International Journal of Soft Computing and Engineering (IJSCE)
ISSN: 2231-2307, Volume-2 Issue-2, May 2012

An Analysis on Update Strategies for Spatio-Temporal Indexing

Lakshmi Balasubramanian, M. Sugumaran

Abstract:- Many applications such as location based systems, traffic monitoring, radio frequency identification, sensor networks etc., benefit from spatio-temporal indexing. R-Tree based index structures are widely used for indexing the spatial information. The main issue to be considered is frequent updates. These applications pose frequent updates which have to be reflected in the index structure. Frequent changes to the index structure causes more overhead. Recent research is to handle these frequent updates efficiently. This paper presents the state of art in the update strategies adopted in spatio-temporal indexing. This work provides an idea of the present development in updating techniques for spatio-temporal indexing.

Index Terms:- R-Trees, Spatio-Temporal Indexing, Update Strategies.

I. INTRODUCTION

Applications such as Global Positioning System (GPS), Computer Aided Designing [1] etc., have made storage of multi-dimensional data mandatory nowadays. Indexing these data is necessary for fast retrieval. Spatial index structures such as grid files, R-Trees, Quad trees [3] exist for this purpose. R-Tree [2] and its variants are the most widely used for its efficiency and simplicity. R-Trees represent the objects as Minimum Bounding Rectangles (MBRs). R-Tree can be extended to support spatio-temporal data also. Applications [1] such as traffic monitoring, radio frequency identification, sensor networks, multimedia, location aware systems etc., benefit from spatio-temporal indexing. With the increase in necessity to track moving objects, find the objects nearby during travel, finding a neighbour for some purpose etc., spatio-temporal databases are gaining more importance. Since these data keep moving, accuracy of query results depends on fast retrieval of data. If retrieval takes time, the actual result and the result obtained differ because the objects would have moved by the time the result was retrieved and the result might become less useful. So, for faster retrieval indexing of these data is required. Though indexing provides quicker query results, there are some issues in indexing spatio-temporal data. One of the main issues is to handle frequent updates. Existing spatial index structures are not efficient to handle frequent updates. This poses a lot of overhead to the index structure. Update is normally a costly process as it involves a delete and insert. Frequent updates increase the update cost dramatically. Techniques to handle update efficiently are thus necessary. Techniques like lazy splitting [4], lazy update [5], extended MBR [5], batch update [6], deferred delete [7], deferred delete and insert [8] etc., have been proposed to handle the issue effectively. Lazy splitting avoids splitting the node when it is full by finding free space in some nearby node. Lazy Update [5] updates only when the object moves out of the present MBR. Extended MBR [5] approach expands the MBR of the present leaf to some extent so that zigzag movement of the objects do not pose frequent changes in the structure. Bulk update [6] technique takes advantage of the fact that not all updates need to be considered as separate operations, they might share a common path which when exploited reduces the disk accesses. Buffering technique [7] defers the deletes or both the deletes and inserts [8] and makes them effective in a bulk when the buffer is full. Normally R-Tree is traversed top-down for any operation. But bottom up approach [9] was proposed to tackle updates efficiently.

This work presents an analysis on these update strategies adopted in spatio-temporal indexing. Section 2 provides a brief overview on the most used index structure, the R-Tree and its spatio-temporal variant TPR-Tree. Section 3 provides an analysis on the state of art update strategies in spatio-temporal indexing and finally Section 4 provides the conclusion.

Manuscript received on April 26, 2012.

Lakshmi Balasubramanian, Department of Computer Science and Engineering, Pondicherry Engineering College, Puducherry, India, (e-mail: lakshmi@pec.edu).

Sugumaran. M, Department of Computer Science and Engineering, Pondicherry Engineering College, India, (e-mail: sugu@pec.edu).

II. INDEX STRUCTURE

The main index structures used for spatial indexing include R-Tree [2] and its variants. TPR-Tree [10] is an efficient variant of R-Tree supporting spatio-temporal indexing. The following sub sections provide basic details of R-Tree and the TPR Tree.

A. R-Tree

R-Tree [2] is a disk resident index structure. It is height balanced and multi-dimensional version of B-Tree [11]. The structure is designed such that a search requires visiting only a small number of nodes. Inserts, deletes, updates and queries to R-Tree can be mixed and hence the structure is dynamic. R-Tree consists of a root node, non leaf nodes and leaf nodes. The leaf nodes contain the unique ID of the object, MBR of the object, and pointer to the actual data. MBR is the smallest rectangle that covers the object. The non-leaf nodes contain the MBR which covers all its children and pointer to the children. A set of properties [2] must be satisfied for the structure to be a valid R-Tree. The properties includes details on maximum and minimum

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication

limit of entries that a node can hold, the fan out of the root, index and leaf nodes and balance property.

