PLACE: A Scalable Location-aware Database Server for Spatio-temporal Data Streams

Mohamed F. Mokbel
Department of Computer Science and Engineering, University of Minnesota
Minneapolis, MN, 55455
mokbel@cs.umn.edu

Walid G. Aref
Department of Computer Science
Purdue University
West Lafayette, IN 47907
aref@cs.purdue.edu

Abstract

*In this paper, we overview the PLACE server (***P***ervasive* **L***ocation-***A***ware* **C***omputing* **E***nvironments); a scalable location-aware database server developed at Purdue University. The PLACE server extends data streaming management systems to support location-aware environments. Location-aware environments are characterized by the large number of continuous spatio-temporal queries and the infinite nature of spatio-temporal data streams. The PLACE server employs spatio-temporal query operators that support a wide variety of continuous spatio-temporal queries. In addition, the PLACE server is equipped with scalable operators that provide shared execution among multiple continuous spatio-temporal queries. To cope with intervals of high workload of data objects and/or continuous queries, the PLACE server utilizes object and query load shedding techniques to support a larger number of continuous queries with approximate answers.*

1 Introduction

The wide spread of *location-detection* devices (e.g., GPS-like devices, RFIDs, handheld devices, and cellular phones) results in *location-aware* environments where massive spatio-temporal data streams are continuously sent from these devices to database servers. Location-aware environments are characterized by the large numbers of continuously moving objects and moving queries (also known as spatio-temporal queries). Such environments call for new scalable database servers that deal with the continuous movement and frequent updates of both spatio-temporal objects and spatio-temporal queries.

In this paper, we overview the PLACE server (**P**ervasive **L**ocation-**A**ware **C**omputing **E**nvironments) [AHP03, MXA$^+$04b, MXHA04]; a scalable location-aware database server developed at Purdue University. The PLACE server combines the advanced technologies of spatio-temporal databases and data stream management systems to support efficient execution of a large number of continuous spatio-temporal queries over spatio-temporal data streams. The PLACE server is equipped with spatio-temporal pipelined query operators that interact with traditional query operators (e.g., join, distinct, and aggregates) to support a wide variety of continuous spatio-temporal queries. Furthermore, the PLACE server employs *incremental evaluation*, *shared execution*, and *load shedding* techniques to support large number of concurrent spatio-temporal queries.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

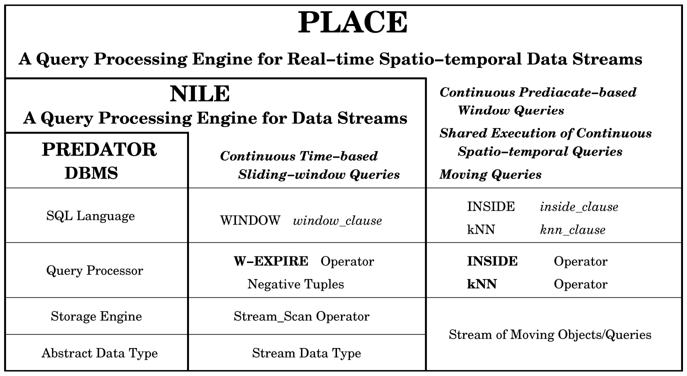

PLACE
A Query Processing Engine for Real–time Spatio–temporal Data Streams
NILE
A Query Processing Engine for Data Streams
PREDATOR
DBMS
Continuous Time–based Sliding–window Queries
Continuous Prediacate–based Window Queries
Shared Execution of Continuous Spatio–temporal Queries
Moving Queries
SQL Language
WINDOW window_clause
INSIDE inside_clause
kNN knn_clause
Query Processor
W–EXPIRE Operator
Negative Tuples
INSIDE Operator
kNN Operator
Storage Engine
Stream_Scan Operator
Abstract Data Type
Stream Data Type
Stream of Moving Objects/Queries

Figure 1: The PLACE Server.

As given in Figure 1, the PLACE server extends the NILE data stream management system [HMA$^+$04] and the PREDATOR database management system [Ses98] to support: (1) Continuous spatio-temporal queries over spatio-temporal data streams (i.e., streams of moving objects) [MXHA04]. (2) The concept of *predicate-based* window queries [GAE05]. (3) Incremental evaluation of continuous spatio-temporal queries [MXA04a]. Incremental evaluation is achieved through updating the query answer by *positive* and *negative* updates. A positive/negative update indicates that a certain object needs to be added to/removed from the query answer. (4) Scalable execution of a large number of continuous spatio-temporal queries [MA05b, MXA04a]. (5) Load shedding techniques for spatio-temporal data streams [MA05b].

