International Journal of Computer Applications (0975 – 8887)
Volume 69– No.16, May 2013

Moving Object indexing using Crossbreed Update

K. Appathurai
Ph.D Research Scholar
Karpagam University
Coimbatore – 21

S. Karthikeyan, PhD.
Director, School of computer Science
Karpagam University
Coimbatore – 21

ABSTRACT

Although lot of spatio-temporal indexing techniques for moving objects are availed, some more intelligence has been given to the advance of techniques that competently support queries about the past, present, and future positions of moving objects. This paper proposes the new index structure called SOBBx (Space Based Optimal BBx) which indexes the positions of moving objects, given as linear functions of time, at any time. In a Time t, more objects are updated to the tree than usual. It saves the cost of regular update as well. The simulation results shows that the proposed algorithm provides superior performance than POBBx index structure.

Keywords

Moving Objects, BBx-tree, OBBx index, POBBx index, Migration, Regular Update, Crossbreed Update and SOBBX index.

1. INTRODUCTION

Spatio-temporal databases deals with moving objects that change their locations over time. In common, moving objects report their locations obtained via location-aware instrument to a spatio-temporal database server. Spatiotemporal access methods are subversive 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]. To predict future positions of moving objects, the spatio-temporal database server may need to store auxiliary 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]. The online moving object index tuning method also contribute to improve the performance [22].

2. POBBx index Structure

The POBBx-index consists of nodes that consist of entries, each of which is of the form (x _rep; tstart; tend; pointer.) For leaf nodes, pointer points to the objects with the equivalent x_rep, where x_rep is 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 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 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. Unlike the Bx-tree, the POBBx-index is a collection of trees, with each tree having an associated timestamp signature tsg and a lifespan. The timestamp signature parallels the value tlab from the Bx-tree and is obtained by partitioning the time axis in the same way as for the Bx-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.

Objects inserted during the same phase will be stored in the tree with the tsg that is equal to the end timestamp of that phase. In particular, an update with timestamp tstart 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].

The maximum update interval value is making as twice for interval. Figure 1 shows a POBBx-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. In case of POBBx index [21] unlike BBx index method the searching is less overhead. That means, the object is moved from one tree to another along with last tree value and the position in that tree. So during updation or migration the searching process is not complex.

1. Find out the maximum update interval for each object and the maximum interval value is stored in ui.
2. The maximum update interval Ui is multiplied by two and then based on this scalability the linear array is formed for ts1,ts2,ts3, etc.,
3. Array of n equal intervals of ts1, ts2, ts3, etc

4. Each object lifespan are find out that is stored in LE. In this Multi dimensional points representing object paths by Hilbert curve coordinates.
5. Based on the lifespan the data are stored in the tree.
6. If the insertion node C is lesser than the node N then the node C inserted on left else inserted on right. If already the nodes are there the same way created and stored. The insertion time for each object is stored in the variable Arr and total object is inserted is stored in the variable Tot
7. For each move from one tree to another, While Arr not equal to Null, it is checked whether all the moving objects are reached to the new tree or not, if it is reached call the function update or else all the function migration.

Fig 1: Algorithm to Tree Construction, Object Insertion, Updation and Migration

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

Algorithm Update(Eo; En)

```
Input: Eo and En are old and new objects respectively
Oodum ⟵ indexed position of the object Eo.
Thisarr ⟵ last tree value of the object Eo.
Curpos ⟵ current position of the object En.
Curtim ⟵ current time of the object En
Curarr ⟵ current tree of the object En which lifespan contains Curtim.
Remove the object Eo from the tree Thisarr of the position Oodum.
Locate object En in the tree Curarr of the position Curpos.
Oodum ⟵ indexed position of the object En.
Thisarr ⟵ last tree value of the object En.
```

Fig 2 : Algorithm for Update

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

Algorithm Migrate(Eo; En)

```
Input: Eo and En are old and new objects respectively
Oodum ⟵ indexed position of the object Eo.
Thisarr ⟵ last tree value of the object Eo.
Curpos ⟵ current position of the object En.
Curtim ⟵ current time of the object En
Curarr ⟵ current tree of the object En which lifespan contains Curtim.
```

