QU-Trade
“Workload-Aware Indexing of Continuously Moving Objects”

Kostas Tzoumas
Man Lung Yiu, Christian S. Jensen

daisy.aau.dk
Motivation

• The mobile Internet will be “bigger” than the conventional Internet.

• Mobile Internet characteristics
   Powerful mobile devices, e.g., mobile phones, navigation systems
   Positioning via GPS, Galileo, Wi-Fi
   Users are online and on the move.

• We need scalable data management support for location-based services.

• Moving object indexing may become an update bottleneck.

• We need methods that enable indexing techniques to adapt to frequently changing update and query workloads and that are capable of exploiting workload skew.
Query-Update Trade-Off Space

QU-Space

Query performance

Update performance
R-trees and TPR-trees have high query performance, but low update performance.

The naive solution (no spatial index) has high update performance, but low query performance.
An index structure represents a fixed compromise between update and query performance.

- **R/TPR-trees**
- **B^x-trees**
- **Grids**
- **No index**
We render an indexing technique adaptive.

The query and update performance adapt to the incoming workload.

QU-Trade = “Query-Update Trade”
Why Adaptive?

- Update rate per day of a real workload (20 cars driving in Copenhagen)
- Varies significantly (at least by one order of magnitude between days)
- Is an update-efficient or a query-efficient index better for this workload?
- Neither: We want the index to be update-efficient for the part of the workload with high update rate and query-efficient for the rest.
Spatial Skew

- QU-Trade exploits spatial skew.
- When updates and queries are concentrated in small areas, the performance of QU-Trade becomes better in all cases.
Why Spatial Skew?

- Spatial distribution of a real update workload (20 cars driving in Copenhagen)
- Substantial spatial skew
- Evidence from a large scale study advocates that this is an inherent characteristic of human motion [Nature08].
- QU-Trade is designed to take advantage of spatial skew.
Outline

- Motivation
  - The query-update tradeoff space and the need for adaptivity
  - Opportunities to exploit spatial skew

- QU-Trade
  - Architecture
  - Object representation
  - Update and query processing

- Growing and shrinking heuristics

- Experimental results
  - Workload-awareness
  - Transient behavior

- Conclusions and future work
QU-Trade, architecture

A server side solution

Goal: server side performance

A layer between the system interface and the index and DB

Update

Query

QU-Trade

DB

R/TPR-tree

A server side solution

Goal: server side performance

A layer between the system interface and the index and DB

Update

Query
Users update their positions. Any protocol can be used.

Users submit range and NN queries. Easy to extend to other types. We consider range queries.
A moving object is hashed on its oid.

A moving object is hashed on its oid.

Any index that supports bounding rectangles can be used.
• Representation of an object
  ■ The object identifier \( oid \)
  ■ The last reported position \( r \)
  ■ A window \( (c, W) \)

• The database stores the position \( r \) and window \( (c, W) \) of each object, for correctness.

• The index stores only the window \( (c, W) \).
QU-Trade, update

When an object issues an update, its new position is recorded in DB.

update(oid,r,r')

DB

(oid,r,c,W) = retrieve(oid)
delete(oid)
insert(oid,r')

The object window is retrieved and checked.
If the new position is inside the window, we are done.

(update(oid, r, r'))
QU-Trade, update

\[(c', W') = \text{grow}(c, W, r')\]

Otherwise center the window at the new position and expand it. Update the index and the DB with the new window (\(c', W')\).

\[\text{delete}(\text{oid}, (c, W))\]
\[\text{insert}(\text{oid}, (c', W'))\]
When we receive a range query, we forward it to the index as an intersection query.

Correctness: If the query does not intersect a window, the object is not in the query answer.
QU-Trade, query

The index returns a candidate set of real answers and false positives.

Object 1 is a true answer, and object 2 is a false positive.

