Efficient Evaluation of Ad-Hoc Range Aggregates

Christian Ammendola[1], Michael H. Böhlen[2], and Johann Gamper[1]

[1] Free University of Bozen–Bolzano, 39100 Bozen–Bolzano, Italy
christian@ammendola.name, gamper@inf.unibz.it
[2] University of Zurich, 8050 Zurich, Switzerland
boehlen@ifi.uzh.ch

Abstract. θ-MDA is a flexible and efficient operator for complex ad-hoc multi-dimensional aggregation queries. It separates the specification of aggregation groups for which aggregate values are computed (base table $\mathbf{b}$) and the specification of aggregation tuples from which aggregate values are computed. Aggregation tuples are subsets of the detail table $\mathbf{r}$ and are defined by a general θ-condition. The θ-MDA requires one scan of $\mathbf{r}$, during which the aggregates are incrementally updated.

In this paper, we propose a two-step evaluation strategy for θ-MDA to optimize the computation of ad-hoc range aggregates by reducing them to point aggregates. The first step scans $\mathbf{r}$ and computes point aggregates as a partial intermediate result $\tilde{\mathbf{x}}$, which can be done efficiently. The second step combines the point aggregates to the final aggregates. This transformation significantly reduces the number of incremental updates to aggregates and reduces the runtime from $\mathcal{O}(|\mathbf{r}| \cdot |\mathbf{b}|)$ to $\mathcal{O}(|\mathbf{r}|)$, provided that $|\mathbf{b}| < \sqrt{|\mathbf{r}|}$ and $|\tilde{\mathbf{x}}| \approx |\mathbf{b}|$, which is common for OLAP. An empirical evaluation confirms the analytical results and shows the effectiveness of our optimization: range queries are evaluated with almost the same efficiency as point queries.

1 Introduction

Multi-dimensional aggregation queries, such as range aggregates that aggregate sets of tuples identified by a range condition, are an important class of queries in business intelligence and data warehousing applications. To efficiently process such queries, various techniques have been proposed, including extensions to SQL [7], generalized projections [8], pre-aggregation of data cubes [9], and (relative) prefix sums [6,10]. θ-MDA [1] is a flexible and efficient operator for *ad-hoc* multi-dimensional aggregation queries, where pre-computed aggregates are not available and a scan of the detail table is required.

Consider the relation $\mathbf{stays}$ in Fig. 1(a), which stores hospital stays of patients and has the following attributes: admission date (D), patient identifier (P), urgency category (U), and duration of the stay (S). To analyze the impact of urgent stays ($U{=}3$) on the average duration of stays, aggregates for stays with $U{\in}\{1,2\}$ are compared to aggregates for stays with $U{\in}\{1,2,3\}$. Consider the following query Q1: *Compute the cumulative average duration of stays per admission date ($C1$) and per admission date and urgency ($C2$).* The result of Q1 is shown in table $\mathbf{x}$. The first two columns represent the aggregation groups. The other two columns represent the aggregate results. The average is represented as a sum/count pair.

L. Bellatreche and M.K. Mohania (Eds.): DaWaK 2013, LNCS 8057, pp. 46–59, 2013.

stays

	D	P	U	S
r_1	31/01/13	P1	1	2
r_2	31/01/13	P2	1	4
r_3	31/01/13	P3	3	8
r_4	31/01/13	P4	2	4
r_5	31/01/13	P5	1	3
r_6	31/01/13	P6	2	3
r_7	01/02/13	P7	1	2
r_8	01/02/13	P8	1	3
r_9	01/02/13	P9	3	9
r_{10}	01/02/13	P10	3	6

(a) Detail Table

x

	D	U	$C1$	$C2$
x_1	31/01/13	2	24/6	16/5
x_2	31/01/13	3	24/6	24/6
x_3	01/02/13	2	44/10	21/7
x_4	01/02/13	3	44/10	44/10

(b) Result of Query Q1

Fig. 1. Running Example

$C1$ and $C2$ are *range aggregates*. Each aggregate is computed from the aggregation tuples: all detail tuples that satisfy a given range predicate (exemplified by hatched areas in Fig. 1). For instance, aggregate $x_3.C2$ is the average duration of a stay over all detail tuples in **stays** with $D \leq 01/02/13$ and $U \leq 2$, i.e., the tuples $\{r_1, r_2, r_4, r_5, r_6, r_7, r_8\}$. While one-dimensional range aggregates can be solved efficiently with SQL window functions, multi-dimensional range aggregates require expensive joins with range predicates [1].

