Proc. 15th Int. Conf. on Scientific and Statistical Database Management (SSDBM 2003), Cambridge, Massachusetts (USA), 2003.

Acceleration of Relational Index Structures Based on Statistics

Hans-Peter Kriegel*, Peter Kunath*, Martin Pfeifle*, Matthias Renz*
*University of Munich, Institute for Computer Science
{kriegel, kunath, pfeifle, renz}@dbs.informatik.uni-muenchen.de

Abstract

Relational index structures, as for instance the Relational Interval Tree, the Relational R-Tree, or the Linear Quadtree, support efficient processing of queries on top of existing object-relational database systems. Furthermore, there exist effective and efficient models to estimate the selectivity and the I/O cost in order to guide the cost-based optimizer whether and how to include these index structures into the execution plan. By design, the models immediately fit to common extensible indexing/optimization frameworks, and their implementations exploit the built-in statistics facilities of the database server. In this paper, we show how these statistics can also be used for accelerating the access methods themselves by reducing the number of generated join partners. The different join partners are grouped together according to a cost-based grouping algorithm. Our first experiments on an Oracle9i database yield a speed-up of up to 1,000% for the Relational Interval Tree, the Relational R-Tree and for the Linear Quadtree.

1. Introduction

The efficient management of complex objects has become an enabling technology for many novel database applications, including computer aided design (CAD), medical imaging or molecular biology. For commercial use, a seamless and capable integration of spatial indexing into industrial-strength databases is essential. Fortunately, a lot of traditional database servers have evolved into Object-Relational Database Management Systems (ORDBMS). In [5], for instance, it was shown that the Linear Octree and especially the Relational Interval Tree (RI-tree) are suitable index-structures for the efficient management of intervals, and tiles likewise. In order to integrate these index structures into modern ORDBMSs, we need suitable cost models [2], which exploit the built-in statistics facilities of the database server. Based on these statistics it is possible to estimate the selectivity of a given query and to predict the cost of processing that query.

In this paper, we show how these statistics can be used to accelerate the query process using common relational index structures. We introduce our approach in general as well as exemplarily for spatial intersection queries performed on the Linear Quadtree. This discussion should enable the reader to easily adapt our ideas to other relational index structures, e.g. the RI-tree and the Relational R-tree.

The remainder of this paper is organized as follows. In Section 2 we introduce our basic idea in general. In Section 3, we adapt our general approach to the Linear Quadtree and show how it can be integrated into an ORDBMS. In Section 4, we present our first promising experimental results and conclude the paper with a few remarks on future work.

2. Accelerating Relational Index Structures

Our approach aims at reducing the total query I/O cost using a relational access method. The relational access method can be any custom index structure mapped to a fine granular relational schema which is organized by built-in access methods, as for instance the B^+-tree. Our general idea is to minimize the overall navigational cost of the built-in index by applying extended index range scans. Thereby we read false hits from the index, which are filtered out by a successive refinement step.

2.1. Index Range Scan Sequences

For spatial intersection queries, the query object Q leads to many disjoint range queries on the build-in index I. We consider them as a sequence $Seq_{Q,I}$ of index range scans.

I/0 cost. The number of logical reads $LR(s)$ associated with one index range scan $s = (l, u)$ of $Seq_{Q,I} = (\langle s_1,..., s_n\rangle)$ is composed from two parts: $LR_n(s)$ the navigational cost for finding the first page of the result set, and $LR_s(s)$ the cost for scanning the remaining pages containing the complete result set.