Operations like insertion, deletion, updating and querying can be carried out in R-Tree. Insertion involves traversing the tree from root to the leaf node where the object can be put such that the properties of R-Tree are satisfied. During traversal, the node which needs least change to put the new object is chosen and travelled down. After insertion, if the node exceeds the maximum capacity, splitting is done so that the newly formed nodes have entries between the minimum and maximum capacity. Splitting incurs more disk accesses. Hence many splitting methods were proposed to handle situations efficiently. Some of them include linear [2], quadratic [2], R* [12], 2-3 [1], branch grafting [1], clustering [13] etc. Deleting involves multi path search. When an object is to be deleted, first the object is located by traversing from root to some leaf node. The search may be multi path due to overlap. During deletion, if the number of entries becomes less than the minimum capacity then the tree is adjusted to satisfy the properties. Updating a record involves deletion of old value of the entry and insertion of new value of the entry. Searching records which lie within a given window is the most often used query. The tree is always traversed from top to bottom. The non-leaf records which overlap the given window are chosen during traversal for further exploration. This is done at every level until the leaf nodes are reached. The leaf nodes which overlap with the given window are the qualified records.

B. TPR-Tree

A TPR-Tree [11] is a spatio-temporal variant of R-Tree. TPR-Tree represents a moving object with a MBR which describes its extent at time *t* and a VBR (Velocity Bounding Rectangle) which describes the upper and lower bound of the velocity in each dimension. Leaf and non leaf nodes both store MBRs and VBRs. Insertion, deletion and updating processes are same as R-Tree. It can handle queries based on future positions by using the VBR. The extent to which it can predict the future is called the horizon of the tree.

Update methods are not efficient to handle updates from update-intensive applications. Efficient schemes are necessary to handle these frequent updates. Next section highlights on the analysis of the update techniques in the literature for spatio-temporal indexing.

III. ANALYSIS OF STATE OF ART UPDATE STRATEGIES

Indexing moving objects faces many challenges. One of the main challenges is to handle numerous updates efficiently. Some techniques such as lazy splitting [4], lazy update [5], Extended MBR [5], Bulk loading [6], update memos [7] and semi bulk loading [8] are briefed and their advantages and disadvantages are highlighted. A synopsis on generalized bottom up approach [9] is also presented. Updating protocol [14] which could reduce the number of update messages is also discussed.

A. Lazy Splitting

Lazy Splitting [4] is avoiding or deferring the costly splitting process thereby saving disk accesses. When an object is inserted into a leaf that is full, then a nearby leaf which has free space is chosen and the data is added into that leaf and the original leaf's MBR is expanded to contain the new entry. This method reduces update cost by trying to avoid disk access consuming processes like splitting and merging unless they are absolutely necessary. When a data is deleted the existing MBR may have to contract. Though the update cost is reduced with this method, the search cost increases since the extension of MBR would cause overlap. Leaf nodes which are nearby need not be near based on distance. This fact still degrades the query performance.

An improvement to this technique was to search for free nodes only with the siblings. If a data member is inserted into a node and that node is full, the parent of that node will check to see if any other child of the parent node can hold the new entry. If the parent node is full, the child will split. If the parent node does have a free space, it will find a child with a free space. The parent will take a data member from the node directly next to the free node and insert it into the free node. Next, the MBRs for both of the nodes will be adjusted and process repeats. Eventually the node where the new point is to be inserted will have an available spot. This ensures that the leaf nodes will be packed to their maximum capacity. This technique can be effective for updates which are not very frequent.

B. Lazy Update

Lazy Update [5] technique updates the entry only if it moves out of the present MBR. The update operation first finds the old entry and deletes it. During deletion, the changes are propagated till the root. Now, if the object is again inserted into the same leaf, then the change propagated is waste of cost. The cost incurred is still more when the deletions caused underflow and thus lead to merging and after insertion of new entry to the same node caused splitting. The lazy update technique updates the structure of the index only when an object moves out of the corresponding MBR. If a new position of an object is in the MBR, it changes only the position of the object in the leaf node. It can update the position of the object quickly and reduce update cost greatly. This is more useful for slow moving objects which change their positions but not far enough to move away from its MBR.

C. Lazy Update with Extended MBR

Extended MBR [5] approach is to expand the current MBR to some extent so that the objects do not move out of the current MBR. An object that is on the boundary of an MBR can easily move out of the MBR. If an object zigzags along the boundary of an MBR, deletions and insertions can occur continuously even when adopting lazy update technique. This can reduce the improvement in cost savings due to lazy update technique as every time a normal update must take place. To prevent this problem, a slightly large bounding rectangle called the Extended MBR (EMBR) instead of an MBR is used. The EMBR is used only for leaf nodes. The actual MBR is expanded to certain extent to hold the object. If the object is still out of the MBR, then normal update has to take place. Since the EMBR is larger than the corresponding actual MBR, overlaps tend to increase

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication

degrading the search performance. A trade-off between the gain of the update performance and the loss of the search performance should be maintained. This technique also can support only slowly moving objects. If the objects move out of the EMBR, then this method would not be effective.