The θ-MDA operator [1] offers a succinct formulation and efficient computation of ad-hoc multi-dimensional aggregation queries. The operator separates the specification of the aggregation groups in a base table $\mathbf{b}$ (first two columns in table $\mathbf{x}$ in Fig. 1(b)) from the specification of the aggregation tuples. For each aggregation group in $\mathbf{b}$ a θ-condition defines the aggregation tuples in the detail table $\mathbf{r}$ from which aggregate values are computed, e.g., $x_3.C2$ is computed from the aggregation tuples $\{r_1, r_2, r_4, r_5, r_6, r_7, r_8\}$. The evaluation strategy of θ-MDA works as follows: the result table $\mathbf{x}$ is initialized to $\mathbf{b}$ and one additional column for each aggregate, followed by a scan of $\mathbf{r}$ during which the aggregate values are incrementally updated. A major part of the evaluation cost for range aggregates is the incremental update of the aggregate values in $\mathbf{x}$ since each tuple in $\mathbf{r}$ affects many entries in $\mathbf{x}$. For instance, tuple r_1 affects both aggregates ($C1$ and $C2$) in all result tuples (x_1 to x_4). Moreover, for θ-conditions that reference a subset of the attributes in $\mathbf{b}$, redundant updates occur. Consider $C1$ with θ-condition $D \leq 31/01/13$. Entries x_1 and x_2 are updated for all input tuples with $D \leq 31/01/13$.

In this paper, we tackle these problems and propose an efficient two-step evaluation strategy for ad-hoc range aggregates by reducing them to point aggregates. The first step scans $\mathbf{r}$ and computes corresponding point aggregates as a partial intermediate result, $\tilde{\mathbf{x}}$. Point aggregates require much less incremental updates and can be computed very efficiently since only equality conditions are used to define aggregation tuples. The second step combines the result of the point aggregates with the help of the corresponding super aggregates [7] (e.g., SUM to add up COUNTs) and the θ-conditions to obtain the final result relation. To further reduce the number of updates for aggregates with θ-conditions that reference a subset of the attributes in $\mathbf{b}$, we split the intermediate result

table into a separate table for each aggregate function so that each detail tuple affects exactly one entry in each intermediate table. This two-step strategy reduces the number of incremental updates from $\mathcal{O}(|\mathbf{r}| \cdot |\mathbf{b}|)$ to $\mathcal{O}(|\mathbf{r}|)$ if $|\mathbf{b}| < \sqrt{|\mathbf{r}|}$ and $|\tilde{\mathbf{x}}| \approx |\mathbf{b}|$, which is common for OLAP queries, such as for the TPC-H benchmark. We integrate these optimizations into a new evaluation algorithm, termed $\texttt{TCMDA}^+$. An empirical evaluation confirms the analytical results and shows that range aggregates can be computed with almost the same efficiency as point aggregates.

The technical contributions can be summarized as follows:

- we show how the evaluation of range aggregates can be reduced to the evaluation of point aggregates by computing an intermediate table of corresponding point aggregates, which are then combined to the final aggregates;
- to further reduce the number of updates in the intermediate table, we maintain a separate intermediate table for each aggregate;
- we integrate these optimizations in a new evaluation algorithm, termed $\texttt{TCMDA}^+$, which reduces the runtime complexity from $\mathcal{O}(|\mathbf{r}| \cdot |\mathbf{b}|)$ to $\mathcal{O}(|\mathbf{r}|)$ if $|\mathbf{b}| < \sqrt{|\mathbf{r}|}$ and $|\tilde{\mathbf{x}}| \approx |\mathbf{b}|$;
- we report experimental results that confirm the analytical results and show that range aggregates are evaluated with almost the same efficiency as point aggregates.

The paper is organized as follows: Sec. 2 reports related work, followed by a summary of the θ-MDA operator in Sec. 3. In Sec. 4, we present the reduction of range to point aggregates. These optimizations are integrated in a new algorithm in Sec. 5. In Sec. 6, we report experimental results. Section 7 concludes the paper and points to future work.

