Continuous Query Processing in Spatio-Temporal Databases

Mohamed F. Mokbel

Dept. of Computer Sciences, Purdue University, West Lafayette IN, USA
mokbel@cs.purdue.edu

Abstract. In this paper, we aim to develop a framework for continuous query processing in spatio-temporal databases. The proposed framework distinguishes itself from other query processors by employing two main paradigms: (1) Scalability in terms of the number of concurrent continuous spatio-temporal queries. (2) Incremental evaluation of continuous spatio-temporal queries. Scalability is achieved thorough employing a *shared execution* paradigm. Incremental evaluation is achieved through computing only the updates to the previously reported answer. We distinguish between two types of updates; *positive* updates and *negative* updates. Positive or negative updates indicate that a certain object should be added to or removed from the previously reported answer, respectively. The proposed framework is applicable to a wide variety of continuous spatio-temporal queries where we do not have any constraints about the mutability of objects and queries (i.e., both objects and queries can be either stationary or moving) or the movement representation (i.e., movement can be represented either by sampling or trajectory).

1 Introduction

The rapid increase of spatio-temporal applications calls for new query processing techniques to deal with both the spatial and temporal domains. Examples of spatio-temporal applications include location-aware services, traffic monitoring, enhanced 911 service, and multimedia databases. Unlike traditional databases, spatio-temporal databases have the following distinguishing characteristics: (1) Most of spatio-temporal queries are continuous in nature. Unlike snapshot queries that are evaluated only once, continuous queries require continuous evaluation as the query result becomes invalid with the change of information [33]. (2) A large number of mobile and stationary objects, and consequently a large number of mobile and stationary concurrent continuous queries. (3) Any delay of the query response results in an obsolete answer. For example, consider a query that asks about the moving objects that lie in a certain region. If the query answer is delayed, the answer may be outdated where objects are continuously changing their locations.

Spatio-temporal databases need to support a wide variety of continuous spatio-temporal queries. For example, a continuous spatio-temporal range query may have various forms depending on the mutability of objects and queries. In addition, a range query may ask about the past, present, or the future. A naive way to process continuous spatio-temporal queries is to abstract the continuous queries into a series of snapshot spatio-temporal queries. Snapshot queries are issued to the server (e.g., a location-aware server) every T seconds. The naive approach incurs redundant processing where there may be only a slight change in the query answer between any two consecutive evaluations.

1.1 Motivation

The main objective in my PhD is to build a location-aware server [1, 18, 21] that has the ability to efficiently process a large number of stationary and moving objects and queries. In our attempt to build the location-aware server [1], we face the following challenges:

- Most of the existing query processing techniques focus on solving special cases of continuous spatio-temporal queries, e.g., [28, 30, 35, 36] are valid only for moving queries on stationary objects, [4, 8, 10, 23] are valid only for stationary range queries. Also, [15, 17, 23, 28] are valid only for sampling while [2, 25, 30, 31] require a trajectory representation. Trying to support a wide variety of spatio-temporal queries in a location-aware server results in implementing a variety of specific algorithms with different access structures. Maintaining different access structures and algorithms degrades the performance of the location-aware server.
- Most of the existing spatio-temporal algorithms focus on evaluating only one spatio-temporal query (e.g., [2, 14, 16, 28, 30, 32, 35, 36]). In a typical location-aware server [1, 18], there is a huge number of concurrently outstanding continuous spatio-temporal queries. Handling each query as an individual entity dramatically degrades the performance of the location-aware server.
- Most of the existing algorithms for continuous spatio-temporal queries model the continuous queries as a series of snapshot queries. Different approaches (e.g., *valid time* [36], *valid region* [35], *safe region* [23], *safe period* [8], *No-Action* region [34], and *trajectory model* [30]) are employed to allow for longer time intervals T between any two consecutive evaluations of spatio-temporal queries. Reissuing the spatio-temporal queries, even with longer time intervals, incurs redundant processing between each two consecutive executions, hence degrading the performance of a location-aware server.
- Most of the existing algorithms for continuous spatio-temporal queries require that the location-aware server sends a complete answer to the client with each reevaluation. In a typical location-aware server, query results are sent to clients via satellite servers [11]. Sending the whole answer each time consumes the network bandwidth and results in network congestion at the server side, thus degrading the ability of the server to process more queries.

