Optimizing Aggregation Query Processing

Prateek Singhal[1], Velagapudi Sreenivas[2], K V D Kiran[3]

IV/IV B.Tech, Department of CSE, K L University, Andhra Pradesh, India
[1]srkprateek@gmail.com
Asst Professor, Department of CSE, K L University, Andhra Pradesh, India
[2]velagapudi@kluniversity.in
Asst Professor, Department of CSE, K L University, Andhra Pradesh, India
[3] kiran_cse@kluniversity.in

Abstract-The selection query has been one of the most widely used queries in databases. For instance, find the gas stations that are within a given spatial range. Such queries select a subset of records from a large collection of data based on a given selection condition. A straightforward approach to solve the selection-based aggregation problem is: first find all the records that satisfy the selection condition and then perform the aggregation on-the-fly. The problem with this approach is that the query performance is at least linear to the size of the selection result. If many records satisfy the selection condition, the performance is not satisfactory. A better approach is to build some specialized index which can help compute the aggregation result without scanning through the records.

Key words – Aggregation, Query, Hierarchical Temporal Aggregation, Optimizing, Box Model.

I. INTRODUCTION

The first aggregation problem we address in this paper is temporal aggregation. With the rapid increase of historical data in data warehouses, temporal aggregates have become predominant operators for data analysis. Computing temporal aggregates is a significantly more intricate problem than traditional aggregation without the time dimension. This is because each database tuple is accompanied by a time interval during which its attribute values are valid. Consequently, the value of a tuple attribute affects the aggregate computation for all those instants included in the tuple's time interval. Many approaches have been recently proposed to address temporal aggregation queries. Earlier work focused on the *instantaneous* temporal aggregation problem, i.e. compute the aggregate value over objects whose time intervals contains a query time instant.

II PROBLEM DEFINITION

2.1 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 [3] is the first which addresses this problem. 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(logb n) I/Os. The insertion/deletion cost is O(logb K) while the space complexity is O(nb ¢ logb 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 [4].

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 [5], a proposal is made 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-treeFS) systematically aggregates information at coarser granularities to keep the index size within a given limit.

© 2013 IJAIR. ALL RIGHTS RESERVED

III. THE SIMPLE BOX-SUM PROBLEM

We differentiate between two types of objects: point and box objects. Given two d-dimensional points x = (x1, · · · , xd) and y = (y1, · · · , yd), we say that x dominates y if for every i {1, · · · , d}, xi yi. A dimensional box b can be described by two corner points: a low point which is dominated by all other corner points of b and a high point which dominates all other corner points of b. The d-dimensional space is itself a box whose low point and high point are represented as pmin and pmax, respectively. Each object has a value which is used for the aggregation. We refer to the following aggregations:

- (simple) box-sum: given a collection Sb of box objects and a query box q, compute SUM{o.value|o Sb and o.box intersects q};
- range-sum: given a collection Sp of point objects and a query box q, compute SUM{o.value|o Sp and o .point is contained in q};
- dominance- sum: given a collection Sp of point objects and a query point p, compute SUM{ o.value|o Sp and o. point is dominated by p}.

The box-sum problem is the most general since, (i) the range-sum problem is a special case of the box-sum when the box of each object reduces into a point, and, (ii) the dominance-sum problem is a special case of the range-sum problem with query box q = (pmin, p). The COUNT aggregation problems (box-count, range-count and dominance count) are special cases of the SUM aggregations, when the value of every object is 1.

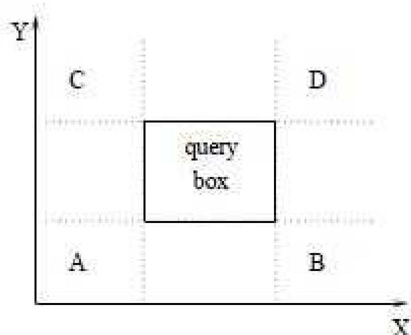

Fig 1. Existing technique reduces a box-sum query into eight dominance-sum queries.

Since it is easy to maintain the sum of all objects, to find the sum of objects intersecting a query box, it is enough to compute the sum of objects NOT intersecting the query box. These objects must be either above, below, to the left of, or to the right of the query box. To find the sum of objects which are to the left of the query box can be done via a 1- dimensional dominance-sum query. I.e. if we maintain the higher x of all objects, the task is to find the dominance- sum regarding the lower x of the query box. Similarly, we can compute the sum for objects above, below, and to the right of the query box. If we add up these results, we get a value larger than the anticipated box-sum. The reason is that any object which resides in the regions A, B, C or D is counted twice. We note that the sum in each of these areas can be answered by a 2-dimensional dominance-sum query. Hence, a 2-dimensional box-sum query is reduced to four 1-dimensional and four 2-dimensional dominance-sum queries.