D. Deferred Delete

Deferred delete [7] technique does only the insertion part of the update immediately and defers the delete part. The update consists of deletion of old entry and insertion of new entry. Multipath search is required for deletion. Thus if the deletion of old entry is not done immediately but done in bulk by a garbage cleaner and only insertion of the new entries is performed immediately, the cost of update reduces significantly. This is the main basis of update memos. Update memo is a structure which stores the object ID, latest timestamp of the object and number of obsolete entries (that has to be deleted). Whenever an update is issued, if the object ID is present in the update memo, then the timestamp is updated to the current and the number of obsolete entries is increased by one. If the object ID is not present in the update memo then a new entry_for it is created in the update memo. Then the new entry is inserted in the tree. The deletion is done by the garbage cleaner in a lazy manner and in batches (i.e.) all obsolete entries in a leaf are deleted at the same time. Thus the multipath search for every delete is reduced to a single path search to a leaf having a number of obsolete entries and then passing it to the next leaf and so on till all the obsolete entries are deleted. This methodology supports indexing of objects' current and future positions. This technique can cause unnecessary overflows due to the presence of obsolete entries. This technique can be adopted for fast moving objects also.

E. Semi Bulk Loading

In semi bulk loading [8], both the insertions and deletions are deferred. A small in memory buffer is exploited to defer the operations. When the buffer is full, flushing algorithms [8] like flushAll, flushLRU, flushLFU are used to flush the entries to the disk. FlushAll flushes out all the entries to the disk. FlushLRU flushes a certain amount of the entries which have least timestamp. FlushLFU flushes a certain amount of entries which have less number of updates. Semi bulk loading technique provides good results on frequently moving objects. Optimization on the amount of entries to be flushed and the buffer size is provided with analysis. More flushing algorithms can be analyzed so that the performance still increases.

F. Bulk Loading

Bulk loading technique [6] exploits the common path that the updates may take and updates group of entries which share a common path in a batch. Update has always been considered as an individual operation. When numerous updates are issued, each is considered as an individual operation and processed. But the updates may share a common path which when utilized can reduce the update cost. The updates which share a common path are grouped and bulk loaded so that the separate disk accesses to every operation is saved and reduced to that of loading to one node. The loading pool is partitioned into buckets and updates are grouped in buckets according to some hash function such that updates in a bucket share a common path. There exist different schemes for loading the buckets [6]. The MBR bias scheme considers the old position of the entries. The entries whose old positions belong to the same leaf node are grouped in one bucket. The Grid bias considers the effect of nearness of new positions of the objects. Though the exact MBR is not known at this stage, a grid is formed and an approximate MBR is calculated and those which would fall in same leaf are put in one bucket. Hybrid Biased scheme considers both the old and new positions. The buckets are indexed with both the leaf node ID of old MBR and grid cell ID of new MBR. The objects which have common leaf node for old MBR and common grid cell ID for new MBR are considered to share a path. When the buckets are full, they are flushed to the disk resident index structure. This technique suits moving objects with any speed.

G. Generalized Bottom Up Approach

Bottom up approach [9] exploits the fact that all entries are stored at the leaf level. So traversing from bottom might save disk accesses. Usually the deletion and insertion operations in an update are processed by traversing from the root to the leaf node. But the operations actually take place at the leaf. So rather than top-down approach, bottom up approach is expected to perform better. A generalized bottom up approach was proposed and proved to outperform the top down approach. A direct access table to the internal nodes is maintained. It stores the ID and MBR of the node. A bit vector on leaf nodes is maintained to find whether the node is full or not. A hash table which provides directs access to the leaf nodes. Using these structures, generalized bottom up approach is as follows; whenever an update is issued, first using the hash table, it is checked whether the updated entry exists within the same old MBR. If so then the position is updated in the leaf node directly. If the object is not in the same MBR, then it is checked whether expansion of the leaf node can hold the entry. If it so then the object is put in the leaf node and the hash table is modified with the new boundaries of the leaf. If the object does not exist in the leaf node even after the expansion, then the direct access table is used to locate the leaf node it can be placed. If the leaf node in which it has to be place is full, the next sibling is found using the bit vector table. The maintenance of the secondary structures is not expensive and they do not occupy more space in main memory. This generalized bottom up approach when integrated with any update technique performs better then the top down technique.