The rest of this paper is organized as follows. Section 2 discusses the challenges of dealing with spatio-temporal streams. The spatio-temporal pipeline query operators are presented in Section 3. Sections 4 and 5 give a brief highlight of the *shared execution* and *load shedding* features of the PLACE server, respectively. Finally, Section 6 concludes the paper.

2 Spatio-temporal Data Streams

Although there are numerous research efforts that deal with data streams (e.g., see [GO03] for a survey), the spatial and temporal properties of data streams are addressed only recently in [CM03, HS04] to solve geometric problems (e.g., computing the convex hull) and in [EMA05, TPZL05] to develop spatio-temporal histograms. On the other side, research efforts in spatio-temporal databases (e.g., [CEP03, MGA03, PCC04]) rely mainly on indexing and/or storing the incoming data in disk storage, which is not suitable for the streaming environment. Up to the authors' knowledge, the PLACE server provides the first attempt to furnish query processors in data stream management systems to support continuous queries over spatio-temporal data streams.

2.1 Managing the Scarce Memory

With the infinite nature of spatio-temporal data streams, it becomes essential to develop in-memory algorithms and data structures to support the efficient execution of continuous queries. The PLACE server optimizes the scarce memory resource by keeping track of only those objects that are considered *"significant"*. A moving object P is considered *significant* if there is at least one outstanding continuous query Q that shows interest in P. Thus, once a new incoming data object P_{new} is received, we go through all the outstanding continuous queries to check if there is any query that is interested in P_{new}. If no query shows interest, we ignore the arrival of P_{new}. Moreover, due to the continuous movements of both objects and queries, we monitor the status of all

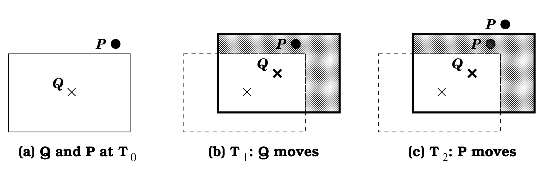


Figure 2: Uncertainty in moving queries.

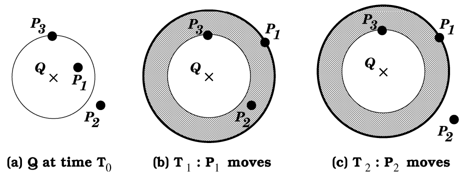


Figure 3: Uncertainty in static NN queries.

the stored objects. If a *significant* stored object becomes *insignificant* at any time, we immediately drop it from memory.

2.2 Uncertainty in Continuous Spatio-temporal Queries

Storing only *significant* objects may result in having *uncertainty* areas in continuous spatio-temporal queries. We define the *uncertainty* area of a query Q as the spatial area that may contain potential moving objects that satisfy Q, with Q not being aware of contents of this area. Uncertainty areas may result in erroneous query results in both moving and stationary contiguous spatio-temporal queries.

- **Moving queries.** Figure 2 gives example *uncertainty* areas that result from moving range queries. Figure 2a represents a snapshot at time T_0 where point P is outside the area of query Q. Thus, P is not physically stored in the database. At time T_1 (Figure 2b), Q is moved to cover a new spatial area. The shaded area in Q represents Qs *uncertainty* area. Although P is inside the new query region, P is not reported in the query answer, simply, because P is not stored in the database. At T_2 (Figure 2c), object P moves out of the query region. Thus, P is never reported at the query result, although it was physically inside the query region in the interval $[T_1, T_2]$.
- **Stationary queries.** Figure 3 gives an example *uncertainty* area in stationary k-nearest-neighbor queries ($k = 2$). At time T_0 (Figure 3a), the query Q has P_1 and P_3 as its answer. P_2 is outside the query spatial region, thus P_2 is not stored in the database. At T_1 (Figure 3b), P_1 moves far from Q. Since Q is aware of P_1 and P_3 only, we extend the spatial region of Q to include the new location of P_1. Thus, an *uncertainty* area is produced. Notice that Q is unaware of P_2 since P_2 is not stored in the database. At T_2 (Figure 3c), P_2 moves out of the new query region. Thus, P_2 never appears as an answer to Q, although P_2 should have been part of the answer in the time interval $[T_1, T_2]$.