Fig 3 : Algorithm for Migrate

3. Statement of Problem

Though POBBx indexing method reduces the processing time, migration hits, storage requirement and node accessibility, it may have the following problems in some scenarios:

1. Larger tree leads to a higher in query cost, while a smaller tree may introduce extra migration cost.

2. Both the updation time and migration time has larger in case of high density in a tree.

4. Proposed Algorithm

The main aim of the proposed work is to improve the performance in superior level than POBBx indexing technique. In this proposed work, the conversion of multidimensional data into single, the interval value, searching technique are all same as POBBx index method. The major work concentrate in node updation in the trees. The new technique applied are called Crossbreed update. In Moving object indexing, the following three factors plays the important role for effective indexing and they can change frequently based on time 1. Location of Objects, 2. Distribution of Objects, and 3. Workload. In order to avoid migration as much as possible while keeping the tree size relatively small, we have applied Crossbreed Update technique in POBBx indexing.

Crossbreed Update :

The principle of Crossbreed update is to update as many objects as possible without increasing the number of input/output accesses. That means, the object identity, current location and velocity for each moving object is known. Based on this information the future is predicted and applied Crossbreed update. So, some of the objects are shifted from current tree to some distanced tree instead of next immediate tree. So less objects are in the old tree and there is more chance to reduce the migration process. Crossbreed Update accesses the same tree nodes as regular update. In a Time t, more objects are updated to the tree than usual. Fewer objects are left in the older sub tree and the migration cost decreases. It saves the cost of regular update as well. Another kind of performance improvisation by considering the density of the object. Intrinsically, moving objects are spatial objects whose positions change with time. Here, once the time is splited with the maximum update time interval, the objects are accessed based on the density in tree. Due to this, dense area can attain more attention than the sparse areas. So that We can effectively index the moving objects than POBx indexing method nearly 15-25% of efficiency. Updatation cost and migration costs are reduced upto 20-30% when compared to POBBx indexing method.

The proposed algorithm of SOBBx index is as follows,

1. Find out the maximum update interval for each object and the maximum interval value is stored in ui.
2. The maximum update interval Ui is make it as twice and then based on this interval the linear array is formed for ts1,ts2,ts3, etc.,
3. Array of n equal intervals of ts1, ts2, ts3, etc
4. Each object lifespan are find out that is stored in LE. In this Multi dimensional points representing object paths by Hilbert curve coordinates.

5. For each lifespan the data are stored in the tree. Crossbreed Update of Node[i] with object identity, Current location and velocity of the object. If the insertion node C is lesser than the node N then the node C inserted on left else inserted on right. If already the nodes are there the same way created and stored. The insertion time for each object is stored in the variable Arr and total object is inserted is stored in the variable Tot
6. For each move from one tree to another, While Arr not equal to Null, it is checked whether all the moving objects are reached to the new tree or not, if

   it is reached, count ⟵ count the number of objects in T.

```
    Region ⟵ find regions based on space present in T.
  while(Region) do
        Slice the tree with density level.
  end
  If Node[i] is in ts[Pos]
        Begin
        Crossbreed Update of Node[i] with object identity, Current location
      and velocity of the object.
          Update Node[i] to ts[Pos] with position value
        End
        Else call Migrate Node[i] to ts[Pos] with position value
    End
```

Fig 4: Algorithm to Tree Construction, Object Insertion, Updation and Migration

Crossbreed Update of Node[i] with object identity, Current location and velocity of the object: (oid, x,v)

```
oid ⟵ Identity of object.
x ⟵ Current location of the object.
v ⟵ Velocity of the object.
L ⟵ indexed position of the object.
oldtree ⟵ last tree value of the object.
oldkey ⟵ Indexed position of the object.last tree value of the object.
curtim ⟵ current time of the object.
x' ⟵ New location of the object [current time, location and velocity of object].
newtree ⟵ Tree value which contains the location x'.
newkey = computeKey(x');
Delete the object from the tree oldtree of the position oldkey.
Insert the object in the tree newtree at the position of newkey.
```

Fig 5 : Algorithm for Crossbreed Update.

