Efficient CNG Indexing in Location-aware Sevices

Yuni Xia Sunil Prabhakar
Department of Computer Science
Purdue University, West Lafayette, IN 47906, USA
{xia, sunil}@cs.purdue.edu

Abstract

In this paper, we propose and evaluate a new in-memory index structure for efficient processing of range queries over moving objects. Traditional spatial index approaches suffer from the need for frequent updates and therefore result in poor performance in moving object environments. To reduce the number of updates, many existing approaches use a linear function to represent the movement of objects, but for numerous real applications, the movements of objects are too complicated to be represented as a simple linear function. The index structure we propose has two levels. The lower level consists of a number of cluster nodes, each of which is in charge of a group of moving objects and can dynamically adjust its location and covering range according to the movement of the objects. The upper level makes use of a grid index to support querying evaluation. With our approach, no assumption is required about the maximum velocity of objects and it is not necessary for objects to move according to some well-behaved patterns.

1 Introduction

The advances of wireless communications technologies, personal locator technology and global positioning systems enable a wide range of location-aware services. Current location-aware services support proximity-based queries including map viewing and navigation, driving directions, searches for restaurants and hotels, weather and traffic information. They include GPS based systems like Vindigo and SnapTrack and cell-phone based systems like TruePosition and Cell-Loc. These technologies are the foundation for pervasive location-aware environments and services. Such services have the potential to improve the quality of life by adding location-awareness to virtually all objects of interest such as humans, cars, laptops, eyeglasses, canes, desktops, pets, wild animals, bicycles, and buildings. Applications can range from proximity-based queries on non-mobile objects, locating lost or stolen objects, tracing small children, helping the visually challenged to navigate, locate, and identify objects around them. The demand for storing, updating and processing continuously moving data arises in a large number of such applications.

In this paper, we address the problem of indexing continuously moving objects, which could be critical for evaluating queries in response to the movement of objects with near real-time responses. Traditional spatial index structures such as Rtree are not appropriate for indexing moving objects because the location changes of objects may cause splitting or merging the nodes constantly or even rebuilding from time to time. To reduce the number of index updates, many previous schemes use a simple linear function to describe the movements of the objects, where the index and the database are updated only when the parameters of the linear function change. However, in reality, the movements of objects are far too complicated to be accurately represented as a linear function that changes infrequently.

We propose a general dynamic structure to efficiently index the positions of the moving objects and reduce the updating cost to a great extent. This indexing structure has two levels. The lower level consists of a number of cluster nodes, each of which is in charge of a group of moving objects and can dynamically adjust its location, covering range and shape according to the movement of the objects. The upper level makes use of a grid index. With the self-adaptive cluster nodes of the lower level, the index updating overhead caused by object movements is significantly reduced. While the grid indexing of the upper level greatly speeds up various queries such as range/point and nearest-neighbor queries and it could also be very helpful for selectivity estimations for spatial-temporal queries and aggregated queries. We call this index structure CNG(Cluster Node and Grid) indexing. Other index structures including Quadtree and Rtree are also investigated and our results show that CNG indexing achieves a better overall performance than existing approaches in most cases.

The rest of the paper proceeds as follows: Related work is discussed in Section 2. In Section 3, we introduce the basic idea and the structure of CNG indexing. we also present

the details of its updating and query processing. Section 4 presents experimental evaluation of the proposed approach and Section 5 concludes the paper.

2 Related Work

Developing efficient index structure is an important research issue of moving object databases. As a naive approach, multi-dimensional spatial index structures can be used for indexing the positions of moving objects. Numerous index structures have been proposed for indexing multi-dimensional data. [2] did a good survey of these indexing schemes. Recently, in [4] Kanth et al. argue that R-trees are generally better than Quadtrees and Oracle now recommends use of only the R-tree. Although traditional spatial index structures can be used, they are not efficient for indexing the positions of moving objects because of frequent and numerous updating operations in moving environment.

Some new index structures have been proposed for indexing moving objects recently. These index structures can be classified into the two type: index the trajectories (histories) and index the current positions of objects. Our approach belongs to the latter category.