3.1 Temporal Aggregation with Fixed Storage

[1] proposed two approaches to incrementally maintain temporal aggregates. Both approaches rely on using two *SB-trees* collectively.[6] In this section we first propose a slightly different approach and then we present a technique to extend the approach under the fixed storage model.

3.2 The 2SB-tree

A single SB-tree as proposed in [YW01] can be used to maintain instantaneous temporal aggregates. One feature about the SB-tree is that a deletion in the base table is treated as an insertion with a negative value. Thus in the rest we focus on the insertion operation. [YW01] also proposed two approaches to maintain the cumulative temporal aggregate. The first approach is called *Dual SB-tree*. Two SB-trees are kept. One maintains the aggregates of records valid at any given time, while the other maintains the aggregates of records valid strictly before any given time.[7] The latter SB-tree can be implemented via the following technique: whenever a database tuple with interval i is inserted in the base table, an interval is inserted into the SB-tree with start time being i.end and end time being + . To compute the aggregation query with query interval I the approach first computes the aggregate value at i.end It then adds the aggregate value of all records with intervals strictly before i.end and finally subtracts the aggregate value of all records with intervals strictly before i.start.

The second approach is called the JSB-tree. Logically, two SB-trees are again maintained. One maintains the aggregates of records valid strictly before any given time, while the other maintains the aggregates of records valid strictly after any given time. Physically, the two SB-trees can be combined into one tree, where each record keeps two aggregate values rather than one.

© 2013 IJAIR. ALL RIGHTS RESERVED

To compute an aggregate with query interval i, we find the total value of records whose start times are less than i.end and then subtract the total value of records whose end times are less than i.start. Note that each such SB-trees takes as input, besides a value, a point instead of an interval. The point implies an interval from it to + . Compared with the Dual SB-tree, to answer an aggregation query we need to perform two SB-tree traversals instead of three. Compared with the JSB-tree approach, there is no need to maintain the total weight of all records, and the two SB-trees have unified point input. In the JSB-tree, we can also let the two SB-trees take point input; however, the input points have different meanings for the two trees. In one tree, it implies an interval from - to the point, while in the other tree, it implies an interval from the point to + .

IV.PERFORMANCE ANALYSIS

We use *SB FTW* to represent the 2SB-tree with fixed time window. Figure [3] compares the index generation time while varying the by-minute window size. The single-granularity indices SB day and SB min are not affected when the by-minute window size varies.[8] As expected, the SB day takes the shortest time to generate, while the SB min takes the longest time. The generation time of the SB FTW is between the other two (11 times less than that of the SB min for 1% window size). As the by-minute window size becomes larger, the generation time of the SB FTW tends to be longer, too. The effect of the size of the by-minute window on the generation time is twofold.

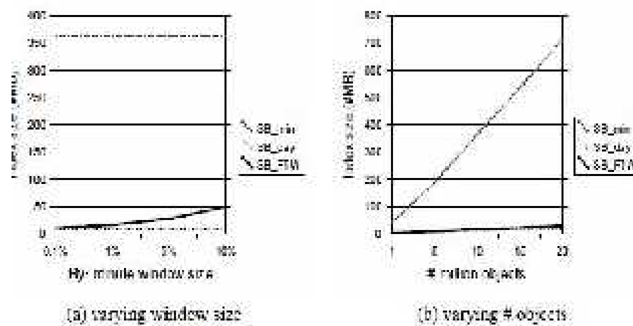

Fig 2.Index sizes of temporal aggregation with fixed time window.

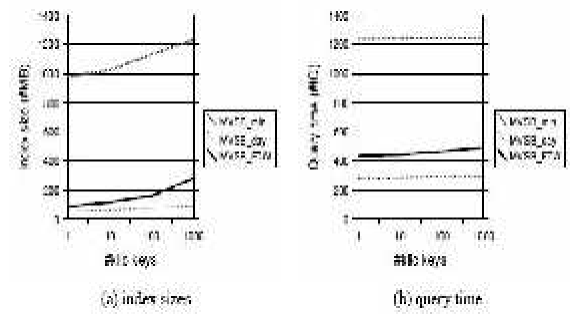

Fig 3. Generation and query time of temporal aggregation with fixed storage.

A larger window size means that the dividing time is increased less often. On the other hand, a larger window implies that the sizes of the indices which exist after the dividing time are larger and thus the updates in them take longer. The combined effect is shown in figure [2]. As shown in figure [4], the SB FTW and the SB day have similar query performance which is much faster than that of the SB min. For 1% window size, the SB FTW is 30 times faster than the SB min.[9] The SB FTW is preferred over the SB day since for the recent history; the SB FTW has the ability to aggregate at a finer granularity.