$LR(s) = LR_n(s) + LR_s(s)$, with the following properties:
(*i*) $LR_n(s) = LR_n(p(r'))$
(*ii*) $LR_s(s) = LR_s(<p(r'),...,p(r'')>)$
where the index entries r' and r'' have the property:
$\forall r \in R : (r < r' \Rightarrow r < l) \wedge (r > r'' \Rightarrow r > u)$
and $p(r)$ denotes the disk page of the index I, which contains the entry r. The number of logical reads $LR(Seq_{Q,I})$ associ-

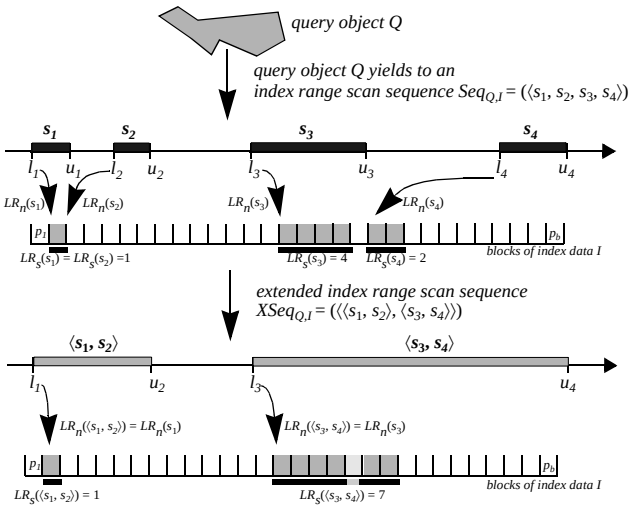

Fig. 1. Accelarated query processing

ated with $Seq_{Q,I} = (\langle s_1,..., s_n\rangle)$ is determined by $LR(Seq_{Q,I}) = \sum_{i=1}^{n} LR(s_i)$.

2.2. Extended Index Range Scan Sequences

The main purpose of our approach is to minimize the overall number of logical reads for the navigational part of the built-in index. Therefore, we try to reduce the number of generated range queries on the index *I*, while only allowing a small increase in the output I/O cost. This can be achieved by merging two suitable adjacent range scans *s'* = (*l'*, *u'*) and *s"* = (*l"*, *u"*) together to one *extended range scan xs* = (*l'*, *u"*).

Intuitively, an *extended range scan* $xs = \langle s_r,..,s_s\rangle$ is an ordered list of index range scans. When carrying it out, we traverse the index directory only once and perform a range scan (l_r, u_s), as for example (l_3, u_4) in Figure 1.

I/O cost. The number of logical reads *LR*(*xs*) associated with one extended range scan $xs = \langle s_r,..,s_s\rangle$ is composed from two parts $LR(xs) = LR_n(xs) + LR_s(xs)$, with the following properties:

(*i*) $LR_n(xs) = LR_n(s_r)$
(*ii*) $LR_s(xs) = LR_s((l_r,u_s))$

The number of logical reads $LR(XSeq_{Q,I})$ associated with an *extended index range scan sequence* $XSeq_{Q,I} = (\langle xs_1,..., xs_m\rangle)$ can be computed as follows:

$$LR(XSeq_{Q,I}) = \sum_{j=1}^{m} LR(xs_j).$$

Obviously, there might exist extended index range scan sequences $XSeq_{Q,I}$ for which $LR(XSeq_{Q,I}) << LR(Seq_{Q,I})$ holds. For each gap *g* between two adjacent range queries *s'* and *s"* we decide, whether the output I/O-cost of scanning over the gap *g* are lower than the navigational I/O cost related to *s"*. The decision whether to merge range scan *s'* and *s"* to one extended range scan and apply an additional refinement step afterwards in order to filter out false hits, is based on statistics, which are necessary for the cost models anyway. In [2] it was shown that the quantile-based approach efficiently and effectively estimates the selectivity by exploiting the built-in index statistics of ORDBMSs.

Definition 1 *(Quantile Vector).*

Let $(M, \leq)$ be a totally ordered multi-set. Without loss of generality, let $M = \{m_1, m_2, \ldots, m_N\}$ with $m_j \leq m_{j+1}$, $1 \leq j < N$. Then, $Q(M, \nu) = (q_0, \ldots, q_\nu) \in M^\nu$ is called a *quantile vector* for *M* and a *resolution* $\nu \in IN$, iff the following conditions hold:

(*i*) $q_0 = m_1$
(*ii*) $\forall i \in 1, \ldots, \nu: \exists j \in 1, \ldots, N: \; q_i = m_j \wedge \frac{j-1}{N} < \frac{i}{\nu} \leq \frac{j}{N}$