In this category, most approaches describe moving object's location by a linear function, and only when the parameters of the function change, for example, when the moving object changes its speed or direction, the database is updated. Saltenis et al. [9] proposed the time-parameterized R-tree (TPR-tree). In this scheme, the position of a moving point was represented by a reference position and a corresponding velocity vector. When splitting nodes, the TPRtree considers both the positions of the moving points and their velocities. Kollios et al. [6] proposed an efficient indexing scheme using partition trees. Tayeb et al. [10] introduced the issue of indexing moving objects to query the present and future positions . They proposed PMR-Quadtree for indexing moving objects. Agarwal et al.[1] proposed various schemes based on the duality and they developed an efficient indexing scheme to answer approximate nearest-neighbor queries. The problem of all these techniques is that there are hardly exist a good function for describing the objects' movements in reality. In many applications, the movement of objects is complicated and nonlinear. In such situations, the approaches based on a linear function cannot work efficiently. Approximation technique using threshold has been proposed to reduce the updating cost. However, this approximation technique can decrease the accuracy, it is not appropriate for the applications that requires a high precision.

In [8], we propose two approaches, namely Query Indexing (QI) and Velocity-Constrained Indexing (VCI), for indexing moving objects. Both approaches achieve significant improvements over traditional approaches. However,

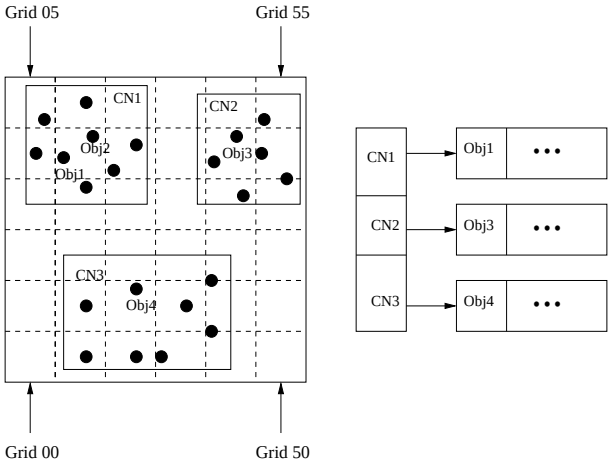

Figure 1. CNG Index

Query Indexing cannot efficiently handle the arrival of new queries, while the VCI index does not have good performance when the number of concurrent queries is large. In [3], several in-memory moving object index structures are investigated and the results show that using a grid-like structure gives the best performance. The evaluation is this paper is also based on Query Indexing(QI).

3 CNG indexing

In this section, we present our index technique, called CNG indexing. It is a general dynamic structure for indexing moving objects. In our work, no assumption is made about the future positions of objects. It is not necessary for objects to move according to well-behaved patterns. And there are no restrictions, like the maximum velocity, placed on objects either. We will first introduce the basic idea of CNG index. followd by its structure. We also present the index updating algorithm and query processing procedure.

3.1 Basic Idea

The CNG index we propose is an in-memory structure. Disk-based indexing may not suit the need for real-time continuous execution of queries over large amount of moving objects. Besides, the dropping main memory costs makes main memory evaluation feasible. In [3], several in-memory moving object index structures are investigated and a grid-like structure gives the best performance. However, for grid indexing, since the space is uniformly partitioned without considering the topography, objects may frequently move from one grid to another grid or move back and forth between two grids. The index updating cost can be high due to the frequent operation of deleting an object from the object list of one grid and insert the object into another grid.

To reduce the updating cost, we hope to first find the natural clusters of objects, create a number of Cluster Nodes(CN). Each CN is in charge of a group of objects that fall in its covering range. When object moves, it reports its movement to the corresponding cluster nodes (CN). The CNs update the object records with latest positions. As long as the object does not move out of the CN's covering range, no other operation needs to be performed, otherwise, the object might be deleted from its current CN and be inserted into another CN. If the CNs are properly selected and reflects the objects' behavior, objects will tend to stay in its CN for a large periods of time, which could significantly reduce the index updating cost.

The CNG index is a two-level structure. The lower level mainly handles object movements while the upper level support query processing. As figure 1 shows, the lower level of the CNG index structure consists of a number of cluster nodes, which divide-and-conquer all moving objects. The upper level is a grid index, which is a two dimensional array of cells. This grid structure can efficiently handle range and point queries.

3.2 Index Structure

Figure 2 shows the structure of CNG index. The whole space is divided into grids. In the Figure 1 example, the space is divided into a 5*5 grids. Each grid has pointers which point to the cluster nodes whose covering ranges intersect with the grid. For example, grid10 and grid11 both intersect with Cluster Node CN3, therefore, they have pointers point to CN3. Since there could be overlaps between the covering ranges of cluster nodes, one grid can have multiple pointers that point to different cluster nodes. Each cluster node has an array of objects. All objects in the array are in the covering range of that cluster node. As Figure 2 shows, Cluster Node 1(CN1) pointers to an array which contains obj1, obj2, etc.

