

# SINA: Scalable Incremental Processing of Continuous Queries in Spatiotemporal Databases

Mohamed F. Mokbel, Xiaopeng Xiong, Walid G. Aref

October 1, 2007

Department of Computer Sciences, Purdue University, West Lafayette, IN 47907-1398

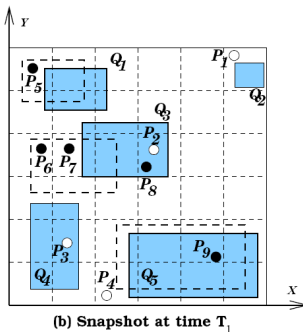
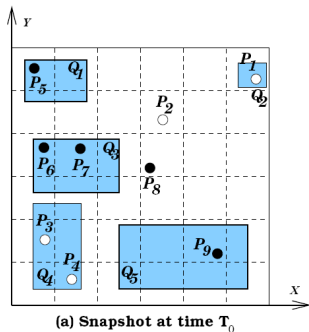
{fmokbel,xxiong,arefg}@cs.purdue.edu

Presented by: Kristian Torp

- 1 Motivation
- 2 Incremental Evaluation Algorithm
  - Hashing
  - Invalidation
  - Joining
- 3 Experimental Results
- 4 Evaluation

- 1 Motivation
- 2 Incremental Evaluation Algorithm
  - Hashing
  - Invalidation
  - Joining
- 3 Experimental Results
- 4 Evaluation

# Example



Positive Update

- $(Q_3, +p_2)$
- $(Q_3, +p_8)$

Negative Update

- $(Q_1, -p_5)$
- $(Q_2, -p_1)$
- $(Q_3, -p_6)$
- $(Q_3, -p_7)$
- $(Q_4, -p_4)$

## Characteristics

- Large number of mobile objects and mobile queries
- Queries are continuous by nature
- Delayed results are obsolete

# Design Criteria

- Scalable
- Incremental computation

# Query Types I

$$\left\{ \begin{array}{l} \text{stationary query} \\ \text{moving query} \end{array} \right\} \times \left\{ \begin{array}{l} \text{stationary object} \\ \text{moving object} \end{array} \right\}$$

## Example (Stationary queries on moving objects)

Continuously report the cars that are within 3 miles of my home

## Example (Moving queries on stationary objects)

Continuously report all gas stations that are within 3 miles of my location

## Example (Moving queries on moving objects)

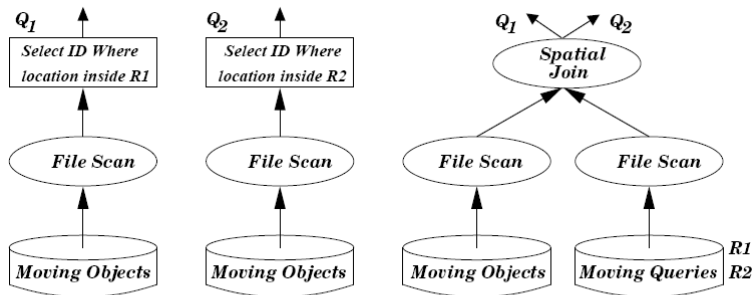
Continuously report all police cars that within 3 miles of my car location

## Query Types II

- Range queries (the focus of the paper)
- Nearest neighbor queries
- Aggregation queries
- Predictive queries



# Shared Execution



Store spatial extend of

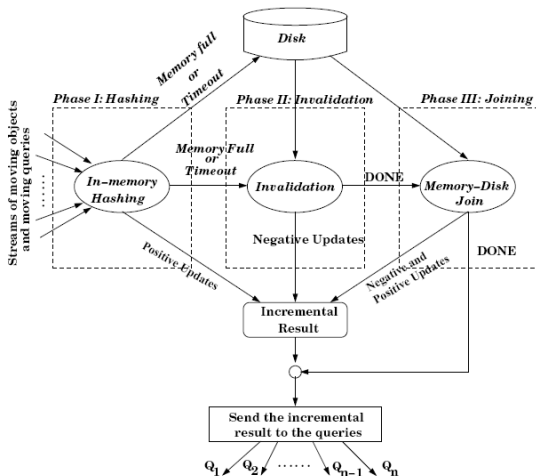
- Moving object
- Moving queries

- 1 Motivation
- 2 Incremental Evaluation Algorithm
  - Hashing
  - Invalidation
  - Joining
- 3 Experimental Results
- 4 Evaluation

