GeoInformatica 9:4, 343–365, 2005

Continuous Query Processing of Spatio-Temporal Data Streams in PLACE*

MOHAMED F. MOKBEL, XIAOPENG XIONG AND WALID G. AREF
Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN, USA
E-mail: mokbel@cs-umn.edu

MOUSTAFA A. HAMMAD
Department of Computer Science, University of Calgary, Calgary, Alberta, Canada
E-mail: hammad@cpsc.ucalgary.ca

Received December 6, 2004; Revised March 7, 2005; Accepted April 6, 2005

Abstract

The tremendous increase in the use of cellular phones, GPS-like devices, and RFIDs results in highly dynamic environments where objects as well as queries are continuously moving. In this paper, we present a continuous query processor designed specifically for highly dynamic environments (e.g., location-aware environments). We implemented the proposed continuous query processor inside the PLACE server (Pervasive Location-Aware Computing Environments); a scalable location-aware database server developed at Purdue University. The PLACE server extends data streaming management systems to support location-aware environments. These environments are characterized by the wide variety of continuous spatio-temporal queries and the unbounded spatio-temporal streams. The proposed continuous query processor includes: (1) New incremental spatio-temporal operators to support a wide variety of continuous spatio-temporal queries, (2) Extended semantics of sliding window queries to deal with spatial sliding windows as well as temporal sliding windows, and (3) A shared-execution framework for scalable execution of a set of concurrent continuous spatio-temporal queries. Experimental evaluation shows promising performance of the continuous query processor of the PLACE server.

Keywords: spatio-temporal databases, continuous queries, data stream management systems, location-aware services

1. Introduction

The rapid increase in spatio-temporal applications calls for new query processing techniques to deal with both the spatial and temporal domains. Examples of these applications include location-aware services [36], traffic monitoring [42], and enhanced 911 service (http://www.fcc.gov/911/enhanced/). Such applications continuously receive data from mobile objects (e.g., moving vehicles in road networks). The streaming nature

* This work was supported in part by the National Science Foundation under Grants IIS-0093116, IIS-0209120, and 0010044-CCR.

of incoming spatio-temporal data poses new challenges that require combining and/or modifying the recent advances in both spatio-temporal database systems and data stream management systems.

Recent research efforts for continuous spatio-temporal query processing (e.g., see [21], [28], [31]–[33], [38], [44], [48], [55], [56]) rely mainly on the ability of storing and indexing spatiotemporal data. Given the dynamic environment of spatio-temporal applications, the main idea is to modify traditional data indices to support frequent updates. Examples of these indices include modified grid structures (e.g., [18], [38]), modified B-trees, (e.g., [29]), modified R-trees (e.g., [31], [33]), and time-parameterized R-trees (e.g., [47], [48], [56]). Although these indexing schemes give better support for updates than their counterpart traditional indices, issues of high arrival rates of both objects and queries, infinite source of data, and spatio-temporal streams are not addressed by these approaches. With the notion of spatio-temporal streams, only in-memory algorithms for continuous queries need to be realized.

Numerous research efforts are devoted to stream query processing (e.g., see [1], [3], [11], [16], [26], [41], [66]). The main focus is to provide the ability to process incoming data streams online against a set of outstanding and continuous queries. However, the spatial and temporal properties of both data and queries are not addressed. A spatio-temporal data stream distinguishes itself from a traditional data stream in the following: (1) Queries as well as data have the ability to change their locations continuously. Thus, the arrival of a new data item (e.g., the location) of an object, say p, at some time t_2 ($t_2 > t_1$) may result in expiring the previous location information of p at time t_1 (*predicate-based* sliding window). This is in contrast to traditional data streams where data is expired only as it becomes old in the system (time-based expiration in sliding window queries). (2) An object may be added to or removed from the answer set of a spatio-temporal query (*positive* and *negative* answers). For example, consider a set of moving vehicles that move in and out of a certain range query. Thus the query answer may be represented *progressively* by a sequence of *positive* and *negative* updates. This is in contrast to traditional queries where only an addition to the query answer is permitted. (3) Due to the mobility of both objects and queries, any delay in processing spatio-temporal queries may result in an obsolete answer. Consider a query that asks about 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. These distinguishing characteristics of spatio-temporal streams require revisiting traditional data stream management systems to have special handling of spatio-temporal streams.

In this paper, we present the PLACE server (**P**ervasive **L**ocation-**A**ware **C**omputing **E**nvironments) [4], [39]; a scalable location-aware database server currently being developed at Purdue University. The PLACE server extends the Nile [26] data stream management system to support continuous query processing of spatio-temporal streams. The PLACE server aims to bridge the areas of spatio-temporal databases and data stream management systems. The main idea is to furnish traditional data stream management systems with the basic functionalities that support processing incoming spatio-temporal streams against a set of outstanding continuous spatio-temporal queries. In particular, the

continuous query processor of the PLACE server [40] has the following distinguishing characteristics:

1. Predicate-based window queries: The PLACE continuous query processor extends the processing of continuous sliding window queries beyond time-based and tuple-count windows to accommodate for the so called *predicate-based* window queries. In predicate-based window queries, objects are qualified to be part of the window once they satisfy a certain query predicate. Similarly, objects are expired only when they no longer satisfy a certain predicate. *Predicate-based* windows are a generalization of time-based and tuple-count sliding windows.
2. Incremental evaluation. The PLACE continuous query processor employs an *incremental* evaluation paradigm by continuously updating the query answer. We distinguish between two types of updates; namely positive and negative updates [38]. A positive/negative update indicates that a certain object needs to be added to/ removed from the query answer.
3. Spatio-temporal operators. The PLACE continuous query processor goes beyond the idea of implementing high level algorithms for continuous spatio-temporal queries. Instead, the PLACE server encapsulates the spatio-temporal query algorithms into a set of primitive spatio-temporal pipelined operators (e.g., INSIDE and *k*NN operators) that can be part of a larger query plan. Having a set of primitive spatio-temporal operators results in supporting a wide variety of continuous spatio-temporal queries and in having flexible query optimizers where multiple candidate query plans can be produced.
4. Scalability. We use a *shared-execution* paradigm as a means of achieving scalability in terms of the number of outstanding continuous spatio-temporal queries.

The rest of the paper is organized as follows. Section 2 highlights the challenges in realizing the continuous query processor of the PLACE server along with the related work of each challenge. In Section 3, we present an overview of the data model and SQL language used by the PLACE server. Section 4 presents *predicate-based* window queries. The *incremental* evaluation of the PLACE server is discussed in Section 5. The scalability in terms of the number of outstanding spatio-temporal queries is addressed in Section 6. The GUI interfaces for the PLACE server and PLACE clients are presented in Section 7. Section 8 presents experimental results that evaluate the performance of the PLACE server. Finally, Section 9 concludes the paper.

2. Challenges and related work

In this section, we highlight some challenges in realizing the continuous query processor of the PLACE location-aware server. With each challenge, we summarize the related work in the areas of spatio-temporal databases and data streams. Then, we highlight briefly how the PLACE continuous query processor deals with these challenges.

2.1. Challenge I: Massive size of incoming spatio-temporal data streams

Spatio-temporal databases. Existing continuous query processors for spatio-temporal databases assume explicitly that all incoming data can be stored in secondary storage. A wide variety of spatio-temporal access methods (e.g., see [37] for a survey) has been introduced to deal with massive sizes of spatio-temporal data. However, with the streaming input, only in-memory algorithms are feasible. There is limited work that exploits the spatial and/or temporal properties of data streams. The spatial properties of data streams are addressed recently in Cormode and Muthukrishnan [15] and Hershberger and Suri [27] to solve geometric problems, e.g., computing the convex hull [27]. In Sun et al. [53], spatio-temporal histograms are used as synopses for approximate query processing on spatio-temporal data streams. Up to the authors' knowledge, there is no existing work that addresses continuous query processing for spatio-temporal streams.

