CRB-Tree: An Efficient Indexing Scheme for Range-Aggregate Queries

Sathish Govindarajan Pankaj K. Agarwal Lars Arge

COMP 670Q PRESENTATION

by WANG Qi

Apr. 24, 2008

Outline

Outline

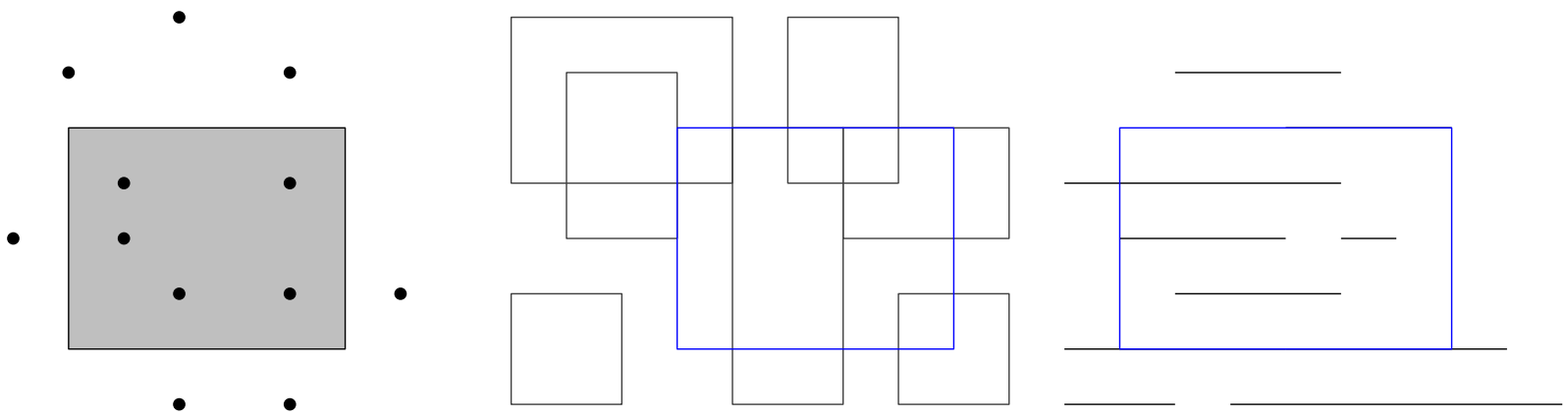
Motivation

- In OLAP spatial databases, range-aggregate queries(e.g. range-COUNT, range-SUM etc) are extremely important.
- Since the use of data warehouses is increasing rapidly, temporal aggregate queries have received much attention.

Outline

Model

- P: a set of N points in $\mathcal{R}^d$; $w: P \rightarrow \mathcal{Z}$: a weight function; $n = \lceil N/B \rceil$.
- The range-aggregate queries such as range-COUNT, SUM, AVG call for computing $|P \cap R|$, $\sum_{p \in P \cap R} w(p)$, and $\sum_{p \in P \cap R} w(p)/|P \cap R|$.
- Range-aggregate query, Rectangle-intersection-aggregate query, and Temporal range-aggregate query.



(i) (ii) (iii)

Outline

Related Work

- kdB-tree uses $O(n)$ disk blocks and can be used to answer a range-aggregate query in $O(\sqrt{n})$ I/Os.
- aP-tree uses $O(nlog_B n)$ disk blocks and can be used to answer a range-aggregate query in $O(log_B n)$ I/Os. (Tao et al.)
- SB-tree (MVSB-tree) uses $O(nlog_B n)$ disk blocks and can be used to answer a temporal aggregation query in $O(log_B n)$ I/Os. (Zhang et al.)
- In internal memory model, the best known data structure is *compressed range-tree*, which uses $O(N)$ space and can be used to answer a range-COUNT query in $O(\log_2 N)$ time. (Chazelle)

Outline

Basic Structure

- a B^+-tree T constructed on the x-coordinates of P, with each internal node v storing a secondary structure $\sum_v$.
- a normal B^+-tree ψ constructed on the y-coordinates of P.
- $P_v = \{p_1, p_2, \cdots\}$ denotes the sequence of points contained in the subtree of T rooted at v, sorted in a non-decrease order of their y-coordinates. Set $N_v = |P_v|$ and $n_v = N_v/B$.