The multi-set *M* of our quantile vector $(q_0, \ldots, q_\nu)$ is formed by the values of the first attribute A_1 of the domain values of our index *I*. By means of these statistics we can estimate the number of logical reads LR_s^{est} associated with one range scan *s* = (*l*, *u*). In the following formula, *b* denotes the number of disk blocks at the leaf level of *I*, *v* denotes the resolution of the quantile vector, *N* denotes the overall number of entries stored in the index *I* and *overlap* returns the intersection length of two intersecting intervals.

$$LR_s((l,u)) \approx LR_s^{est}((l,u)) = \frac{\sum_{i=1}^{\nu} \left(\frac{overlap((l,u),(q_{i-1},q_i))}{q_i - q_{i-1}} \cdot \frac{N}{\nu} \right)}{(N/b)}$$

We can also apply the above formula to estimate the I/O cost $LR_s(g)$ related to scanning over a gap *g* =]*u'*, *l"*[between two adjacent range queries *s'* and *s"*. If $LR_s(g)$ is lower than $LR_n(s")$, we close the gap g. Thus we obtain an *extended index range scan* sequence $XSeq_{Q,I} = (\langle \langle s_{i_0+1},..,s_{i_1}\rangle, ..., \langle s_{i_{m-1}+1},..,s_{i_m}\rangle \rangle)$ for which the following property holds,

$$\forall i \in 1... n-1: \; i \in i_1 ... i_{m-1} <=> LR_n^{est}(s_{i+1}) < LR_s^{est}((u_i, l_{i+1}))$$

Usually, the navigational cost are independent of the actual range scan and can easily be estimated, e.g. by the height of the B^+-directory.

In the next section, we will show how our approach can be applied to a specific index structure.

3. The Intersection Query on the Linear Quadtree - an Example

The classical example for a space partitioning relational access method is the *Linear Quadtree* [7]. In this section, we shortly introduce our approach based on the basic idea of the Linear Quadtree according to the in-depth discussion of Freytag, Flasza and Stillger [1].

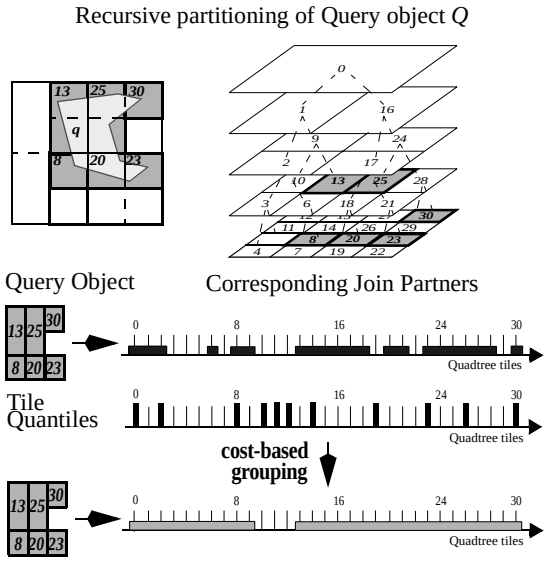

Fig. 2. Cost-Based Tile Grouping

The Linear Quadtree organizes the multidimensional data space using a regular grid. Any spatial object is approximated by a set of *tiles*. By numbering the Z-tiles of the data space according to a depth-first recursion, any set of Z-tiles can be represented by a set of linear values. The linear values of the Z-tiles of each spatial object can be stored in an index table *DBTiles* (*zval*, *id*), where both columns comprise the primary key, i.e we have a B^+-tree on the attributes *zval*, *id*. Furthermore we build a quantile vector on the values of the *zval*-attribute.

Assume object *Q* in Figure 2 is used as query object. Then there are multiple exact match and range scan queries which have to be performed. We can reduce the cost by closing small gaps on the leaf-level of the underlying B^+-tree. By using the information stored in the statistics, i.e. using the tile quantiles, the number of join partners can be reduced drastically, with a marginal increase of the I/O cost. The definition of *tile quantiles* is based on the general definition of a quantile vector (cf. Definition 1). The multi-set *M* of our quantile vector $(q_0, \ldots, q_v)$ is formed by the values stored in the leaf-level of the B^+-tree, i.e $M = \pi_{zval}(DBTiles)$.

