Indexing Large Moving Objects from Past to Future with PCFI$^+$-Index

Zhao-Hong Liu
GIS Research Center
ChongQing Univ. of Posts & Tele. ChongQing, 400065
China (86)-23-62461581
macromliu@hotmail.com

Xiao-Li Liu
Adult Education College
CQUPT
ChongQing, 40065, China
(86)-23-62460400
amuletlizi@yahoo.com

Jun-Wei Ge
GIS Research Center
CQUPT,
ChongQing, 400065, China
(86)-23-62461548
gejw@cqupt.edu.cn

Hae-Young Bae
Dept. of Computer
INHA University
Inchon, 402-751, Korea
(82)-32-860-8712
hybae@inha.ac.kr

ABSTRACT

Ideally, moving object databases should handle the past, current and future positions of moving objects efficiently. However, existing indexes such as SEB-Tree, SETI-Tree, 2+3R-Tree, 2-3RT-Tree and etc. can only provide the capability for past and current query, and the others such as TPR-Tree, and TPR*-Tree can only provide the capability for current and future query. None of them can provide a strategy for indexing the past, current and also the future information of moving objects. In this paper, we present the Past-Current-Future$^+$-Index (PCFI$^+$-Index) which indexes the past, current & future information of the moving objects. The PCFI$^+$-Index builds upon the PCFI-Index which was based on SETI-tree and TPR*-tree. The PCFI$^+$-Index consists of two parts, in memory part with the name frontline, and disk based part. The whole region is partitioned into none-overlapping cells, and a spatial access method is used to index these cells. A set of main-memory TPR*-tree is used to index the moving objects that belong to the cells (one cell, one TPR*-tree). Considering the large update operation triggered by moving objects, the current data file which contains the moving objects' current information is organized as a hash index file. By keeping the restriction in SETI-Index, one page only contains the segments from one cell. Another sparse R*-tree is used to index the lifetimes of the pages. The performance analysis proves that the PCFI$^+$-Index can handle most of the queries efficiently and provides a uniform solution for the trajectory, time-slice, internal and moving queries, and has a better performance than the SETI-Index, TPR*-Index, and PCFI-Index.

1. INTRODUCTION

Spatio-temporal databases deal with large numbers of moving objects that change their location and/or shape continuously. For example: a spatio-temporal database is a collection of moving objects in the *D*-dimensional space. The moving objects get their own positions via location detection techniques such as GPS, TDOA, A-GPS, Hybrid-LDT. Then, the positions are reported to the tracking server using the underlying communication network, e.g., via wireless networks, GPRS, CDMA, and are stored in the database. With the continuously report, the tracking server keeps a large volume of historical trajectories reported by moving objects. In addition, the server stores additional information to help predict the future positions of moving objects. Typical queries that are supported by such a server include time slice queries e.g., "Find all objects that cross a certain area at time *t*" and window queries "Find all objects that cross a certain area in the time interval [t1, t2]". Time slice queries and window queries may ask about the past, current, or future positions. Some queries are concerned only with the past, e.g., trajectory queries "What was the maximum speed of a certain object in the last hour?" Other queries are concerned only with the future, e.g., moving window queries "Find the objects that intersect a moving area in a certain time interval". Numerous researchers have studies the development of spatio-temporal access methods as an auxiliary structure to support spatio-temporal queries. They can be classified into two categories: one is indexing the past and current which can answer the queries involving past and current positions, the other one is indexing the current and future which can answer the queries involving current and future well. But none of them can support the queries involved from past to future. [1]

In this paper, we present a new approach, named PCFI$^+$-index, which based on the SETI-tree and TPR*-tree to support the queries with the time vary from past to future. It is a extension of the PCFI-index.

Section 2 gives an introduction to the query types and the spatial temporal access methods proposed in past researches. Section 3 outlines our approach, the idea, the solution and the algorithm. Performance evaluations are done in Section 4, and conclusion and future work comes in Section 5.

[1] This research was supported by MIC (Ministry of Information and Communication), Korea, under the ITRC (Information Technology Research Center) support program supervised by the IITA (Institute of Information technology Assessment) and Sino-Korea GIS Research Center, China.

