Sort-Based Parallel Loading of R-trees

Daniar Achakeev, Marc Seidemann, Markus Schmidt, Bernhard Seeger

Department of Mathematics and Computer Science
Philipps-Universität Marburg, Germany
achakeye@mathematik.uni-marburg.de

ACM SIGSPATIAL BigSpatial 2012
Agenda

- Introduction
- Sequential sort-based query-adaptive loading
  - Sorted-Set Partitioning
- Parallel Loading
  - MapReduce
- Results
- Conclusion
Motivation

- R-tree
  - Spatial and multidimensional data

- Emerging applications
  - Location Based Services
    - kNN, Reverse kNN, Spatial keyword search etc…

- Tuple-by-Tuple loading is inefficient
  - Trade-off loading time query efficiency
  - NP-hard

- Parallelism
  - modern hardware
  - low cost parallel architecture e.g. Hadoop
Agenda

- Introduction
- **Sequential sort-based query-adaptive loading**
  - Sorted-Set Partitioning
- Parallel Loading
  - MapReduce
- Results
- Conclusion
R-tree

I/O Model:

R-Tree nodes mapped to disk blocks
Maximal capacity: $B$
Minimal capacity: $b \leq \left\lceil \frac{B}{2} \right\rceil$

Minimal Bounding Rectangle MBR
R-tree Query Types

Goal: minimize node accesses!
R-tree Query Types

Goal: minimize node accesses!
R-tree Query Types

**kNN Query**

Goal: minimize node accesses!
Cost Model

Minimize sum of areas of node MBRs ⇔ Minimize node accesses!

The average number of rectangles intersecting the query window

\[ \sum_{i=1}^{N} (dx_i + sx) \cdot (dy_i + sy) \]

Query profile QP is given by \((sx, sy)\)

I. Kamel and C. Faloutsos. On packing r-trees, In CIKM 1993
B.-U. Pagel et. al. Towards an analysis of range query performance in spatial data structures. In PODS 1993
Y. Theodoridis and T. Sellis A model for the prediction of r-tree performance. In PODS 1996
Objective

Rectangles

Queries

Minimal bounding rectangles (MBR) of R-tree leaf level
Sort-based Query-Adaptive Loading [1]

- NP-Hardness of optimal partitioning

- Conceptual Easy Heuristic Algorithm
  - Sorting according Space Filling Curve
  - Dynamic Programming
  - Adaptive SFC

- Excellent I/O performance
  - I/O Complexity is bounded by external sort $O\left(\frac{N}{B} \cdot \log \frac{M}{B} \frac{N}{B}\right)$

- Experiments
  - Non trivial test framework
  - Average better query performance
  - Robustness for different query and data distribution

- Parallel Version

Sort-based Query-Adaptive Loading

**Bottom-Up and Level-by-Level**

1. **Determination of Sort Order and Sorting:** For a given QP determine a sort order that minimizes cost.
   - Quadratic queries (aspect ratio 1:1); Hilbert or Z-Curve
   - Otherwise asymmetric Z-Curve

2. **Sorted set partitioning.** Partition the sorted sequence of rectangles into subsequences of size between minimal page capacity $b$ and maximal page capacity $B$

3. **Recursive Step:** Generation of index entries and recursion
The problem of optimal partitioning is **NP-hard**!

**Idea:**
1. Space Filling Curves
2. Dynamic Programming

Hilbert: \((r_{31}, r_{42}, r_{13}, r_{24}, r_{55}, r_{96}, r_{77}, r_{68}, r_{89})\)

**Example:**
- Max page capacity \(B=3\)
- Min page capacity \(b=2\)

Cost function:
- Sum of MBR areas

**Standard approach**  **Our approach**
Storage-Bounded Partitioning

**Dynamic Programming (DP)**

Hilbert: \{r_{31}, r_{42}, r_{13}, r_{24}, r_{55}, r_{96}, r_{77}, r_{68}, r_{89}\}  

\(b=2, B=3\)

\[ C[5][2] = \min \{C[2][1] + \text{area}_{QP}(\text{MBR}\{3,4,5\}), C[3][1] + \text{area}_{QP}(\text{MBR}\{4,5\})\} \]

- V-Optimal Histograms
- \(N/B \leq m \leq N/b\)
- Quadratic time \(O(N^2 \cdot B)\) and space \(O(N^2)\)

\[ \text{opt}^*(i,k) = \min_{b \leq j \leq B} \{\text{opt}^*(i-j, k-1) + \text{Area}_{QP}(\text{MBR}(p_{i-j+1,i}))\} \]
Query-Optimal Partitioning GOPT

Hilbert: \((r_{31}, r_{42}, r_{13}, r_{24}, r_{55}, r_{96}, r_{77}, r_{68}, r_{89})\)

- Linear time \(O(N \cdot B)\) and linear space \(O(N)\)
- Number of output partitions \(m\) is bounded by \(N/B \leq m \leq N/b\)

\[
gopt^*(i) = \min_{b \leq j \leq B} \{ gopt^*(i - j) + Area_{QP}(MBR(p_{i-j+1}, i)) \}\]