Based on these challenges, we specify our goal as not to propose another spatio-temporal algorithm for very specific spatio-temporal queries. Instead, we aim to develop a general framework for spatio-temporal query processing that is scalable, incremental, and applicable to a wide variety of spatio-temporal queries.

1.2 The PhD Contribution

Although my PhD contributions are geared towards building scalable location-aware servers [1, 18], the main ideas and concepts can be utilized individually or together for any other spatio-temporal application. In general, the PhD contributions can be summarized as follows:

1. We go beyond the idea of reevaluating continuous spatio-temporal queries for every change of information. Instead, we employ an incremental evaluation paradigm that updates the query answer rather than evaluating it. By employing the incremental evaluation paradigm, we achieve two goals: (a) Reducing the required computations to evaluate continuous spatio-temporal queries. (b) Better utilization of the network bandwidth where we limit the data sent to queries to only the updates rather than the whole query answer.
2. We distinguish between two types of updates; *positive* updates and *negative* updates. Positive or negative updates indicate that a certain object should be added to or removed from the previously reported answer, respectively.
3. We support a wide variety of continuous spatio-temporal queries through a general framework. The proposed framework does not make any assumptions about the mutability of objects and queries or the movement representation.
4. We employ the *shared execution* paradigm as a means of achieving scalability in terms of the number of concurrently executed continuous queries.
5. We employ a recovery algorithm for *out-of-sync* clients; clients that are disconnected from the server for a short period of time. The recovery algorithm aims to keep out-of-sync clients updated with their results whenever they reconnect to the system.
6. We aim to realize our spatio-temporal query processor inside the Predator [27] database management system. In addition, we use a storage manager that is based on Shore [5] to store information and access structures for moving objects and moving queries.

1.3 Environment

This PhD work is part of the *Pervasive Location-Aware Computing Environments* project (PLACE, for short) [1], developed at Purdue University. The PLACE server receives information from moving objects and moving queries through GPS-like devices. In addition, the PLACE server keeps track of the locations of stationary objects (e.g., gas stations, hospitals, etc.). Once a moving object or query sends new information, the old information becomes persistent and is stored in a repository server.

The PLACE server [21] is implemented on top of the NILE query processor [13]; an extended version of the PREDATOR database management system [27] to handle continuously incoming data. For more details about the architecture and the query processor of the PLACE server, the reader is referred to [18, 20, 21].

2 Related Work

Most of the recent research in spatio-temporal query processing (e.g., [2, 16, 28, 30, 35, 36]) focus on continuously evaluating one spatio-temporal query at a time. Issues of scalability, incremental evaluation, mutability of both objects and queries, and client overhead are examples of challenges that either are overlooked wholly or partially by these approaches. Mainly, three different approaches are investigated: (1) The validity of the results. With each query answer, the server returns a valid time [36] or a valid region [35] of the answer. Once the valid time is expired or the client goes out of the valid region, the client resubmits the continuous query for reevaluation. The time T between each two consecutive evaluations relies on the accuracy of the valid time and the valid region. (2) Caching the results. The main idea is to cache the previous result either in the client side [28] or in the server side [16]. Previously cached results are used to prune the search for the new results of k-nearest-neighbor queries [28] and range queries [16]. (3) Precomputing the result. If the trajectory of the query movement is known apriori, then by using computational geometry for stationary objects [30] or velocity information for moving objects [16], we can identify which objects will be nearest-neighbors [2, 30] to or within a range [16, 25] from the query trajectory. If the trajectory information is changed, then the query needs to be reevaluated. The time T between each two consecutive evaluations relies on the accuracy of determining the future trajectory.