Data stream management systems. A common challenge for all data stream management systems is the infinite size of the incoming data streams. With the inability to store all the incoming data into memory, data stream management systems tend to store only data that is of interest to any outstanding continuous query. Once a stored data becomes out of interest of all outstanding queries, it is expired from the memory leaving its space to a more important incoming tuple. One approach of expiring in-memory data is to use *punctuation* [61]. A *punctuated* tuple indicates the expiration of a certain set of stored tuples. To decide whether a certain incoming tuple is important or not, every continuous query is associated with a *historical* window that limits the important tuples to the most recent ones. A *historical* window could be time-based (e.g., the last 1 hour) or tuple-based (e.g., the last 100 tuples). Such queries are termed sliding-window queries (e.g., see [3], [5], [11], [20], [24], [30], [34], [52]). In *sliding-window* queries, tuples are dropped (expired) from the system in a first-in-first-expire policy. In case of large window sizes, *load shedding* techniques (e.g., [59]) are utilized to drop some tuples from memory.

The PLACE approach. Trying to deploy the idea of *sliding-window* queries from traditional data streams would result in limiting the functionality of the PLACE server. A wide variety of continuous spatiotemporal queries are considered as NOW queries [14] (i.e., no historical information is needed). Thus, in the PLACE server, we generalize the time-based and tuple-based sliding window queries to *predicate-based* window queries. A tuple is considered important if it satisfies at least one query predicate from all outstanding continuous queries. Once a tuple no longer satisfies any query predicate, it is expired from the server. This implies that tuple expiration is predicate-based rather than first-in-first-expire as in the commonly used time-based or tuple-count data streams. Expiration in *predicate-based* window queries is different from *punctuated* streams [61] in that tuples in *predicate-based* windows are expired one at a time, rather than expiring a set of tuples using a certain *punctuated* tuple.

2.2. Challenge II: Continuous evaluation of continuous queries

Spatio-temporal databases. Most of the existing techniques in spatio-temporal databases abstract the continuous query into a series of snapshot queries executed at regular interval times. Mainly, three different approaches are investigated: (1) The validity of the results [67], [68]. With each query answer, the server returns a *valid time* [68] or a *valid region* [67] 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. (2) Caching the results. The main idea is to cache the previous result either in the client side [51] or in the server side [32]. Previously cached results are used to prune the search for the new results of *k*-nearest-neighbor queries [51] and range queries [32]. (3) Precomputing the result [32], [55]. If the trajectory of query movement is known apriori, then by using computational geometry for stationary objects [55] or velocity information for moving objects [32], we can identify which objects will be nearest-neighbors [55] to or within a range [32] from the query trajectory. If the trajectory information changes, then the query needs to be reevaluated. Up to the authors' knowledge, our earlier work, SINA [38], is the only work that addresses incremental evaluation of continuous queries in spatio-temporal databases.

Data stream management systems. Due to the newly introduced *sliding-window* queries, several algorithms are proposed for each query operator, e.g., window join Golab et al. [20], Hammad et al. [23], [24], Kang et al. [30], and Srivastava and Widom [52] and window aggregates Arasu and Widom [2] and Datar et al. [17]. All window algorithms for traditional data streaming provide progressive updates of the query answer.

The PLACE approach. A distinguished characteristic of spatiotemporal streams is that we need to have the ability to *remove* some parts of the query answer (e.g., an object moves out of the range query). This feature is not available in traditional data streams where the query answer is append-only. In the PLACE server, we apply a progressive evaluation paradigm by extending the ideas of SINA [38] to be applicable to spatio-temporal streams rather than being tied to disk storage.

2.3. Challenge III: Wide variety of continuous query types

Spatio-temporal databases. A major challenge for spatio-temporal query processors is the wide variety of spatio-temporal query types (e.g., see [36] for a thorough classification of spatio-temporal queries). Thus, it becomes a difficult task to provide a database system with the ability to support all kinds of spatio-temporal queries. The DOMINO database system [63] is the first attempt to build a database of moving objects on top of existing DBMSs [62]. One main focus of DOMINO is to support new kinds of spatio-temporal attributes and query language for moving objects. However, the query

processing issues are not addressed. Other than DOMINO, most of the existing query processing techniques focus on solving special cases of continuous spatio-temporal queries. (For example, [51], [55], [67], [68]) focus on *moving* queries on *stationary* objects while Cai et al. [8], Gedik and Liu [18], Hadjieleftheriou et al. [21] and Prabhakar et al. [44]) focus on *stationary* range queries on *moving* objects. Other work focuses on aggregate queries (e.g., see [21], [53], [58]), *k*-NN queries (e.g., see [28], [51]) and reverse nearest-neighbor queries [6].

Data stream management systems. Existing data stream management systems (e.g., Borealis [1], Telegraph [11], Nile [26], and STREAM [41]) provide new algorithms for (almost) all traditional query operators. However, there is no special handling and/or optimization for spatio-temporal queries.

The PLACE approach. In the PLACE server, we go beyond the idea of having tailored high-level algorithms for each specific spatiotemporal query. Instead, we furnish existing data stream management systems by a set of primitive spatio-temporal pipeline operators (e.g., the `INSIDE` and `kNN` operators). Spatio-temporal operators are combined with traditional streaming operators to provide the ability of having complex query plans that represent a wide variety of continuous spatio-temporal queries. In addition, the PLACE server provides a uniform framework that is applicable to all mutability combinations of objects and queries, e.g., *moving* queries on *stationary* objects, *stationary* queries on *moving* objects, and *moving* queries on *moving* objects.

2.4. Challenge IV: Large number of concurrent continuous queries

Spatio-temporal databases. Most of the existing spatio-temporal algorithms focus on evaluating only one spatio-temporal query (e.g., [6], [28], [32], [51], [55], [57], [67], [68]). Optimization techniques for evaluating a set of continuous spatio-temporal queries are addressed recently for centralized [38], [44] and distributed environments [8], [18]. In centralized environments, the Q-index [44] is presented as an R-tree-like index structure to index the *stationary* queries instead of objects. SINA [38] uses a *shared* grid structure to index both objects and queries. Then, evaluating a set of continuous queries is abstracted as a spatial join (using the grid index) between objects and queries. In distributed environments, the main idea of Cai et al. [8] and Gedik and Liu [18] is to ship part of the query processing down to the moving objects, while the server acts mainly as a mediator among moving objects.

Data stream management systems. There is a lot of research in sharing the execution of concurrent continuous queries (e.g., NiagaraCQ [12], [13] and PSoup [9], [10]). The main idea is to have a shared query plan for all continuous queries. Other forms of sharing at the operator level are investigated, for window join [19], [24] and window aggregate operators [2].

The PLACE approach. In the PLACE server, we employ a *shared-execution* paradigm similar to that in NiagaraCQ [12] and SINA [39]. The execution of a set of concurrent continuous spatio-temporal queries is performed as a spatial join between two incoming streams. The first stream represents the streaming objects while the second stream represents the streaming queries.[1]

3. The PLACE server

In this section, we present the data modelling and SQL language used by the PLACE server.

3.1. Data model

By subscribing with the PLACE server, moving objects are required to send their location updates periodically to the PLACE server. A location update from the client (moving object) to the server has the format (*OID*, *x*, *y*), where *OID* is the object identifier, (*x*, *y*) is the location of the moving object in the two-dimensional space. An update is time-stamped upon its arrival at the server side. Once an object *P* stops moving (e.g., *P* reaches its destination or *P* is shut down), either *P* sends an explicit *disappear* message to the server or the server will timeout due to not receiving any updates from *P* for a certain time $T_{Timeout}$. In both cases, the server recognizes that object *P* is no further moving.