The author's full address: Sino-Korea ChongQing GIS Research Center, ChongQing University of Posts and Telecommunications(CQUPT), ChongQing, 400065, P.R.China

ADVANCES IN DATA MANAGEMENT 2005
Jayant Haritsa, T.M. Vijayaraman (Editors)

2. PROBLEM DEFINITION AND RELATED WORKS

2.1 Spatio-temporal Data Model (Trajectory)

Most conventional models for data representations are static in nature, but trajectory data for moving objects is continuously changing as the object moves. Representing the location of the object at all times is a challenge. A commonly used model for representing trajectory data approximates the motion of an object as a straight line segment between two consecutive updates [16]. In this paper, we also use this model for representing trajectories. The position and velocity of a moving object is sampled at discrete times, and a series of straight lines connecting successive positions is used to represent the movement of the object. In this paper, this line is referred to as a segment, and the sequence of the connected segments for a single moving object is referred to as a trajectory. Furthermore, to simplify the complexity, we will assume that an object moves in a two-dimensional space.

Stated more formally, a trajectory is represented as $tr_j(tid, u0, u1, u2 \ldots un \ldots)$, where *tid* is a unique trajectory id, and <u0, u1, u2 . . . un . . > is a sequence of points reflecting the positions of the moving object. Each point *ui* is a three-tuple $ui(xi, yi, ti)$, where x_i and y_i represents the spatial position of the object along the x and y dimension respectively at time ti. The only restriction on the sequence is that $u_i < u_{i+1}$ to ensure that the time parameter in the trajectory sequence is monotonically increasing.

A trajectory segment is represented as $si(tid, sid_i, u_{i-1}, u_i)$, where sid_i is a unique segment number for this segment of the trajectory (trivially one can set sid_i to i), and u_{i-1} and u_i are the two update end-points. The model can be easily extended to associate additional variables with each segment or update point; for example, in a trajectory data set of moving vehicles, each update point may have an additional reading recording the engine temperature at the time of the update.

2.2 Query Types

Based upon time, queries on moving data can be broadly classified into three categories: queries that ask questions about the historical positions of moving objects, queries that ask questions about the current positions of moving objects, and queries that ask questions about the future positions of moving objects. The second and third category of queries can be answered by storing current position, speed and the direction of the moving objects [5, 6] such as approaches in TPR-Tree[2], TPR*-Tree[11]. For the first category of queries, Pfoser et al. [19] further classified historical queries into two different sub-classes: coordinate-based queries and trajectory based queries. Coordinate-based queries include (a) time-interval, which select all objects within a given area and given time period, (b) time-slice, which select all the objects present in a given area at a time instant, and (c) nearest neighbor queries. Trajectory-based queries involve information about a trajectory such as topology and velocity. In the real world, the queries can belong to one of these categories, or combinations of the first and second category, or second and third category, or all of the three.

In this paper, we focus on coordinate-based queries in general, time-interval and time-slice queries in particular.

2.3 Related Works

2.3.1 Indexing the past and current positions

There has been a lot of work in this area. In some indexes such as MR-tree[13], HR-tree, HR+-tree[37], MV3R-tree[10], the temporal dimension is distinguished from the spatial dimensions. The goal is to keep all spatial data that are alive at one time instance together in one index structure (e.g., the R-tree). The ultimate goal is to build a separate R-tree for each time instance. This approach requires excessive storage. Some other indexes such as TB-tree[20], 2+3 R-tree[21], 2-3 TR-tree[1], SEB-tree[22] and SETI-tree[3] focus on trajectory-oriented queries, and can be classified into trajectory-oriented access methods.

The SETI-tree (Scalable and Efficient Trajectory Index) also has considered the scalability, focusing on indexing large moving objects. The SETI-index partitions the spatial dimensions into static, non-overlapping partitions. The main observation is that the change of the spatial dimension is limited while the temporal dimension is continuously evolving. Thus, the spatial dimensions are partitioned statically. Within each partition the trajectory segments are indexed using an R-tree. Line segments of the same trajectory stored in the same partition. Thus, trajectory preservation is achieved by minimizing the effect of the spatial dimensions in the R-tree. A segment that crosses the boundary of two spatial partitions is clipped at the boundary and is stored twice in both partitions. Each trajectory segment is a tuple in the data file with the restriction that one data page only contains trajectory segments that belongs to only one cell. Another sparse R*-tree is used to indexing the lifetime and every data page (one data page, one entry in the sparse R*-tree). A main-memory structure named *frontline* is used to maintain the last update location for all moving objects. It is organized with a hash structured index on the unique id of moving objects, and can be implemented with the persistent index such as hash or B^+-index.

