SINA-Scalable Incremental Processing of Continuous Queries in Spatio-Temporal Databases
Mohamed F. Mokbel, Xiaopeng Xiong, Walid G. Aref

Revised Narrative written by:
Nilu Thakur, Prasad Sriram,
CSCI 8701

Introduction:
Spatio-temporal databases consist of a large number of continuously arriving queries from a large number of moving objects. The changing locations and times of objects add a temporal dimension to spatio-temporal queries. Some of the features of spatio-temporal queries include persistent read-only data and a large number of historical trajectories. Also, these queries have the ability to query along both spatial and temporal dimensions.

Any algorithm or technique that handles continuous query processing should be able to produce fast, complete and accurate query results. Also, the limited network bandwidth and the large number of continuous queries make it desirable for the algorithm to return incremental results rather than send full results. Above all, the algorithm should be scalable to handle any number of queries and objects. This problem is especially important in today’s world where we have a large number of location-based/location-aware devices like PDA, laptops and other mobile devices continuously sending queries to the database server in order to obtain responses. In this paper, the authors introduce a novel technique to evaluate continuously and concurrently arriving spatio-temporal queries.

Motivation:
Example of these application include many services which rely on large spatio-temporal datasets like:
- Location aware services
- Traffic Monitoring
- Enhanced 911 services

Few Illustrations:
**Moving query on stationary object:** Find the nearest gas station within 1 mile of moving red car.

![Diagram](image_url)

*Figure 1: Circled dot represent moving cars and Gas stations are stationary objects*
**Moving query on moving object:** Continuously find all Police cars within 3 miles of the moving red car.

![Image](image1.png)

*Figure 2: Police cars are in blue and red car is moving along the dotted path*

**Stationary Query on moving object:** Continuously find all vehicles within 1 mile of my house.

![Image](image2.png)

*Figure 3: All colored circle represent different moving cars.*

**Problem Statement:**

**Input:**
A large number of *mobile and stationary* objects and *continuous* spatio-temporal queries

**Output:**
Find complete and accurate query evaluation results in minimum response time.

**Objective:**
Continuous queries require continuous evaluation.
Scalability in terms of the number of queries.
Update only those results in the query that changed from the previous answer.

**Constraints:**
Send response to queries without any delay as a delayed response might become obsolete, outdated or invalid due to the highly dynamic nature of queries and objects in a spatio-temporal environment.
Because of the limited network bandwidth, the continuous query processing algorithm should use the resources frugally as a large number of queries arrive continuously.
Contributions:
Some of the main contributions of the authors in this paper are
- The use of shared execution paradigm
- Incremental evaluation of concurrently executing spatio-temporal queries by sending only results that changed from the previous answer
- The proof of completeness, uniqueness and progressiveness of their algorithm
- Experimental results through a comparative analysis of their algorithm (SINA) with some other popular techniques.

While there have been previous techniques to process continuous spatio-temporal queries, there has always been a trade-off between various factors like scalability, completeness of the results and performance. This paper describes an algorithm that simultaneously performs well in all these factors.

Strengths:
Some of the strengths of the algorithm mentioned in this paper are
- It doesn’t assume that clients which submit queries have computational capabilities
- It conserves network bandwidth by sending only incremental results
- It takes into account the mutability of both objects and queries
- Above all, the algorithm is scalable to handle multiple and concurrently executing spatio-temporal queries.

Proposed Solution:
One of the highlights of this paper is the use of shared execution technique together with incremental evaluation where a spatial join is done between queries and objects. Its incremental nature as opposed to some other techniques that use FUR tree, Q-Index and RSJ, makes the algorithm more scalable to handle a large number of concurrent queries. An incremental algorithm is the one, which returns just the results of the query, which are different from the previous response. It sends incremental results to these queries in the form of two kinds of updates, positive update and a negative update. A positive update means that a particular objects needs to be added while a negative update means that a particular object needs to be removed from the result part of the query. For example (Q₃,+p₁) means that the object or point p₁ needs to be added from the result of Query Q₃ while (Q₄,-p₅) means that the object or point p₅ needs to be removed from the result of query Q₄.

SINA’ Key Concepts:

Key Concepts 1: Hashing
Hashing is the first phase of incremental evaluation. It primarily creates two hash tables, each with N buckets in memory for moving objects and moving queries respectively. Apart from this, an in-memory query table is also maintained to keep track of the upper left and lower right corners of the query region.
Figure 4: Hashing [1]

**Hashing a moving object:** Each moving object is hashed on its x and y co-ordinates. This hash value is then probed with all the matching queries from the query hash table and stored in their matching object hash buckets. All the matching results are then updated as positive updates to an in-memory table called Updated_Answer table.

**Hashing a moving query:** Each moving query tuple is hashed on its lower and upper x and y co-ordinates. This hash value is then probed with all the matching objects that intersect with this query. As a result, all the matching results are updated as positive updates to the in-memory Updated_Answer table. Also, the query tuple ‘t’ is stored in all the hash buckets corresponding to the grid cells lying inside its query region.

Key concepts 2: Invalidation

Figure 5: Invalidation [1]

Invalidation phase starts when the memory gets full or after a certain specified time. The working of the invalidation phase is based on partitioning the two-dimensional space of moving objects and moving queries into N*N disk based grid cells. That means all the objects and queries are mapped to one or more disk based grid cells.