Due to the highly dynamic nature of location-aware environments and the infinite size of incoming spatio-temporal streams, we cannot store all incoming data. Thus, the PLACE server employs a three-level storage hierarchy. First, a subset of the incoming data streams is stored in in-memory buffers. In-memory buffers are associated with the outstanding continuous queries at the server. Each query determines which tuples are needed to be in its buffer and when these tuples are expired, i.e., deleted from the buffer. Second, we keep an in-disk storage that keeps track with only one reading of each moving object and query. Since, we cannot update the disk storage every time we receive an update from moving objects, we sample the input data by choosing every *k*th reading to flush to disk. Moreover, we cache the readings of moving objects/queries and flush them once to the secondary storage every *T* time units. Data on the secondary storage are indexed using a simple grid structure [38]. Third, every $T_{archive}$ time units, we take a snapshot of the in-disk database and flush it to a repository server. The repository server acts as a multi-version structure of the moving objects that supports historical queries. Stationary objects (e.g., gas stations, hospitals, restaurants) are preloaded to the system as relational tables that are infrequently updated.

In this paper, we focus on the memory part of the PLACE server where continuous queries over spatio-temporal streams can be evaluated. For processing and querying spatio-temporal data in the secondary storage, the reader is referred to Mokbel et al. [38], and Xiong et al. [65]. Archival storage is used to query historical spatio-temporal data which is not the focus of this paper.

3.2. Extended SQL syntax

As the PLACE server [39] extends both PREDATOR [50] and NILE [26], we extend the SQL language provided by both systems to support spatio-temporal operators. As a proof of concept, we focus on two main operators, namely the `INSIDE` and `kNN` operators to support continuous range queries and k-nearest-neighbor queries, respectively. Other operators (e.g., reverse-nearest-neighbor [6], trajectory-based operators [43], [60], and navigation [54]) are subject to future research. A continuous query is registered at the PLACE server using the SQL:

```
REGISTER QUERY query_name AS
SELECT select_clause
FROM from_clause
WHERE where_clause
INSIDE inside_clause
kNN knn_clause
WINDOW window_clause
```

The `REGISTER QUERY` statement registers the continuous query at the PLACE server with the *query_name* as its identifier. The *select_clause*, *from_clause* and *where_clause* are inherited from the PREDATOR [50] database management statement. The *window_clause* is inherited from the NILE [26] stream query processor to support continuous sliding window queries [24]. A continuous query is dropped form the system using the SQL `DROP QUERY` *query_name*.

The *inside_clause* may represent stationary/moving rectangular or circular range queries. Moving queries are tied to *focal* objects. As the *focal* object reports movement update to the server, we update the query region. Notice that we do not need to represent the object and query movement with a motion vector. Instead, the motion is presented in terms of frequent location updates. One *focal* object may issue several queries. A rectangular range query can have one of the following two forms:

- Static range query (x_1, y_1, x_2, y_2), where (x_1, y_1) and (x_2, y_2) represent the top left and bottom right corners of the rectangular range query.
- Moving rectangular range query ($'M'$, *ID*, *xdist*, *ydist*), where $'M'$ is a flag to indicate that the query is moving, *ID* is the identifier of the query focal point, and *xdist* and *ydist* are the length and width of the query rectangle.

A circular range query has the same syntax except that we define only the radius instead of (x, y). Similarly, the *knn_clause* for continous k-nearest-neighbor queries may have one of the following two forms:

- Static kNN query (k, x, y), where k is the number of the neighbors to be maintained, and (x, y) is the center of the query point.

– Moving kNN query ($'M'$, k, ID), where $'M'$ is a flag to indicate that the query is moving, k is the number of neighbors to be maintained, and ID is the identifier of the query focal point.

4. Predicate-based sliding windows

With the unbounded incoming spatio-temporal streams, it becomes infeasible to store all incoming tuples. However, some input tuples may be buffered in memory for a limited time. The choice of the stored tuples is mainly query dependent, i.e., we store only the tuples that are of interest to any outstanding continuous query. In addition, there should be a mechanism to expire (delete) some of the stored tuples and replace them with other tuples that become more relevant to the outstanding continuous queries. The PLACE server employs a *predicate-based* window policy, where each query is associated with a certain predicate. Only tuples that satisfy at least one query predicate are stored in memory. Based on *predicate-based* window queries, the PLACE server supports three types of tuple expiration, namely, *temporal* expiration (sliding-window queries), *spatial* expiration, and *predicate-based* expiration. In general, expired tuples may result in a query *uncertainty* where a future query or a moving query would ask for these tuples. In PLACE, we deal with different types of query *uncertainty* using a *caching* technique. For more detailed about *uncertainty* and *caching* in PLACE, the reader is referred to Mokbel and Aref [35].

4.1. Temporal expiration

Temporal expiration is used commonly to support continuous sliding-window queries in data streams. A sliding window query involves a *historical* time window w. Any object that has a timestamp within the current sliding window of any outstanding query Q is buffered in-memory with the associated buffer of Q. Stored tuples follow strictly a first-in-first-expire policy.

An example for a *historical* sliding window query submitted to the PLACE server is: Q_1: "*Continuously, report the number of cars that passed by region R in the last hour*."

```
SELECT COUNT(ObjectID)
FROM MovingObjects
WHERE type = Car
INSIDE R
WINDOW 1 hour
```

Notice that Q_1 buffers all incoming tuples during the previous hour. A tuple is expired (i.e., deleted from the query buffer) once it goes out of the sliding time window (i.e., when the car tuple becomes more than one hour old).

4.2. Spatial expiration

NOW queries are more common in spatio-temporal data stream applications than historical queries. For example, monitoring applications are concerned with the actual current answer not the accumulated historical one. Such applications cannot be realized using only temporal expiration. Thus, the PLACE server introduces a new type of expiration that depends on the *spatial* location of the moving objects instead of their timestamps. An incoming tuple, say o, is stored in the in-memory buffer associated with a query Q only if o satisfies the *spatial* window (e.g., region) of Q. A stored tuple is expired only when it steps out of the *spatial* window.

An example of a spatial expiration query is: Q_2: "*Continuously, report the number of cars in a certain area*." Notice that unlike Q_1, in Q_2, we are concerned about the actual current number of cars not the number of cars in the recent history. The SQL of Q_2 is similar to that of Q_1 with only the removal of the window statement.

Figure 1 gives the difference between a temporal-expiration and a spatial-expiration in a range query. Figure 1a gives a snapshot of the database at time T_0. The vertical **bold** line represents the spatial predicate window in the one-dimensional space (along the y axis). The shaded rectangular area represents the temporal window along the time dimension (the x axis). A query with a temporal window is interested in objects that lie in the shaded area while a query with a spatial window is interested in the actual objects that lie in the spatial region regardless of their timestamps. The answer to both the temporal and spatial window queries at time T_0 is (P_1, P_3, P_5). Figure 1b gives a snapshot of the database at time T_1. Only objects P_3 and P_4 change their locations (the new locations are plotted as white circle while the old location of P_3 is plotted as a gray circle). For the temporal query, although P_1 still satisfies the query spatial predicate, P_1 is discarded from the server. Also, although P_3 becomes out of the query predicate, the

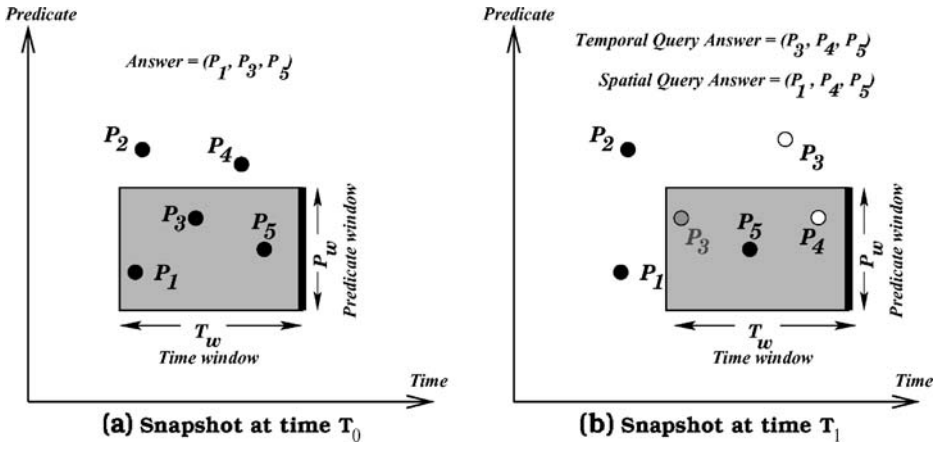

Figure 1. Temporal and spatial expiration.

old value of P_3 has not yet expired. Thus, the temporal query answer at T_1 would be (P_3, P_4, P_5). However, the answer of the spatial window query would be (P_1, P_4, P_5).

