Towards Increasingly Update Efficient Moving-Object Indexing

Christian S. Jensen and Simonas Šaltenis
Department of Computer Science
Aalborg University
{csj,simas}@cs.auc.edu *

Abstract

Current moving-object indexing concentrates on point-objects capable of continuous movement in one-, two-, and three-dimensional Euclidean spaces, and most approaches are based on well-known, conventional spatial indices. Approaches that aim at indexing the current and anticipated future positions of moving objects generally must contend with very large update loads because of the agility of the objects indexed. At the same time, conventional spatial indices were often originally proposed in settings characterized by few updates and focus on query performance. In this paper, we characterize the challenge of moving-object indexing and discuss a range of techniques, the use of which may lead to better update performance.

1 Introduction

Several trends in hardware technologies combine to provide the enabling foundation for a class of mobile e-services where the locations of the moving objects play a central role. These trends encompass continued advances in the *miniaturization* of electronics, in *display devices*, and in *wireless communications*. Other trends include the improved *performance* of general computing technologies and the general improvement in the *performance/price ratio* of electronics. Perhaps most importantly, *positioning technologies* such as GPS (global positioning system) are becoming increasingly accurate and usable.

The coming years are expected to witness very large quantities of wirelessly on-line, i.e., Internet-worked, objects that are location-enabled and capable of movement to varying degrees. Example objects include consumers using WAP-enabled mobile-phone terminals and personal digital assistants and vehicles with computing and navigation equipment. Some predict that each of us will soon have approximately 100 on-line objects, most of which will be special-purpose, embedded computers.

These developments pave the way to a range of qualitatively new types of Internet-based services, which either make little sense or are of limited interest in the traditional context of fixed-location, PC- or workstation-based computing. Such services encompass traffic coordination, management, and way-finding; location-aware

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*The authors' research is supported in part by the Danish National Centre for IT Research, the Nordic Academy of Advanced Study, and the Nykredit Corporation.

advertising; integrated information services, e.g., tourist services; safety-related services; and location-based games that merge virtual and physical spaces.

In location-enabled m-services, moving objects disclose their positional data (position, speed, velocity, etc.) to the services, which in turn use this and other information to provide specific functionality. Our focus is on location-enabled services that rely on access to the up-to-date locations of large volumes of moving objects. Due to the volumes of data, the data must be assumed to be disk resident; and to obtain adequate query performance, some form of indexing must be employed. The aim of indexing is to make it possible for multiple users to concurrently and efficiently retrieve desired data from very large databases. Indexing techniques are becoming increasingly important because the improvement in the rate of transfer of data between disk and main memory cannot keep pace with the growth in storage capacities and processor speeds. Disk I/O is becoming an increasingly pronounced bottleneck.

The techniques that have been proposed for indexing the current and near-future positions of moving objects are based on spatial indexing techniques, most prominently the R-tree, that were conceived in settings where updates were relatively few and where focus was on queries. Therefore, efficient update processing represents a substantial, and as yet unmet, challenge. Specifically, without more efficient update processing the applicability of moving-object indexing techniques will remain restricted to scenarios with relatively few objects or scenarios where few updates are needed per object per time unit.

This paper briefly describes a range of techniques that seem to offer additional opportunities for further improving the update processing capabilities of moving-object indices. We initially describe a setting for moving-object indexing. Then follows a section that considers six classes of techniques that may be applied to make a variety of moving-object indices more update efficient. A brief summary ends the paper.

2 Problem Setting

The application setting for the problem of moving-object indexing descibed here aims to concisely capture the complexities of the problem. We have at times opted for a concrete setting as opposed to a very general and abstract setting.

At the core of the problem setting is a set of so-called *moving objects*, which are capable of continuous movement. We assume that the movements of these objects are embedded in two-dimensional space, meaning that we ignore altitude. As examples, the moving objects can be pedestrians or people traveling in cars.

We also assume that one or more *services*, each with a database, are available to the moving objects. The moving objects are capable of communicating wirelessly with the services. Further, the moving objects are capable of reporting their movement information, including and most prominently their current position, to the services. This capability is achieved by means of one of a range of geo-location technologies. Indeed, we are particularly interested in the class of services where a moving object reports its movement information to the service. Such a location-enabled service records the movement of each object. It may record the past, current, and projected, future movement.

A service's record of an object's movement is inherently imprecise, for several reasons. First, the movement of an object is captured via some kind of sampling, so that movement information pertaining only to discrete instances of time is obtained. Movement information pertaining to all other times must be derived via some kind of interpolation or extrapolation. Second, the movement information in each sample is imprecise [7].

