A New Proposed Algorithm for BBx-Index Structure

K. Appathurai[1] and Dr. S. Karthikeyan[2]
[1]Department of Information Technology, Karpagam University
Coimbatore, Tamil Nadu

Dr.S.Karthikeyan
Department of Information Technology, College of Applied Sciences,
Sultanate of Oman

Abstract

Even if major effort has been put into the development of capable spatio-temporal indexing techniques for moving objects, some more mind has been given to the advance of techniques that professionally support queries about the past, present, and future positions of moving objects. The specification of such techniques is difficult, by the nature of the data, which reflects continuous movement, and because of the types of queries to be supported. This paper proposes the new index structure called OBB^x (Optimized BB^x) which indexes the positions of moving objects, given as linear functions of time, at any time. The index stores linearized moving-object locations in a minimum of B+-trees. The index supports queries that select objects based on temporal and spatial constraints, such as queries that retrieve all objects whose positions fall within a spatial range during a set of time intervals. The proposed work reduces lot of efforts done by the existing method and minimized time complexity. The simulation results shows that the proposed algorithm provides better performance than BB^x index structure.

Keywords: *Moving Objects, BB^x index, OBB^x index, Migration and B+-trees.*

1. Introduction

Spatio-temporal databases deals with moving objects that change their locations over time. In common, moving objects account their locations obtained via location-aware instrument to a spatio-temporal database server. Spatiotemporal access methods are secret into four categories: (1) Indexing the past data (2) Indexing the current data (3) Indexing the future data and (4) Indexing data at all points of time. All the above categories are having set of indexing structure algorithms [1- 4, 10, 13]. The server store all updates from the moving objects so that it is capable of answering queries about the past [4, 5, 8, 9, 15]. Some applications need to know current locations of moving objects only. This case, the server may only store the current status of the moving objects. To predict future positions of moving objects, the spatio-temporal database server may need to store additional information, e.g., the objects' velocities [7, 17]. Many query types are maintained by a spatio-temporal database server, e.g., range queries "Find all objects that intersect a certain spatial range during a given time interval", k-nearest neighbor queries "Find k restaurants that are closest to a given moving point", or trajectory queries "Find the trajectory of a given object for the past hour". These queries may execute on past, current, or future time data. A large number of spatio-temporal index structures have been proposed to support spatio-temporal queries efficiently [12, 13]. This paper is based on the source paper [10].

2. Related work

Several recent reviews of moving-object indexing techniques exist that focus on different aspects [1, 6, 7]. The first variant of indices include the TPRtree (Time-Parameterized R-tree) family of indexes [2, 5]. One of the initial works is the Historical R-tree (HR-tree) [18], which logically constructs a "new" R-tree each time an update occurs. Duplication of object is the major drawback of R-tree. After R-tree Pfoser et al. propose the Spatio-Temporal R-tree (STR-tree) and the Trajectory-Bundle tree (TB-tree). Yongquan Xia, Weili Li , and Shaohui Ning, Moving Object Detection Algorithm Based on Variance Analysis [16] is derived.

Besides Muiti-Version 3D R-tree (MV3R-tree) [19] is proposed by Tao and Papadias. Then, B. Liu. Querying about the Past, the Present, and the Future in Spatio-Temporal Databases [20]. A recent proposal by Dan Lin, Christian S. Jensen, and Ooi [10] supports queries about the past, present, and future. However, on approximate aggregate query results can be computed. In applications were accurate results are needed, other proposals are needed

3. BB^x INDEX Structure

The BB^x-index consists of nodes that consist of entries, each of which is of the form (x _rep; t_{start}; t_{end}; pointer.) For leaf nodes, pointer points to the objects with the equivalent x_rep, where x_rep is obtained from the space-filling curve; t_{start} denotes the time when the object was inserted into the database (matching to the tu in the description of the B^x-tree), and t_{end} denotes the time that the position was deleted, updated, or migrated (migration pass on to the update of a position done by the system automatically). For non-leaf nodes, pointer points to a (child) node at the next level of the index: tstart and tend are the minimum and maximum t_{start} and t_{end} values of all the entries in the child node, respectively. In addition, each node contains a pointer to its right sibling to facilitate query processing. Unlike the B^x-tree, the BB^x-index is a group of trees, with each tree having an associated timestamp signature tsg and a lifespan (see Figure 1). The timestamp signature parallels the value tlab from the B^x-tree and is obtained by partitioning the time axis in the same way as for the B^x-tree. The lifespan of each tree corresponds to the minimum and maximum lifespan of objects indexed in the tree. The roots of the trees are stored in an array, and they can be accessed efficiently according to their lifespan. This array is relatively small and can usually be stored in main memory.