In this phase, completely full or non-empty buckets of the moved objects and moved queries are flushed in to the corresponding grid cells of the disk and then for each moving
Invalidating a moving object: The object is mapped to a grid cell and if that grid cell already has the entry for that object, then that means the object has not crossed a cell boundary. So, only the timestamp and query list id modified for that object.

If the object is mapped to a grid cell which did not have the any entry for that object then a new entry is inserted with current timestamp and query list obtained from update answer table. Then auxiliary structure of object index is used to get the old entry for the object. And for all the queries in the old entry list, negative updates are reported to the update answer table and finally the old entry is deleted.

Invalidating a moving query: For each query if it is mapped to a new grid cell then completely new entry is entered in the format of (QID, query region, timestamp, object list). Then in-memory query table is compared with the in-disk query index. All the cells (S_k) that were part of old region of query but are no more in the query region are found. Then for each object that was part of query answer in each grid cell of S_k set of query are sent negative updates. And finally the old entry of moving query is deleted.

Key Concepts 3: Joining
In this phase, for each grid cell, the in-memory objects are joined with the in-disk queries while the in-memory queries are joined with the in-disk objects. So for example, if both the object and query had arrived at the same time, this would have been taken care of in the hashing phase. The invalidation is mainly responsible or just cancelling the positive updates through negative updates. But, there might be a situation when the moving object had arrived much before the moving query inside which it is present. As there will be not any corresponding query hash buckets, the object will not be a part of the result. It is the joining phase that takes care of this situation by joining the in-disk objects/queries and in-memory queries/objects.

The use of spatial join on moving objects and moving queries together with incremental evaluation precludes the need of spatio-temporal indices like R* Tree, TPR-Tree, TPR* Tree. Spatio-temporal indices are special purpose indexing structures developed to support predictive queries by focusing on indexing the future position of the object. As SINA does not involve any indexing, one might think that it can’t predict future queries. Predictive spatio-temporal queries are queries which try to infer the objects that will be covered by the region of the query in a future time interval. SINA clearly can be extended to support such predictive query processing.
Validation Methodology:
In this paper, the authors generate synthetic data by using Network-Based Object Generator and compare their result with other algorithms that use various indexing techniques like FUR tree, Q-Index and RSJ. For comparison they consider various metrics like size of the results returned, performance in terms of CPU time, I/O time. The use of synthetic data for experiments is very useful in situations where it is very difficult to have datasets containing all the required problem features (dimensions). The use of Network-Based Object Generator to get synthetic data looks convincing and their comparison clearly shows that their technique is much better than the existing techniques. However, unless one conducts real life experiments where the conditions might be different, we really can’t be 100% sure about the results obtained from synthetic data.

Assumptions:
The paper makes very few assumptions and the assumptions look reasonable, considering that they arise from the limitations of the previous work in the same area. Assumptions can be listed as follows:

- No computational capabilities on the client side.
- No storage capabilities on the client side.
- No velocity Assumptions.
- Optimal value of time Interval assumed to be 10 seconds.

Both the first and second assumptions are fair considering that many times client uses cheap, low battery and passive devices that do not have computational or storage capabilities. About the assumptions for optimal value of time Interval, authors did not discuss how they came up with this value and we do feel that this value can either be determined by performing large number of experiments or by gradually learning the value through some statistics.

How will I rewrite this paper today?
Some improvements:
Also, if we were to rewrite this paper today, we would preserve most of the parts. But, based on our observations, we will develop the algorithm by taking into account the following points.

- Object histories
- Optimal value of Time interval

Improvements for Object histories:
For example consider the following query ‘Q’

\[ Q: \text{Given a huge history of a moving car, find all objects that were within a radius of 1 mile.} \]

These kinds of queries occur very naturally in applications where a large amount of history or past data is collected about objects and queries are posted to understand the relationship or interaction of objects over that particular period of history.

Though this algorithm is developed to handle continuously arriving queries involving moving objects, it is not quite capable of handling such moving objects with huge
histories which are usually gigabytes of data. Although, it is beyond the scope and objective of this paper, we would take this factor into account and make it capable of handling such moving objects with large histories.

**Improvements for Optimal value of Time interval:**
As a part of their algorithm, the paper talks about sending updates to queries every $T$ time units. In their experiments, they use a value of $T = 10$ seconds. But they do not discuss about this value. In our opinion, the value of ‘$T$’ also plays an important role in the query evaluation process. If ‘$T$’ value were set too high, then the updates would be sent much after the queries were issued thus leading to a situation where the results sent are no longer valid as objects and queries would have moved their locations. If ‘$T$’ were set too low, then updates would be sent frequently than necessary, thus leading to an increased network bandwidth which is undesirable in this environment where there is a large number of concurrently executing queries.

It is surprising to see why the authors haven’t discussed about this value. In our opinion, deciding the value of ‘$T$’ would involve one or more of the following techniques

- Set a value of ‘$T$’ based on large number of experiments
- The ‘$T$’ value can be set to an initial value and the value can be learned gradually after getting some kind of statistics on how much the results sent were valid in the past.

In a nutshell, the paper describes a scalable and continuous query processing algorithm that employs shared execution and incremental evaluation techniques. The comprehensive experiments that the authors conduct clearly emphasize the fact that SINA is better than other R-Tree based techniques.

**References:**