4.3. Predicate-based expiration

Predicate-based expiration generalizes both the temporal and spatial-based expiration methods. *Punctuation-based* expiration [61] can be considered as a special case of *predicate-based* when we consider the predicate as the arrival of a certain artificial tuple (the *punctuated* tuple). An incoming tuple is stored in-memory only when it satisfies a certain query predicate. A predicate could be as simple as a temporal or a spatial predicate. Due to the nature of spatio-temporal streams, other forms of predicates may arise. For example, consider the query Q_3: "*For each moving object, continuously report the elapsed time between each two consecutive readings*". Such a query contains a self-join where objects from the stream of moving objects are self-joined based on the object identifier. The query buffer needs to maintain only the latest reading of each moving object. Once the reading of a certain object is reported, the previous reading is expired.

Predicate-based expiration is expressed with the same SQL as the spatial expiration except that there is no limit on the predicate complexity. For any query Q, predicate evaluation is triggered in two cases: (1) Movement of an object P that is candidate to cross any of the query Q boundary, (2) Movement of query Q. To avoid excessive evaluations of the query predicate, the underlying data structure (e.g., a grid structure) is used to limit the evaluation to those tuple that are candidate to produce results.

5. Incremental evaluation

To avoid reevaluating continuous spatio-temporal queries, we employ an *incremental evaluation* paradigm in the PLACE continuous query processor. The main idea is to only report the changes of the answer from the last evaluation time. By employing *incremental evaluation*, the PLACE server achieves the following goals: (1) Fast query evaluation, since we compute only the updates of the answer not the whole answer. (2) In a typical location-aware server, query results are sent to the users via satellite servers [22]. Thus, limiting the amount of transmitted data to the updates only rather than the whole query answer saves in network bandwidth. (3) When encapsulating incremental algorithms into physical pipelined query operators, limiting the tuples that go through the whole query pipeline to only the updates reduces the flow in the pipeline. Thus, efficient query processing is achieved.

To realize the incremental evaluation processing in the PLACE server, we go through three main steps. First, we define the high level concept of incremental updates, by defining two types of updates; *positive* and *negative* updates [38]. Second, we encapsulate the processing of incremental algorithms into pipelined query operators. Third, we

modify traditional pipelined query operators (e.g., distinct and join) to deal with the concept of negative tuples [25].

5.1. Positive/negative updates

Incremental evaluation is achieved through updating the previous query answer. Mainly, we distinguish between two types of updates; *positive* updates and *negative* updates. A positive/negative update indicates that a certain object needs to be added to/removed from the query answer. A query answer is represented in the form (*QID*, *OList*), where *QID* is the query identifier and *OList* is the query answer. The PLACE server continuously updates the query answer with updates of the form (*QID*, ±, *OID*) where ± indicates the type of the update and *OID* is the object identifier.

Figure 2 gives an example of applying the concept of positive/negative updates on a set of continuous range queries. The snapshot of the database at time T_0 is given in Figure 2a with nine moving objects, p_1 to p_9, and five continuous range queries, Q_1 to Q_5. The answer of the queries at time T_0 is represented as (Q_1, P_5), (Q_2, P_1), (Q_3, P_6, P_7), (Q_4, P_3, P_4), and (Q_5, P_9). 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. As a result, the PLACE server reports the following updates: (Q_1, $-P_5$), (Q_3, $-P_6$), (Q_3, $+P_8$), and (Q_4, $-p_4$).

5.2. Spatio-temporal incremental pipelined operators

Two alternative approaches can be utilized in implementing spatiotemporal algorithms inside the PLACE server: using SQL *table functions* [45] or by encapsulating the algo-

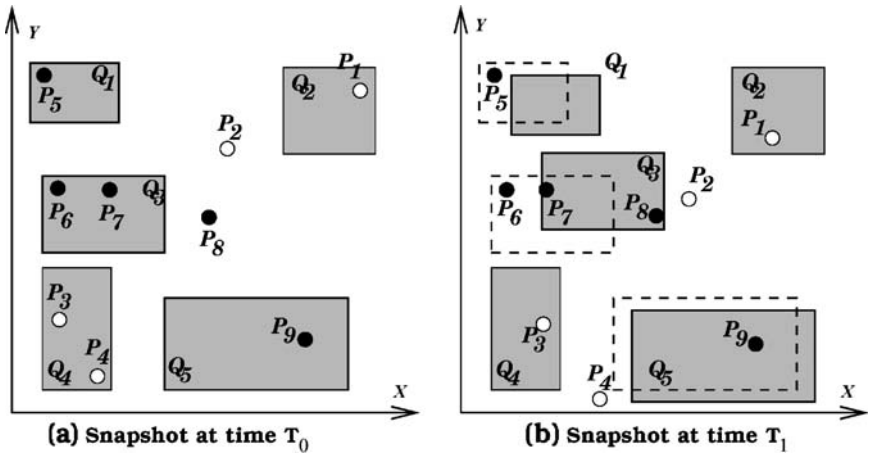

Figure 2. Incremental evaluation of range queries.

rithms in physical query operators. Since there is no straightforward method for pushing query predicates into table functions [46], the performances is limited and the approach does not give enough flexibility in optimizing the issued queries. In the PLACE server, we encapsulate our algorithms inside physical pipelined query operators that can be part of a query execution plan. By having pipelined query operators, we achieve three goals: (1) Spatio-temporal operators can be combined with other operators (e.g., distinct, aggregate, and join operators) to support incremental evaluation for a wide variety of continuous spatio-temporal queries. (2) Pushing spatio-temporal operators deep in the query execution plan reduces the number of tuples in the query pipeline. This reduction comes from the fact that spatio-temporal operators act as filters to the above operators. (3) Flexibility in the query optimizer where multiple candidate execution plans can be produced.

The main idea of spatio-temporal operators is to keep track of the recently reported answer of each query Q in a query buffer termed *Q.Answer*. Then, for each newly incoming tuple P, we perform two tests: Test I: Is P part of the previously reported *Q.Answer*? Test II: Does P qualify to be part of the current answer? Based on the results of the two tests, we distinguish among four cases:

- **Case I:** P is part of *Q.Answer* and P still qualify to be part of the current answer. As we process only the updates of the previously reported result, P will not be processed.
- **Case II:** P is part of *Q.Answer*, however, P does not qualify to be part of the answer anymore. In this case, we report a *negative* update P^- to the above query operator. The negative update indicates that P is spatially expired from the answer.
- **Case III:** P is not part of *Q.Answer*, however, P qualifies to be part of the current answer. In this case, we report a positive update to the above query operator.
- **Case IV:** P is not part of *Q.Answer* and P still does not qualify to be part of the current answer. In this case, P has no effect on Q.