3.3 Index Updating

At each cycle, objects report their new positions to the corresponding cluster nodes. If an object's new position is still within the covering range, the cluster node updates the record of that object with its new position. If the object moves out of the covering range, the cluster node can choose either expand its current covering range, or to pass it to another cluster node, depending on which cluster node needs expanding the least to contain the new position.

If the object should be passed to another cluster node, then, the current cluster node will delete the object from its object array and the new cluster node will insert it into its array. If the number of objects one cluster node contains is too large, which might slow down the query, a cluster

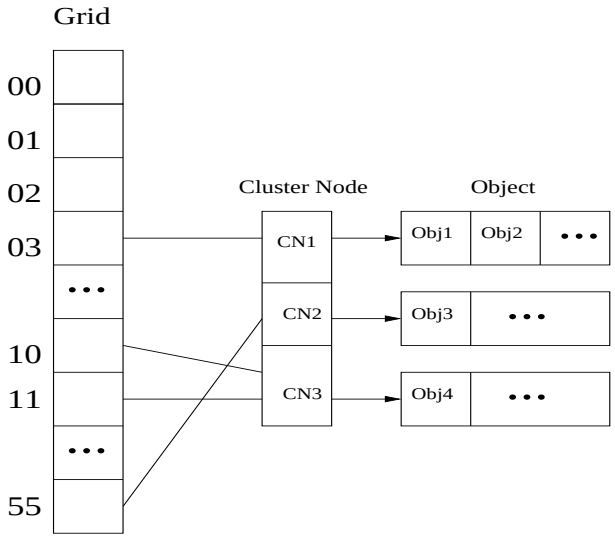

Figure 2. Index Structure

node can split into two. Also, If the number of objects one cluster node contains is getting very small, we can merge it with neighboring cluster nodes. When cluster nodes are split or merged or expanded, the pointers between grids and cluster nodes should be updated so that grids always points to the relevant set of cluster nodes.

Unlike the MBR(Minimal Bounding Rectangle) in Rtree, the covering ranges of cluster nodes are not minimal so the chances that objects move out of the current covering range can be smaller. Therefore, the coordination overhead between cluster nodes and the updating cost will be less.

Algorithm: CN Updates
Procedure Updates(Oid, newPos)
Input: an object's ID oid, the new position newpos
begin

```
1:      E = GetEntry(oid);
2:      if newpos in CN.range then
3:          E.pos = newpos;
4:      else
5:          CNnum = LeastExpand(newpos);
6:          if (CNnum == currentCNnum) then
7:              E.pos = newpos;
8:              ExpandRange(newpos);
9:          else
10:             Delete(oid);
11:             CN[CNnum].insert(oid, newpos);
```

end

Since CNG index is a simple two-level structure, this updating process is much simpler than that of the Rtree and the Quadtree in terms of the amount of computation and operations need to be performed.

If most objects within a cluster node move in the same direction and roughly at the safe speed, we can move the

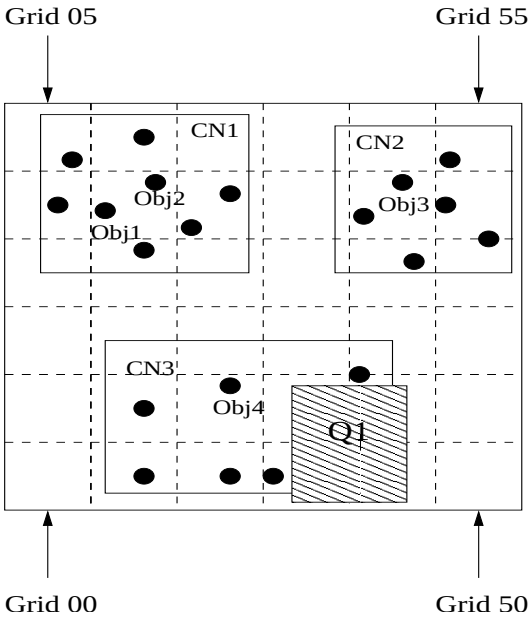

Figure 3. Query Processing

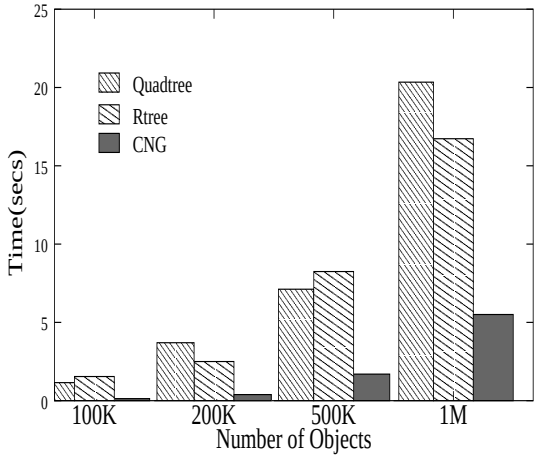

Figure 4. Index Updating time

cluster node correspondingly, which could reduce the updating cost even more. In our current implementation, we have not included this feature yet.