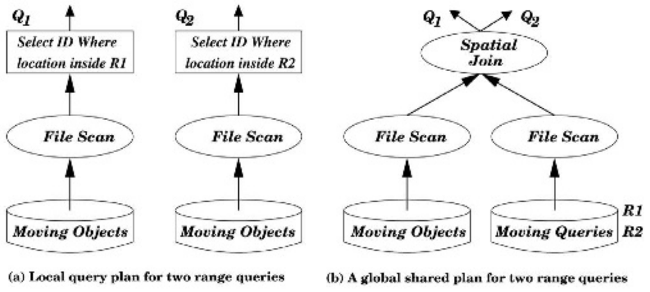
There is lot of research in optimizing the execution of multiple queries in traditional databases [26], continuous web queries [7], and continuous streaming queries [6, 12]. Optimization techniques for evaluating a set of continuous spatio-temporal queries are recently addressed for centralized [23] and distributed environments [4, 8]. Distributed environments assume that clients have computational and storage capabilities to share query processing with the server. The main idea of [4, 8] is to ship some part of the query processing down to the moving objects, while the server mainly acts as a mediator among moving objects. This assumption is not always realistic. In many cases, clients use cheap, low battery, and passive devices that do not have any computational or storage capabilities. For centralized environments, the Q-Index [23] does not assume any client overhead. The main idea of the Q-index is to build an R-tree-like [9] index structure on the queries instead of the objects. Then, at each time interval T, moving objects probe the Q-index to find the queries they belong to. The Q-index is limited in two aspects: (1) It performs reevaluation of all the queries every T time units. (2) It is applicable only for stationary queries.

In general, spatio-temporal queries can be evaluated using a spatio-temporal access method [19]. The TPR-tree [25] and its variants (e.g., the R^{EXP}-tree [24] and the TPR*-tree [31]) are used to index objects with future trajectories. However, there are no special mechanisms to support the continuous spatio-temporal queries in any of these access methods.

Our proposed framework distinguishes itself from other approaches, where we go beyond the idea of reevaluating continuous queries. Instead, we use incremental evaluation to compute only the updates of the previously reported result. In addition, unlike [4, 8], we do not assume any computational capabilities on the client side. Moreover, our framework is scalable to support a large number of concurrently outstanding continuous queries and can deal with many variations of continuous spatio-temporal queries.

3 Scalable Incremental Processing of Continuous Spatio-Temporal Queries

In this section, we present a scalable and incremental framework for continuously evaluating continuous spatio-temporal queries. The scalability is achieved by employing a *shared execution* paradigm for continuous spatio-temporal queries [18, 20]. With the shared execution, queries are indexed in the same way as data. Thus, evaluating a set of concurrent continuous spatio-temporal queries is reduced to a join between a set of moving objects and a set of moving queries [34]. Figure 1 gives an example of having a



Q1
Select ID Where location inside R1
File Scan
Moving Objects
Q2
Select ID Where location inside R2
File Scan
Moving Objects
(a) Local query plan for two range queries
Q1
Q2
Spatial Join
File Scan
File Scan
Moving Objects
Moving Queries
R1
R2
(b) A global shared plan for two range queries

Fig. 1. Shared execution of continuous queries

shared plan (Figure 1b) for two concurrent continuous queries (Figure 1a). Incremental evaluation is achieved through computing only the updates to the previously reported answer. We distinguish between two types of updates; *positive* updates and *negative* updates. A positive update of the form $(Q, +A)$ indicates that object A needs to be added to the answer set of query Q. Similarly, a negative update of the form $(Q, -A)$ indicates that object A is no longer part of the answer set of query Q. In general, we distinguish between three types of objects: *Stationary* objects, *moving* objects, and *predictive* objects. Moving objects can send only their current locations, while predictive objects have the ability to report their velocity vector. Thus, their future location can be predicted. Similarly, we have the same classification of queries, i.e., stationary, moving, and predictive queries.

3.1 Algorithms and Data Structures

The main idea of the continuous query processor is to treat both objects and queries similarly. Thus, we store both objects and queries in the same data structure. We use a simple grid structure that divides the space evenly into $N \times N$ equal sized grid cells. We utilize one grid structure that holds both objects and queries. Stationary and moving objects are mapped to specific grid cells using their locations. Predictive objects and all query types are clipped to multiple grid cells that overlap with the movement trajectory or query region, respectively.