2.3 Avoiding Uncertainty by Caching

In PLACE, we avoid *uncertainty* areas in continuous spatio-temporal queries using a *caching* technique. The main idea is to predict the uncertainty area of a continuous query Q and *cache* in-memory all moving objects that lie in Q's uncertainty area. Whenever an uncertainty area is produced, we probe the in-memory *cache* and produce the result immediately. A *conservative* approach for *caching* is to expand the query region in all directions with the maximum possible distance that a moving object can travel between any two consecutive updates. Such *conservative* approach completely avoids uncertainty areas where it is guaranteed that all objects in the uncertainty area are stored in the *cache*.

3 Spatio-temporal Operators

In the PLACE server we encapsulate query processing 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.
- **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.

Similarly, whenever a query reports movement, it classifies in-memory stored objects into four categories C_1 to C_4 as follows:

- $C_1 \subset Q.Answer$ and C_1 satisfies the new $Q.Region$. Such objects are not processed where they keep their status as part of the query answer.
- $C_2 \subset Q.Answer$, C_2 does not satisfy the query region. For each data object in C_2, we report a *negative* update.
- $C_3 \not\subset Q.Answer$ and C_3 satisfies the new $Q.Region$. For each data object in C_3, we report a *positive* update.
- $C_4 \not\subset Q.Answer$ and C_4 does not satisfy $Q.Region$. Such objects are not processed where they keep their status as they do not belong to the query answer.

3.1 Generic Pipelined Query Operators

PLACE utilizes a unified framework (e.g., see [MA05a]) to deal with continuous range queries as well as continuous k-nearest-neighbor queries. In addition, there is no distinction between stationary and moving queries where both of them are treated similarly. The main idea for dealing with moving queries is to treat data and queries similarly. Thus, data as well as queries have the ability to change their location and size over time. For k-nearest-neighbor queries, a kNN query is represented as a circular range query. The only difference is that the size of the query range may grow or shrink based on the movement of the query and objects of interest. Once the kNN query is registered in the PLACE server, the first incoming k objects are considered as the initial query answer. The radius of the circular region is determined by the distance from the query center to the current kth farthest neighbor. Then, the query execution continues as a regular range query, yet with a variable size. Whenever a newly coming object P lies inside the circular query region, P removes the kth farthest neighbor from the answer set (with a *negative* update) and adds itself to the answer set (with a *positive* update). The query circular region is *shrunk* to reflect the new kth neighbor. Similarly, if an object P, that is one of the k neighbors, updates its location to be outside the circular region, we expand the query circular region to reflect the fact that

P is considered the farthest kth neighbor. Notice that in case of expanding the query region, we do not output any updates.

3.2 SQL Syntax

As the PLACE server extends both PREDATOR [Ses98] and NILE [HMA$^+$04], we extend the SQL language provided by both systems to support spatio-temporal operators. 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. A continuous query is dropped form the system using the SQL `DROP QUERY` *query_name*. The *select_clause*, *from_clause*, and *where_clause* are the same as those in the PREDATOR [Ses98] database management statement. The *window_clause* is the same as that in the NILE [HMA$^+$04] stream query processor to support continuous sliding window queries [HFAE03]. 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. A rectangular range query can have one of the following two forms:

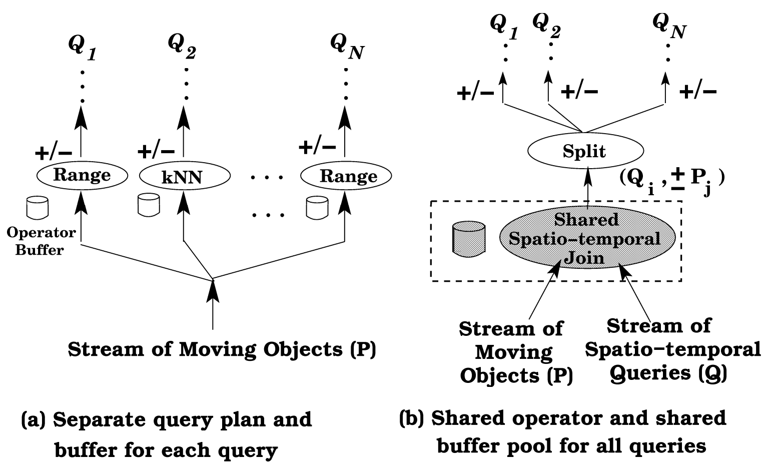
- Static range query (x_1, y_1, x_2, y_2), where (x_1, y_1) and (x_2, y_2) represent the top-left and bottom-right corners of the rectangular range query.
- Moving rectangular range query $('M', ID, xdist, ydist)$, where $'M'$ is a flag to indicate that the query is moving, ID is the identifier of the 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 continuous k-nearest-neighbor queries may have one of the following two forms:

- Static kNN query (k, x, y), where k is the number of 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 Shared Execution

In a typical spatio-temporal application (e.g., location-aware services), there are large numbers of concurrent spatio-temporal continuous queries. Dealing with each query as a separate entity would easily consume the system resources and degrade the system performance. Figure 4a gives the pipelined execution of N queries (Q_1 to Q_N) of various types with no sharing, i.e., each query is considered a separate entity. The input data stream goes through each spatio-temporal query operator separately. With each operator, we keep track of a separate buffer that contains all the objects that are needed by this query (e.g., objects that are inside the query



Q1
Q2
QN
+/−
Range
kNN
Range
Operator Buffer
Stream of Moving Objects (P)
(a) Separate query plan and buffer for each query
Split
(Qi, ± Pj)
Shared Spatio-temporal Join
Stream of Moving Objects (P)
Stream of Spatio-temporal Queries (Q)
(b) Shared operator and shared buffer pool for all queries

Figure 4: Overview of shared execution in the PLACE server.

region or its cache area). With a separate buffer for each single query, the memory can be exhausted with a small number of continuous queries. Figure 4b gives the pipelined execution of the same N queries as in Figure 4a, yet with the shared pipelined query operator [MA05b]. The problem of evaluating concurrent continuous queries is reduced to a spatio-temporal join between two streams; a stream of moving objects and a stream of continuous spatio-temporal queries. The *shared* spatio-temporal join operator has a shared buffer pool that is accessible by all continuous queries. The output of the *shared* query operator has the form $(Q_i, \pm P_j)$ that indicates the addition or removal of object P_j to/from query Q_i. The shared query operator is followed by a *split* operator that distributes the output either to the users or to the various query operators.

5 Load Shedding

Even with the shared execution paradigm in PLACE, the memory resource may be exhausted at intervals of unexpected massive numbers of queries and moving objects (e.g., during rush hours). To cope with such intervals, the PLACE server is equipped with a *self-tuning* approach that tunes the memory load to support a large number of concurrent queries, yet with an approximate answer. The main idea is to tune the definition of *significant* objects based on the current workload. By adapting the definition of *significant* objects, the memory load will be *shed* in two ways: (1) In-memory stored objects will be revisited for the new meaning of *significant* objects. If an *insignificant* object is found, it will be *shed* from memory. (2) Some of the newly input data will be *shed* at the input level.

Figure 5 gives the architecture of *self-tuning* in the PLACE server. Once the shared join operator incurs high resource consumption, e.g., the memory becomes almost full, the join operator triggers the execution of the *load shedding* procedure. The *load shedding* procedure may consult some statistics that are collected during the course of execution to decide on a new meaning of *significant* objects. While the shared join operator is running with the new definition of *significant* objects, it may send updates of the current memory load to the *load shedding* procedure. The load shedding procedure replies back by continuously adopting the notion of *significant* objects based on the continuously changing memory load. Finally, once the memory load returns to a stable state, the shared join operator retains the original meaning of *significant* objects and stops the execution of the *load shedding* procedure. Solid lines in Figure 5 indicate the mandatory steps that should be taken by any *load shedding* technique. Dashed lines indicate a set of operations that may or may not be employed based on the underlying *load shedding* technique.