- Generalized methods for all levels are also investigated
GOPT Example ...

\[ b = 2, B = 3 \]

Hilbert: \((r_{21}, r_{42}, r_{13}, r_{24}, r_{55}, r_{96}, r_{77}, r_{68}, r_{89})\)

\[ \ldots C[6] = \min \{ C[3] + \text{area}_{QP}(\text{MBR}\{4,5,6\}), C[4] + \text{area}_{QP}(\text{MBR}\{5,6\})\} \]
GOPT Example …

b=2, B=3

Hilbert: \((r_{31}, r_{42}, r_{13}, r_{24}, r_{56}, r_{96}, r_{77}, r_{68}, r_{89})\)

…\(C[6] = \min \{C[3] + \text{area}_{QP}(\text{MBR}\{\{4,5,6\}\}), C[4] + \text{area}_{QP}(\text{MBR}\{\{5,6\}\})\}\)
GOPT Example …

Hilbert: \(r_3, r_4, r_1, r_2, r_5, r_6, r_7, r_8, r_9\)

End Result
Practical Considerations

- Reduce CPU and memory costs
  - Use main memory efficiently
  - Simple heuristic: chunking

\[ \geq B^2 \quad \geq B^2 \quad \geq B^2 \]
Practical Considerations

- Sort data only once for leaf level
- Use the sorting order of the produced output

Sorted data: \{r_{31}, r_{42}, r_{13}, r_{24}, r_{55}, r_{96}, r_{77}, r_{68}, r_{89}\}

Partitioning output: L1 = \{r_{31}, r_{42}\}, L2 = \{r_{13}, r_{24}\}, L3 = \{r_{55}, r_{96}, r_{77}\}, L4 = \{r_{68}, r_{89}\}

Index Level: \{L1, L2, L3, L4\}
Results

**H**: standard sort-based bulk loading, Hilbert-Order

**H-GO**: our approach GOPT, Hilbert-Order

**STR**: STR loading

4 KB Pages;

Queries follow data distribution.

The query size is defined by the number of results.
Agenda

- Introduction
- Sequential Loading
  - Sorted-Set Partitioning
- Parallel Loading
  - MapReduce
- Results
- Conclusion
Introduction

Forest-Approach [1][2]


Introduction

Our approach
Level-by-level [1]

[1] Daniar Achakeev, Marc Seidemann, Markus Schmidt, Bernhard Seeger: Sort-based Parallel Loading of R-trees, ACM SIGSPATIAL BigSpatial-2012;
Parallel Level-By-Level Loading

1. Computation of initial split vector V
2. Parallel sort using SFC
3. Data distribution over machines using V
4. Computation of optimal partitioning GOPT
5. Computation of split vector for the next level
6. Recursion: Step 3 using output of step 4
Leaf node generation:

- Mapper
  \[ (\text{null, MBRdata}), \ldots \] -> \[ (\text{SFC-Key, Data}), \ldots \] 

- Split vector \( V \) \( [p_1, \ldots, p_m] \) computed using parallel random sampling

- Partitioner distributes data using \( V \)

- Reducer runs gopt for its sorted key interval
  \( (\text{SFC-Key, [MBRdata]})) \) -> \[ ((\text{reducerRank, MBRRank}), \text{info}), \ldots \] 

Reducer Output:

- Sorted input data
- Leaf node MBR
  - Key (reducerRank, localRank)
MapReduce

Reducer 1

Reducer 2

p1:[0,5]

p2:[6,11]
MapReduce

- **Index node generation**
  - Mapper Identity
  - Partitioner: lexicographical order (reducerRank, MBRRank)
  - Reducer runs GOPT on ((reducerRank, MBRRank), info) objects

- **Final R-tree**
  - level files in parallel
  - level file sequentially
Results

- **Settings:**
  - Java, Hadoop 0.20.205.0, XXL-Java-Library
  - Amazon: medium machines
  - Data set TIGER USA Streets  72M MBR approx. 3.6 GB
  - Extended TIGER USA approx. 13 GB
  - Machines (1),2,4,8,16 (+ 1 Jobtracker)
  - Random Sampling 3%
Results

![Graph showing speedup vs. number of machines](image)

![Bar chart showing time in seconds](image)

Legend:
- **[Color] Sampling**
- **[Color] Leaf level**
- **[Color] Index level**

Number of machines:
- 1
- 2
- 4
- 8
- 16

Time in seconds:
- 0
- 2000
- 4000
- 6000
- 8000
Results

- **p-nopt**: level-by-level, fixed-size partitioning
- **p-trees**: forest approach [1]
- **p-gopt**: out approach

Conclusions & Next

- Novel parallel level-by-level approach
  - Excellent I/O performance
  - Almost linear speedup
  - Robust query performance
  - Conceptual Simplicity

- Efficient partitioning for load balancing
  - Minimize overlap/MBR Area

- Loading algorithms for parallel R-trees
  - Balancing query performance over set of machines
Thank You!

Q&A