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

Algorithm Migrate(Eo; En)

```
Input: Eo and En are old and new objects respectively
Oodum ⟵ indexed position of the object Eo.
Thisarr ⟵ last tree value of the object Eo.
Curpos ⟵ current position of the object En.
Curtim ⟵ current time of the object En
Curarr ⟵ current tree of the object En which lifespan contains Curtim.
Remove the object Eo from the tree Thisarr of the position Oodum.
```

Fig 6 : Algorithm for Migrate

5. Performance Studies

The figure 7 shows how the objects moving randomly in vague path and it describes the clear path of the every moving objects. In this example 4 moving objects are consider for indexing. The starting time is 13 ms and the ending time is 187.93755639 ms, this is clearly shown in the figure 7. In figure 7 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).

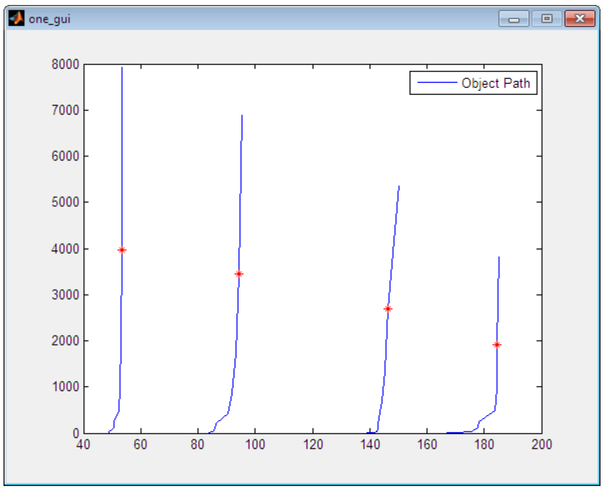


Fig 7 : This figure shows how the objects moving randomly in un specified path. And It describes the clear path of the every moving objects.

The figure 8 shows how the processing speed are vary for all the four cases. The SOBBx index performance is better than other methods.

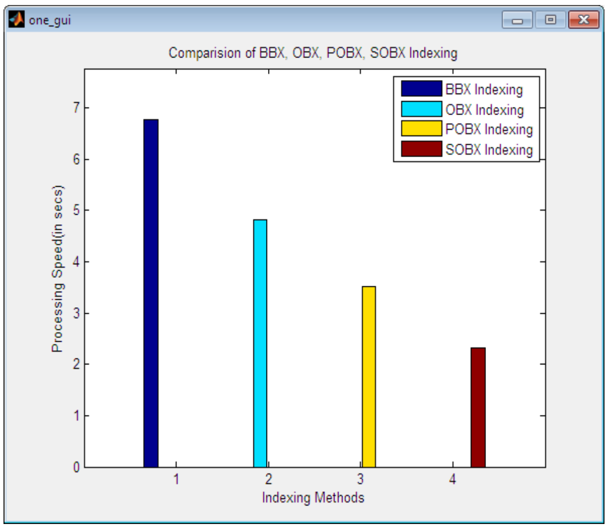


Fig 8 : Comparison of processing speed of BBx,OBBx , POBBx and SOBBx

Actually in this strategy the number of created tree is same in OBBx index method, POBBx Index and SOBBx techniques. But vary in BBx index method because of interval. This is clearly shown in the figure 9.