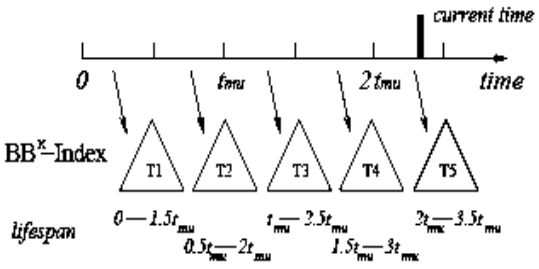

Fig 1 The BB^x index [10]

Objects inserted during the same phase will be stored in the tree with the tag that is equal to the end timestamp of that phase. In particular, an update with timestamp t_{start} is assigned a timestamp signature tsg = [tstart]t, where x[t] returns the smallest timestamp signature that does not precede x. Using space-filling curve the position of an object is represented by a single-dimensional value x rep. In order to retain the proximity-preserving property of the space-filling curve, we index objects within a time interval by their positions as of the time given by the timestamp signature of this interval. Hence, we need to determine an object's position at the timestamp signature according to its moving function [6].

An object's linear movement O = $(\vec{x}, \vec{v})$ is given by a position and a velocity at the time of update, t_{start}. The transformation from the current position to the position $\vec{x}_{ind}$ that will be indexed. We thus place the position x rep computed by applying the space filling curve to $\vec{x}_{ind}$ in the tree with timestamp signature tsg. Note that we do not concatenate the timestamp signature and x rep as in the B^x-tree. There are two reasons for this. First, our index aims to handle moving objects from the past to the future. Thus, the index must contend with timestamps that keep growing in value. Inclusion of such values in the key would poses an efficiency problem since we must then allocate substantial space for the key in order to cater to its growth. In contrast, the B^x-tree only indexes current positions of moving objects and hence is able to fix the length of the key value (by using the modulo function). Second, without considering the timestamp, we obtain a shorter key and a simpler mapping function. Imagining that the index runs for one year, the accumulated timestamp value (224 minutes) would require a long key value representation, which will significantly reduce the node capacity and fanout, which increases index size and decreases query performance. Let us illustrate the BB^x - index with an example. Figure 1 shows a BB^x-index with n equal to 2. Objects inserted between timestamps 0 and 0:5tmu are stored in tree T1 with their positions as of time 0:5tmu; those inserted between timestamp 0:5tmu and tmu are stored in tree T2 with their positions as of time tmu; and so on. Each tree has a

maximum lifespan: T1's lifespan is from 0 to 1:5tmu because objects are inserted starting at timestamp 0 and because those inserted at timestamp 0:5tmu may be alive throughout the maximum update interval tmu, which is thus until 1:5tmu; the same applies to the other trees [10].

4. Statement of Problem

In BB^x index structure the migration is one of the major problems, even though the past information also indexed unlike B+ tree indexing structure. BB^x take more effort and time for the whole process of indexing. Due to this high effort the memory space utilization, processor utilization, execution time and cost increases very high. Besides in tree the node insertion, deletion also complex process when the number of moving objects are high.

5. Proposed Algorithm

The main aim of the proposed algorithm is to decreases the complexity of BB^x index structure. Besides the overall performance of the proposed algorithm is good than BB^x index about 40%. The proposed algorithm is called OBB^x-index (Optimized Broad B^x). The scalability is considered as twice for the better result. The OBB^x-index the nodes consist of the form (x _rep; t_{start}; t_{end}; pointer.) where x_rep is nothing but one dimensional data obtained from the space-filling curve; tstart denotes the time when the object was inserted into the database and tend denotes the time that the position was deleted, updated, or migrated (migration refers to the update of a location done by the system). t_{start} and t_{end} are the minimum and maximum tstart and tend values of all the entries in the child node, respectively. In addition, each node contains a pointer to its right sibling to facilitate query processing. The OBB^x-index is a forest of trees, with each tree having an associated timestamp signature tsg and a lifespan. The timestamp signature parallels the value tlab from the B^x-tree and is obtained by partitioning the time axis in the same way as for the B^x-tree. The lifespan of each tree corresponds to the minimum and maximum lifespan of objects indexed in the tree. The roots of the trees are stored in an array, and they can be accessed efficiently according to their lifespan. This array is relatively small and can usually be stored in main memory. Initially the maximum update interval is found out among all the moving objects.