2 Related Work

Various research work investigates multi-dimensional data aggregation techniques to gain more flexibility and/or performance. The *CUBE* operator [7] is part of the SQL standard and allows to express aggregation queries with equality constraints over several attributes in a concise way. For aggregation queries over a part of the data cube, grouping sets can be used. Additional support for aggregation queries over one dimension is provided by *window functions* in SQL:2003. The support is based on the ordering of tuples. Currently, SQL does not support the efficient evaluation of range aggregates over multiple dimensions [1].

An orthogonal approach to improve the query performance is to pre-compute aggregates. Harinarayan et al. [9] propose a strategy for the selection of a subset of all possible data cubes to be materialized. To avoid the complete re-computation of cubes when source relations change, incremental update strategies have been proposed [11,12,13,15]. Although the pre-computation of data cubes works well for point aggregates, the performance of range aggregates suffers since the cells in the data cube must be accessed repeatedly. To tackle this problem, Ho et al. [10] propose to maintain additionally a so-called *prefix sum cube*. Subsequent work has studied techniques to lower the comparably high update costs of the prefix sum cube [5,6,14].

The goal in this paper is the efficient evaluation of complex *ad-hoc* OLAP queries when pre-computed aggregates (cubes or prefix sum cubes) are *not* available. Such

an approach provides more flexibility for the exploration of large data sets when the requirements for the analysis and queries are not known a priori.

To efficiently answer ad-hoc OLAP queries with a single scan of the detail table, Akinde et al. [2] propose the *multi-dimensional join* (MDJ) and later the *generalized multi-dimensional join* (GMDJ) [3,4]. The operator has been used in complex OLAP settings to transform general sub-query expressions into expressions that use the GMDJ instead of joins, outer joins, or set difference. Sridhar et al. [16] use the GMDJ in combination with MapReduce to compute aggregation queries over RDF data.

The *θ-constrained multi-dimensional aggregation* (θ-MDA) operator [1] extends the MDJ and presents a detailed cost model together with algebraic transformation rules. θ-MDA outperforms SQL for complex multi-dimensional aggregation queries, such as range aggregates over multiple dimensions. In this paper, we propose an alternative evaluation strategy for θ-MDA, which significantly reduces the cost of the computation of range aggregates to almost the same cost as point aggregates.

3 Preliminaries

We assume two relations, $\mathbf{b}$ and $\mathbf{r}$, with schema $\mathbf{B} = (B_1, \ldots, B_t)$ and $\mathbf{R} = (A_1, \ldots, A_p)$, respectively. For a tuple x we write $x.\mathbf{B}$ as an abbreviation for $(x.B_1, \ldots, x.B_t)$. E/C denotes the renaming of E to C, $attr(\theta)$ denotes the set of attributes used in θ, and f_i denotes an aggregate function.

Definition 1. *(θ-MDA [1]) Let $\mathbf{b}$ and $\mathbf{r}$ be relations with schema $\mathbf{B}$ and $\mathbf{R}$, respectively, $F = (f_1/C_1, \ldots, f_m/C_m)$ be aggregate functions over attributes in $\mathbf{R}$, and $\Theta = (\theta_1, \ldots, \theta_m)$ be conditions with $attr(\theta_i) \subseteq \mathbf{B} \cup \mathbf{R}$. The θ-MDA operator is defined as*

$$\mathcal{G}^{\theta}(\mathbf{b}, \mathbf{r}, F, \Theta) = \{b \circ v \mid b \in \mathbf{b} \wedge v = (f_1(\mathbf{r}_{[b,\theta_1]}), \ldots, f_m(\mathbf{r}_{[b,\theta_m]}))\},$$

where $\mathbf{r}_{[b,\theta_i]} = \{r \in \mathbf{r} \mid \theta_i(r, b)\}$ are the aggregate tuples from which the aggregate values for aggregation group b are computed.