H. Updating Protocol

The new updating protocol [14] suggests a tolerance region which reduces the updates since updates are not necessary until the object is within the tolerance region. The protocol for updates can be modified to reduce the number of messages between objects and database server. This reduces the overall workload of the system and eventually reduces update cost. In conventional systems, exact location and velocity of the object is used to predict the future positions. Spatio-Temporal Safe Region is a tolerance provided to the objects such that they are free from the velocity and location updates as long as

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication

they are within the error bounds. Safe region is calculated on two objectives: reduced update workload and query accuracy. Query accuracy is achieved by the database server by probing exact location of the object when the safe region does not provide adequate information. These probing are called passive updates. Thus the new protocol supports both active updates (object to database server) and passive updates (database server asking for objects' update).
The update techniques in the literature can thus be classified as those supporting slow moving and fast moving objects, those that are done in bulk and those done individually, those that utilizes top-down and those that utilizes bottom-up approaches.

IV. CONCLUSION

Spatio-temporal indexing is used by many applications and most of them are update intensive. The index is faced with numerous updates but is not competent to handle them. Update involves a multipath delete and insert and thus is a costly process. Several techniques have been proposed in the literature to handle updates. These methods may either support slow moving or fast moving objects; can be processed individually or in bulk; utilize top-down or bottom-up approaches. The techniques analyzed in this work include lazy splitting, lazy update, lazy update with extended MBR, deferred delete, semi bulk loading and bulk loading. The major drawback with lazy splitting and extended MBR was overlap which could degrade search performance. Lazy update could support only slow moving objects. Deferred delete may cause more overflows due to the presence of the objects which have to be deleted. Every technique was analyzed and bottom up technique provided a promising cost saving. Update protocol which suggested less number of messages between server and objects and supported both passive and active update is also highlighted. Further research works can take advantage of the rich resource available in this field and proceed further to design a more efficient update strategy to handle frequent updates.

REFERENCES

1. Yannis Manolopoulos, Alexandros Nanopoulos, Apostolos N. Papadopoulos and Yannis Theodoridis, *R-Trees: Theory and Applications*, London: Springer, 2006, 1st Ed.
2. A. Guttman, "R-Trees: A Dynamic Index Structure for Spatial Searching", *Proceedings of the 1984* ACM SIGMOD *International Conference on Management of Data,* 1984, pp.47-57.
3. Raphael A. Finkel and Jon Louis Bentley, "Quad Trees: A Data Structure for Retrieval on Composite Keys", *Journal of Acts Informtica*, vol.4, no.1, 1974, pp.1-9.
4. Barrios. J, Makki. S.K and Karimi. M, "An Indexing Structure for Mobile Objects Utilizing Late Update", *Proceedings of the 7th International Confernce on Information Technolofy: New Generations,* 2010, pp.162-167.
5. Dongseop Kwon, Sangjun Lee and Sukho Lee, "Indexing the Current Positions of Moving Objects Using the Lazy Update R-Tree", *Proceedings of the 3rdInternational Conference on Mobile Data Management,* 2002, pp.113–120.
6. Xiaoyuan Wang, Weiwei Sun and Wei Wang, "Bulkloading Updates for Moving Objects", *Proceedings of the 7thInternational Conference on Web-Age Information Management*, 2006.
7. Xiaopeng Xiong and Walid G. Aref, "R-Trees with Update Memos", *Proceedings of the 22nd International Conference on Data Engineering,* 2006.
8. MoonBae Song and Hiroyuki Kitagawa, "Managing Frequent Updates in R-Trees for Update-Intensive Applications", *IEEE Transactions on Knowledge and Data Engineering*, vol.21, no.11, 2009, pp.1573-1589.
9. M.-L. Lee, W. Hsu, C. S. Jensen, and K. L. Teo, "Supporting Frequent Updates in R-Trees: A Bottom-Up Approach", *Proceedings of of the International Conference on Very Large Databases,* 2006.
10. S. Saltenis, C.S. Jensen, S. Leutenegger and M. Lopez, "Indexing the Positions of Continuously Moving Objects", *Proceedings of ACM SIGMOD Conference on Management of Data*, 2000, pp.331-342.
11. Douglas Comer , "Ubiquitous B-Tree", *Journal of CAN Computing Surveys,* vol. 11, no.2, 1979, pp.121-137.
12. N. Beckmann, H.-P. Kriegel, R. Schneider and B. Seeger, "The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles", *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1990, pp.322-331.
13. Pan Jin and Quanyou Song, "A Novel Index Structure R*Q-Tree based on Lazy Splitting and Clustering", *Proceedings of the International Conference on Computer Science and Automation Engineering,* 2011, pp.405-407.
14. Su Chen, Beng Chin Ooi and Zhenjie Zhang, "An Adaptive Updating Protocol for Reducing Moving Object Database Workload", *Journal Proceedings of VLDB Environment*, vol.3, no.1, 2010, pp.735-746.