The maximum interval value is making it as twice for scalability. Figure 1 shows a BB^x-index with n = 2. Objects inserted between timestamps 0 and 0:5tmu are stored in tree T1 with their positions as of time 0:5tmu; those inserted between timestamp 0:5tmu and tmu are stored in tree T2 with their positions as of time tmu; and so on. Each tree has a maximum lifespan: T1's lifespan is from 0 to 1:5tmu because objects are inserted starting at timestamp 0 and because those inserted at timestamp 0:5tmu may be alive throughout the maximum update interval tmu, which is thus until 1:5tmu; the same applies to the other trees.

```
Begin( )
For each E do
Begin( )
uie <-- update interval of E
if  uie is greater than ui
Begin
ui <--uie
End
Else
Begin
ui <--ui
End
i ←i+1
End( )
tmu = 2 * ui
ts1 ← timeaxis from 0 to tmu
ts2 ← timeaxis from tmu to 2tmu
ts3 ← timeaxis from 2tmu to 3tmu

T ←Array of  n equal intervals of ts1, ts2, ts3, etc
For each T do
Begin( )
LE <--objects in the lifespan of T
Pos = 2
For each LE do
Begin( )
create a new node N
C ← current node in the tree
Iterate ←  true
While(Iterate) do
Begin( )
If key of N lesser than key of C then
Begin
If C has left node then
Begin
C ←left node of C
```

End
Else
Begin
Iterate ← false
Set N as left node of C
End
End
Else
Begin
If C has right node then
Begin
C ← right node of C
End
Else
Begin
Iterate ← false
Set N as right node of C
End
End
Initialize an Array variable Arr,
Arr = Store the time value of Node N
End( )
Tot = Count(Arr)
End( )
For each T of move from ts[Pos-1] to ts[Pos]
Begin
While Arr not equal to Null
Begin
If Node[i] is in ts[Pos]
Begin
Update Node[i] to ts[Pos]
End
Else
Begin
Migrate Node[i] to ts[Pos]
End
End
End
Initialize an Array variable Oit1,Oit2,Oit3,etc,.
Oitn ← Store the indexed objects in the timeslice of ts/2
End( )
End( )

Fig 2: Algorithm to Tree Construction, Object Insertion, Updation and Migration

Each tree has lifespan after that the tree values are updated to next tree. So first check whether all the objects are reached or not if it is reached then update all the objects to next tree and then the objects are removed or deleted from the existing old tree because to avoid duplication of index. The following algorithm shows how the updation takes place in OBB^x. In this algorithm first identify the tree where the update object is located and then find out the position of the object in that tree and then the object is removed and updated in new tree from old tree.

Update Node[i] to ts[Pos-1]

Algorithm Update(Eo; En)

Input: Eo and En are old and new objects respectively
tindex ← time Eo is indexed in the tree
find tree Tx whose lifespan contain tindex
posindex ← position of Eo at tindex
keyo ← x-value of the posindex
locate Eo in Tx according to keyo
modify the end time of Eo's lifespan to current time
t'index ← time En will be indexed
pos'index ← position of En at t'index
keyn ← x-value of the pos'index
insert En into the latest tree according to keyn

Fig 3 Algorithm for Update

Each tree has lifespan after that the tree values are updated to next tree. So first check whether all the objects are reached or not if any object is not reached then that object is identified and then migrated to next tree. Next that objects are removed or deleted from the existing old tree because to avoid duplication of index. The following algorithm shows how the migration process takes place in OBB^x. In this algorithm first identify the tree where the migrate object is located and then find out the position of the object in that tree and then the object is removed and migrated in new tree from old tree.
Migrate Node[i] to ts[Pos-1]

Algorithm Migrate(Eo; En)
Input: Eo and En are old and new objects respectively
tindex ← time Eo is indexed in the tree
find tree Tx whose lifespan contain tindex
posindex ← position of Eo at tindex
keyo ← x-value of the posindex
locate Eo in Tx according to keyo
modify the end time of Eo's lifespan to current time

Fig 4 Algorithm for Migrate

6. Performance Studies

The below figure 5 shows how the objects moving randomly in un specified path and it describes the clear path of the every moving objects. In this example 5 moving objects are consider for indexing. The starting time is 45 ms and the ending time is 205.98214875 ms, this is clearly shown in the figure 5. In figure 5 the x axis is time and y axis is points i.e. by Hilbert curve the multidimensional data is converted as points (single dimensional data).

