In this framework, an index tree is segmented into several TSIs (Time Segmented Index), but these TSIs compose a virtual index, which is shown as single tree from outside. A virtual index has also TSI manager, INSERT manager, SEARCH manager, and DELETE manager. Due to the space limitation, we do not explain the detail operations of the virtual index and the strategies of time segmentation.

6. LOCATION STORAGE MODULE (LSM)

The role of Location Storage Module (LSM) is to insert and search the locations of moving objects in efficient. As shown in Figure 12, LSM consists of a storage manager, a server manager, a connection manager, a disk manager and storage drivers.

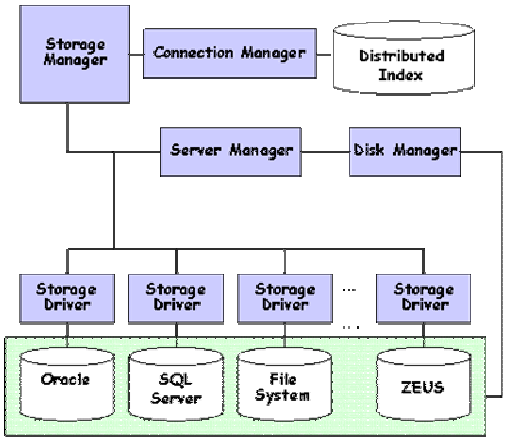

Figure 12. Location Storage Module

The Storage Manager has two main operations, INSERT and SEARCH, which are usually called by BMM. When BMM issues INSERT or SEARCH, the Storage Manager looks for a suitable storage system by referencing the Connect Manager and the Server Manager. Then, it calls INSERT and SEARCH of the storage driver, which is corresponded with the storage system.

The Server Manager maintains the registered storage drivers for balancing the loads of INSERT operations. That is, it checks the status of storage system corresponding with the storage driver, and then informs the Storage Manager which storage system is most available to execute INSERT operations.

The Connection Manager manages the distributed-index database that contains where locations are in distributed environment. When Storage Manager inserts locations into storage system through storage driver by issuing INSERT operation, Connection Manager inserts metadata into distributed-index database. In case of SEARCH operation, a Connection Manager looks up the servers in distributed-index database, and returns the list of relevant server information to Storage Manager.

The Disk Manager examines periodically storage systems, such as Oracle, MS-SQL Server, ZEUS, or etc., which are registered on Server Manager. When it detects one of the storage systems full, it prohibits Storage Manager from inserting into the storage system. Storage Manager, however, could access the storage system for the purpose of retrieving locations. Disk Manager also provides system administrator with several utilities, such as IMPORT, EXPORT, BACKUP and RESTORE.

The Storage Driver inserts locations of moving objects into corresponding storage system and search locations with specific predicates.

7. CONCLUSION

In this paper, we have designed the moving objects database for a large number of moving objects. And we have also implemented moving object components and SQL processing system on Microsoft Windows .Net Environment using C#. We integrated various kinds of works related to moving objects into one system, and newly proposed a data model for moving objects, a location query language, an indexing framework, and a method for storing moving objects. The system we proposed supports a diverse set of location acquisition policies, location indexes and location storages. Therefore, it is expected to be applied into various kinds of location based services practically.

As future work, we should make extensive experiments with real environments to measure the performance of MODB, and develop algorithms to enhance the performance of location query processing.

REFERENCES

Beckmann, N., and Kriegel, H. P., 1990. "The R*-tree: An Efficient and Robust Access Method for Points and Rectangles," In Proc. ACM SIGMOD, pp. 332-331.

Erwig, M., Guiting, R. H., Schneider, M., and Vazirgiannis, M., 1999. "Spatio-Temporal Data Types : An Approach to Modeling and Querying Moving Obejct in Databases," GeoInfomatica, Vol.3, No.3, pp.269-296.

Forlizzi, L., Güting, R. H., Nardelli, E., and Schneider, M., 2000. "A Data Model and Data Structures for Moving Objects Databases," Proc. ACM SIGMOD Conf. (Dallas, Texas), pp. 319-330.

Güting, R. H., Böhlen, M. H., Erwig, M., Jensen, C. S., Lorentzos, N. A., Schneider, M., and Vazirgiannis, M., 1998. "A Foundation for Representing and Querying Moving Objects," Fern Universität Hagen, Informatik-Report 238, September 1998, ACM Transactions on Database Systems, 25(1), pp. 1-42.

Guttman, A., 1984. "R-trees: A dynamic index structure for spatial searching," ACM SIGMOD Conference, pp. 47-54.

IBM, http://www.alphaworks.ibm.com/tech/citysimulator (accessed 29 Apr. 2004)

Jensen, C. S., Friis-Christensen, A., Pedersen, T. B., Pfoser, D., Saltenis, S., and Tryfona, N., 2001. "Location-Based Services - A Database Perspective,"' Proceedings of the Eighth Scandinavian Research Conference on Geographical Information Science, As, Norway, June 25-27, pp. 59-68.

Kollios, G., Gunopulos, D., and Tsotras, V. J., 1999. "On Indexing Mobile Objects," ACM Symposium on Principles of Database Systems, pp. 261-272.

Nascimento M. A., and Silva, J. R. O., 1998. "Towards historical R-trees," ACM SAC.

Pfoser, D., and Theodoridis, Y., 2000. "Generating Semantics-Based Trajectories of Moving Objects," International Workshop on Emerging Technologies for Geo-Based Applications, Ascona, Switzerland.

Pfoser, D., Jensen, C. S., and Theodoridis, Y., 2000. "Novel Approaches in Query Processing for Moving Object Trajectories," VLDB 2000, pp. 395-406.

Sistla, A. P., Wolfson, O., Chamberlain, S., and Dao, S., 1997. "Modeling and Querying Moving Objects," ICDE, pp. 422-432.

Song, Z., and Roussopoulos, N., 2001. "Hashing Moving Objects," MDM 2001, LNCS 1987, pp. 161-17.

Tao, Y. and Papadias, D., 2001. "MV3R-Tree: A Spatio-Temporal Access Method for Timestamp and Interval Queries," VLDB.

Theodoridis, Y., Silva, J. R. O., and Naschimento, M. A., 1999. "On the Generation of Spatiotemporal Datasets," CHOROCHRONOS Technical Report CH-99-01, Proceedings of the 16th Int'l Symposium on Spatial Databases (SSD).

Vazirgiannis, M., Theodoridis, Y., and Sellis, T. K., 1998. "Spatio-Temporal Composition and Indexing for Large Multimedia Applications," Multimedia Systems 6(4), pp. 284-298.

Wolfson, O., Xu, B., Chamberlain, S., and Jiang, L., 1998. "Moving Objects Databases: Issues and Solutions," SSDBM 1998, pp. 111-122.

Wolfson, O., Chamberlain, B. X. S., Sistla, P., Xu, B., and Zhou, X., 1999. "DOMINO: Databases fOr MovINg Objects tracking," ACM International Conference on SIGMOD , pp. 547-549.

Wolfson, O., Jiang, L. A., Sistla, P., Chamberlain, S., and Deng, M., 1999. "Databases for Tracking Mobile Units in Real Time," International Conference on Database Theory, pp.169-186.