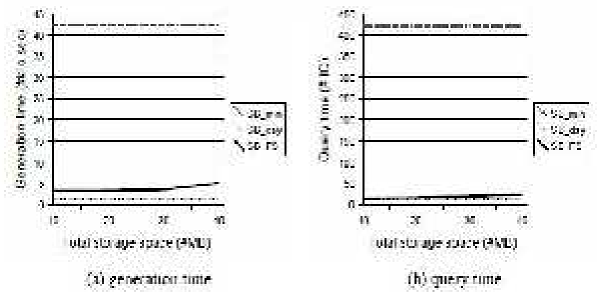

Fig 4. Performance of the range temporal aggregation with fixed time window, varying number of keys.

4.1 Performance of Spatio-Temporal Aggregation with Fixed Time Window

We use *BA FTW* to represent our spatio-temporal aggregation index with fixed time window while *BA day* and *BA min* denote the corresponding single-granularity indices. For the data sets we generated, besides the time dimension there are two spatial dimensions. The number of different locations per spatial dimension varies from 50 to 1000. Again, the by-minute window size is set to 10% of the time space,as the number of locations per spatial dimension increases, both the index sizes and the query time of the indices become larger. The reason is that the BA-tree is sensitive to the number of different spatial locations a point object may reside at. This is

© 2013 IJAIR. ALL RIGHTS RESERVED

more apparent with the BA FTW since to perform a query, besides the main index, we also query a separate

V. BOX MAX AGGREGATION

We focus on the MAX aggregation and the techniques we propose can be directly applied to compute MIN aggregates. More formally, the problem we focus on is defined as follows .box-max: given a collection S of box objects and a query box q, compute MAX{ o. value | o E S and o.box intersects q}. An example of box-max query is: "find the max precipitation in the Los Angeles district". Again, straightforward approaches find the actual objects that intersect with the query and thus are not efficient. Our aim is to provide specialized indices which focus on the MAX aggregate computation.

Although the box-max problem is similar to the box-sum problem, the solutions we proposed for the box-sum problems cannot be applied to the box-max problem. The reason lies in a inherent difference between the MAX problem and the SUM problem. The SUM operator has a inverse operator SUBTRACT, which means that to compute the SUM of a set of records (figure 5 a), we can compute the SUM of a larger set of records (figure 5 b) and SUBTRACT from it the SUM of records in the difference between the superset and the original set (figure 5 c).

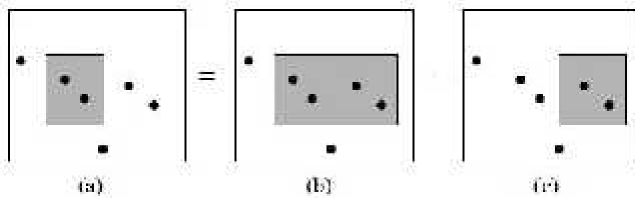

Fig 5.1: SUM aggregate has an inverse operator, but MAX aggregate does not.

In this chapter, we present a specialized aggregate index, the Min/Max R-tree (MR-tree) for the MIN/MAX aggregation. Based on the R*-tree, we propose four optimizations. One of the optimizations (the k-max) attempts to eliminate more paths from the index traversal when the aggregate is computed. As such, it can be used either on the SAM that indexes the objects, or, on a specialized aggregate index. The other optimizations (union, box-elimination and area-reduction) eliminate or resize object MBRs when they do not affect the MIN/MAX computation. Thus they apply only to specialized MIN/MAX aggregate indices.

As a by-product, we discuss how a specialized aggregate index, the MSB-tree [YW00], can be improved by applying the box-elimination optimization. The MSB-tree efficiently solves the MIN/MAX problem for the special case of one-dimensional interval objects. The optimization allows the MSB-tree to avoid frequent reconstructions that were needed in its original version.

VI. THE PROPOSED OPTIMIZATIONS

To compute box-max aggregates, we could use an R-tree to index the objects and reduce the box-max computation to a range search, which means to locate the actual objects that intersect the query box while keeping a running max value of the objects seen so far. Based on the R-tree index, in this section we propose four optimizations that improve the performance.

We first introduce some notations. An index/leaf record is an entry in an internal/leaf node of the tree.[10] Given an leaf record r,let r.box and r.value denote the MBR and the value of the record, respectively. Given an index record r, let r.box denote its MBR, r.value denote the maximum value of all records in sub tree (r) and r.child denote the child page pointed by r.

6.1 The k-max optimization

The a R-tree [PKZ+01] is an R-tree where each index record stores the aggregate (in this case, maximum) value of all leaf records in the sub-tree.[11] If a query box contains the MBR of an index record, the value stored at the record contributes to the query answer and the examination of the sub-tree is omitted. However, the index records at higher levels of the a R-tree have large MBRs. So the box-max query is not likely to stop at higher levels of the a R-tree. The M-max optimization is an extension such that even if the query box does not contain the MBR of an index record, the examination of the sub-tree may be omitted.