The diagram illustrates the QU-Trade query process. The DB (database) returns a candidate set of real answers and false positives. The R/TPR-tree is used to filter the results. The query range (Q) is intersected with the database to find relevant objects. The objects (oid, c, W) are then evaluated to determine if they are true answers or false positives.
QU-Trade, query

If an object is a false positive, its window is re-centered and shrunk.

The index and DB are updated with the new window.
Intuition

• When an object updates often, its window will grow larger and larger.
• This will save us from updating the index, but deteriorates its (query performance) quality.
• When an object is “hit” by a query, its windows is shrunk.
• This improves query performance.
• **Adaptation:** The continuous competition of growing and shrinking will lead to a good window allocation, minimizing the combined update and query cost.
• **Exploitation of skew:** Objects in frequently queried regions will have small windows. Objects in non-queried regions that update often are allowed to have large windows. We expect the best performance for spatially skewed workloads.
Outline

• Motivation
  ■ The query-update tradeoff space and the need for adaptivity
  ■ Opportunities to exploit spatial skew

• QU-Trade
  ■ Architecture
  ■ Object representation
  ■ Update and query processing

• Growing and shrinking heuristics

• Experimental results
  ■ Workload-awareness
  ■ Transient behavior

• Conclusions and future work
Growing Heuristics

- Square windows only for good R-tree packing

A. No grow
- \( \text{grow}(c, W, r') = (r', W) \)

B. Add a system-wide value
- \( \text{grow}(c, W, r') = (r', \min(W + \text{thr}_g, W_{\text{max}})) \)

C. Motion adaptive
- \( \text{grow}(c, W, r') = (r_{cm}, W + \sqrt{R_g}) \)
  - \( r_{cm} \) is the (weighted) mean position so far
  - \( R_g \) is the (weighted) deviation of subsequent updates (radius of gyration)
Shrinking Heuristics

I. No shrink
   \[ shrink(c, W, r, Q) = (r, W) \]

II. “Reset”
   \[ shrink(c, W, r, Q) = (r, W_{\text{min}}) \]

III. Subtract a system-wide value
   \[ shrink(c, W, r, Q) = (r, \max(W - \text{thr}_s, W_{\text{min}})) \]

IV. “Just enough”
   \[ shrink(c, W, r, Q) = (r, \text{mindist}(r, Q)) \]
Outline

- Motivation
  - The query-update tradeoff space and the need for adaptivity
  - Opportunities to exploit spatial skew
- QU-Trade
  - Architecture
  - Object representation
  - Update and query processing
- Growing and shrinking heuristics
- Experimental results
  - Workload-awareness
  - Transient behavior
- Conclusions and future work
Experimental Setup

• Three variants of QU-Trade
  • Aggressive QU-Trade (B + II): Best performance
    - \( \text{grow}(c,W,r') = (r', \min(W+20\text{thr},200\text{thr})) \)
    - \( \text{shrink}(c,W,r,Q) = (r,0.1\text{thr}) \)
  • Conservative QU-Trade (B + III): Too conservative
    - \( \text{grow}(c,W,r') = (r', \min(W+2\text{thr},20\text{thr})) \)
    - \( \text{shrink}(c,W,r,Q) = (r,\max(W-2\text{thr},\text{thr})) \)
  • Adaptive QU-Trade (C + IV): 2\text{nd} best performance, no parameters
    - \( \text{grow}(c,W,r') = (r_{cm}, W+\sqrt{R_g}), \alpha=\gamma=0.9 \)
    - \( \text{shrink}(c,W,r,Q) = (r,\text{mindist}(r,Q)) \)
Experimental Setup