Different geo-location technologies yield different precisions, and the precisions obtained when using a single technology also varies, depending on the circumstances under which the technology is used. For example, the cellular infrastructure itself, the positioning technologies offered by companies such as SnapTrack and Cambridge Positioning Technologies, and GPS and server-assisted GPS offer quite different precisions. And, for example, GPS technology is dependent on lines of sight to several satellites, which affects the robustness of the technology. In other words, the accuracy of the positioning is highly dependent on a number of aspects, including the user's location.

Different services require movement information with different minimum precisions for them to work. For example, a weather information service requires user positions with very low precision, while an advanced location-based game, where the participants interact with geo-located, virtual objects, requires high precision. Stated in general terms, the highest precision that may be obtained is that offered by the geo-location technology. One may get close to the highest precision by sampling very frequently. Services that require higher precision cannot be accommodated. We will assume that a required precision is given by the service under consideration and that this precision can indeed be achieved. Further, each object is assumed to be aware of the movement information kept for it by the service. The object is then able to issue an update to the service when its actual movement information deviates by more than the required precision from the service's record [11].

A set of geo-referenced objects not capable of continuous movement is also part of the problem setting. Examples include schools, recreational facilities, and gas stations. These objects are part of the "content" queried by the services.

It should also be noted that the objects are not simply moving in a perfect, two-dimensional Euclidean space. Rather, objects are subjected to movement constraints, one category of which consists of objects that block the movements of objects. For example, a private property and a lake block the movement of a hiker. Another category consists of infrastructure that intuitively takes the movement of an object to a lower-dimensional space. For example, cars may be confined to a road network.

The workload experienced by the database at a service then consists of a sequence of updates intermixed with queries. The amount of updates is dependent on factors such as the number of objects, the required precision, the agility of the objects, and the service's representation of the objects' movements. The queries may be range queries, ranked or unranked k-nearest neighbor queries, and reverse nearest neighbor queries, to name but a few. In addition, these queries may be attached to moving objects, making them moving queries; and they may be active, meaning that they are evaluated continuously for a time period. It is assumed that the data at a service is stored on disk and that some kind of indexing is necessary to support the queries issued.

3 Update Techniques

We proceed to explore a range of techniques, the application of which may reduce the processing needed to accommodate the index updates implied by a workload. These techniques aim to process the individual updates more efficiently or to reduce the number of updates.

Representing Positions as Functions One approach to reduce the number of updates is to model the position of an object as a function of time. This reduces the need for updates because a function better estimates the real locations of an object for a longer period than does a constant position. An update is needed when the difference between the real location of the object and the position believed by the database (the "database position") exceeds the threshold dictated by the services being supported. This is illustrated in Figure 1, where the movement of a one-dimensional moving point is shown together with linear functions representing the object's movement between the updates. When linear functions are used, for a time point t, the database position of an object $\bar{x}(t) = \bar{x}(t_{upd}) + \bar{v}(t_{upd})(t - t_{upd})$ is described by two parameters—the position of the object, $\bar{x}(t_{upd})$, as recorded at the time of the last update t_{upd} ($t_{upd} \leq t$), and the velocity vector, $\bar{v}(t_{upd})$, as recorded at t_{upd}.

The velocity vector is the first derivative of the position function. Similarly, the acceleration vector is the second derivative of the position. In general, one could employ $n + 1$ parameters and n-th degree polynomials to approximate an object's movement, using the Taylor series:

$$\bar{x}(t) = \bar{x}(t_{upd}) + \bar{x}'(t_{upd})(t - t_{upd}) + \bar{x}''(t_{upd})\frac{(t - t_{upd})^2}{2!} + \ldots + \bar{x}^{(n)}(t_{upd})\frac{(t - t_{upd})^n}{n!}$$

Above, it is assumed that at any point in time, there is one function that models the position of a moving object in the index. Another direction involves the use of several functions at a time for capturing an object's position. Specifically, the time interval during which an object's position is modeled may be partitioned into sub-intervals, and one function may be given for each sub-interval. It is natural to require that the position given by a function at the end of its interval is equal to the position obtained from the function assigned to the next interval when applied to the start of this interval. If all functions are linear, a polyline in $n+1$-dimensional space results for an object moving in n-dimensional space. This direction may be useful when the path of an object in a transportation network is known.

Several researchers have explored the use of linear functions for representing and indexing the current and future positions of moving objects in one-dimensional to three-dimensional spaces [4, 8, 9, 10]. The more general approaches outlined here have yet to be explored, as does the combined indexing of the past, current, and future positions of moving objects.

Capturing and Using Expiration Times Independently of how an object's position is represented in an index, the accuracy of the database position and, thus, its utility for any application will tend to decrease as time passes. When an object has not reported its position to the database for a certain period of time, the database position is likely to be of little use to the services being supported, and the object is also unlikely to be interested in the services. The object's position should then simply be discarded from the database and the index.

