

Advanced Database Aggregation Query Processing

Donghui Zhang

Computer Science Department, University of California, Riverside, CA 92521
donghui@cs.ucr.edu

Abstract. The aggregation query is an important but costly operation in database management systems. In the worst case, to compute an aggregate, all tuples in a database needs to be examined. However, in many applications (on-line analysis, etc.), we need to compute the aggregates very fast. In the Ph.D. research, we focus on devising specialized indices for the box-aggregation query and its relatives. Our research shows that the newly designed structures have much better query performance than the existing solutions, sometimes over a hundred times faster. In this paper, we summarize our findings.

1 Introduction

In this Ph.D. work, we consider the *box-aggregation problem*: “given n weighted rectangular objects and a query rectangle r in the d -dimensional space, find the aggregate of all the objects which intersect r ”. Previous work on multi-dimensional aggregation (e.g. [5]) considers only point objects (i.e., objects with zero extent in all dimensions) that fall on a fixed multi-dimensional grid. However, in temporal, spatial, or spatiotemporal environment, the objects have non-zero extents. For example, a record in a spatial database is normally represented by a *minimum bounding rectangle (MBR)*.

Without utilizing any index structure, to compute the box-aggregates we need to scan through the objects, determine those which intersect the query rectangle and aggregate their values on the fly. This is clearly inefficient. We can utilize index structures to speed up the query processing. One such index is the R^* -tree [3]. An optimization is that we can store aggregate information along with internal records in the tree so that when the query rectangle fully contains a sub-tree, the examination of the sub-tree can be omitted. This approach is called the *Aggregation R-tree (aR-tree)* [7, 6].

Depending on the application (temporal, spatial, etc.) and type of aggregation (we consider SUM, COUNT, AVG, MIN, and MAX), the general box-aggregation problem has various special cases and related problems. In section 2 we discuss the five aggregation problems that we have addressed during the Ph.D. research and summarize our solutions. Section 3 provides our conclusions.

2 Problem Definition & Proposed Solutions

2.1 Range-Temporal Aggregation

Definition 1. range-temporal aggregation (RTA): *given a set of temporal records, each having a key, a time interval and a value, compute the total value of records whose keys are in a given range and whose intervals intersect a given time interval.*

Our approach [9] is the first which addresses this problem. Previous work on temporal aggregation (the most efficient one being [8]) aggregate on the whole key space. To solve the RTA problem using the previous approaches, we would need to maintain a separate index for each possible key range, which is prohibitively expensive. We proposed a new index structure called the *Multi-version SB-tree (MVSB-tree)* and we were able to prove the following theorem:

Theorem 1. *Using the MVSB-tree, a SUM, AVG, COUNT RTA query is answered in $O(\log_b n)$ I/Os. The insertion/deletion cost is $O(\log_b K)$ while the space complexity is $O(\frac{n}{b} \cdot \log_b K)$.*

Here n is the total number of insertions, K is the number of different keys ever inserted, and b is the page capacity in number of records.

The approach has much better query performance than the existing approach which indexes all the temporal records in an index structure since the approach has guaranteed worst-case logarithmic performance, while the existing approach has linear query complexity. More details of the solution appear in [9].

2.2 Hierarchical Temporal Aggregation

We are interested in computing temporal aggregates (both with and without the key-range predicate) with fixed storage space. Since historical data accumulates over time, while with fixed storage space we cannot keep all of them, we have to live with storing partial information. One approach is to keep just the most recent information. However, it leads to lose of the ability to answer queries for the past. Another approach is to aggregate all information at coarser granularity (e.g. instead of aggregating on days, aggregate on months). However, this approach leads to lose of aggregation details.

In [12], we proposed to maintain the temporal aggregates under multiple granularities, with more recent information being aggregated at finer granularities. Initially, all information are aggregated at the finest granularity (e.g. by day). As time advances, the storage size of the aggregate information at the finest granularity increases. If we have fixed storage space, once and a while we need to shrink the storage size by removing part of the aggregated information and aggregate it at coarser granularity (e.g. by month). Our proposed structure (the *2SB-tree^{FS}*) systematically aggregates information at coarser granularities to keep the index size within a given limit.

Theorem 2. *Given a storage limit S , expressed in number of disk pages, the complexities of the 2SB-tree^{FS} with k time granularities are as follows. To compute a temporal aggregate takes $O(\log_B (S/k))$ I/Os, and the amortized insertion cost is $O(k \log_B (S/k))$. Here B is the page capacity in number of records.*

We also provided solutions under the *Fixed Time Window Model*, and we have extended our solutions to the range-temporal and the spatiotemporal environment. More details appear in [12].

2.3 Box-Sum Aggregation

This problem is a special case of the box-aggregation problem for the SUM/COUNT/AVG aggregates. In [11] we proposed efficient solutions to this problem. Compared to the existing work (the aR-tree), our solution has much better query performance.