An object entry O has the form $(OID, loc, t, QList)$, where OID is the object identifier, loc is the recent location of the object, t is the timestamp of the recently reported location loc, and $QList$ is the list of the queries that O is satisfying. A query Q is clipped to all grid cells that Q overlaps with. For any grid cell C, a query entry has the form $(QID, region, t, OList)$, where QID is the query identifier, $region$ is the recent rectangular region of Q that intersects with C, t is the timestamp of the recently reported region, and $OList$ is the list of objects in C that satisfy $Q.region$. k-nearest-neighbor queries are stored in the grid structure by considering the query region as the smallest circular region that contains the k nearest objects. Simple grid structures are commonly used to support different spatio-temporal queries (e.g., range queries [8], future queries [29], and aggregate queries [10]). In addition to the grid structure, we keep track of two auxiliary data structures; the object index and the query index. The object and query indexes are indexed on the OID and QID, respectively, and are used to provide the ability for searching the old locations of moving objects and queries given their identifiers. Using auxiliary data structures to keep track of the old locations is utilized in the LUR-tree [15] as a linked list and in the frequently updated R-tree [17] as a hash table.

Since a typical location-aware server receives a massive amount of updates from moving objects and queries, it becomes a huge overhead to handle each update individually. Thus, we buffer a set of updates from moving objects and queries for bulk processing. Basically the bulk processing is reduced to a spatial join between a set of objects (either stationary or moving) and a set of queries (either stationary or moving). Since, we are utilizing a grid structure, we use a spatial join algorithm similar to the one proposed in [22]. For each moving query Q, we keep track of the old (A_{old}) and new (A_{new}) query regions. A set of negative updates are produced for all objects that are in $Q.OList$ and lie in the area $A_{old} - A_{new}$. Then, we need only to evaluate the area $A_{new} - A_{old}$ to produce a set of *positive* updates. The area $A_{new} \cap A_{old}$ does not need to be reevaluated where the query result of this area is already reported to Q before. The efficiency of this incremental approach comes from the fact that the area $A_{new} \cap A_{old}$ is much larger than $A_{new} - A_{old}$. For any moving object O, we check the set of candidate queries that can intersect with the new location of O. Candidate queries are the queries that are stored in the same grid cell with O. The queries that are joined with O are compared with $O.QList$ to determine the set of positive and negative updates to be sent to the clients.

3.2 Examples

Example 1 (Spatio-Temporal Range-Queries). Figure 2a gives a snapshot of the database at time T_0 with nine moving objects, p_1 to p_9, and five continuous range queries, Q_1 to Q_5. At time T_1 (Figure 2b), only the objects p_1, p_2, p_3, and p_4 and the queries Q_1, Q_3, and Q_5 change their locations. The old query locations are plotted with dotted borders. Black objects are stationary, while white objects are moving. As a result of the change of database status from T_0 to T_1, the location-aware server reports the following updates: $(Q_1, -p_5)$, $(Q_2, -p_1)$, $(Q_3, +p_2)$, $(Q_3, -p_7)$, $(Q_3, -p_6)$, $(Q_3, +p_8)$, and $(Q_4, -p_4)$.

Example 2 (Spatio-Temporal k-NN Queries). Figure 3a gives an example of two kNN queries where $k = 3$ issued at points Q_1 and Q_2. Assuming that both queries are issued at time T_0, we compute the first-time answer using any of the traditional algorithms of

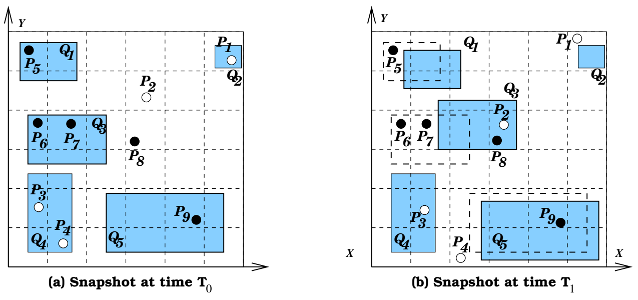


Fig. 2. Range queries

kNN queries. For Q_1, the answer would be $Q_1 = p_1, p_2, p_3$ while for Q_2, the answer would be $Q_2 = p_5, p_6, p_7$. In this case, we present Q_1 and Q_2 as circular range queries with radius equal to the distance of the kth neighbor. Later, at time T_1 (Figure 3b), objects