5.3. *Negative tuples in traditional operators*

Having the spatio-temporal operators at the bottom or at the middle of the query evaluation pipeline requires that all the above operators be equipped with special handling of negative tuples. Fortunately, recent data stream management systems (e.g., Borealis [1], NILE [26], STREAM [41]) have the ability to process such negative tuples. The NILE query processor [26] handles negative tuples in pipelined operators as follows: *Selection* and *Join* operators handle *negative* tuples in the same way as the regular *positive* tuples. The only difference is that the output will be in the form of a *negative* tuple. *Aggregates* update their aggregate functions by considering the received *negative* tuple. The *Distinct* operator reports a *negative* tuple at the output only if the corresponding *positive* tuple is in the recently reported result. For more detail about handling the *negative* tuples in various query operators, the reader is referred to Hammad et al. [25].

6. Scalability

The PLACE continuous query processor exploits a *shared-execution* paradigm [36], [38], [64] as a means for achieving scalability in terms of the number of concurrently executing continuous spatio-temporal queries. The main idea is to group similar queries in a query table. Then, the evaluation of a set of continuous queries is modelled as a spatial join between moving objects and moving queries. Similar ideas of shared-execution have been exploited in the NiagaraCQ [12] for web queries and PSoup [9], [10] for streaming queries.

Figure 3a gives the execution plans of two simple continuous spatiotemporal queries, Q_1: "*Find the objects inside region* R_1," and Q_2: "*Find the objects inside region* R_2." With *shared-execution*, we have the execution plan of Figure 3b. Shared-execution for a collection of spatio-temporal range queries can be expressed in the PLACE server by issuing the following continuous query:

```
SELECT Q.ID, O.ID
FROM QueryTable Q, ObjectTable O
WHERE O.location inside Q.region
```

7. User interface in PLACE

Figures 4 and 5 give snapshots of the server and client graphical user interface (GUI) of PLACE, respectively. The server GUI displays all moving objects on the map.[2] The client GUI simulates a client end-device used by the users. Users can choose the type of

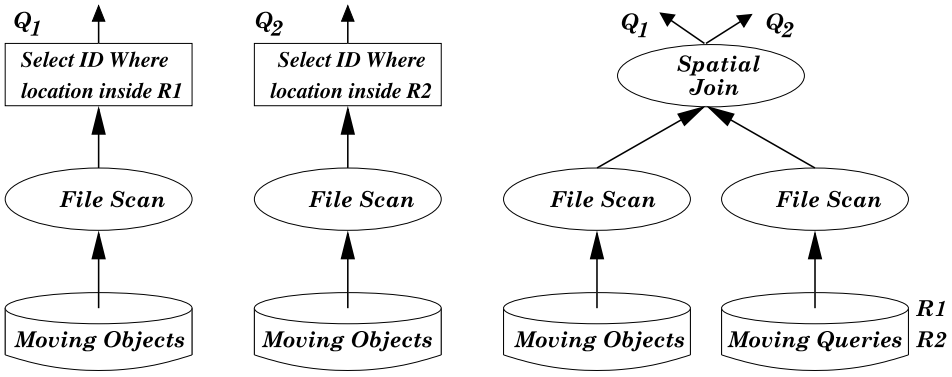

Figure 3. Shared-execution of continuous queries.

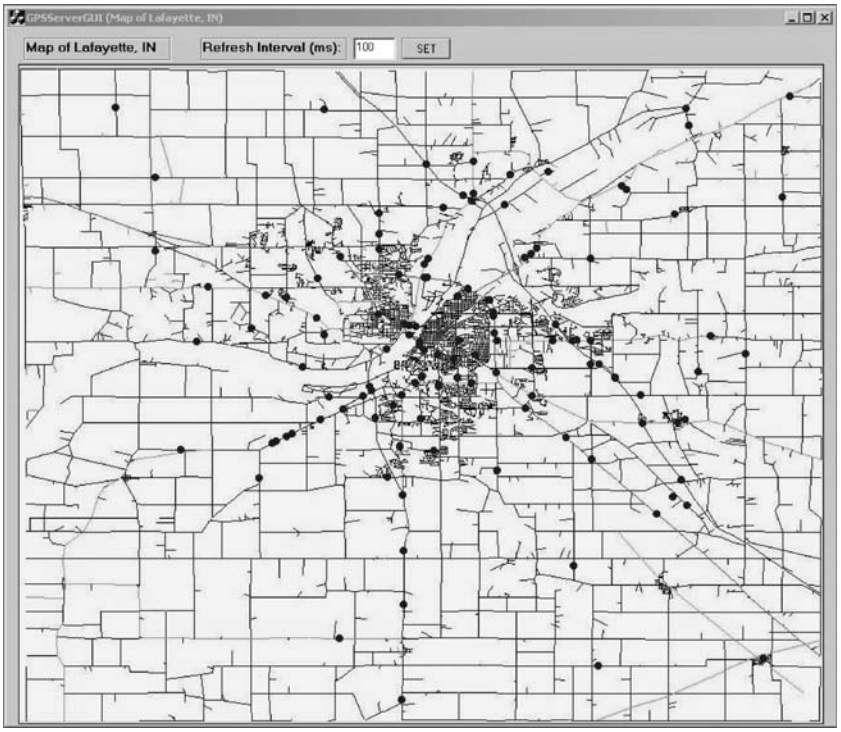

Figure 4. Server GUI.

query from a list of available query types (stationary/moving range queries and stationary/moving *k*-nearest-neighbor queries). The spatial region of the query can be determined using the map of the area of interest. By pressing the *submit* button, the client translates the query into SQL language and transmits it to the PLACE server. The result appears both in the list of Figure 5 and as moving objects on the map. A client can see only, on its map, the objects that belong to its issued query.

8. Performance evaluation

In this section, we present experiments that show the promising performance of the continuous query processor in the PLACE server. We use the *Network-based Generator of Moving Objects* [7] 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 range queries.

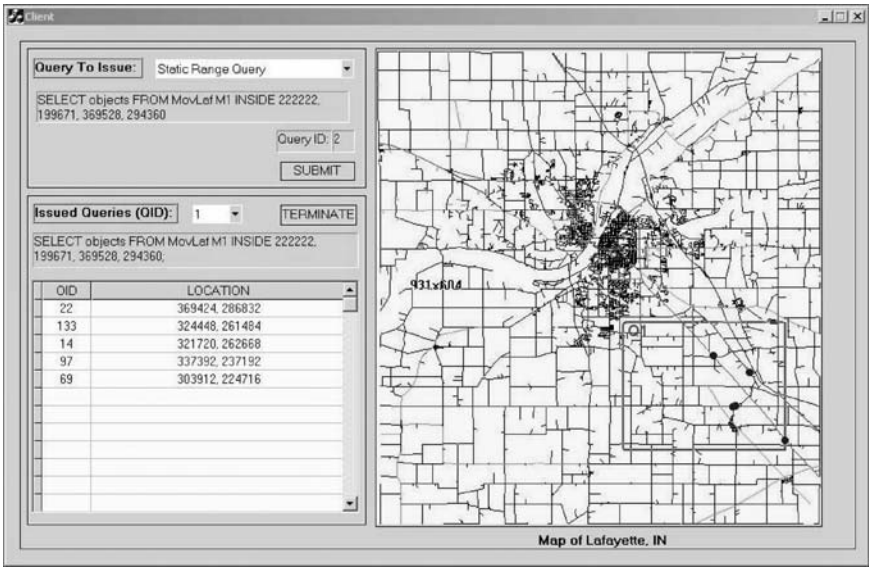

Figure 5. Client GUI.