The *base table* $\mathbf{b}$ specifies the aggregation groups for which a result tuple is reported. The *detail table* $\mathbf{r}$ contains the data from which aggregate values are computed. F is a list of aggregate functions. Each f_i gets as argument a subset of $\mathbf{r}$, $\mathbf{r}_{[b,\theta_i]} \subseteq \mathbf{r}$, that is determined by a condition θ_i, and aggregates one of the attributes. Each entry in the *result table* $\mathbf{x}$ consists of a $\mathbf{b}$-tuple and the aggregation results stored in $C_1, \ldots, C_m$. Query Q1 can be formulated as $\mathcal{G}^{\theta}(\mathbf{b}, \mathbf{r}, F, \Theta)$ with $\mathbf{b} = \pi_{D,U}(\sigma_{U \in \{2,3\}}(\mathbf{stays}))$, $\mathbf{r} = \mathbf{stays}$, $F = ((\mathrm{SUM}(S)/\mathrm{COUNT}(S))/C1, (\mathrm{SUM}(S)/\mathrm{COUNT}(S))/C2)$, and $\Theta = (\theta_1, \theta_2)$ with $\theta_1 \equiv (r.D \leq b.D)$ and $\theta_2 \equiv (r.D \leq b.D \wedge r.U \leq b.U)$.

The evaluation of θ-MDA queries works as follows: (1) initialize the result table $\mathbf{x}$ to $\mathbf{b}$ and the neutral value for each aggregate function; (2) scan $\mathbf{r}$ and incrementally update the aggregates f_i in $\mathbf{x}$ that are affected by an $r \in \mathbf{r}$, i.e., satisfy condition θ_i. After processing all $\mathbf{r}$-tuples, $\mathbf{x}$ contains the result relation. The runtime complexity of this evaluation strategy is $\mathcal{O}(|\mathbf{r}| \cdot |\mathbf{b}|)$ with one scan of $\mathbf{r}$. Figure 2 shows the result table during the evaluation of the first three tuples. The first two detail tuples affect all eight aggregate values in $\mathbf{x}$, whereas r_3 requires six updates.

Observe that the number of aggregates that are updated for an $\mathbf{r}$-tuple depends on the θ_i-conditions. Range aggregates require many more updates than point aggregates.

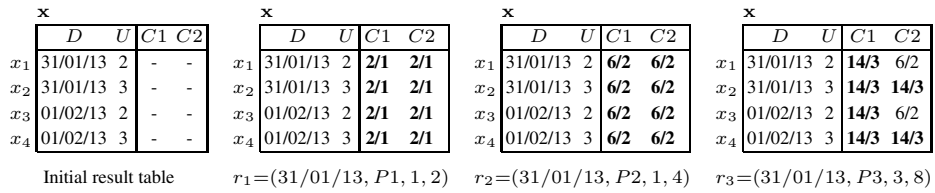


Fig. 2. Processing Tuples in θ-MDA Queries

4 A New Evaluation Strategy for θ-MDA Queries

4.1 Reducing Range to Point Queries

To tackle the problem of a large number of incremental updates for range aggregates and take advantage of the efficient computation of point aggregates, we propose an evaluation strategy that reduces range to point aggregates.

Proposition 1 (Reduction to Point Aggregates). *Let* $\mathbf{b}$, $\mathbf{r}$, $\mathbf{B}$, $\mathbf{R}$, F, Θ *be as in Def. 1,* $G = (g_1, \ldots, g_m)$ *be the super aggregates of the* $f_i \in F$, *and* $\mathbf{R}_i = \mathbf{R} \cap attr(\theta_i)$ *be the attributes in* $\mathbf{R}$ *that occur in* θ_i. *Then,* $\mathbf{x} = \mathcal{G}^\theta(\mathbf{b}, \mathbf{r}, F, \Theta)$ *can be computed as follows:*

1. *construct* $\tilde{\Theta} = (\tilde{\theta}_1, \ldots, \tilde{\theta}_m)$, *where* $\tilde{\theta}_i(r, b) = \bigwedge_{A \in \mathbf{R}_i} r.A = b.A$;
2. *compute an intermediate result table* $\tilde{\mathbf{x}} = \mathcal{G}^\theta(\pi_{\mathbf{R}_1 \cup \cdots \cup \mathbf{R}_m}(\mathbf{r}), \mathbf{r}, F, \tilde{\Theta})$;
3. *compute the result table* $\mathbf{x} = \{b \circ v \mid b \in \mathbf{b} \wedge v = (g_1(\tilde{\mathbf{x}}_{[b,\theta_1]}), \ldots, g_m(\tilde{\mathbf{x}}_{[b,\theta_m]}))\}$, *where* $\tilde{\mathbf{x}}_{[b,\theta_i]} = \pi_{\mathbf{R}_i, C_i}\{\tilde{x} \in \tilde{\mathbf{x}} \mid \theta_i(\tilde{x}, b)\}$.