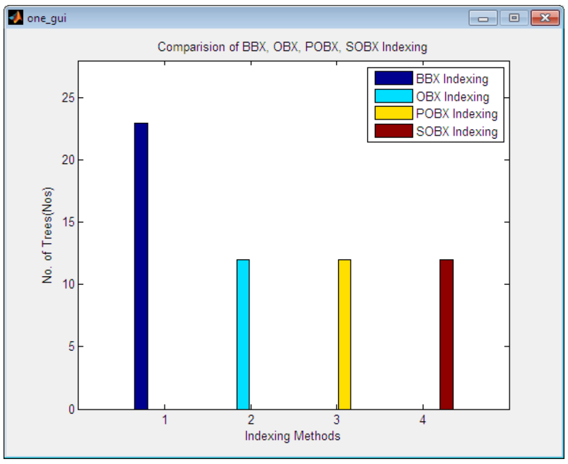


Fig 9 : Comparison of creation of number of trees in BBx,OBBx , POBBx and SOBBx

The figure 10 shows the number of migration hits in all the four cases. The number of migration hits are same in OBBx index method, POBBx Index and SOBBx techniques. But vary in BBx index method because of interval.

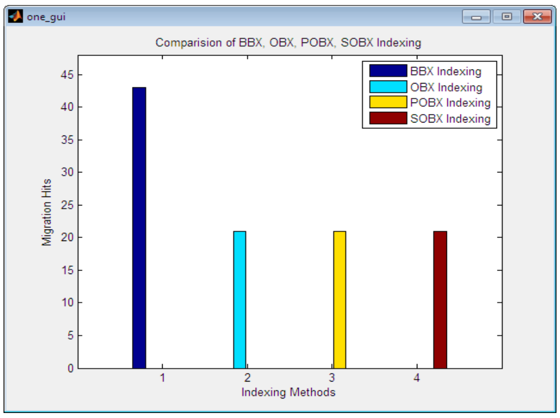


Fig 10 : Comparison of number of migration hits of BBx,OBBx , POBBx and SOBBx

The figure 11 shows the number of node access in all the four cases. In SOBBx the number of node access is very less than other methods. This is because of Crossbreed update and the objects are accessed based on the density in tree. Due to this, dense area can attain more attention than the sparse areas.

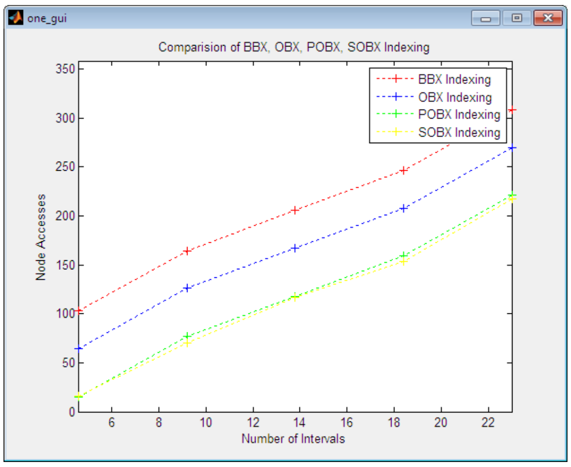


Fig 11 : Comparison of number of node access of BBx,OBBx , POBBx and SOBBx

The figure 12 shows the storage requirement of all the four cases. In SOBBx index method the storage requirement is less when compared with other three methods, so automatically the storage requirement also very less than other methods.

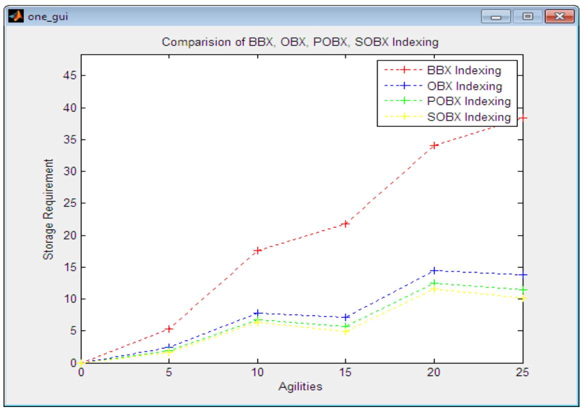


Fig 12 : Comparison of storage requirement of BBx,OBBx , POBBx and SOBBx

The figure 13 shows the total update time of all the four methods. The SOBBx Index method update time is very less than other methods.

The figure 14 shows the total Migration time of all the four methods. The SOBBx Index method Migration time is very less than other methods.