8.1. Size of incremental answer

Figure 6 compares between the size of the incremental answer returned by utilizing the incremental approach and the size of the complete answer. The location-aware server buffers the received updates from moving objects and queries and evaluates the queries 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

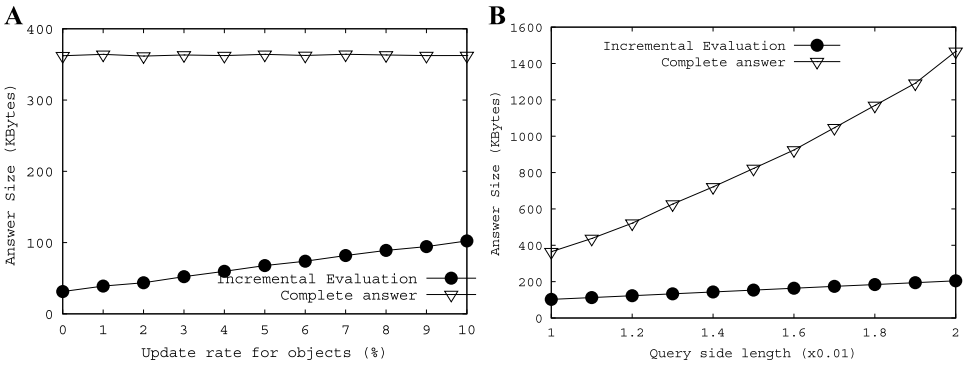

Figure 6. A) Moving objects (%). B) Query size.

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

8.2. Pipelined spatio-temporal operators

In this section, we compare the implementation of spatio-temporal algorithms at the application-level (e.g., table functions) with the encapsulation of the spatio-temporal algorithms inside query operators. In our experiments, we direct the query optimizer to have the spatio-temporal operators at the bottom of the pipeline. However, appropriate query optimization techniques and/or cost models would ensure more optimized query pipelines. Query optimizations along with cost models are in our top list of future research directions.

8.2.1. Pipeline with a select operator. Consider the query *Q*: "*Continuously report all trucks that are within MyArea*." *MyArea* can be either a stationary or moving range query. A high-level implementation of this query has only a selection operator that selects only the "*trucks*." Then, a high-level algorithm implementation would take the selection output and produce incrementally the query result. However, an encapsulation of the `INSIDE` algorithm into a physical operator allows for more flexible plans.

Figure 7 compares the high-level implementation of the above query with pipelined operators for both stationary and moving queries. The selectivity of the queries varies from 2% to 64%. The selectivity of the selection operator is 5%. Our measure of comparison is the number of tuples that go through the query evaluation pipeline. When algorithms are implemented at the application level, the performance is not affected by the selectivity. However, when `INSIDE` is pushed before the *selection*, it acts as a filter

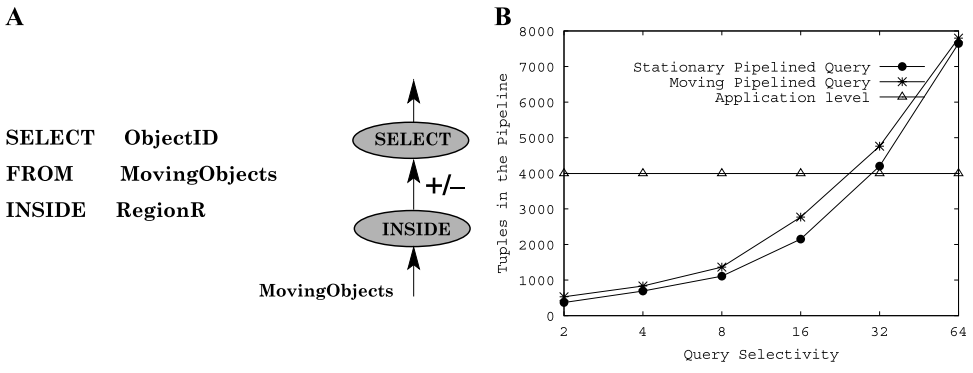

Figure 7. A) Query SQL and pipeline. B) Pipelined operator with Select.

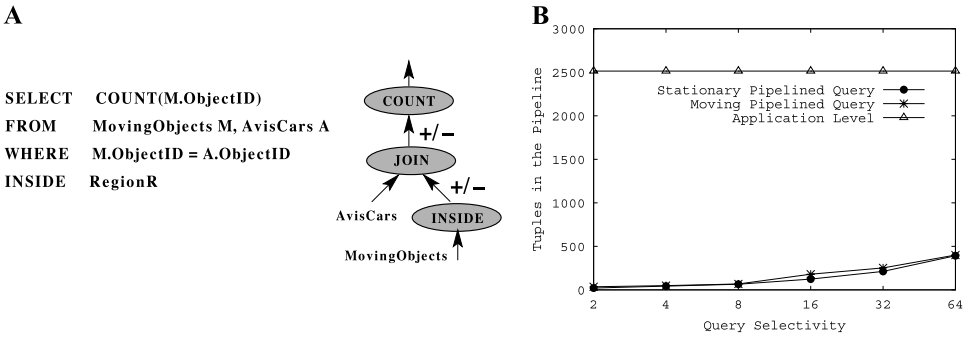

Figure 8. A) Query SQL and pipeline. B) Pipelined operator with Join.

for the query evaluation pipeline, thus, limiting the tuples through the pipeline to only the incremental updates. With `INSIDE` selectivity less than 32%, pushing `INSIDE` before the selection affects the performance greatly.

8.2.2. Pipeline with a join operator. In this section, we consider a more complex query plan that contains a join operator. Consider the query *Q*: "*Continuously report moving objects that belong to my favorite set of objects and lie within MyArea*." A high-level implementation would probe a streaming database engine to join all moving objects with my favorite set of objects. Then, the output of the join is sent to the algorithm for further processing. However, with the `INSIDE` operator, we can have a query evaluation plan as that of Figure 8a where the `INSIDE` operator is pushed below the *Join* operator.

Figure 8 compares the high-level implementation of the above query with the pipelined `INSIDE` operator for both stationary and moving queries. The selectivity of the queries varies from 2% to 64%. Unlike the case of *selection* operators, there is a dramatic increase in performance when `INSIDE` is implemented as a pipelined operator. The main reason in this dramatic gain in performance is the high overhead incurred when evaluating the *join* operation. Thus, the `INSIDE` operator filters out the input tuples and limits the input to the *join* operator to only the incremental *positive* and *negative* updates.

9. Conclusion

In this paper, we presented the continuous query processor of the PLACE (Pervasive Location-Aware Computing Environments) server; a database server for location-aware environments currently developed at Purdue University. The PLACE server extends both the PREDATOR database management system and the NILE stream query processor to deal with unbounded spatio-temporal streams. In addition to the temporal tuple expiration defined in sliding-window queries, we maintain other forms of tuple expirations (e.g., spatial expiration). To efficiently handle large number of continuous queries, we employ an incremental evaluation paradigm that contains: (1) Defining the concept of

positive and *negative* updates, (2) Encapsulating the algorithms for incremental processing into pipelined spatio-temporal operators, and (3) Modifying traditional query operators (e.g., distinct and join) to deal with the *negative* updates that comes from the spatio-temporal operators. Shared-execution is employed by the continuous query processor as a means of achieving scalability in terms of the number of concurrently continuous queries. Experimental results show the promising performance of the PLACE continuous query processor.

Notes

1. For simplicity, we present the spatial join in the context of a uniform grid structure. However, the uniform grid can be substituted by more sophisticated structures e.g., the FUR-tree [33] or quad-tree-like structures [49]. A detailed experimental study between the usage of grid structures and the usage of R-tree-like structures is given in Mokbel et al. [38].
2. The map in Figures 4 and 5 is for the Greater Lafayette area, Indiana, USA.