First, m point aggregates are constructed by creating conditions $\tilde{\Theta} = \{\tilde{\theta}_1, \ldots, \tilde{\theta}_m\}$ such that each $\tilde{\theta}_i$ contains an equality constraint, $r.A = b.A$, for each attribute $A \in \mathbf{R}_i$ that is used in the corresponding θ_i. Second, a $\mathcal{G}^\theta$-call computes an intermediate result table, $\tilde{\mathbf{x}}$, with m point queries, where the base table is a projection of $\mathbf{r}$ to all $\mathbf{R}$-attributes that are used in Θ. This requires significantly less updates in $\tilde{\mathbf{x}}$ than the range queries would do. The final result table, $\mathbf{x}$, is derived from $\tilde{\mathbf{x}}$ using the aggregates g_i in combination with the original conditions θ_i. Following Gray et al. [7], we call the functions g_i that are needed to aggregate the intermediate values the super aggregates. For the standard aggregate functions we have the following pairs of aggregate/super aggregate: MAX/MAX, MIN/MIN, SUM/SUM, COUNT/SUM; average is replaced by sum divided by count. The super aggregates are computed over groups of entries, $\tilde{\mathbf{x}}_{[b,\theta_i]} \subseteq \tilde{\mathbf{x}}$, that are assigned to tuples $b \in \mathbf{b}$ using the original conditions θ_i. Note the projection to the aggregation group attributes $\mathbf{R}_i$ and the aggregate C_i. This is required to eliminate duplicates in situations when a condition $\tilde{\theta}_i$ references only a subset of the aggregation group attributes in $\tilde{\mathbf{x}}$, i.e., $\mathbf{R}_i \subset \mathbf{R}_1 \cup \cdots \cup \mathbf{R}_m$. Although in step 3 each tuple of $\tilde{\mathbf{x}}$ may affect multiple tuples in $\mathbf{x}$, the overall runtime is significantly reduced, provided that $\tilde{\mathbf{x}}$ is much smaller than $\mathbf{r}$, which is frequently the case in OLAP.

Figure 3 shows the evaluation of Query Q1. We have the attribute sets $\mathbf{R}_1 = \{D\}$ and $\mathbf{R}_2 = \{D, U\}$ and the conditions $\tilde{\Theta} = \{\tilde{\theta}_1, \tilde{\theta}_2\}$ with $\tilde{\theta}_1 \equiv (r.D{=}b.D)$ and

$\tilde{\theta}_2 \equiv (r.D{=}b.D \wedge r.U{=}b.U)$. These conditions together with the aggregate functions represent point aggregates, which are computed in the intermediate result table as $\tilde{\mathbf{x}} = \mathcal{G}^{\theta}(\pi_{D,U}(\mathbf{stays}), \mathbf{r}, F, \tilde{\Theta})$. Note the significant reduction of incremental updates in $\tilde{\mathbf{x}}$. For instance, tuple r_1 affects only four aggregates (both aggregates in $\tilde{x}_1$ and aggregate $C1$ in $\tilde{x}_2$ and $\tilde{x}_3$) instead of eight as in Fig. 2. To derive the final result table $\mathbf{x}$, the original conditions, θ_i, are used to determine the subsets $\tilde{\mathbf{x}}_{[b,\theta_i]}$. For result tuple x_1 with aggregation group $b = (31/01/13, 2)$ we have the following subsets:

$$\begin{aligned}
\tilde{\mathbf{x}}_{[(31/01/13,2),\theta_1]} &= \pi_{D,C1}\{\tilde{x} \in \tilde{\mathbf{x}} \mid \tilde{x}.D \leq 31/01/13\} \\
&= \pi_{D,C1}\{\tilde{x}_1, \tilde{x}_2, \tilde{x}_3\} = \{(31/01/13, 24/6)\}, \\
\tilde{\mathbf{x}}_{[(31/01/13,2),\theta_2]} &= \pi_{D,U,C2}\{\tilde{x} \in \tilde{\mathbf{x}} \mid \tilde{x}.D \leq 31/01/13 \wedge \tilde{x}.U \leq 2\} \\
&= \pi_{D,U,C2}\{\tilde{x}_1, \tilde{x}_2\} = \{(31/01/13, 1, 9/3), (31/01/13, 2, 7/2)\}.
\end{aligned}$$