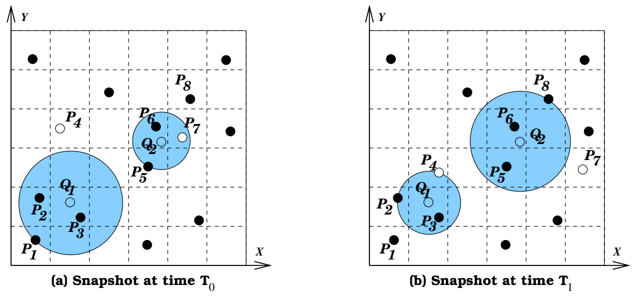


Fig. 3. k-NN queries

p_4 and p_7 are moved. For Q_1, object p_4 intersects with the query region. This results in invalidating the furthest neighbor of Q_1, which is p_1. Thus, two update tuples are reported $(Q_1, -p_1)$ and $(Q_1, +p_4)$. For Q_2, the object p_7 was part of the answer at time T_0. However, after p_7 moves, we find that p_8 becomes more near to Q_2 than p_7. Thus, two updates are reported, $(Q_2, -p_7)$ and $(Q_2, +p_8)$. Notice that unlike range queries, k-NN queries can change their location and size over time.

Example 3 (Predictive Spatio-Temporal Range-Queries). Figure 4a gives an example of querying the future. Five moving objects p_1 to p_5, have the ability to report their current locations at time T_0 and a velocity vector that is used to predict their future locations at times T_1 and T_2. The range query Q is interested in objects that will intersect with its region at time $T_2 > T_0$. At time T_0 the rectangular query region is joined with the lines representation of the moving objects. The returned answer set of Q is (p_1, p_3). At T_1 (Figure 4b), only the objects p_2, p_3, and p_4 change their locations. Based on the new information, we report only the positive update $(Q, +p_2)$ and negative update $(Q, -p_3)$ that indicate that p_2 is considered now as part of the answer set of Q while p_3 is no

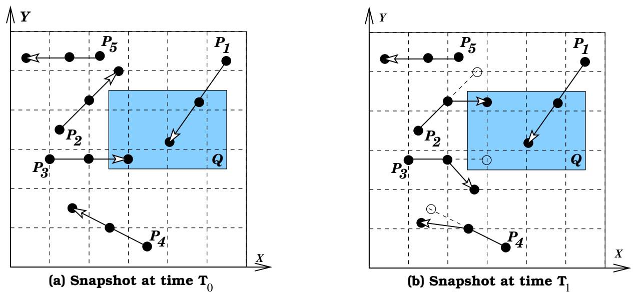


Fig. 4. Predictive queries

longer in the answer set of Q. Notice that no tuples are produced for object p_1 where it does not change its information from the previously reported result at time T_0.

3.3 Out-of-Sync Clients

Mobile objects tend to be disconnected and reconnected several times from the server for some reasons beyond their control, i.e., being out of battery, losing communication signals, being in a congested network, etc. This *out-of-sync* behavior may lead to erroneous query results in any incremental approach. Figure 5 gives an example of erroneous query

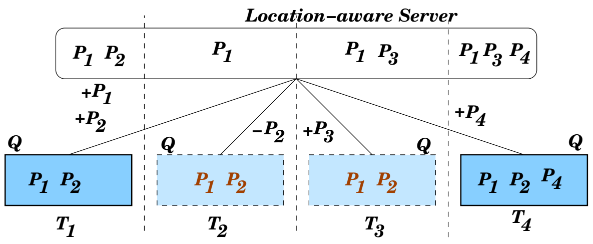


Fig. 5. Example of Out-of-Sync queries

result. The answer of query Q that is stored at both the client and server at time T_1 is (p_1, p_2). At time T_2, the client is disconnected from the server. However, the server keeps computing the answer of Q, and sends the negative update $(Q, -p_2)$. Since the client is disconnected, the client could not receive this negative update. Notice the inconsistency of the stored result at the server side (p_2) and the client side (p_1, p_2). Similarly, at time T_3, the client is still disconnected. The client is connected again at time T_4. The server computes the incremental result from T_3 and sends only the positive update $(Q, +p_4)$. At this time, the client is able to update its result to be (p_1, p_2, p_4). However, this is a wrong answer, where the correct answer is kept at the server (p_1, p_3, p_4).