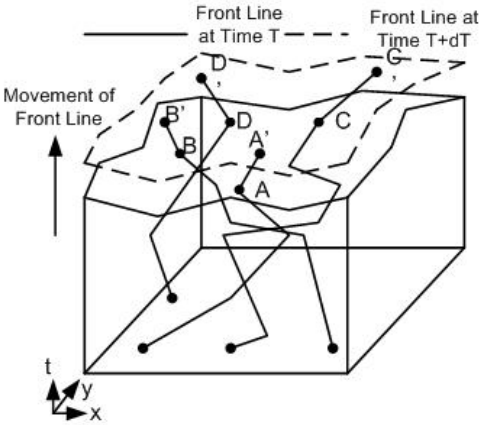

Figure 1. Movement of the front-line structure

There are some shortcomings of the SETI-index: 1) it cannot efficiently answer queries over current question since the frontline is implement using a hash or B^+-tree (eg: currently, how many moving object is hold in the selected area?); 2) it can not answer any query which ask questions about the future since no velocity information is stored in the front line; 3) this index suffers from several problems including: simulating the sparse R*-tree by existing dense R*-Tree is a little complex. A new sparse R*-tree should be implemented instead of simulating to fulfill the condition: one entry for one data page in sparse R*-tree.

2.3.2 Indexing the Current and Future Positions (NOW and the Future)

To predict the future positions of moving objects, we need to store extra information (e.g., the velocity and the destination). By introducing parametric bounding rectangles in R-tree, the TPR-tree and TPR*-tree[11] provides the capability to answer the queries which ask about current positions or future positions.

A moving object o is represented with (i) an MBR o_R that denotes its extent at reference time 0, and (ii) a velocity bounding rectangle (VBR) $o_V=\{o_{V1-},o_{V1+},o_{V2-},o_{V2+}\}$ where o_{Vi-} (o_{Vi+}) describes the velocity of the lower (upper) boundary of o_R along the i-th dimension (1 i 2). Figure 2a shows the MBRs and VBRs of 4 objects *a,b,c,d*. The arrows (numbers) denote the directions (values) of their velocities. A non-leaf entry is also represented with an MBR and a VBR. Specifically, the MBR (VBR) tightly bounds the MBRs (VBRs) of the entries in its child node. In Figure 2a, the objects are clustered into two leaf nodes N1, N2, whose VBRs are $N1_V=\{-2,1,-2,1\}$ and $N2_V=\{-2,0,-1,2\}$ (their directions are indicated using white arrows).

Figure 2b shows the MBRs at timestamp 1 (notice that each edge moves according to its velocity). The MBR of a non-leaf entry always encloses those of the objects in its subtree, but it is not necessarily tight.

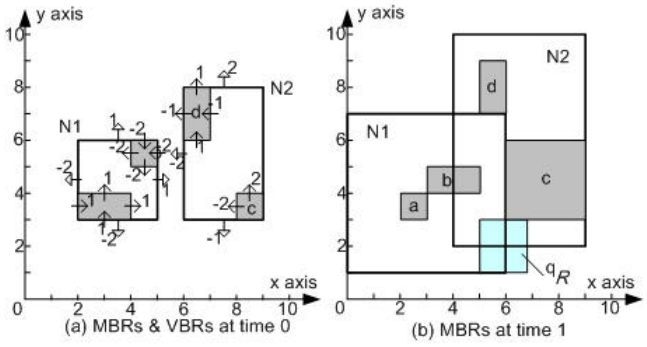

Figure 2. Entry Representation in a TPR*-tree

The TPR*-tree uses the structure and assumptions with the TPR-tree, only a new set of insertion and deletion algorithms that aim at minimizing a certain cost function is adopted.

The TPR-tree or TPR*-tree can answer the queries which ask the current or future questions. But they can not answer the queries which ask the historical questions since they do not store the trajectory.

3. PCFI$^+$-INDEX

3.1 Main Idea

In the PCFI-Index, there is only one TPR*-tree in the frontline which will be a bottleneck while processing large update operation on TPR*-tree since the whole TPR*-tree will be locked in some special update operation. In the PCFI$^+$-Index, the space dimension is partitioned into cells as in the SETI-tree, and there is one TPR*-Tree for one cell to index the moving objects' current positions and velocities, and a spare R*-tree is used to index the past positions of moving object trajectories. Considering the huge update operations on TPR*-tree of large population, the current data file is organized as main memory hash index file.

3.2 Index Structure

The PCFI-index consists of two parts: one located in memory, the other one located on disk.

