Supporting Frequent Updates in R-Trees: A Bottom-Up Approach

Christian S. Jensen

Aalborg University, Denmark

Mong Li Lee   Wynne Hsu   Bin Cui   Keng Lik Teo

National University of Singapore, Singapore

VLDB 2003

presented by

Simonas Šaltenis
Motivation

• New data management applications monitor continuous processes.
  • Tracking 2D moving objects
• Updates are frequent.
• Updates are likely to exhibit locality.

• Existing R-tree updates work in a top-down manner, performing two index traversals.
• Particularly the delete operation is expensive.
  • Traverses several partial or full paths from the root to the leaf level

• Key idea: do localized updates that consider less placements of updated values.
Outline

• Motivation
• Background – the R-tree
• Generalized bottom-up update
  • Data structure
  • Algorithms
  • Optimizations, tuning parameters
• Related work
• Performance study
• Strong and weak points
• Conclusion
R-tree updates

- An update in the R-tree is a pair of operations:
  - Delete($obj_id$, ($x_{old}$, $y_{old}$))
  - Insert ($obj_id$, ($x_{new}$, $y_{new}$))

- Insert:
  - Traverse one path down the tree, at each node using a heuristic choice of a subtree
  - Traverse up the tree as high as necessary propagating splits and/or adjustments of MBRs

- Delete:
  - Perform a query ($x_{old}$, $y_{old}$) to find the point
    - Potentially **several** paths down the tree are traversed!
  - Traverse up the tree as high as necessary propagating adjustments of MBRs

- Four tree traversals in total!
Outline

- Motivation
- Background – the R-tree
  - Generalized bottom-up update
    - Data structure
    - Algorithms
    - Optimizations, tuning parameters
- Related work
- Performance study
- Strong and weak points
- Conclusion
Data structure

• Unmodified R-tree is used
• ID-index is added
  • A disk-based hash table mapping objIDs to leaf page numbers
• Main-memory summary of the tree is maintained:
  • For each non-leaf node: level, MBR, child pointers, pointer to a corresponding disk page
  • For each leaf node: one bit recording whether the node is full

• For standard node fan-outs, the size of the main-memory summary is much less than 1% of the total index size
Case 1: new location is outside the root BR
- standard top-down update
Case 2: new location remains inside its rectangle
- write new p2 location
Case 3: new location is outside its rectangle

- enlarge rectangle
- if p3 inside R4
  - write new R4
  - write new p3 location
Case 4: new location is far outside its rectangle
- enlarging does not help
- deletion does not cause an underflow
- if new p5 is in the BR of a non-full sibling
  - delete old p5
  - get sibling node
  - insert new p5 into sibling
Case 5: new location is far outside its rectangle
- enlarging does not help
- no siblings, no underflow
  - delete old p2
  - findParent(pNode, newLocation)
  - do a standard R-tree insert at
    - the found parent node
Epsilon $\varepsilon$
Movement of Objects Between Siblings

- When moving an object to a sibling, redistribute other objects
Outline

• Motivation
• Background – the R-tree
• Generalized bottom-up update
  • Data structure
  • Algorithms
  • Optimizations, tuning parameters
• Related work
• Performance study
• Strong and weak points
• Conclusion
Related Work

• Lazy updates for R-tree [Kwon et al. 2002]
  • Leaf-level bounding rectangles are enlarged equally in all directions.
  • Parent pointers are added to the R-tree:
    • Expensive to maintain!
  • Query performance deteriorates because of increases in BR overlap.
  • It can be called localized bottom-up update (LBU) approach
Effect of $\varepsilon$

Updates
Effect of $\varepsilon$

Queries
Varying Buffer Size

Updates

Queries
Scalability

Updates

Queries
Outline

• Motivation
• Background – the R-tree
• Generalized bottom-up update
  • Data structure
  • Algorithms
  • Optimizations, tuning parameters
• Related work
• Performance study
• Strong and weak points
• Conclusion
Strong points

• Content:
  • The study is rather deep – explores the possibilities in-between localized updates and top-down updates
  • Concurrency is addressed
  • Extensive experiments
  • Cost model is presented showing theoretically the merit of the proposed approach

• Form
  • Good order of presentation:
    • First simpler algorithm, then a general one
Weak points

• Content:
  • It requires a non-constant amount of main-memory to work
  • It does not utilize all the available main-memory
  • Data structure and algorithms are rather complex
  • Too many parameters to adjust

• Form:
  • A couple of errors in the pseudo-codes
  • Pseudo-code does not have line numbers
  • Algorithm 3 pseudo-code is not very clear
  • Symbols used in formulas are not always explained (e.g., section 4.2)
Conclusion

• Addressed the problem of handling frequent updates in R-trees
• Proposed a generalized bottom-up update strategy for R-trees
• Significantly better performance than top-down and localized bottom-up update.

• Future work
  • Application to other multi-dimensional indexes
  • Better theoretical analysis of tradeoff between global-ness and update cost

• Acknowledgment:
  • Christian S. Jensen for most of the slides