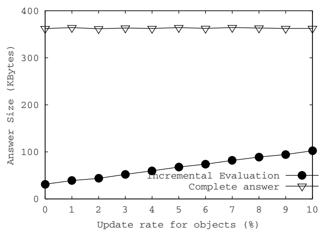
A naive solution for the out-of-sync problem is once the client wakes up, it empties its previous result and sends a *wakeup* message to the server. The server replies by the query answer stored at the server side. For example, in Figure 5, at time T_4, the location-aware server will send the whole answer (p_1, p_3, p_4). This approach is simple to implement and process in the server side. However, it may result in significant delay due to the network cost in sending the whole answer. Consider a moving query with hundreds of objects in its result that gets disconnected for a short period of time. Although, the query has missed a couple of points during its disconnected time, the server would send the complete answer to the query.

To deal with out-of-sync clients, we maintain a repository of committed query answers. An answer is considered committed if it is guaranteed that the client has received it. Once the client wakes up from the disconnected mode, it sends a *wakeup* message to the server. Then, the server compares the latest answer for the query with the committed answer, and sends the difference of the answer in the form of positive and negative updates. For example, in Figure 5, the server stores the committed answer of Q at time T_1 as (p_1, p_2). Then, at time T_4, the server compares the current answer with the committed one, and send the updates $(Q, -p_2, +p_3, +p_4)$. Once the server receives any information from a moving query, it considers its latest answer as a committed one. However, stationary queries are required to send explicit *commit* message to the location-aware server to enable committing the latest result. *Commit* messages can be sent at the convenient times of the clients.

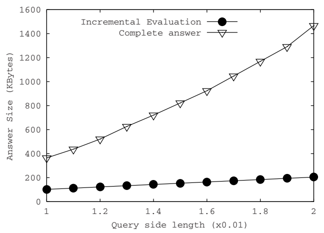
4 Experimental Performance

In this section, we present a preliminary experiment that shows the promising performance of the continuous query processor. Figure 6 compares between the size incremental answer returned by utilizing the incremental approach and the size of the complete answer. We use the *Network-based Generator of Moving Objects* [3] to generate a set of 100K moving objects and 100K moving queries. The output of the generator is a set of moving objects that move on the road network of a given city. We choose some points randomly and consider them as centers of square queries.

The location-aware server buffers the received updates from moving objects and queries and evaluates them every 5 seconds. Figure 6a gives the effect of the number of moving objects that reported a change of location within the last 5 seconds. The size of the complete answer is constant and is orders of magnitude of the size of the worst-case incremental answer. In Figure 6b, the query side length varies from 0.01 to 0.02. The size of the complete answer increases dramatically to up to seven times that of the incremental



(a) Moving objects (%)



(b) Query size

Fig. 6. The answer size

result. The saving in the size of the answer directly affects the communication cost from the server to the clients.

5 Conclusion

In this paper, we presented a scalable and incremental framework for continuous query processing in location-aware servers as an application of spatio-temporal databases. Mainly, we emphasize on the scalability and incremental evaluation of continuous spatio-temporal queries. The applicability of the proposed framework to a wide variety of continuous spatio-temporal queries is discussed. A recovery mechanism for updating the results of clients that are disconnected from the server for a short period of time is presented. Preliminary results show that the size of the result of the incremental approach is around 10% of the size of a complete result. Thus, the proposed scalable and incremental framework has promising savings in both computational processing and network bandwidth.