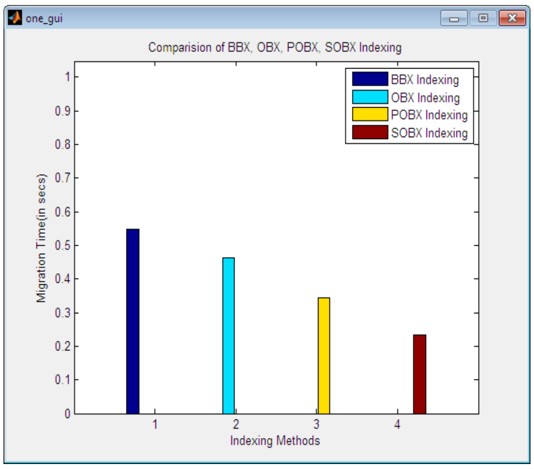
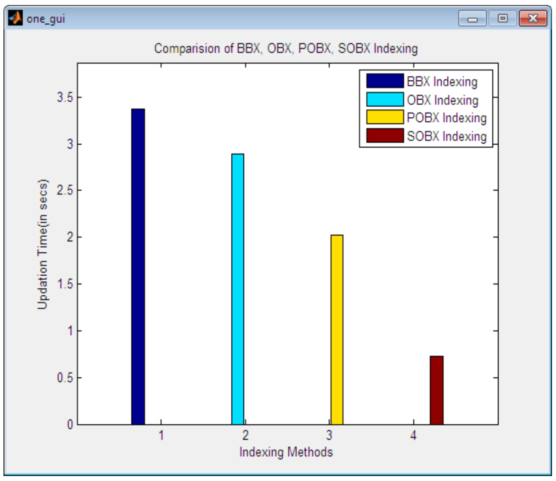


Fig 14 : Total Migration Time

Aspects	**BB^x**	**OBB^x**	**POBB^x**	**SOBBx**
Processing Time	7.768861e+000	4.365904e+000	4.404340e+000	3.466709e+000
No. of Trees	28	14	14	14
Migration Hits	73	37	37	37
Node Accesses	4.893000e+002	3.719000e+002	3.323000e+002	3.288220e+002
Storage Requirement	4.740000e+001	1.660000e+001	1.380000e+001	1.240000e+001
Updation Time	3.280862e+000	2.541900e+000	2.468457e+000	1.260982e+000
Migration Time	4.619041e-001	4.516481e-001	3.683524e-001	2.654879e-001

Table 1 : comparison of BBx , OBBx and POBBx



Figu 13 : Total Updation Time

6. Results

Using MATLAB the following results are produced.

The number of Moving Objects consider is : 4

Starting Time : 13.00000000

Ending Time : 187.93755639

For BBX, Maximum Anticipated Time Interval : 6.34004697

For OBX, Maximum Anticipated Time Interval : 12.68009393

7. Conclusion

This paper applied a new advanced indexing technique, the SOBBx-index (Space Based Optimal BBx-index), which can answer queries about the past, the present and the future. The SOBBx -index is based on the concepts underlying the POBBx-index. Besides the SOBBx -index is compared with BBx index, OBBx index and POBBx index methods under 7 different aspects that is mentioned in table 1. There is no change in number of trees created, number of migration hits in

OBBx-index, POBBx-index and SOBBx index methods. The SOBBx -index performance is better than other three methods in all the aspects like the processing speed, update time, migration time, storage requirement and node accesses. The Future work is planed to improve the performance in non-linear function.

8. 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 Sˇ 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, oustafa 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. owards 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.

[21] K. Appathurai, Dr. S. Karthikeyan 2012, “A New Proposed Algorithm for OBBx-index Structure”, International Journal of Computer Applications, Vol. 50[11], 0975 – 8887.

[22] SU CHEN, BENG CHIN OOI and KIAN-LEE TAN, “Continuous Online Index Tuning in Moving Object Databases”, ACM Transactions on Database Systems, Vol. V, No. N, Month 20YY, Pages 1–45.

AUTHOR’S PROFILE

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.