# Algorithm Phases

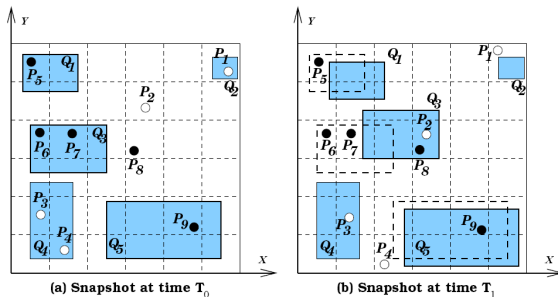
- Hash phase
  - Find positive updates
  - Main memory hash join
- Invalidation Phase
  - Find negative updates
- Joining Phase
  - Find additional positive and negative update
  - Merge positive and negative update
  - Ship changes to clients
  - Clear main-memory data structures

# Overview State Diagram



- 1 Motivation
- 2 Incremental Evaluation Algorithm
  - Hashing
  - Invalidation
  - Joining
- 3 Experimental Results
- 4 Evaluation

# Example



Updated points

- $p_1, p_2, p_3, p_4$

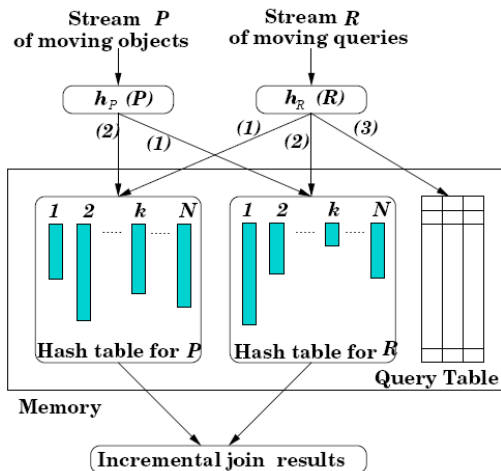
Updated queries

- $Q_1, Q_3, Q_5$

Positive Updates after join

- $(Q_3, +p_2)$

# Data Structures



**Procedure** HashingPhase(tuple  $t$ , source ( $P/ R$ ))

**Begin**

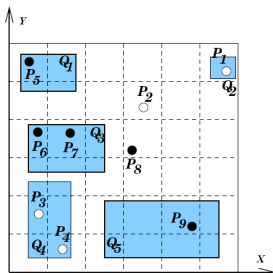
1. *If there is not enough memory to accommodate  $t$ , start the InvalidationPhase(), return*
2. *If (source==P) //Moving object*
  - (a)  $k =$  the hash value  $h_P(t)$  of tuple  $t$ .
  - (b)  $S_q =$  Set of queries from joining  $t$  with queries in  $R_k$
  - (c) For each  $Q \in S_q$ , add  $(Q, +t)$  to Updated\_Answer
  - (d) Store  $t$  in Bucket  $P_k$
  - (e) **return**
3.  $S_k =$  Set of buckets result from hash function  $h_R(t)$
4. For each bucket  $k \in S_k$ 
  - (a)  $S_o =$  Set of objects from joining  $t$  with objects in  $P_k$
  - (b) For each  $O \in S_o$ , add  $(t, +O)$  to Updated\_Answer
  - (c) Store a clipped part of  $t$  in Bucket  $R_k$
5. Store  $t$  in the query table

**End.**

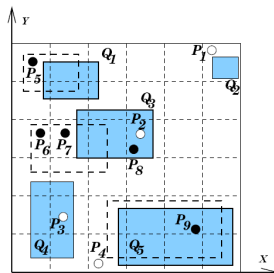


- 1 Motivation
- 2 Incremental Evaluation Algorithm
  - Hashing
  - **Invalidation**
  - Joining
- 3 Experimental Results
- 4 Evaluation

# Example



(a) Snapshot at time  $T_0$



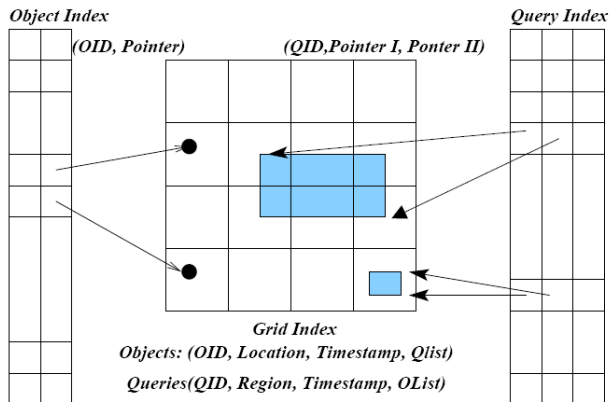
(b) Snapshot at time  $T_1$

- $p_1$  does not cross cell boundary
- $p_2$  not involved in query at  $T_0$
- $p_3$  crosses cell boundary
- $p_4$  is outside  $Q_4$
- $Q_1$  and  $Q_5$  no result
- $Q_2$  and  $Q_4$  do not move
- $Q_3$  leaves cell with  $p_6$  but not cell with  $p_7$