References

1. Walid G. Aref, Susanne E. Hambrusch, and Sunil Prabhakar. Pervasive Location Aware Computing Environments (PLACE). `http://www.cs.purdue.edu/place/`.
2. Rimantas Benetis, Christian S. Jensen, Gytis Karciauskas, and Simonas Saltenis. Nearest Neighbor and Reverse Nearest Neighbor Queries for Moving Objects. In *IDEAS*, 2002.
3. Thomas Brinkhoff. A Framework for Generating Network-Based Moving Objects. *GeoInformatica*, 6(2), 2002.
4. Ying Cai, Kien A. Hua, and Guohong Cao. Processing Range-Monitoring Queries on Heterogeneous Mobile Objects. In *Mobile Data Management, MDM*, 2004.
5. Michael J. Carey et al. Shoring up persistent applications. In *Proceedings of the ACM International Conference on Management of Data, SIGMOD*, 1994.
6. Sirish Chandrasekaran and Michael J. Franklin. Streaming Queries over Streaming Data. In *Proceedings of the International Conference on Very Large Data Bases, VLDB*, 2002.
7. Jianjun Chen, David J. DeWitt, Feng Tian, and Yuan Wang. NiagaraCQ: A Scalable Continuous Query System for Internet Databases. In *Proceedings of the ACM International Conference on Management of Data, SIGMOD*, 2000.
8. Bugra Gedik and Ling Liu. MobiEyes: Distributed Processing of Continuously Moving Queries on Moving Objects in a Mobile System. In *Proceedings of the International Conference on Extending Database Technology, EDBT*, 2004.
9. Antonin Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In *Proceedings of the ACM International Conference on Management of Data, SIGMOD*, 1984.
10. Marios Hadjieleftheriou, George Kollios, Dimitrios Gunopulos, and Vassilis J. Tsotras. On-Line Discovery of Dense Areas in Spatio-temporal Databases. In *Proceedings of the International Symposium on Advances in Spatial and Temporal Databases, SSTD*, 2003.
11. Susanne E. Hambrusch, Chuan-Ming Liu, Walid G. Aref, and Sunil Prabhakar. Query Processing in Broadcasted Spatial Index Trees. In *Proceedings of the International Symposium on Advances in Spatial and Temporal Databases, SSTD*, 2001.
12. Moustafa A. Hammad, Michael J. Franklin, Walid G. Aref, and Ahmed K. Elmagarmid. Scheduling for shared window joins over data streams. In *Proceedings of the International Conference on Very Large Data Bases, VLDB*, 2003.