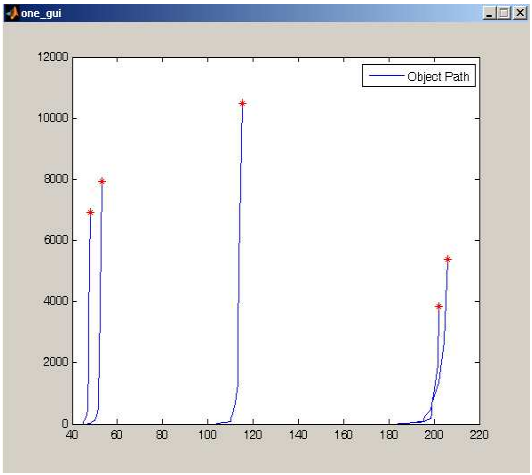

Fig 5 This figure shows how the objects moving randomly in un specified path. And It describes the clear path of the every moving objects.

The below figure 6 shows the total indexing time for both the methods like BB^x index and OBB^x index. The total processing time for BBX Indexing is 5.542051e+000 and the total processing time for OB^X Indexing is 2.815064e+000. so it clearly says the OBB^x method is much better than BB^x method.

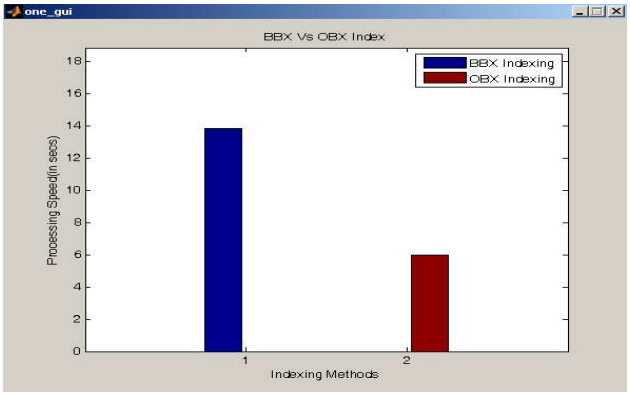

Fig 6 Comparison of BBX and OBX indexing in terms of Processing Speed

The below figure 7 indicates the number of migration hit occur in both the techniques. As per this concern also the OBB^x index techniques is much better than BB^x index techniques. The migration hits for BB^X Indexing is 44 and the migration hits for OBB^X Indexing is 22. This reducing of migration hit improves the total performance of OBB^x index method, reducing the processor utilization time and it deceases the total cost also.

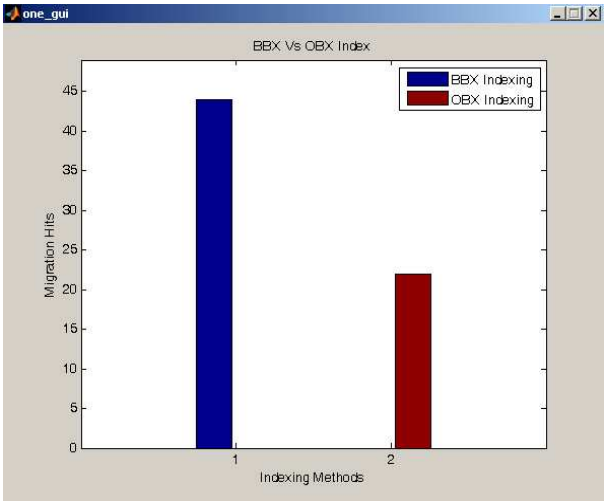

Fig 7 Comparison between BB^X and OBB^X in terms of Migration hits

7. Results

Using MATLAB the following results are produced.

S.No	No of Objects	Time (Milliseconds)	
		BB^x	OBB^x
1	5	5.542051e+000	2.815064e+000
2	9	1.059695e+001	6.200636e+000

Table 1 Time Analysis

8. Conclusion

This paper presents a new indexing technique, the OBBx-index (Optimized BB^x-index), which can answer queries about the past, the present and the future. The OBB^x -index is based on the concepts underlying the BB^x-tree. Like the BB^x-index, the indexing of historical information, it avoids duplicating objects and thus achieves significant space saving and efficient query processing. Moreover it reduces almost half of the number of trees used in BB^x-index. So the energy efficiency is very good than BB^x index and hardly reduces time complexity. Extensive performance studies were conducted that indicate that the OBB^x-index outperforms the existing state of-the-art method, with respect of historical, present and predictive queries. The Future work is planed to further reducing of migration problem without affecting the efficiency.