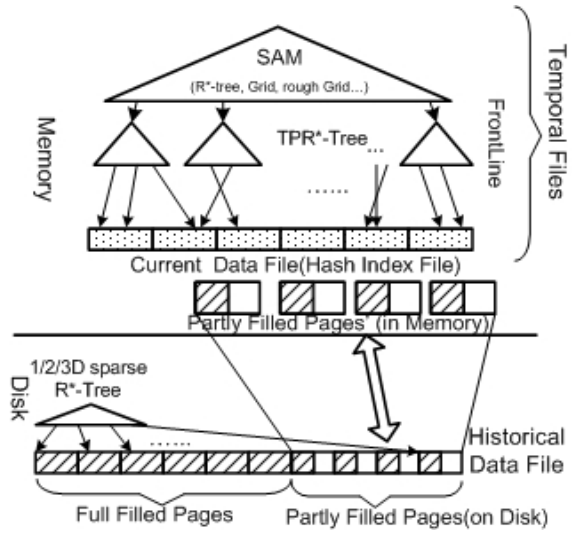

Figure 3. Data Structure of the PCFI$^+$-Index

The in-memory component is called frontline. It consists of a current data file, a spatial index (SAM) to index the non-overlapped partitions, and a set of TPR*-trees to index the records in the current data file.

The current data file is located in the temporal space without time-consuming log, and initialized when it is the first invocation of the spatio-temporal table's open method after system up. A hash index file organization method used in the current data with UID as the hash key. It can provide fast retrieval of records from the data file. Since there is a set of TPR*-tree, an additional pointer (slot id in SAM leaf page, or TPR*-tree ID, or pointer to the entry in TRP*-tree) can add to the tuple together with a latch for concurrency control.

The SAM is used to index the non-overlapping partitions. Many spatial access methods can be used to implement the partition strategy. For example, the R-tree, R*-tree can be used when we partition the area into irregular shapes; the Grid-file, rough Grid file and quadtree can be used when we partition the area into regular rectangles. Partitioning the area into irregular cells raises another issue: how to send the partition information to the index manager? We can use a spatial table to store the partition data,

and pass the table ID and field description when other modules invoke the index's open method. If the irregular partition is used, the index file can be used (the cell's spatial description resident in the leaf page) to reduce the disk I/O cost of the partition information.

For the TPR*-tree, a leaf node entry consists of the MBR and VBR of a moving object and a pointer (Record Identifier, RID) to the moving object in the current position data file; and the entry in an internal node is a pair of a pointer (Page Identifier, PID) to a subtree and a rectangle (MBR and VBR) that bounds the positions and velocities of all moving objects in subtree. Like the current data file, the TPR*-tree is also located in the temporal space and initialized when the table's open method is first invoked after start up. Each tuple in the current data file (hash index file) has an entry in the TPR*-tree. Considering the frequent update on TPR*-tree, the localized bottom-up update or generalized bottom-up update strategy[24] can be used but the epsilon adjustment is not applied on the root page of TPR*-tree with the pointer in the current data file tuple is entry of TPR*-tree leaf page. To simplify the data structure, we can just use the TPR*-tree id or entry in SAM leaf page in the tuple. The TPR*-trees resident in memory, the cache conscious R-tree: CR-Tree [25] can be used to improve the search speed.

The sparse R*-Tree is located on disk. For partitioning, the 1 dimensional, 2 dimensional, or 3 dimensional sparse R*-tree can be used. A historical data page's lifetime (*starttime, endtime*) is mapped to one-dimensional line (s*tarttime, endtime*), we call this the lifetime dimension. The first is the same as the SETI-tree--just one dimension sparse R*-tree is used. If we use a regular partition method such as grid, the Hilbert curve or Z-order curve can be used to translate the 2-dimensional area into 1-dimensional, thus the 2-dimensional sparse R*-tree can be used. If we use irregular partition method, the 2-D space dimension and the lifetime dimension consists of the 3-D spare R*-tree.

When a new page is allocate for the historical data file, an entry is inserted in the sparse R*-tree with an empty *endtime* in the lifetime. When a data page is fully filled by the subsequence segments, the corresponding entry in sparse R*-tree will be updated to set the *endtime*. If the 3-dimensional R*-tree is used, the entries in leaf pages will contain the historical data file's PageID, partition's spatial description, page's life time; for the rest two solutions leaf page's entry will consist of a pointer to a tuple in historical data file's page, lifetime, and *cellid* (restriction: one page only contains the segments belong to one cell). Entries in internal nodes are pairs of pointers to a subtree and a lifetime that bounds the lifetime of all segments or other bounding lifetime in that subtree. The *cellid* in leaf page is used to reduce disk read cost of the candidate page when historical slice query is processed.