We investigate all gaps included in the sequence of our generated join partners and decide for each gap whether it is beneficial to close it. Assume the height of our B^+-directory is *n*. If the database tiles of the actual investigated gap do not cover more than *n* leaf blocks on our index (*zval*), we close this gap. Thus we reduce the $join_{I/O}$ cost by *n*, while not increasing the $output_{I/O}$ cost by more than *n*. This procedure is depicted in Figure 2.

The above mentioned cost-based grouping step can be carried out in a procedural preparation step *JoinPartGen* by using bind variables [2], leading to an efficient cursor driven SQL-statement [4] (cf. Figure 3). This approach reduces

```
SELECT DISTINCT idx.id
FROM    DBTiles idx,
        TABLE(JoinPartGen(BOX((0,0),(10,10)))) tiles,
WHERE   (idx.zval BETWEEN tiles.ZvalLow AND tiles.ZvalHigh)
        AND TestZval(idx.zval, tiles.ExactZvalList);
```

Fig. 3. Accelarated window

the overhead of barrier crossings between the declarative and procedural environments to a minimum. The resulting table *tiles* contains entries of a type which consists of three attributes *ZvalLow*, *ZvalHigh* and an *ExactZvalList*. The attribute *ExactZvalList* is needed for an additional refinement step to filter out false index hits, called *TestZval*.

4. Conclusions and Future Work

We have implemented our cost-based grouping algorithm for the RI-tree, the Relational R-tree as well as for the Linear Quadtree on top of the Oracle9*i* database system. According to our first experiments, we achieved a speed-up of up to 1,000% depending on the query object and the data distribution. Obviously, for highly selective queries our approach yields very promising results. This is especially true for high resolution data spaces as the number of generated join partners exponentially depends on the resolution [3].

You could also look at our approach from another point of view. The traditional error-and size bound decomposition approaches [6] decompose a large query object into smaller query objects optimizing the trade off between accuracy and redundancy. In contrast, the idea of taking the actual data distribution into account in order to decompose the query object, could lead to a new *database driven decomposition approach*, which tries to minimize the overall number of logical reads.

5. Future Work

In our future work, we plan to elaborate our ideas further and try to present a sound and convincing experimental evaluation, where we want to investigate several index-structures as well as different query predicates.

We will analyze which level of detail for the statistics is most suitable for the acceleration of relational index-structures, and whether these statistics correlate with the ones necessary for the corresponding cost models.

References

[1] Freytag J.-C., Flasza M., Stillger M.: *Implementing Geospatial Operations in an Object-Relational Database System.* Proc. 12th Int. Conf. on Scientific and Statistical Database Management, 209-219, 2000.

[2] Kriegel H.-P., Pfeifle M., Pötke M., Seidl T.: *A Cost Model for Interval Intersection Queries on RI-Trees*. Proc. 14th Int. Conf. on Scientific and Statistical Database Management, 131-141, 2002.

[3] Kriegel H.-P., Pfeifle M., Pötke M., Seidl T.: *Spatial Query Processing for High Resolution.* Proc. 8th Int. Conf. on Database Systems for Advanced Applications, 2003.

[4] Kriegel H.-P., Pfeifle M., Pötke M., Seidl T.: *The Paradigm of Relational Indexing: A Survey.* 10. GI-Fachtagung Datenbanksysteme für Business, Technologie und Web, 2003.

[5] Kriegel H.-P., Pötke M., Seidl T.: *Managing Intervals Efficiently in Object-Relational Databases*. Proc. VLDB, 407-418, 2000.

[6] Orenstein J. A.: *Redundancy in Spatial Databases*. Proc. ACM SIGMOD Int. Conf. on Management of Data, 294-305, 1989.

[7] Samet H.: *Applications of Spatial Data Structures.* Addison Wesley Longman, Boston, MA, 1990.