3.4 Query Processing

To process a range query or a point query, the search starts with the grids, find the grids that relevant to the query. For example, as shown in Figure 3, Q1 is a range query represented by the dark area. Q1 intersects with grid30, grid 40, grid31 and grid41. According to the pointers between grids and cluster nodes, we find that cluster nodes CN3 is related. Search the object array of cluster node CN3 and find the objects that fall in the range Q1. This grid index can also be very helpful for nearest-neighbor queries and aggregate queries.

4 Experimental Evaluations

In this section, we present the results of some experiments to analyze the performance of our index structure with respect to updating and searching performance. We compare our index with the Quadtree and Rtree. Both the Quadtree and Rtree use Lazy Update approach [7] for better updating performance. Experiments show that comparing with disk-based Rtree and Quadtree, CNG indexing is always much better in terms of both index updating and query processing. In order to make a fair comparison, we compare the performance of CNG indexing with memory-based Rtree and Quadtree. The results report the actual execution time for the various cases. The experimental settings are described first, followed by the results and discussion.

4.1 Experimental Setup

In all our experiments, we used a 1Ghz Pentium III machine with 2GB of memory. This machine has 32K of level 1 cache, of which 16K is for instructions and 16K for data, and 256K level 2 cache. Due to the unavailability of actual object movement data, we used a synthetic dataset generated by the City Simulator, developed at IBM Almaden [5]. City Simulator is a scalable, three-dimensional model city that enables creation of dynamic spatial data simulating the motion of up to 1 million people. It is designed to generate realistic data for evaluation of database algorithms for indexing and storing dynamic location data.

We generate 4 datasets with 100K, 200K, 500K and 1M objects respectively. Each cycle consists of two steps: updating index and evaluation queries. The performance of each step is measured separately. In each graph, we present the results for 100k, 200k, 500K, and 1M objects. The number of queries is 100K. The x-axis gives the number of cycles for which the tests were run, and the y-axis gives the actual total execution times observed. We test the performance for 20 cycles and compute the average value.

4.2 Comparing Efficiency of Indexes

In the first experiment, we compared the four approaches in terms of the time to process updating operations for objects in each cycle. Figure 4 shows that for index updating, the CNG index performs much better than Quadtree and Rtree. For example, for 100,000 objects, it takes 1.15 seconds for Quadtree and 1.53 seconds for Rtree to perform the updating, while it only cost the CNG indexing 0.13 seconds. For 1 million objects, Quadtree takes around 20.3

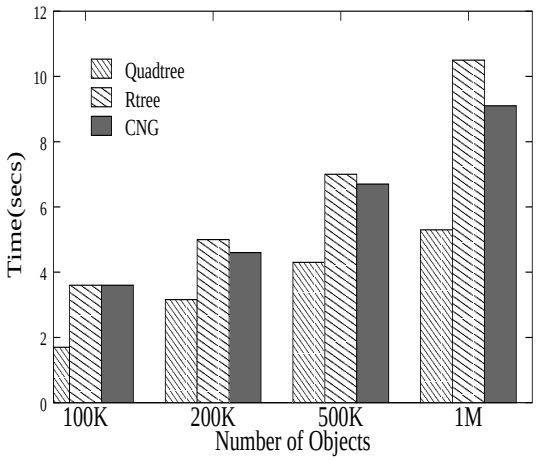

Figure 5. Query Evaluation time

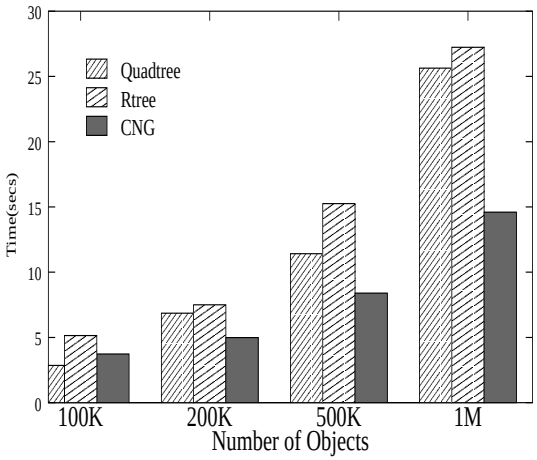

Figure 6. Total time

seconds and Rtree needs 16.7 seconds for updating, while CNG takes only 5.5 seconds. Generally, the updat time for CNG indexing is only 8% to 35% of the Quadtree and Rtree.