Basic Structure

the secondary structure $\sum_v$

- $\sum_v$ is used to count the number of points of P_v that belong to a given child of v and whose y-coordinates $\leq$ a given y-value y_0.
- $\sum_v$ consists of two arrays:
  - a child index array CI_v: $CI_v[i]$- the index b_i of the subtree of v storing the ith point p_i of P_v.
  - a prefix count array PC_v: a two-dimensional $r \times B$ array. For $1 \leq i \leq r$ and $0 \leq j < B$, $PC_v[i,j] = |\{p_1, \cdots, p_{i\mu}\} \cap P_{v_j}|$. ($\mu = B \log_B N$, $r = N_v/\mu$)
- the total space used by T is $\sum_{v \in T} n_v / \log_B n = O(n)$ blocks.

Basic Structure

Bulk loading

- construct T by sorting the points in P in non-decreasing order of x-coordinates —— $O(sort(N))$ I/Os.
- compute P_v by using a single scan through the P_{v_i}'s —— $O(scan(N)) = O(n_v)$ I/Os.
- record the index of the child of v from which each point of P_v came from to construct CI_v —— $O(scan(N)) = O(n_v)$ I/Os.
- compute $PC_v[i, j]$ by scanning the points $p_{(i-1)\mu+1}, \cdots, p_{i\mu}$ —— $O(scan(N)) = O(n_v)$ I/Os.
- the total complexity is $O(sort(N))$ I/Os.

Outline

Query Procedure

- Suppose the query rectangle is $Q = [x_1, x_2] \times [y_1, y_2]$.
- I_v: the interval of x-coordinate.
- Traverse T topdown to find intervals I_λ, I_ρ in which x_1 and x_2 lie in respcetively.
- Compute $C = \sum_{\lambda < j < \rho} |P_{v_j} \cap Q|$.
- α_v: the rank of the first point in P_v whose y-coordinate is at least y_1; β_v: the rank of the last point in P_v whose y-coordinate is at least y_2.
- $\varphi(j, r)$: the number of points in P_v of rank at most r that belong to P_{v_j}, $\varphi(j, r) = |\{k | k \leq r \text{ and } CI_v[k] = j\}|$.
- $|P_{v_j} \cap Q| = \varphi(j, \beta_v) - \varphi(j, \alpha_v)$

Query Procedure

Complexity

- When we reach a leaf w, we compute $|P_w \cap Q|$ in $O(1)$ I/Os by scanning all the (at most B) points in P_v.
- Since we visit $O(\log_B n)$ nodes, we need $O(\log_B n)$ I/Os to answer a query.

Outline

Extensions of CRB-tree

- Range-SUM queries
  - Bulk loading: $O(n \log_B ((W \log_2 W)/N) \log_B n)$ I/Os ($W = \sum_{p \in P} w(p)$)
  - Space: $O(n \log_B ((W \log_2 W)/N))$ disk blocks.
  - Answering: $O(\log_B n)$ I/Os.
- Indexing in higher dimensions
  - Bulk loading: $O(n \log_B^{d-1} n)$ I/Os.
  - Space: $O(n \log_B^{d-2} n)$ disk blocks.
  - Answering range-COUNT: $O(\log_B^{d-1} n)$ I/Os.

Outline

Experiments & Results

- for $2D$ range-COUNT queries.
- compare the performance with kdB-tree .
- for a dataset of around 100 million points, the CRB-tree is $8 - 10$ times faster.

Conclusion

- CRB-tree is efficient for range-aggregate queries.
- CRB-tree can be extended to improve the results on temporal-aggregate queries.
- CRB-tree can be dynamized and extended to higher dimensions.

Q & A

Thank You!