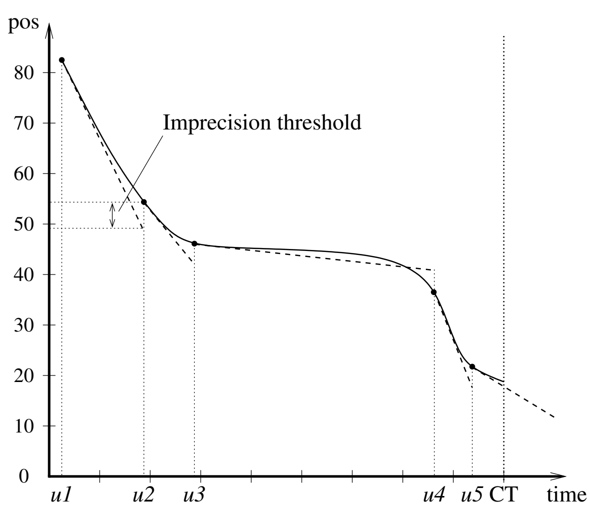

Figure 1: Approximating a One-Dimensional Moving Object by Linear Functions

In this scenario, it is natural to associate an expiration time with each position, thus enabling automatical removal of "expired" positions as well as filtering of yet-to-be-removed, expired entries in queries. As a result, the scheduling of explicit deletion operations and the reporting of expired objects in query results are avoided.

When expiration times are associated with object positions at the times they are inserted or updated, an opportunity exists for the index used to exploit the expiration times to obtain more efficient update processing. Indeed, experiments with the R$^{\mathrm{EXP}}$-tree [10] demonstrate that in order to avoid scheduling of explicit deletion operations to remove expired objects, expiration times can be introduced in the index structure, upon which the expired index entries can be removed in an automatic and lazy fashion. This reduces the average update costs. While the R$^{\mathrm{EXP}}$-tree introduces expiration times in the TPR-tree, the proposed technique is more general. It can be used with any other index structure.

Utilization of Buffering A promising technique for increasing the efficiency of index update operations is to use the ideas underlying Buffer trees [2], the aim being to remedy the inefficiencies of transferring blocks with little useful data between main memory and disk. In a Buffer tree, each node is associated with a main-memory-sized buffer that buffers the update operations concerning the subtree rooted at the node. The operations buffered at some tree node are performed in bulk whenever the node's buffer becomes overfull. With B being the number of (data value, pointer)-pairs that fit in a disk block, such bulk operations are almost B times faster, in amortized cost, than when performing the operations in the usual fashion. With B usually being on the order of hundreds, the performance of update operations may be boosted dramatically.

While some work has been done on buffering in R-trees in the context of workloads that intermix insertions and deletions [3], the existing Buffer-tree approaches do not allow queries to be performed if there are nonempty buffers. however, workloads that intermix queries and updates is a key characteristic of our application scenario, where queries should also be answered in a timely, on-line fashion, i.e., it is not acceptable to buffer queries the same way insertions and deletions are buffered.

It should be possible to extend the Buffer-tree techniques to support queries that are performed simultaneously with insertions and deletions. One approach to achieving this is to organize the buffers as index-like structures, so that queries can be efficiently performed on the buffers. This may reduce the performance of the queries, as they will have to perform additional I/O operations to search the relevant buffers. The goal is to provide a flexible mechanism, whereby one can tune how much the query performance should be sacrificed in order to gain the necessary update performance.

Exploitation of All Available Main Memory Main memory storage is much faster than disk storage and becomes increasingly voluminous and inexpensive. More efficient processing of updates against moving-object indices may be obtained through aggressive use of all available main memory. While buffering tends to result in the use of more main memory, simply using buffering does not imply that all available main memory is being utilized in any optimal fashion.

When aggressively using main memory, one may expect much of an inherently disk-based index to reside in main memory, in a form that is optimized for the particular main memory and processor environment. The challenge then becomes one of maintaining some parts of an index in its disk-based format and some parts in its main-memory format. Such main-memory variants of disk-based moving-object indices deserve study.

The areas of main-memory and real-time database management aggressively exploit main memory and may have ideas to offer. For example, while main-memory page buffers are traditionally organized simply as collections of pages, main-memory databases employ more elaborate index structures to optimize CPU performance. Aggressive use of main memory is also seen in an application area such as telecommunications, where there is a need to record large amounts of real-time discrete events.

Taking Movement Constraints Into Account Existing work on moving-object indexing typically assumes that the underlying objects live in 1-, 2-, or 3-dimensional Euclidean spaces (e.g., [1, 5, 8, 9]).