3.3 Insert Algorithm

Figure 4 illustrates the insert procedure in PCFI-index. The key to the performance of the insert algorithm in PCFI is the use of the hash index file, which maintains the last updated location of all moving objects. It is a cache of the last positions of all objects and is updated with the new location of the moving object.

For simplify, the following method will use the 3-D sparse R*-tree to index the historical data file's page life, and the SAM leaf page's PID and entry id in the current data file tuple.

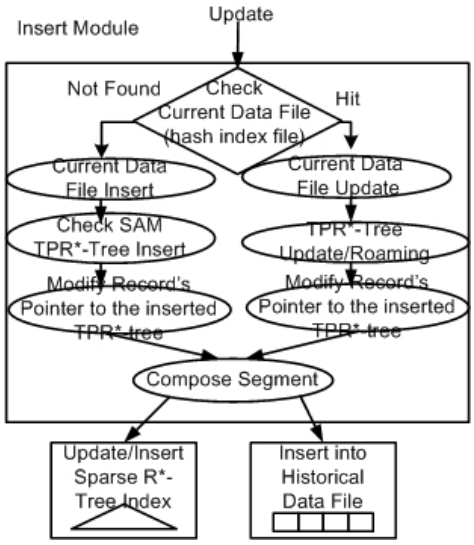

Figure 4. Schematic Diagram of the Insert Procedure in PCFI$^+$-index.

Since there is a set of TPR*-trees, a moving object will move out the original TPR*-trees, we call it tree roaming. When an update (this moving object has been stored in the current data file) is received, if the new location resides inside the current TPR*-tree, a new segment is composed and inserted into the historical data file; otherwise, a split operation is performed, and two segments are inserted into historical data file's different pages. The corresponding entry in old TPR*-tree will be removed, and inserted into another TPR*-tree. If necessary, the sparse R*-tree is also updated when the insertion of a new trajectory into the data file results in fully filling the data page used in the insertion operation.

3.4 Delete, Update Algorithm

There are three types of deletions. Two of them are: whole trajectory deletion and a particular trajectory segment deletion. The algorithms for these two deletions are similar to the original SETI's deletion algorithm. The third is caused when a moving object triggers a tracking termination requirement. A deletion operation on TPR*-tree is performed, following a tuple deletion in the current data file.

3.5 Query Procedure

Figure 5 shows the steps in the query algorithm. The input to the search algorithm for a time-interval query can be considered as a three dimensional query box, which consists of a temporal predicate range and a spatial predicate box.

The search algorithm executes the following steps:

1. *Compare with the Current time*: In this step, the temporal predicate range is compared with the current time. There are three cases:

Case 1: if the temporal predicate range falls completely in past, step 2.1 to 2.4 is performed;

Case 2: if the temporal predicate range falls completely in now or future, step 3 is performed.

Case 3: if the temporal predicate range covers past, now and future, we separate it into two parts: one corresponds to the past for which the procedures in Case 1 are adopted; the other corresponds to the current and future for which the procedures in Case 2 are adopted.

2 *Search for historical segments*

2.1. *Search candidate pages*:

Case 1: if 1-D sparse R*-tree is used, the spatial partitions that overlap with the spatial predicate box are computed via SAM, and a list of pair *(cellid, full_ overlap_or_ not)* is produced. Next the 1D sparse R*-tree is searched with the temporal predicate range. Each entry in the sparse R*-tree leaf page is checked to check whether the cell id falls into the cell list without fetching the corresponding data page into memory. If the cell is fully overlapped by the query region, all the segments belonging to this cell can be putted to the result segment list directly without refinement, or, the page id is putted to candidate pages list.

Case 2: if 2-D sparse R*-tree is used, first, the 2-D area (query area) is mapped to one dimension, then a 2-D filter is constructed by the one dimension mapped from the query area and the lifetime dimension. The sparse R*-tree is searched with this 2-D filter, and a list of candidate pages produced.

Case 3: if 3-D sparse R*-tee is used, a 3-D search box is composed by the life time dimension and the 2-D search area, and then applied to the sparse R*-tee, thus, the candidate page list is produced.

2.2. *Refinement Step*: This procedure is the same as the original SETI-tree. A list of segments that overlap with the query box and time period is produced in this step.

2.3. *Duplicate Elimination*: This procedure is the same as the original SETI-tree. It produces a list of trajectory ids or a list of segments.

3. *Spatial & Temporal Filtering*: This procedure is the same as the query algorithm of original TPR-tree.