In the second experiment, we measured the performance for range queries. The number of queries is fixed at 100,000. Query windows are uniformly distributed in the space. Figure 5 shows the result for this experiment. As in Figure 5, the Quadtree outperforms both Rtree and CNG in searching performance, while CNG performs a little bit better than Rtree. The query evaluation time for Quadtree is almost always around half of that of the Rtree. For example, for 1 million objects, it takes quadtree 5.3 seconds to evaluation all queries, while Rtree takes about 10.5 seconds. CNG takes 9.1 seconds.

Figure 6 shows the total cost of the four schemes in each cycle by adding up their updating cost and the searching cost. For 100K objects, the performance of the Quadtree is the best. For the 200K, 500K, 1M cases, CNG outperforms both the Quadtree and the Rtree. As the number of objects grow higher, the difference between the Quadtree/Rtree and the CNG index becomes more significant. For example, for 200K objects, CNG is 27% better than the Quadtree and 33% better than the Rtree, while for 1M objects, CNG's performance is 44% and 46% better than the Quadtree and the Rtree respectively. This shows that CNG is an scalable and efficient index structure.

5 Conclusions

Traditional spatial index structures do not work well in moving object environments, which are characterized by large numbers of continuously moving objects and concurrent active queries over these objects. The need for frequent index updating results in poor performance. Some early techniques try to reduce the number of updates by approximating the movement of moving objects as a linear function, but the movement of real objects are too complicated to be described as a linear function.

We present a novel CNG indexing technique for scalable execution. CNG index is a two-level index structure. The self-adaptive dynamic cluster node of the lower level handle the movements of the objects and significantly reduce the index updating cost. The grid indexing of the upper level is a simple but very efficient structure for speeding up the query. Our experiments demonstrate that CNG Indexing achieves significant improvement over the traditional approaches in terms of index updating and has a better overall performance in most cases.

References

[1] P. K. Agarwal, L. Arge, and J. Erickson. Indexing moving points. In *Symposium on Principles of Database Systems*, pages 175–186, 2000.

[2] V. Gaede and O. Günther. Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–231, 1998.

[3] D. V. Kalashnikov, S. Prabhakar, S. Hambrusch, and W. Aref. Efficient evaluation of continuous range queries on moving objects. In *DEXA 2002, Proc. of the 13th International Conference and Workshop on Database and Expert Systems Applications*, Aix en Provence, France, September 2–6 2002.

[4] K. V. R. Kanth, S. Ravada, and D. Abugov. Quadtree and R-tree indexes in oracle spatial: a comparison using GIS data. *Proceedings of ACM SIGMOD Conference*, 2002.

[5] J. Kaufman, J. Myllymaki, and J. Jackson. City simulator developed by IBM Almaden. http://www.alphaworks.ibm.com/tech/citysimulator.

[6] G. Kollios, D. Gunopulos, and V. J. Tsotras. On indexing mobile objects. In *Proceedings of the Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 31 - June 2, 1999, Philadelphia, Pennsylvania*, pages 261–272. ACM Press, 1999.

[7] D. Kwon, S. J. Lee, and S. Lee. Indexing the current positions of moving objects using the lazy update r-tree. *3rd International Conference on Mobile Data Management*, Jan 2002.

[8] S. Prabhakar, Y. Xia, D. Kalashnikov, W. Aref, and S. Hambrusch. Query indexing and velocity constrained indexing: Scalable techniques for continuous queries on moving objects. *IEEE TC 2002, IEEE Transactions on Computers, Special section on data management and mobile computing*, 51(10):1124–1140, Oct. 2002.

[9] S. Saltenis, C. Jensen, S. Leutenegger, and M. Lopez. Indexing the position of continuously moving objects. *Proceedings of ACM SIGMOD Conference*, pages 261–272, 2000.

[10] J. Tayeb, O. Ulusoy, and O. Wolfson. A quadtree-based dynamic ttribute indexing method. *The Computer Journal*, pages 185–200, 1998.