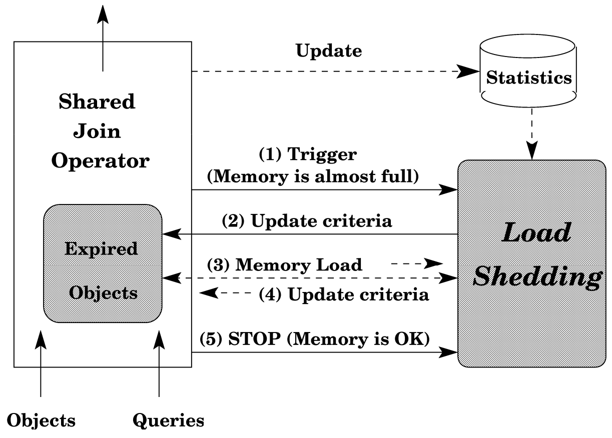


Figure 5: *Self-tuning* in the PLACE server

6 Conclusion

In this paper, we presented the PLACE (Pervasive Location-Aware Computing Environments) server; a database server for location-aware environments developed at Purdue University. The PLACE server combines the recent advances in spatio-temporal query processors and data stream management systems to provide a location-aware database server that efficiently execute large number of concurrent continuous spatio-temporal queries over spatio-temporal data streams. The PLACE server is realized by extending both the PREDATOR database management system and the NILE stream query processor to support the following main features: (1) New physical *spatio-temporal query operators* that can interact with traditional query operators in a large query plan. (2) *Incremental evaluation* through the concepts of positive and negative updates. (3) *Shared execution* of large number of concurrent continuous spatio-temporal queries. (4) *Load shedding* techniques to cope with time intervals of high workload of incoming data objects and/or queries. (5) Unified framework for a wide variety of continuous spatio-temporal queries.

Acknowledgments

This work was supported in part by the National Science Foundation under Grants IIS-0093116, IIS-0209120, and 0010044-CCR. The authors also would like to thank Hicham Elmongui, Thanaa Ghanem, Susanne Hambrusch, Moustafa Hammad, Sunil Prabhakar, and Xiaopeng Xiong for contributing in different stages in realizing the PLACE server.

References

[AHP03] Walid G. Aref, Susanne E. Hambrusch, and Sunil Prabhakar. Pervasive Location Aware Computing Environments (PLACE). http://www.cs.purdue.edu/place/, 2003.

[CEP03] V. Prasad Chakka, Adam Everspaugh, and Jignesh M. Patel. Indexing Large Trajectory Data Sets with SETI. In *Proc. of the Conf. on Innovative Data Systems Research, CIDR*, 2003.

[CM03] Graham Cormode and S. Muthukrishnan. Radial Histograms for Spatial Streams. Technical Report DIMACS TR: 2003-11, Rutgers University, 2003.

[EMA05] Hicham G. Elmongui, Mohamed F. Mokbel, and Walid G. Aref. Spatio-Temporal Histograms. In *SSTD*, 2005.

[GAE05] Thanaa M. Ghanem, Walid G. Aref, and Ahmed K. Elmagarmid. Exploiting Predicate-window Semantics over Data Streams. *SIGMOD Record. To Appear*, 2005.

[GO03] Lukasz Golab and M. Tamer Ozsu. Issues in data stream management. *SIGMOD Record*, 32(2):5–14, 2003.

[HFAE03] Moustafa A. Hammad, Michael J. Franklin, Walid G. Aref, and Ahmed K. Elmagarmid. Scheduling for shared window joins over data streams. In *VLDB*, 2003.

[HMA$^+$04] 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 (Demo). In *ICDE*, 2004.

[HS04] John Hershberger and Subhash Suri. Adaptive Sampling for Geometric Problems over Data Streams. In *PODS*, 2004.

[MA05a] Mohamed F. Mokbel and Walid G. Aref. GPAC: Generic and Progressive Processing of Mobile Queries over Mobile Data. In *MDM*, 2005.

[MA05b] Mohamed F. Mokbel and Walid G. Aref. SOLE: Scalable Online Execution of Continuous Queries on Spatio-temporal Data Streams. Technical Report TR CSD-05-016, Purdue University Department of Computer Sciences, July 2005.

[MGA03] Mohamed F. Mokbel, Thanaa M. Ghanem, and Walid G. Aref. Spatio-temporal Access Methods. *IEEE Data Engineering Bulletin*, 26(2), 2003.

[MXA04a] Mohamed F. Mokbel, Xiaopeng Xiong, and Walid G. Aref. SINA: Scalable Incremental Processing of Continuous Queries in Spatio-temporal Databases. In *SIGMOD*, 2004.

[MXA$^+$04b] 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 *VLDB*, 2004.

[MXHA04] Mohamed F. Mokbel, Xiaopeng Xiong, Moustafa A. Hammad, and Walid 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.

[PCC04] Jignesh M. Patel, Yun Chen, and V. Prasad Chakka. STRIPES: An Efficient Index for Predicted Trajectories. In *SIGMOD*, 2004.

[Ses98] P. Seshadri. Predator: A Resource for Database Research. *SIGMOD Record*, 27(1):16–20, 1998.

[TPZL05] Yufei Tao, Dimitris Papadias, Jian Zhai, and Qing Li. Venn Sampling: A Novel Prediction Technique for Moving Objects. In *ICDE*, 2005.