However, as pointed out earlier, in many situations, some objects constrain the locations of other objects. For example, the movements of hikers in a forest are constrained by fenced, private properties. As another example, the movement of a ship is constrained by shallow water and land. These blocking objects do not reduce the dimensionality of the space in which the objects are embedded. In other application contexts, the locations of objects are constrained to a transportation network embedded in, typically, 2-dimensional Euclidean space [4]. This in some sense reduces the dimensionality of the space available for movement—the term 1.5-dimensional space has been used. Examples abound. Cars typically move in transportation networks, and the destinations such as private residences, hotels, and shops may be given locations in transportation networks. Next, folklore has it that 80-90% of all automobile drivers move towards a destination. This suggests that drivers typically follow network paths that are known ahead of time.

Moving-object indices should be able to exploit these constraints to obtain better update performance. This may be achieved by exploiting the constraints to better represent the moving objects' locations in the indices and to better estimate the positions of the objects, both of which then lead to less frequent updates.

It should be noted that Euclidean distances are either not the only interesting notions of distance or are not of interest at all in these settings. Rather, indexing techniques that apply to settings with obstacles or network-constrained objects must contend with other notions of distance. For some such notions, the distance between two stationary objects varies over time.

Using Application Semantics It may be observed that modifications on moving-object indices have special properties that may be exploited: most modifications occur in the form of updates that combine deletions with insertions, and the newly reported, inserted location for an object is relatively close to its previously reported, deleted location. In the indexing literature, the insertion and deletion operations are usually considered as separate operations. However, in our application scenario, most update operations are deletion-insertion pairs. Processing these pairs as two separate operations results in two descents and two (partial) ascents of the index tree. For a portion of updates, especially those that change the updated data only slightly, the combined deletion-insertion operation can be performed in a single descent and a (partial) ascent of the tree, which may save I/O operations [6].

4 Summary

With the continued proliferation of wireless networks, visionaries predict that the Internet will soon extend to many billions of devices, or objects. A substantial fraction of these will offer their changing positions to the location-enabled services, they either use or support. As a result, software technologies that enable the management of the positions of objects capable of continuous movement are in increasingly high demand.

This paper argues that although indexing of the current and anticipated, future locations of moving objects is needed, there exists an as of yet unmet need for more efficient update processing in moving-object indexing. The paper then proceeds to describe a number of possible techniques, the use of which may render moving-object indices increasingly update efficient.

Many opportunities exist for testing out more specific incarnations of the described techniques in the contexts of concrete indexes. In addition, techniques not described in this paper exist that may also improve update efficiency. Distributed update processing is one such type of technique. Another promising direction is to exploit approximation techniques to offer monotonically improving, as-good-as-possible answers to queries within specified soft or hard deadlines. This may enable the querying of almost up-to-date data, which in turn may reduce the need for prompt updates. Yet another direction involves the use of relaxed index properties.

References

[1] P. K. Agarwal, L. Arge, and J. Erickson. Indexing Moving Points. *Proc. of the PODS Conf.*, pp. 175–186 (2000).

[2] L. Arge. External-Memory Algorithms with Applications in GIS. In *Algorithmic Foundations of Geographic Information Systems*, LNCS 1340, pp. 213–254 (1997).

[3] L. Arge, K. Hinrichs, J. Vahrenhold, J. S. Vitter. Efficient Bulk Operations on Dynamic R-trees. In *Proc. of Algorithm Engineering and Experimentation*, pp. 328–348 (1999).

[4] G. Kollios, D. Gunopulos, and V. J. Tsotras. On Indexing Mobile Objects. *Proc. of the PODS Conf.*, pp. 261–272 (1999).

[5] G. Kollios et al. Indexing Animated Objects Using Spatiotemporal Access Methods. TimeCenter TR-54 (2001).

[6] D. Kwon, S. Lee, and S. Lee. Indexing the Current Positions of Moving Objects Using the Lazy Update R-tree. *Proc. of the 3rd Intl. Conf. on Mobile Data Management*, pp. 113–120 (2002).

[7] D. Pfoser and C. S. Jensen. Capturing the Uncertainty of Moving-Object Representations. In *the Proc. of the SSDBM Conf.*, pp. 111–132 (1999).

[8] C. M. Procopiuc, P. K. Agarwal, and S. Har-Peled. STAR-Tree: An Efficient Self-Adjusting Index for Moving Objects. Manuscript (2001).

[9] S. Šaltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez. Indexing the Positions of Continuously Moving Objects. *Proc. of the ACM SIGMOD Conf.*, pp. 331–342 (2000).

[10] S. Šaltenis and C. S. Jensen. Indexing of Moving Objects for Location-Based Services. In *Proc. of the IEEE International Conf. on Data Engineering*, pp. 463–472 (2002).

[11] O. Wolfson, A. P. Sistla, S. Chamberlain, and Y. Yesha. Updating and Querying Databases that Track Mobile Units. *Distributed and Parallel Databases* 7(3): 257–387 (1999).