## Negative Updates

- $(Q_4, -p_3)$
- $(Q_4, -p_4)$
- $(Q_3, -p_6)$

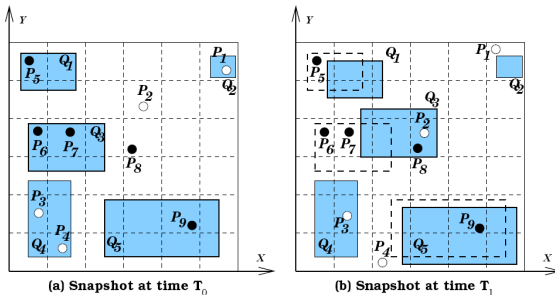
# Data Structures



- Grid can easily be changed to handle skew data

- 1 Motivation
- 2 Incremental Evaluation Algorithm
  - Hashing
  - Invalidation
  - **Joining**
- 3 Experimental Results
- 4 Evaluation

# Joining



Hash join moving objects with stationary queries in grid

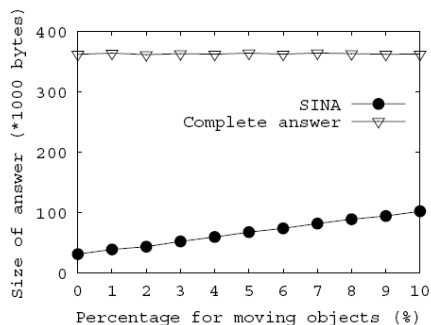
- $(Q_2, -p_1)$
- $(Q_4, +p_3)$

Hash join moving queries with stationary objects in grid

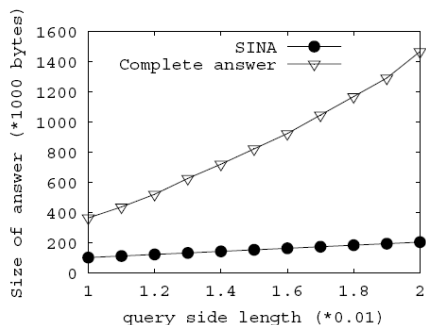
- $(Q_1, -p_5)$
- $(Q_3, -p_7)$
- $(Q_3, +p_8)$

- 1 Motivation
- 2 Incremental Evaluation Algorithm
  - Hashing
  - Invalidation
  - Joining
- 3 Experimental Results
- 4 Evaluation

# Incremental Computation

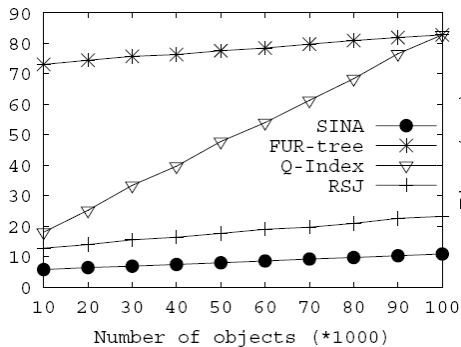


(a) Moving objects (%)

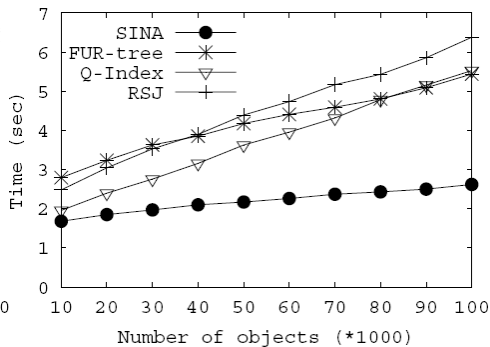


(b) Query size

# Scalability



(a) I/O



(b) CPU Time



- 1 Motivation
- 2 Incremental Evaluation Algorithm
  - Hashing
  - Invalidation
  - Joining
- 3 Experimental Results
- 4 Evaluation

# Good Points

- Good description of the characteristics of spatio-temporal apps
- Good overview of related work, like Table 1
- Contributions of the paper are clearly outlined
- Nice that the complicated Section 4 is split into data structures, algorithms, example, and discussion
- Nice running example that is used to explain each step in the core algorithm
- Good experimental validation of incremental and scalability claims
- Complete paper: Clear idea, algorithms, proof, and experiments

- Section 3 “Shared Execution” is at a very high level of abstraction
- The claim shared execution speeds up query processing could be better supported in the experimentation section
- The refreshment time  $T$  is set to 10 seconds. This is a “magic” value
- The discussions in Section 4 could be moved to the end