• Three variants of QU-Trade
  • Aggressive QU-Trade (B + II): Best performance
    - \( \text{grow}(c, W, r') = (r', \min(W + 20\text{thr}, 200\text{thr})) \)
    - \( \text{shrink}(c, W, r, Q) = (r, 0.1\text{thr}) \)
  • Conservative QU-Trade (B + III): Too conservative
    - \( \text{grow}(c, W, r') = (r', \min(W + 2\text{thr}, 20\text{thr})) \)
    - \( \text{shrink}(c, W, r, Q) = (r, \max(W - 2\text{thr}, \text{thr})) \)
  • Adaptive QU-Trade (C + IV): 2\text{nd} best performance, no parameters
    - \( \text{grow}(c, W, r') = (r_{cm}, W + \sqrt{R_g}), \alpha = \gamma = 0.9 \)
    - \( \text{shrink}(c, W, r, Q) = (r, \text{mindist}(r, Q)) \)
Two Experiments

• **Workload-awareness (WA):** The query and update cost of QU-Trade are a function of the workload's $r_U/r_Q$. Together, they always minimize the total cost.

• **Transient behavior (TB):** When $r_U/r_Q$ changes rapidly over time, QU-Trade adapts to these changes fast.
Update performance same as the R-tree for query-intensive workloads.

Very good update performance in update-intensive workloads.

Update cost is not fixed but a function of $r_u/r_Q$. 
WA: Query Performance

Query performance same as the R-tree for query-intensive workloads.

Becomes worse for update intensive workloads, but always within sensible limits.

Query cost is not fixed but a function of $r_u/r_Q$.
As a result, the overall performance is the best both for update-intensive and query-intensive workloads.
Transient Behavior

A near-constant update rate of 5000 updates/time unit.

Query rate 20000 queries/time unit at times [4,5], zero otherwise.
TB: Measuring I/O Rate

How many I/Os per operation per time unit are needed to process the workload. Smaller is better.
TB: Update-only Workload

What is the update overhead before the queries come?
TB: Update-only Workload

- R-tree has an update overhead of 10 I/Os per time unit.
- Grid has an update overhead of 3 I/Os per time unit.
- QU-Trade has a near-zero overhead (with oscillations).
How fast can the windows shrink when the queries come?
The R-tree is the “optimal” target.

After an initial oscillation, QU-Trade shrinks the windows pretty fast, achieving performance close to the R-tree.

The grid has the worst query performance.
TB: After the query burst

How fast can the windows grow again?
TB: After the query burst

Updates from objects that were not “hit” by the queries are fast, since the windows did not shrink.

Updates from objects that were “hit” by the queries have to grow the windows back.
Also in the paper

• QU-Trade with a TPR-tree
• Cost model: Even in the worst case (uniformity assumptions), the window mechanism is beneficial.
• Additional experiments
  - QU-Trade exploits spatial skew: Its performance becomes increasingly better when we gradually introduce skew to the workload.
  - Similar tradeoff exists in a main memory setting.
• Related work
  - Approximate caching (Olston et al. SIGMOD 01)
  - Online physical desing tuning (Bruno & Chaudhuri ICDE 07)
  - AGILE indexing (Dittrich et al. SIGMOD 05)
  - Oracle proposal is a special case of QU-Trade with no growing or shrinking (Kothuri et al. TIME 08).
  - Continuous spatial queries
Outline

• Motivation
  - The query-update tradeoff space and the need for adaptivity
  - Opportunities to exploit spatial skew

• QU-Trade
  - Architecture
  - Object representation
  - Update and query processing

• Growing and shrinking heuristics

• Experimental results
  - Workload-awareness
  - Transient behavior

• Conclusions and future work
Conclusions and Future Work

• Workload-awareness
  - Traditional indexes are suitable for a specific $r_U/r_Q$.
  - QU-Trade is best for the whole $r_U/r_Q$ spectrum.

• Exploitation of spatial skew
  - Indexes based on uniform space partitioning suffer from spatial skew.
  - QU-Trade is explicitly designed to benefit from spatial skew.

• Fast adaptation in the presence of temporal skew

• Future work
  - A principled, cost-based approach for growing and shrinking
Thank you!

Questions?