References

[1]. Long-Van Nguyen-Dinh, Walid G. Aref, Mohamed F. Mokbel 2010. Spatio-Temporal Access Methods: Part 2 (2003 - 2010). Bulletin of the IEEE Computer SocietyTechnical Committee on Data Engineering

[2]. M. Pelanis, S. ˇ Saltenis, and C. Jensen. Indexing the past, present, and anticipated future positions of moving objects.TODS, 31(1):255–298, 2006.

[3]. Z.-H. Liu, X.-L. Liu, J.-W. Ge, and H.-Y. Bae. Indexing large moving objects from past to future with PCFI+-index. In COMAD, pages 131–137, 2005.

[4]. V. Chakka, A. Everspaugh, and J. Patel. Indexing large trajectory data sets with SETI. In CIDR, 2003

[5]. Y. Tao, D. Papadias, and J. Sun. The TPR*-tree: An optimized spatio-temporal access method for predictive queries. In VLDB, 2003.

[6]. C. Jensen, D. Lin, and B. Ooi. Query and update efficient B+-tree based indexing of moving objects. In VLDB, 2004.

[7]. M. Mokbel, T. Ghanem, andW. G. Aref. Spatio-temporal access methods. IEEE Data Eng. Bull., 26(2):40–49, 2003.

[8]. J. Ni and C. V. Ravishankar. PA-tree: A parametric indexing scheme for spatio-temporal trajectories. In SSTD, 2005.

[9]. P. Zhou, D. Zhang, B. Salzberg, G. Cooperman, and G. Kollios. Close pair queries in moving object databases. In GIS, pages 2–11, 2005.

[10].Dan Lin, Christian S. Jensen, Beng Chin Ooi, Simonas S˘ altenis, BBx index :Efficient Indexing of the Historical, Present, and Future Positions of Moving Objects, MDM 2005 Ayia Napa Cyprus

[11].P. K. Agarwal and C. M. Procopiuc. Advances in Indexing for Mobile Objects. IEEE Data Eng. Bull., 25(2): 25–34, 2002.

[12].G. Kollios, D. Gunopulos, V. J. Tsotras. On Indexing Mobile Objects. In Proc. PODS, pp. 261–272, 1999.

[13]. K.Appathurai, Dr. S. Karthikeyan. A Survey on Spatiotemporal Access Methods.International Journal of Computer Appliations. Volume 18, No 4, 2011.

[14].Mohamed F. Mokbel, Xiaopeng Xiong, Moustafa A. Hammad, and Walid G. Aref, Continuous Query Processing of Spatio-temporal Data Streams in PLACE, 2004 Kluwer Academic Publishers. Printed in the Netherlands

[15].Su Chen · Beng Chin Ooi · Zhenjie Zhang, An Adaptive Updating Protocol for Reducing Moving Object Database Workload.

[16].Yongquan Xia, Weili Li , and Shaohui Ning, Moving Object Detection algorithm Based on Variance Analysis, 2009, Second International Workshop on Computer Science and Engineering Qingdao, China

[17].Arash Gholami Rad, Abbas Dehghani and Mohamed Rehan Karim, Vehicle speed detection in video image sequences using CVS method, 2010, International Journal of the Physical Sciences Vol. 5(17), pp. 2555-2563.

[18].M. A. Nascimento and J. R. O. Silva. Towards Historical R-trees. In Proc. ACM Symposium on Applied Computing, pp. 235–240, 1998.

[19]. Y. Tao and D. Papadias. MV3R-Tree: A Spatio-Temporal Access Method for Timestamp and Interval Queries. In Proc. VLDB, pp. 431–440, 2001.

[20].J. Sun, D. Papadias, Y. Tao, and B. Liu. Querying about the Past, the Present, and the Future in Spatio-Temporal Databases. In Proc. ICDE, pp. 202–213, 2004.

K. Appathurai was born on 12th May 1974. He received his Master degree in Computer Applications from University of Bharathidasn in 1998. He completed his M.Phil from Manonmaniam Sundaranar University in 2003. He is working as an Asst. Professor and Head of the Department of Information Technology at Karpagam University, Coimbatore. Currently He is pursuing Ph.D. His fields of interest are Spatial Database.

Dr. S. Karthikeyan received the Ph.D. Degree in Computer Science and Engineering from Alagappa University, Karaikudi in 2008. He is working as a Professor and Director in School of Computer Science and Applications, Karpagam University, Coimbatore. At present he is in deputation and working as Assistant Professor in Information Technology, College of Applied Sciences, Sohar, Sulatanate of Oman. He has published more than 14 papers in Natrional/International Journals. His research interests include Cryptography and Network Security.