A NOVEL INDEXING METHOD FOR VEHICULAR NETWORKS

K. Appathurai [1] *and M. Anandkumar* [2]

Abstract

Spatiotemporal applications are widely used in the research area. The Spatiotemporal access methods are secret into four categories. The BB^x-index Structure algorithm is fourth category of Spatiotemporal access method. The new algorithm is proposed for past, present and future detection of moving objects of VANETs. In this proposed algorithm contains tree construction, object insertion, updation and migration. Multidimensional object data is converted to single dimensional data using Hilbert curve. This paper precisely focuses on to reduce the migration process done by the existing BB^x index method and minimized time complexity especially in VANETs.

Keywords: BB^x index, Moving Objects , Hilbert Curve and VANETs.

I. Introduction

Moving objects are changing their locations over time in Spatio-temporal databases. The moving objects report their location to the server through devices. Spatiotemporal access methods are 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, 2, 3, 6, 13]. The server stores all updates from the moving objects. Some algorithms are answering queries about the past [4, 5, 9, 10,15] information only. 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. In one case Moving Object Detection Algorithm Based on Variance Analysis [16]. To predict future positions of moving objects in VANETs, the spatio-temporal database server may need to store additional information, e.g., the objects' speed [8, 17]. 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 [6]. This proposed algorithm reduces the migration process, so the total performance is better than BB^x index structure.

II. Related work

The BBx index Structure

The BB^x index is the extension of B^x tree index [7]. The B^x tree index support only for the present and future positions, but in BB^x index [6] it extend to the past information also. 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

[1] Associate Professor and Head, Dept. of Information Technology, Karpagam University, Coimbatore – 21.

[2] Associate Professor Dept. of Information Technology Karpagam University, Coimbatore – 21.

from the space-filling curve; t_{start} indicate the time when the object was inserted into the database (matching to the tu in the description of the Bx-tree), and t_{end} indicate 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: t_{start} and t_{end} 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 3). 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 life spans 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. In query processing based on the timestamp signature it expand either backward for past information and expand forward for future information.

III. Statement of Problem

In BB^x index structure in certain cases half objects are updated and half objects are forced to update. This causes more work to the entire process and automatically it take more time for indexing and it take more memory space for VANETs. In addition, in tree the node insertion, deletion also complex process when the number of moving objects is high.

IV. Proposed Algorithm

The main aim of the proposed algorithm is to decreases the complexity of BB^x index structure in VANETs. Besides the overall performance of the proposed algorithm is good than BB^x index about 50% for Vehicular Networks. The proposed algorithm is called $VOBB^x$-index (Vehicular Optimized Broad B^x). The scalability is considered as twice for the better result. The scalability is try to make it as thrice or fours the total performance is not good, because the depth of the tree is more so the searching time is high while the nodes are inserted or deleted. So, the scalability is make it as twice we get the optimum result and the performance also good than BB^x. It is proved by MATLAB implementation.

The $VOBB^x$-index the nodes consist of the form (x _rep; tstart; tend; pointer.) where x_rep is nothing but one dimensional data obtained from the space-filling curve; t_{start} 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 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. The $VOBB^x$-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 life spans 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 fairly small and can usually be stored in main memory. Initially the maximum update interval is found out among all the moving objects. 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.

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.
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.

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

All trees have lifespan after that the tree values are updated to next tree. So initially 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 below algorithm shows how the updation takes place in $VOBB^x$. 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*)

1. Here Eo and En are old and new objects respectively Input:

tindex ¬ time Eo is indexed in the tree

find tree Tx whose lifespan contain tindex

2. Find the position of the object in the tree

posindex ¬ position of Eo at tindex

3. locate Eo in Tx according to keyo

keyo ¬ x-value of the posindex

4. 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

5. insert En into the latest tree according to keyn

Figure 2: 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 VOBB^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*)

1. Here Eo and En are old and new objects respectively find tree Tx whose lifespan contain tindex tindex ¬ time Eo is indexed in the tree

2. Based on tindex the position of the object is find out posindex ¬ position of Eo at tindex

3. locate Eo in Tx according to keyo keyo ¬ x-value of the posindex

4. modify the end time of Eo's lifespan to current time

Figure 3 : Algorithm for Migrate

V. PERFORMANCE ANALYSIS

The below figure 4 shows how the objects moving randomly in un specified path and it describes the clear path of the every moving objects. In this example 9 moving objects are consider for indexing. The starting time is 32 ms and the ending time is 210.79660866 ms, this is clearly shown in the figure 4. In this figure 4 the 'x' axis is time and 'y' axis is point's i.e. by Hilbert curve the multidimensional data is converted as points (single dimensional data).

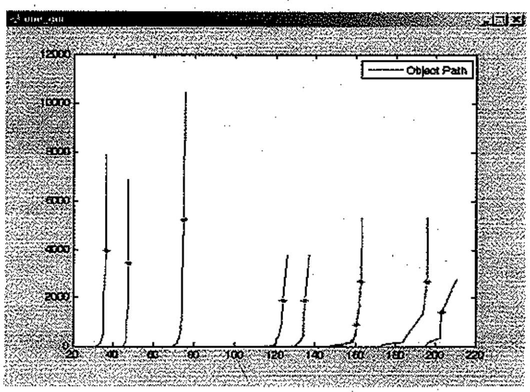


Figure 4: This figure shows how the objects moving randomly in un specified path. And It describes the clear path of the every moving object.