The M-max optimization Along with each index record r, store the objects (for a small constant k) which are in subtree (r) and have the largest values among the objects in the subtree. When examining record r during a box-max query, if the query box intersects with any of the k max-value objects in r, the examination of subtree (r) is omitted.

Clearly, the k-max optimization allows for more paths to be omitted during the index traversal.[12] However, the benefit of M-max on the query performance is not provided for free. The overall space is increased (since each node stores more information) as well as the update time (effort is needed to maintain the M objects). Hence in practice the constant M should be kept small.

As pointed out, the next three optimizations apply for an index explicitly maintained for the MIN/MAX aggregation (to avoid confusion we call such an explicit index the MIN/MAX index). Since the MIN/MAX problem is not

© 2013 IJAIR. ALL RIGHTS RESERVED

incrementally maintainable when tuples are deleted from the database [YW01], the following discussion assumes an append-only database (i.e., spatial objects are inserted in the database but never deleted). When a spatial object o with MBR o.box and value o.value is inserted in the database, o.box accompanied by o.value is inserted as a leaf record in the MIN/MAX index as well. However, as we will describe, some of these insertions may not be applied to the MIN/MAX index, or may cause existing MBRs to be deleted or altered from the MIN/MAX index. As such, we can use an R*- tree to implement the MIN/MAX index. The result after applying all four optimizations will be the MR-tree.

VII. CONCLUSION

Temporal aggregation has become predominant operators in analyzing time-evolving data. Many applications produce massive temporal data in the form of streams. For such applications the temporal data should be processed (pre-aggregation, etc.) in a single pass. In this chapter we examined the problem of computing temporal aggregates over data streams. Furthermore, aggregates are maintained using multiple levels of temporal granularities: older data is aggregated using coarser granularities while more recent data is aggregated with finer detail. We presented two models of operation. In the fixed storage model it is assumed that the available storage is limited.

The fixed time window model guarantees the length of every aggregation granularity. For both models we presented specialized indexing schemes for dynamically and progressively maintaining temporal aggregates. An advantage of our approach is that the levels of granularity as well as their corresponding index sizes and validity lengths can be dynamically adjusted. This provides a useful trade-off between aggregation detail and storage space. Based on our temporal aggregation work, we summarized a framework for computing aggregates over time-evolving data and we discussed how the solutions can be extended to solve the more general range temporal and spatio-temporal aggregation problems under the fixed time window model. Finally, an extended performance evaluation validated the advantages of the proposed structures.[13] As future work we plan to further investigate techniques to aggregate at multiple spatial granularities as well. Moreover, we are extending our framework to the multiple data stream environment.

In this paper, we have summarized and 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.

VIII. REFERENCES:

[1] [YK97] X. Ye and J. Keane "Processing temporal aggregates in parallel", *Proc. of Int. Conf. on Systems, Man, and Cybernetics*, pp. 1373-1378, 1997..

[2] J. L. Bentley, Multidimensional Divide-and-Conquer", *Communications of theACM* 23(4), 1980.

[3]. B. Becker, S. Gschwind, T. Ohler, B. Seeger and P. Widmayer, \An Asymptotically Optimal Multiversion B-Tree", *VLDB Journal* 5(4), 1996.

[4]. 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.

[5]. H. Edelsbrunner and M. H. Overmars, \On the Equivalence of Some Rectangle Problems", *Information Processing Letters* 14(3), 1982.

[6]. S. Ge®ner, D. Agrawal and A. El Abbadi, \The Dynamic Data Cube", *Proc. Of EDBT*, 2000.

[7]. I. Lazaridis and S. Mehrotra, \Progressive Approximate Aggregate Queries with a Multi-Resolution Tree Structure", *Proc. of SIGMOD*, 2001.

[8]. D. Papadias, P. Kalnis, J. Zhang and Y. Tao, \E±cient OLAP Operations in Spatial Data Warehouses", *Proc. of SSTD*, 2001.

[9]. J. Yang and J. Widom, \Incremental Computation and Maintenance of Temporal Aggregates", *Proc. of ICDE*, 2001.

[10] C. Chung, S. Chun, J. Lee and S. Lee, \Dynamic Up-date Cube for Range-Sum Queries", Proc. of VLDB, 2001.

[11] C. Chan, Y. E. Ioannidis, \Hierarchical Cubes for Range-Sum Queries", Proc. of VLDB, 1999.

[12] Y. Chiang and R. Tamassia, \Dynamic Algorithms in Computational Geometry", Proc. of the IEEE, Special Issue on Computational Geometry, G. Toussaint (Ed.), 80(9), 1992.

[13] H. Edelsbrunner and M. H. Overmars, \On the Equiv-alence of Some Rectangle Problems", Information Pro-cessing Letters 14(3), 1982.

© 2013 IJAIR. ALL RIGHTS RESERVED