We first reduced the box-sum problem to the simpler *dominance-sum* problem: “given a collection S_p of point objects and a query point p , compute $\text{SUM}\{o.\text{value} \mid o \in S_p \text{ and } o.\text{point} \text{ is dominated by } p\}$ ”. Here point p_1 dominates point p_2 if the projection of p_1 to every dimension is no less than the projection of p_2 in the same dimension. Our reduction technique reduces a box-sum query into 2^d dominance-sum queries, which is more efficient than the existing technique [4] which reduces into $\Omega(3^d/\sqrt{d})$ dominance-sums.

To solve the dominance-sum problem (and thus the box-sum problem), we proposed two structures that extend the main-memory, static ECDF-tree [1] to the external-memory environment and to handle dynamic updates. One approach has more efficient update while the other has better query time. Finally, we proposed the BA-tree which tends to combine the good things of the two structures. Performance results proved the efficiency of the BA-tree when compared with the other approaches. More details appear in [11].

2.4 Functional Box-Sum Aggregation

Definition 2. functional box-sum aggregation: *given a set of objects, each having a box and a value function, and a query box q , compute the total value of all objects that intersect q , where the value contributed by an object r is the integral of the value function of r over the intersection between r and q .*

This problem is a variation of the box-sum problem. In some cases, it more accurately captures the application needs. This is because an object contributes to the query result proportional to how large it intersects the query rectangle. In [11], we proposed techniques to solve the functional box-sum problem for polynomial value functions. To the best of our knowledge, this is the first work which addresses this problem.

Our approach is to reduce the functional box-sum problem into a special dominance-sum problem, where the values stored in the index are not single values, but coefficients of some polynomial functions. Then, we can use the existing box-sum structures (e.g. the BA-tree) to compute the functional box-sums. The reduction technique can be found in [11].

2.5 Box-Max Aggregation

This problem is a special case of the box-aggregation problem for the MIN/MAX aggregates. In [10] we proposed a new index structure called the *MR-tree* which

efficiently computes the box-max query. Compared with the existing approach (the aR-tree), our approach has much better query performance and much smaller index size.

Our approach is based on four novel optimization techniques, namely, the *k-max*, the *box-elimination*, the *union*, and the *area-reduction* optimizations. More details appear in [10].

3 Conclusions

In this paper, we have summarized our research results as reported in [9–12]. We have presented specialized aggregation index structures for the range-temporal aggregation, the hierarchical temporal aggregation, the box-sum aggregation, the functional box-sum aggregation, and the box-max aggregation problems. In all cases, our proposed specialized index structures have much better query performance than the existing non-specialized indices. Furthermore, the proposed indices are all disk-based and suit for dynamic updates. Their index sizes are either smaller or very close to the existing structures. Based on these findings, we recommend that the proposed index structures should be implemented in commercial DBMSs when the aggregation query performance is crucial.

References

1. J. L. Bentley, “Multidimensional Divide-and-Conquer”, *Communications of the ACM* 23(4), 1980.
2. B. Becker, S. Gschwind, T. Ohler, B. Seeger and P. Widmayer, “An Asymptotically Optimal Multiversion B-Tree”, *VLDB Journal* 5(4), 1996.
3. N. Beckmann, H. Kriegel, R. Schneider and B. Seeger, “The R* tree: An Efficient and Robust Access Method for Points and Rectangles”, *Proc. of SIGMOD*, 1990.
4. H. Edelsbrunner and M. H. Overmars, “On the Equivalence of Some Rectangle Problems”, *Information Processing Letters* 14(3), 1982.
5. S. Geffner, D. Agrawal and A. El Abbadi, “The Dynamic Data Cube”, *Proc. of EDBT*, 2000.
6. I. Lazaridis and S. Mehrotra, “Progressive Approximate Aggregate Queries with a Multi-Resolution Tree Structure”, *Proc. of SIGMOD*, 2001.
7. D. Papadias, P. Kalnis, J. Zhang and Y. Tao, “Efficient OLAP Operations in Spatial Data Warehouses”, *Proc. of SSTD*, 2001.
8. J. Yang and J. Widom, “Incremental Computation and Maintenance of Temporal Aggregates”, *Proc. of ICDE*, 2001.
9. D. Zhang, A. Markowetz, V. J. Tsotras, D. Gunopulos and B. Seeger, “Efficient Computation of Temporal Aggregates with Range Predicates”, *Proc. of PODS*, 2001.
10. D. Zhang and V. J. Tsotras, “Improving Min/Max Aggregation over Spatial Objects”, *Proc. of GIS*, 2001.
11. D. Zhang, V. J. Tsotras and D. Gunopulos, “Efficient Aggregation over Objects with Extent”, *Proc. of PODS*, 2002.
12. D. Zhang, V. J. Tsotras, D. Gunopulos and B. Seeger, “Temporal Aggregation over Data Streams using Multiple Granularities”, *Proc. of EDBT*, 2002.