In figure 5 shows the total indexing time for both the methods like BB^x index and $VOBB^x$ index. The total processing time for BB^X Indexing is 1.059695e+001and the total processing time for $VOBB^X$ Indexing is 6.200636e+000. so it clearly says the $VOBB^x$ method is much better than BB^xmethod.

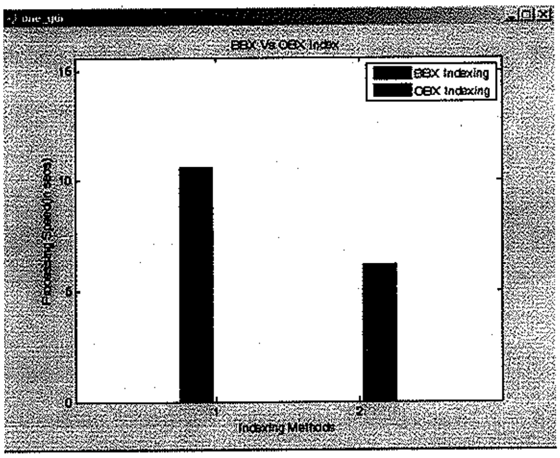


Figure 5: Comparison of BB^X and $VOBB^X$ indexing in terms of Processing Speed

In figure 6 indicates the number of migration hits occur in both the techniques. As per this concern also the $VOBB^x$ index techniques is much better than BB^x index techniques. The migration hits for BB^X Indexing is 68 and the migration hits for $VOBB^X$ Indexing is 34. This reducing of migration hit in $VOBB^x$ index method improves the total performance of $VOBB^x$ index method, reducing the processor utilization time and it decreases the total cost.

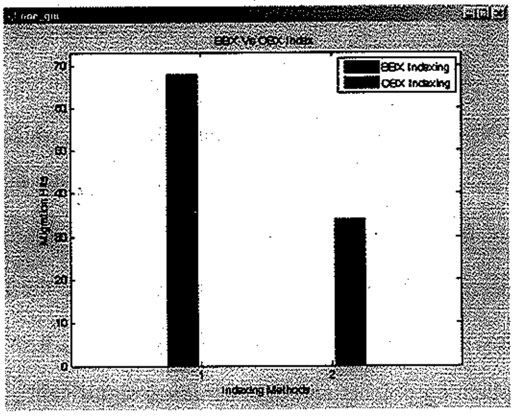


Figure 6: Comparison between BBX and VOBBX

VI. Results

In this section both techniques results are mentioned. This is reported by MATLAB.

The number of Moving Objects considers is: 9

Starting Time: 32.00000000

Ending Time: 210.79660866

For BB^X, Maximum Anticipated Time Interval: 10.79877393

For $VOBB^X$, Maximum Anticipated Time Interval: 21.59754786

Processing Time for BB^X Indexing: 1.059695e+001

Processing Time for $VOBB^X$ Indexing: 6.200636e+000

Migration Hits for BB^X Indexing: 68

Migration Hits for $VOBB^X$ Indexing: 34

VII. Conclusion

This paper proposed a new indexing algorithm, the VOBB^x-index (Vehicular Optimized BB^x-index), which can answer queries about the past, the present and the future. This indexing techniques based on the concepts underlying the BB^x-tree index structure. Like the BB^x-index, the indexing of historical information, it avoids duplicating objects and thus achieves significant space saving and efficient query processing. Also 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 barely reduces time complexity. Extensive performance studies were conducted that indicate that the VOBB^x-index outperforms the existing state of-the-art method, with respect of historical, present and predictive queries. This proposed work is best suited for Vehicular Networks.

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. ¡ 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] D. Lin, C. Jensen, B. Ooi, and S. ¡ Saltenis. Efficient indexing of the historical, present, and future positions of moving objects. In MDM, pages 59–66, 2005.

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

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

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

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

[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.

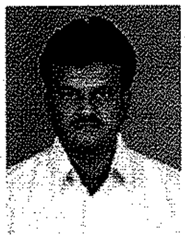
[14] Mohamed F. Mokbel, Xiaopeng Xiong, Moustafa A. Hammad, and Walid G Aref, Continuous Query Processing of Spatio-temporal Data Streams in PLACE, 2004 Kluwer Academic Publishers. Printed in the Netherlands

[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.

Authors' Biography



Dr. K.Appathurai working as Associate professor and Head department of Information Technology in Karpagam University, having 15 years experience in teaching. His area of interest is spatial database. He has published 12 papers in the reputed journals. He is the Editorial member in several international and national journals. He has presented twenty papers in national conferences and four papers in international conferences.

Dr. M. Anand Kumar has completed M.Sc and M.Phil in computer science from Bharathiar University. He has Completed Ph.D in Karpagam University and currently working as an Associate Professor in karpagam University having ten years experience in teaching. His area of research includes network security and information security. He has presented twenty papers in national conferences and four papers in international conferences. He has published twelve papers in international journals.