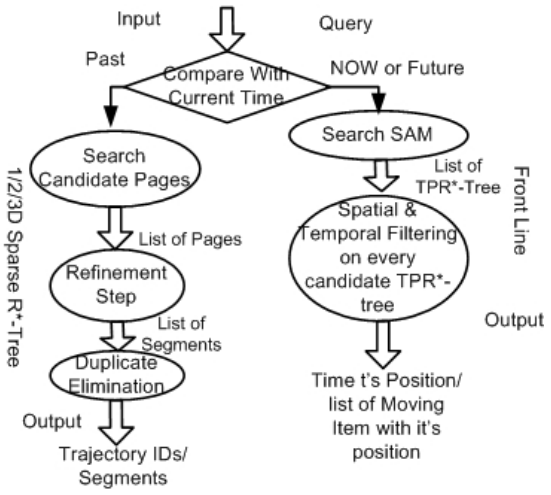

Figure 5. Schematic Diagram of the Query Algorithm in PCFI$^+$-index

For the slice query, the temporal predicate range consists of only a single value, the query can be answered with the sparse R*-tree (past time) or frontline (TPR-tree, current of future).

For the window query, there are three cases: 1) if both of [Ts, Te] are located in the past, the query can be answered via sparse R*-tee; 2) if both of [Ts,Te] are located in the present or future, we can process the query via frontline part; 3) if the Ts is located in the past and Te is located in the future, the [Ts, now) part is answered via sparse R*-tee part, and the [now, Te] part is answered via frontline part.

For the moving query, it is similar to the window query, except the query range should be divided into two parts according to the current time if the time range contains current time.

4. EXPERIMENT EVALUATION

In this section we present the experimental results evaluating the behavior of PCFI$^+$-Tree and two other trajectory indexing structures: the SETI-index and the TPR*-tree.

4.1 Experimental Setup and Workload Generation

The development platform is a dual Intel Pentium III Xeon 600MHz HP server that is configured with 512 MB of main memory, and a 16GB Seagate 10000 RPM Ultra SCSI 2 disk, running Windows 2000 advanced server English version.

The index method is implemented in the GEOMania Millennium Server (GMS)[26], which is a spatial DBMS implemented at database lab., Inha Univ. This system uses Storm/NT as its storage manager, and all indexing techniques that we examine in this section are implemented in Storm/NT. Storm/NT currently supports 2D R*-trees. SETI is implemented using the existing 2D R*-trees. The spatial extent is partitioned using a fixed uniform rectangular grid, and the SAM is hard coded cache conscious R*-Tee. The sparse R*-tree is implemented as 3-D R*-tree. We implemented TPR*-tree with the node size equal to Xeon's cache size (16KB) as an additional indexing mechanism in Storm/NT. The current data file (hash index file) is implemented with the chunk size of 16KB and the pointers are TPR*-tree's IDs. In all our experiments, we use a buffer pool of 128MB. The disk page size used in all the experiments is 8KB. A value in the time dimension is represented using 14 bytes, and a value in the spatial dimension is represented using a 4 byte double.

All trajectory segments are stored in the GMS database. Each tuple has a unique trajectory id, a segment number, and the two end points of the trajectory segments.

Since no real trajectory data sets are available, we generated simulated data sets using the GSTD [18] data generator. The GSTD data generator produces trajectory data sets for a specified number of moving objects, with a specified number of segments per moving object. The GSTD generator has numerous parameters for changing the distribution of the initial positions of the moving objects, the direction of the movements, trajectory segment length, etc. We experimented with a number of data sets produced by varying some of these parameters, and found that the results using the default uniform distributions were representative of these other data sets too. In the interest of space, we only present results

for GSTD data sets that were generated using the default data generation parameter values.

4.2 Performance Evaluation

We examined the time-interval query for the past with 0.1% selectivity as the number of moving objects is increased. The experiment [Figure 6] shows that the PCFI$^+$-index outperforms the SETI-index and PCFI-index. The reason for this is that the PCFI$^+$-index has three dimensions whereas SETI-Index and PCFI-Index have only one dimensional spare R*-tree.