References

1. D. Abadi, Y. Ahmad, H. Balakrishnan, M. Balazinska, U. Cetintemel, M. Cherniack, J.-H. Hwang, J. Janotti, W. Lindner, S. Madden, A. Rasin, M. Stonebraker, N. Tatbul, Y. Xing, and S. Zdonik. "The design of the Borealis stream processing engine," in *Proceedings of the International Conference on Innovative Data Systems Research, CIDR*, 2005.
2. A. Arasu and J. Widom. "Resource sharing in continuous sliding-window aggregates," in *Proceedings of the International Conference on Very Large Data Bases, VLDB*, 2004.
3. A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, M. Datar, K. Ito, R. Motwani, U. Srivastava, and J. Widom. STREAM: The Stanford Data Stream Management System, 2004.
4. W.G. Aref, S.E. Hambrusch, and S. Prabhakar. "Pervasive Location Aware Computing Environments (PLACE)," http://www.cs.purdue.edu/place/, 2003.
5. S. Babu and J. Widom. "Continuous queries over data streams," *SIGMOD Record*, Vol. 30(3), 2001.
6. R. Benetis, C.S. Jensen, G. Karciauskas, and S. Saltenis. "Nearest neighbor and reverse nearest neighbor queries for moving objects," in *Proceedings of the International Database Engineering and Applications Symposium, IDEAS*, 2002.
7. T. Brinkhoff. "A framework for generating network-based moving objects," *GeoInformatica*, Vol. 6(2), 2002.
8. Y. Cai, K.A. Hua, and G. Cao. "Processing range-monitoring queries on heterogeneous mobile objects," in *Mobile Data Management, MDM*, 2004.
9. S. Chandrasekaran and M.J. Franklin. "Streaming queries over streaming data," in *Proceedings of the International Conference on Very Large Data Bases, VLDB*, 2002.
10. S. Chandrasekaran and M.J. Franklin, "PSoup: A system for streaming queries over streaming data,'' *VLDB Journal*, Vol. 12(2):140–156, 2003.
11. S. Chandrasekaran, O. Cooper, A. Deshpande, M.J. Franklin, J.M. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, and M.A. Shah. "TelegraphCQ: Continuous dataflow processing for an uncertain world," in *Proceedings of the International Conference on Innovative Data Systems Research, CIDR*, 2003.
12. J. Chen, D.J. DeWitt, F. Tian, and Y. Wang. "NiagaraCQ: A scalable continuous query system for internet databases," in *Proceedings of the ACM International Conference on Management of Data, SIGMOD*, 2000.

13. J. Chen, D.J. DeWitt, and J.F. Naughton. "Design and evaluation of alternative selection placement strategies in optimizing continuous queries," in *Proceedings of the International Conference on Data Engineering, ICDE*, 2002.
14. J. Clifford, C.E. Dyreson, T. Isakowitz, C.S. Jensen, and R.T. Snodgrass. "On the semantics of "now" in databases," *ACM Transactions on Database Systems, TODS*, Vol. 22(2), 1997.
15. G. Cormode and S. Muthukrishnan. "Radial histograms for spatial streams," Technical Report DIMACS TR: 2003-11, Rutgers University, 2003.
16. C. Cranor, T. Johnson, O. Spataschek, and V. Shkapenyuk. "Gigascope: A stream database for network applications," in *Proceedings of the ACM International Conference on Management of Data, SIGMOD*, 2003.
17. M. Datar, A. Gionis, P. Indyk, and R. Motwani. "Maintaining stream statistics over sliding windows," in *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms, SODA*, 2002.
18. B. Gedik and L. 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.
19. L. Golab and M.T. Ozsu. "Processing sliding window multi-joins in continuous queries over data streams," in *Proceedings of the International Conference on Very Large Data Bases, VLDB*, 2003.
20. L. Golab, S. Garg, and M.T. Ozsu. "On indexing sliding windows over online data streams," in *Proceedings of the International Conference on Extending Database Technology, EDBT*, 2004.
21. M. Hadjieleftheriou, G. Kollios, D. Gunopulos, and V.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.
22. S.E. Hambrusch, C.-M. Liu, W.G. Aref, and S. Prabhakar. "Query processing in broadcasted spatial index trees," in *Proceedings of the International Symposium on Advances in Spatial and Temporal Databases, SSTD*, 2001.
23. M.A. Hammad, W.G. Aref, and A.K. Elmagarmid. "Stream window join: Tracking moving objects in sensor-network databases," in *Proceedings of the International Conference on Scientific and Statistical Database Management, SSDBM*, 2003.
24. M.A. Hammad, M.J. Franklin, W.G. Aref, and A.K. Elmagarmid. "Scheduling for shared window joins over data streams," in *Proceedings of the International Conference on Very Large Data Bases, VLDB*, 2003.
25. M.A. Hammad, T.M. Ghanem, W.G. Aref, A.K. Elmagarmid, and M.F. Mokbel. Efficient Pipelined Execution of Sliding-Window Queries Over Data Streams. Technical Report TR CSD-03-035, Purdue University Department of Computer Sciences, 2003.
26. M.A. Hammad, M.F. Mokbel, M.H. Ali, W.G. Aref, A.C. Catlin, A.K. Elmagarmid, M. Eltabakh, M.G. Elfeky, T.M. Ghanem, R. Gwadera, I.F. Ilyas, M. Marzouk, and X. Xiong. "Nile: A query processing engine for data streams (demo)," in *Proceedings of the International Conference on Data Engineering, ICDE*, 2004.
27. J. Hershberger and S. Suri. "Adaptive sampling for geometric problems over data streams," in *Proceedings of the ACM Symposium on Principles of Database Systems, PODS*, 2004.
28. G.S. Iwerks, H. Samet, and K. 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.
29. C.S. Jensen, D. Lin, and B.C. Ooi. "Query and update efficient B+tree based indexing of moving objects," in *Proceedings of the International Conference on Very Large Data Bases, VLDB*, 2004.
30. J. Kang, J.F. Naughton, and S. Viglas. "Evaluating window joins over unbounded streams," in *Proceedings of the International Conference on Data Engineering, ICDE*, 2003.
31. D. Kwon, S. Lee, and S. Lee. "Indexing the current positions of moving objects using the lazy update R-tree," in *Mobile Data Management, MDM*, 2002.
32. I. Lazaridis, K. Porkaew, and S. Mehrotra. "Dynamic queries over mobile objects," in *Proceedings of the International Conference on Extending Database Technology, EDBT*, 2002.
33. M.-L. Lee, W. Hsu, C.S. Jensen, and K.L. Teo. "Supporting frequent updates in R-trees: A bottom-up approach," in *Proceedings of the International Conference on Very Large Data Bases, VLDB*, 2003.