13. Moustafa A. Hammad, Mohamed F. Mokbel, Mohamed H. Ali, Walid G. Aref, Ann C. Catlin, Ahmed K. Elmagarmid, Mohamed Eltabakh, Mohamed G. Elfeky, Thanaa M. Ghanem, Robert Gwadera, Ihab F. Ilyas, Mirette Marzouk, and Xiaopeng Xiong. Nile: A Query Processing Engine for Data Streams. In *Proceedings of the International Conference on Data Engineering, ICDE*, 2004.
14. Glenn S. Iwerks, Hanan Samet, and Ken Smith. Continuous K-Nearest Neighbor Queries for Continuously Moving Points with Updates. In *Proceedings of the International Conference on Very Large Data Bases, VLDB*, 2003.
15. Dongseop Kwon, Sangjun Lee, and Sukho Lee. Indexing the Current Positions of Moving Objects Using the Lazy Update R-tree. In *Mobile Data Management, MDM*, 2002.
16. Iosif Lazaridis, Kriengkrai Porkaew, and Sharad Mehrotra. Dynamic Queries over Mobile Objects. In *Proceedings of the International Conference on Extending Database Technology, EDBT*, 2002.
17. Mong-Li Lee, Wynne Hsu, Christian S. Jensen, and Keng Lik Teo. Supporting Frequent Updates in R-Trees: A Bottom-Up Approach. In *Proceedings of the International Conference on Very Large Data Bases, VLDB*, 2003.
18. Mohamed F. Mokbel, Walid G. Aref, Susanne E. Hambrusch, and Sunil Prabhakar. Towards Scalable Location-aware Services: Requirements and Research Issues. In *Proceedings of the ACM workshop on Advances in Geographic Information Systems, ACM GIS*, 2003.
19. Mohamed F. Mokbel, Thanaa M. Ghanem, and Walid G. Aref. Spatio-temporal Access Methods. *IEEE Data Engineering Bulletin*, 26(2), 2003.
20. Mohamed F. Mokbel, Xiaopeng Xiong, and Walid G. Aref. SINA: Scalable Incremental Processing of Continuous Queries in Spatio-temporal Databases. In *Proceedings of the ACM International Conference on Management of Data, SIGMOD*, 2004.
21. Mohamed F. Mokbel, Xiaopeng Xiong, Walid G. Aref, Susanne Hambrusch, Sunil Prabhakar, and Moustafa Hammad. PLACE: A Query Processor for Handling Real-time Spatio-temporal Data Streams (Demo). In *Proceedings of the International Conference on Very Large Data Bases, VLDB*, 2004.
22. Jignesh M. Patel and David J. DeWitt. Partition Based Spatial-Merge Join. In *Proceedings of the ACM International Conference on Management of Data, SIGMOD*, 1996.
23. Sunil Prabhakar, Yuni Xia, Dmitri V. Kalashnikov, Walid G. Aref, and Susanne E. Hambrusch. Query Indexing and Velocity Constrained Indexing: Scalable Techniques for Continuous Queries on Moving Objects. *IEEE Trans. on Computers*, 51(10), 2002.
24. Simonas Saltenis and Christian S. Jensen. Indexing of Moving Objects for Location-Based Services. In *Proceedings of the International Conference on Data Engineering, ICDE*, 2002.
25. Simonas Saltenis, Christian S. Jensen, Scott T. Leutenegger, and Mario A. Lopez. Indexing the Positions of Continuously Moving Objects. In *Proceedings of the ACM International Conference on Management of Data, SIGMOD*, 2000.
26. Timos K. Sellis. Multiple-Query Optimization. *ACM Transactions on Database Systems, TODS*, 13(1), 1988.
27. Praveen Seshadri. Predator: A Resource for Database Research. *SIGMOD Record*, 27(1):16–20, 1998.
28. Zhexuan Song and Nick Roussopoulos. K-Nearest Neighbor Search for Moving Query Point. In *Proceedings of the International Symposium on Advances in Spatial and Temporal Databases, SSTD*, 2001.
29. Jimeng Sun, Dimitris Papadias, Yufei Tao, and Bin Liu. Querying about the Past, the Present and the Future in Spatio-Temporal Databases. In *Proceedings of the International Conference on Data Engineering, ICDE*, 2004.
30. Yufei Tao, Dimitris Papadias, and Qiongmao Shen. Continuous Nearest Neighbor Search. In *Proceedings of the International Conference on Very Large Data Bases, VLDB*, 2002.

31. Yufei Tao, Dimitris Papadias, and Jimeng Sun. The TPR*-Tree: An Optimized Spatio-temporal Access Method for Predictive Queries. In *Proceedings of the International Conference on Very Large Data Bases, VLDB*, 2003.
32. Yufei Tao, Jimeng Sun, and Dimitris Papadias. Analysis of Predictive Spatio-Temporal Queries. *ACM Transactions on Database Systems, TODS*, 28(4), 2003.
33. Douglas B. Terry, David Goldberg, David Nichols, and Brian M. Oki. Continuous Queries over Append-Only Databases. In *Proceedings of the ACM International Conference on Management of Data, SIGMOD*, 1992.
34. Xiaopeng Xiong, Mohamed F. Mokbel, Walid G. Aref, Susanne Hambrusch, and Sunil Prabhakar. Scalable Spatio-temporal Continuous Query Processing for Location-aware Services. In *Proceedings of the International Conference on Scientific and Statistical Database Management, SSDBM*, June 2004.
35. Jun Zhang, Manli Zhu, Dimitris Papadias, Yufei Tao, and Dik Lun Lee. Location-based Spatial Queries. In *Proceedings of the ACM International Conference on Management of Data, SIGMOD*, 2003.
36. Baihua Zheng and Dik Lun Lee. Semantic Caching in Location-Dependent Query Processing. In *Proceedings of the International Symposium on Advances in Spatial and Temporal Databases, SSTD*, 2001.