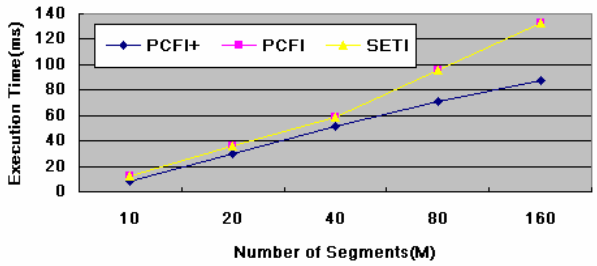

Figure 6. Past Time-interval query for variable of moving objects, 0.1% selectivity

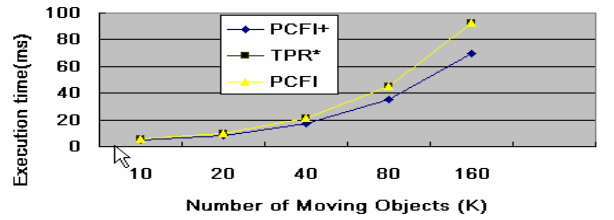

Figure 7. Current query for variable of moving objects, 1% selectivity

We examined the current query for PCFI-index, PCFI$^+$-index, TPR*-tree, and SETI. The experiment result [Figure 7] shows that PCFI$^+$-index outperforms all other indices. Furthermore, the PCFI-index has the same performance with TPR*-tree for current queries, and the SETI-index has the highest execution time. This is because PCFI has the same index structure and algorithm as the TPR* for current queries, both of them only contain one TPR*-tree, but PCFI+-index has a set of TPR*-tree which has a higher degree of concurrency and smaller

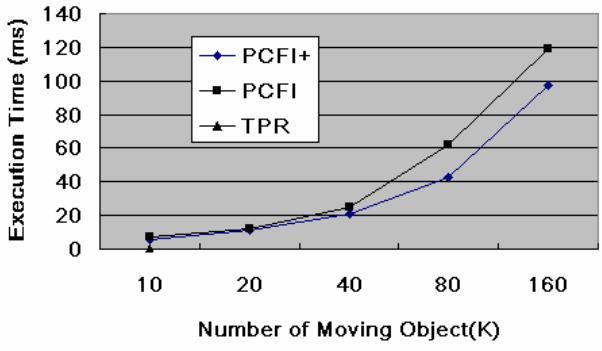

Figure 8. Future Time-slice query for variable of moving objects, 1% selectivity

We also examined the future slice queries for PCFI-index, PCFI$^+$-index and TPR*-tree. The experiment result [Figure 8] shows that both the PCFI-index and TPR*-tree have the same execution time. This is because the PCFI-index uses the same algorithm as the TPR*-index. The PCFI$^+$-index outperforms the other two.

5. Conclusions and Future Work

In this paper we have proposed and evaluated a new spatio-temporal indexing mechanism, called PCFI$^+$-Index. Based on the implementation in Storm/NT, we have demonstrated that PCFI$^+$-index challenges with the original SETI for both historical time-interval and historical time-slice queries with better performance, but overcomes the SETI for current queries, and has a better performance than the TPR*-Index and PCFI-Index for current or future query. PCFI$^+$-Index also provides a uniform processing method for the querying the past, current and future positions. Compared with the SETI–index and PCFI-Index, the PCFI$^+$–index is more realizable since there is a set of TPR*-tree, which provide better concurrency degree. For future work, we plan on investigating the use of PCFI for evaluating trajectory queries, which require fetching entire trajectories, and computing derived values such as average speed.

6. REFERENCES

[1] M. Abdelguerfi, J. Givaudan, K. Shaw, and R. Ladner. The 2-3 TR-tree, A Trajectory-Oriented Index Structure for Fully Evolving Valid-time Spatio-temporal Datasets. In Proc. of the ACM workshop on Adv. in Geographic Info. Sys., ACM GIS, pages 29–34, Nov. 2002.

[2] M. Cai and P. Revesz. Parametric R-Tree: An Index Structure for Moving Objects. In Proc. of the Intl. Conf. on Management of Data, COMAD, Dec. 2000.

[3] V. P. Chakka, A. Everspaugh, and J. M. Patel. Indexing Large Trajectory Data Sets with SETI. In Proc. of the Conf. on Innovative Data Systems Research, CIDR, Asilomar, CA, Jan. 2003.

[4] N. Katayama and S. Satoh. The SR-tree: An Index Structure for High-Dimensional Nearest Neighbor Queries. In Proc. of the ACM Intl. Conf. on Management of Data, SIGMOD, pages 369–380, May 1997.

[5] G. Kollios, D. Gunopulos, and V. J. Tsotras. On Indexing Mobile Objects. In Proc. of the ACM Symp. on Principles of Database Systems, PODS, pages 261–272, June 1999.