34. S. Madden, M. Shah, J.M. Hellerstein, and V. Raman. "Continuously adaptive continuous queries over streams," in *Proceedings of the ACM International Conference on Management of Data, SIGMOD*, 2002.
35. M.F. Mokbel and W.G. Aref. "GPAC: Generic and progressive processing of mobile queries over mobile data," in *Mobile Data Management, MDM*, 2005.
36. M.F. Mokbel, W.G. Aref, S.E. Hambrusch, and S. Prabhakarx. "Towards scalable location-aware services: requirements and research issues," in *Proceedings of the ACM Symposium on Advances in Geographic Information Systems, ACM GIS*, 2003.
37. M.F. Mokbel, T.M. Ghanem, and W.G. Aref. "Spatio-temporal access methods," *IEEE Data Engineering Bulletin*, Vol. 26(2), 2003.
38. M.F. Mokbel, X. Xiong, and W.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.
39. M.F. Mokbel, X. Xiong, W.G. Aref, S. Hambrusch, S. Prabhakar, and M. 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.
40. M.F. Mokbel, X. Xiong, M.A. Hammad, and W.G. Aref. "Continuous query processing of spatio-temporal data streams in PLACE," in *Proceedings of the second workshop on Spatio-Temporal Database Management, STDBM*, 2004.
41. R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G.S. Manku, C. Olston, J. Rosenstein, and R. Varma. "Query processing, approximation, and resource management in a data stream management system," in *Proceedings of the International Conference on Innovative Data Systems Research, CIDR*, 2003.
42. T. Nadeem, S. Dashtinezhad, C. Liao, and L. Iftode. "TrafficView: A scalable traffic monitoring system," in *Mobile Data Management, MDM*, 2004.
43. D. Pfoser, C.S. Jensen, and Y. Theodoridis. "Novel approaches in query processing for moving object trajectories," in *Proceedings of the International Conference on Very Large Data Bases, VLDB*, 2000.
44. S. Prabhakar, Y. Xia, D.V. Kalashnikov, W.G. Aref, and S.E. Hambrusch. "Query indexing and velocity constrained indexing: Scalable techniques for continuous queries on moving objects," *IEEE Transactions on Computers*, Vol. 51(10), 2002.
45. B. Reinwald and H. Pirahesh. "SQL open heterogeneous data access," in *Proceedings of the ACM International Conference on Management of Data, SIGMOD*, 1998.
46. B. Reinwald, H. Pirahesh, G. Krishnamoorthy, G. Lapis, B.T. Tran, and S. Vora. "Heterogeneous query processing through SQL table functions," in *Proceedings of the International Conference on Data Engineering, ICDE*, 1999.
47. S. Saltenis and C.S. Jensen. "Indexing of moving objects for location-based services," in *Proceedings of the International Conference on Data Engineering, ICDE*, 2002.
48. S. Saltenis, C.S. Jensen, S.T. Leutenegger, and M.A. Lopez. "Indexing the positions of continuously moving objects," in *Proceedings of the ACM International Conference on Management of Data, SIGMOD*, 2000.
49. H. Samet. "The quadtree and related hierarchical data structures," *ACM Computing Surveys*, Vol. 16(2), 1984.
50. P. Seshadri, "Predator: A resource for database research," *SIGMOD Record*, Vol. 27(1):16–20, 1998.
51. Z. Song and N. Roussopoulos. "*k*-nearest neighbor search for moving query point," in *Proceedings of the International Symposium on Advances in Spatial and Temporal Databases, SSTD*, 2001.
52. U. Srivastava and J. Widom. "Memory-limited execution of windowed stream joins," in *Proceedings of the International Conference on Very Large Data Bases, VLDB*, 2004.
53. J. Sun, D. Papadias, Y. Tao, and B. 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.
54. G. Swedberg. "Ericsson's mobile location solution," *Ericsson Review*, 1999.
55. Y. Tao, D. Papadias, and Q. Shen. "Continuous nearest neighbor search," in *Proceedings of the International Conference on Very Large Data Bases, VLDB*, 2002.

56. Y. Tao, D. Papadias, and J. Sun. "The TPR*-tree: An optimized spatiotemporal access method for predictive queries," in *Proceedings of the International Conference on Very Large Data Bases, VLDB*, 2003.
57. Y. Tao, J. Sun, and D. Papadias. "Analysis of predictive spatio-temporal queries," *ACM Transactions on Database Systems, TODS*, Vol. 28(4), 2003.
58. Y. Tao, G. Kollios, J. Considine, F. Li, and D. Papadias. "Spatio-temporal aggregation using sketches," in *Proceedings of the International Conference on Data Engineering, ICDE*, 2004.
59. N. Tatbul, U. Cetintemel, S.B. Zdonik, M. Cherniack, and M. Stonebraker. "Load shedding in a data stream manager," in *Proceedings of the International Conference on Very Large Data Bases, VLDB*, 2003.
60. Y. Theodoridis, "Ten benchmark database queries for location-based services,'' *The Computer Journal*, Vol. 46(6):713–725, 2003.
61. P.A. Tucker, D. Maier, T. Sheard, and L. Fegaras, "Exploiting punctuation semantics in continuous data streams,'' *IEEE Transactions on Knowledge and Data Engineering, TKDE*, Vol. 15(3):555–568, 2003.
62. O. Wolfson, B. Xu, S. Chamberlain, and L. Jiang. "Moving objects databases: Issues and solutions," in *Proceedings of the International Conference on Scientific and Statistical Database Management, SSDBM*, 1998.
63. O. Wolfson, A.P. Sistla, B. Xu, J. Zhou, and S. Chamberlain. "DOMINO: Databases for moving objects tracking (demo)," in *Proceedings of the ACM International Conference on Management of Data, SIGMOD*, 1999.
64. X. Xiong, M.F. Mokbel, W.G. Aref, S. Hambrusch, and S. Prabhakar. "Scalable spatio-temporal continuous query processing for location-aware services," in *Proceedings of the International Conference on Scientific and Statistical Database Management, SSDBM*, 2004.
65. X. Xiong, M.F. Mokbel, and W.G. Aref. "SEA-CNN: Scalable processing of continuous *k*-nearest neighbor queries in spatio-temporal databases," in *Proceedings of the International Conference on Data Engineering, ICDE*, 2005.
66. Y. Yao and J. Gehrke. "The cougar approach to in-network query processing in sensor networks," *SIGMOD Record*, Vol. 31(3), 2002.
67. J. Zhang, M. Zhu, D. Papadias, Y. Tao, and D.L. Lee. "Location-based spatial queries," in *Proceedings of the ACM International Conference on Management of Data, SIGMOD*, 2003.
68. B. Zheng and D.L. Lee. "Semantic caching in location-dependent query processing," in *Proceedings of the International Symposium on Advances in Spatial and Temporal Databases, SSTD*, 2001.

Mohamed F. Mokbel received his Ph.D. degree from the Department of Computer Science, Purdue University in summer 2005. Starting from August 2005, he will be an Assistant Professor in the Department of Computer Science, University of Minnesota. His main research interests lie in the broad area of database systems. In particular, his work focuses in advancing the state of the art in the design and implementation of database engines to cope with the requirements of emerging applications (e.g., location-aware applications and sensor networks). In his recent Ph.D. thesis, he has introduced novel techniques to support efficient execution of large number of concurrent continuous spatio-temporal queries in both data stream management systems and relational database systems. For more information, please visit http://www.cs.purdue.edu/~mokbel.

Xiaopeng Xiong received the B.S. degree in computer science from the University of Science and Technology of China in 2001, and received the M.S. degree in computer science from Purdue University in 2004. He is a fourth year Ph.D. student in the Department of Computer Science at Purdue University. He is also a member of Indiana Center for Database Systems. His research interests are in database management, database access methods, spatio-temporal databases and streaming databases.

Walid G. Aref is an associate professor of computer science at Purdue. His research interests are in developing database technologies for emerging applications, e.g., spatial, multimedia, genomics, and sensor databases. He is also interested in indexing, data mining, and geographic information systems (GIS). Professor Aref's research has been supported by the NSF, Purdue Research Foundation, CERIAS, Panasonic, and Microsoft Corp. In 2001, he received the CAREER Award from the National Science Foundation. Professor Aref is in the editorial board of the VLDB Journal and is a member of the ACM and the IEEE.

Moustafa A. Hammad is an Assistant Professor at the Department of Computer Science, University of Calgary, Calgary, Alberta, Canada, since August 2004. He obtained his Ph.D. degree from the Department of Computer Sciences, Purdue University, West Lafayette, IN., USA in June 2004. He holds the Natural Science and Engineering Research Council of Canada (NSERC) Discovery Grant and the University of Calgary Starter Grant. His research interests lie in the area of Database Systems. Specifically, his work focuses on proposing, building, and evaluating database technologies for non-traditional data types such as stream, sensor, spatio-temporal, and multimedia data. He is a member of the ACM and the IEEE.