[6] D. Kwon, S. Lee, and S. Lee. Indexing the Current Positions of Moving Objects Using the Lazy Update R-tree. In Mobile Data Management, MDM, pages 113–120, Jan. 2002.

[7] D. Pfoser, C. S. Jensen, and Y. Theodoridis. Novel Approaches in Query Processing for Moving Object Trajectories. In Proc. of the Intl. Conf. on Very Large Data Bases, VLDB, pages 395–406, Sept. 2000.

[8] K. Porkaew, I. Lazaridis, and S. Mehrotra. Querying Mobile Objects in Spatio-Temporal Databases. In Proc. of the Intl. Symp. on Advances in Spatial and Temporal Databases, SSTD, pages 59–78, Redondo Beach, CA, July 2001.

[9] Y. Tao and D. Papadias. Efficient Historical R-trees. In Proc. of the Intl. Conf. on Scientific and Statistical Database Management, SSDBM, pages 223–232, July 2001.

[10] Y. Tao and D. Papadias. MV3R-Tree: A Spatio-Temporal Access Method for Timestamp and Interval Queries. In Proc. of the Intl. Conf. on Very Large Data Bases, VLDB, pages 431–440, Sept. 2001.

[11] Y. Tao, D. Papadias, and J. Sun. The TPR*-Tree: An Optimized Spatio-temporal Access Method for Predictive Queries. In Proc. of the Intl. Conf. on Very Large Data Bases, VLDB, Sept. 2003.

[12] S. Saltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez. Indexing the Positions of Continuously Moving Objects. In Proc. of the ACM Intl. Conf. on Management of Data, SIGMOD, pages 331–342, May 2000.

[13] X. Xu, J. Han, and W. Lu. RT-Tree: An Improved R-Tree Indexing Structure for Temporal Spatial Databases. In Proc. of the Intl. Symp. on Spatial Data Handling, SDH, pages 1040–1049, July 1990.

[14] Mohamed F. Mokbel , Thanaa M. Ghanem and Walid G. Aref . Spatio-Temporal Access Methods. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 2003

[15] Steve Loughran, Code for Speed :Writing High Performance Win32 Code, http://www.iseran.com/Win32/CodeForSpeed/

[16] Guting, R. H., B¨OHLEN, M. H., Erwig, M., Jensen, C. S., Lorentzos, N. A., Schneider, M., and Vazirgiannis, M. A Foundation for Representing and Quering Moving Objects. ACM Transactions on Database Systems 25, 1, pp1–42, Mar, 2000

[17] Code for speed, http://www.iseran.com/Win32/CodeForSpeed/

[18] Theodordis, Y., SILVA, J. R. O., and Nascimento, M. A. On the Generation of spatiotemporal Datasets. In Advances in Spatial Databases, 6th International Symposium (1999), Lecture Notes in Computer Science, Springer, pp.147–164, 1999

[19] Pfoser, D., Jensen, C. S., and Theodordis, Y. Novel Approaches in Query Processing for Moving Object Trajectories. In Proceedings of the 26st VLDB Conf. (Cairo, Egypt, September 2000), pp. 395–406, 2000

[20] Dieter Pfoser, Indexing the Trajectories of Moving Objects, Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, pages 1-7, 2002

[21] Mario A. Nascimento, Je_erson R. O. Silva, and Yannis Theodoridis, Evaluation of Access Structures for Discretely Moving Points, In Proc. of the Intl. Workshop on Spatio-Temporal Database Management, STDBM, pages 171–188, Sept. 1999.

[22] Z. Song and N. Roussopoulos. SEB-tree: An Approach to Index Continuously Moving Objects. In Mobile Data Management, MDM, pages 340–344, Jan. 2003.

[23] Liu Zhao-Hong, C.H. Lee, J. W. Ge and H. Y. Bae, Indexing Large Moving Objects from Past to Future with PCFI. In Proc. Of the 2nd Asian Symposium on GISs from Computer Science & Engineering View(2004), pages 25-34, May. 2004

[24] Mong Li Lee , Wynne Hsu, Christian S. Jensen , Bin Cui and Keng Lik Teo, Supporting Frequent Updates in R-Trees: A Bottom-Up Approach, Proceedings of the 29th VLDB Conference, Berlin, Germany, 2003

[25] Kihong Kim, Sang K. Cha, Keunjoo Kwon, Optimizing Multidimensional Index Trees for Main Memory Access, In Proc. of the ACM Intl. Conf. on Management of Data, SIMMOD, page139-150, 2001

[26] GEOMania Co. LTD, http://www.geomania.com