The projection in $\tilde{\mathbf{x}}_{[(31/01/13,2),\theta_1]}$ removes duplicates that originate from $C1$ grouping only by D. Since the super aggregate of SUM and COUNT is SUM, the final aggregates are obtained by summing up the individual sums and counts, respectively. For instance, for the result tuple x_1 we get

$$\begin{aligned}
x_1.C1 &= \text{SUM/SUM}_{C1}(\tilde{\mathbf{x}}_{[(31/01/13,2),\theta_1]}) = \text{SUM/SUM}_{C1}(\{(31/01/13, 24/6)\}) \\
&= 24/6, \\
x_1.C2 &= \text{SUM/SUM}_{C2}(\tilde{\mathbf{x}}_{[(31/01/13,2),\theta_2]}) \\
&= \text{SUM/SUM}_{C2}(\{(31/01/13, 1, 9/3), (31/01/13, 2, 7/2)\} \\
&= (9{+}7)/(3{+}2) = 16/5.
\end{aligned}$$

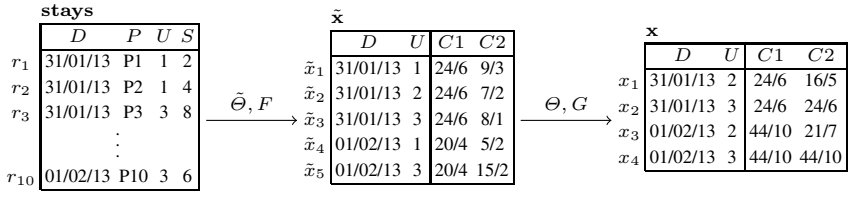


Fig. 3. θ-MDA Evaluation by Reduction to Point Queries

Theorem 1. *The evaluation strategy in Proposition 1 correctly computes the θ-MDA.*

Proof. The theorem applies the following reduction rule for distributive aggregates: an aggregate function, f, over a set of tuples, $\mathbf{r}$, can be reduced to the computation of partial aggregates over a partitioning $\mathbf{r}_1, \ldots, \mathbf{r}_k$ of $\mathbf{r}$, followed by an application of the corresponding super aggregate g, i.e., $f(\mathbf{r}) = g(f(\mathbf{r}_1), \ldots, f(\mathbf{r}_k))$. In step two, each $\tilde{\theta}_i$ induces a partitioning of $\mathbf{r}$ such that all tuples in the same partition have identical values for the attributes $\mathbf{R}_i = \mathbf{R} \cap attr(\theta_i)$. The call to $\mathcal{G}^{\theta}$ with base table $\pi_{\mathbf{R}_1 \cup \cdots \cup \mathbf{R}_m}(\mathbf{r})$ computes for each f_i the partial aggregation results over the individual partitions, i.e., $\tilde{x}.C_i = f_i(\{r \in \mathbf{r} \mid \tilde{\theta}(r, \tilde{x})\})$. In step three, the original $\mathbf{b}$ and θ_is are used to determine which entries $\tilde{x} \in \tilde{\mathbf{x}}$ to combine for each $b \in \mathbf{b}$. Each $\tilde{\mathbf{x}}_{[b,\theta_i]}$ collects all intermediate

tuples, $\tilde{x} \in \tilde{\mathbf{x}}$, that are assigned to b through θ_i. Since all $\mathbf{r}$-tuples in a partition that contributed to an intermediate aggregate, $\tilde{x}.C_i$, have identical values for the attributes $\mathbf{R}_i$, they would have been assigned by θ_i to the same b. The projection to $\mathbf{R}_i, C_i$ eliminates duplicates when $\mathbf{R}_i \subset \mathbf{R}_1 \cup \cdots \cup \mathbf{R}_m$. In such cases, several entries in $\tilde{\mathbf{x}}$ might store the same aggregation group for C_i, but only one can be considered for the final aggregation result. Thus, no spurious tuples are combined for the aggregates in the final result table, and the use of the original θ_i in step three guarantees that all input tuples are considered. □

The following lemma shows that the reduction of range aggregates to point aggregates can be expressed by a nested θ-MDA if all θ_is use the same set of aggregation group attributes $\mathbf{B}$.

Lemma 1. *Let* $\mathbf{b}$, $\mathbf{r}$, F, G, Θ, *and* $\tilde{\Theta}$ *be as in Proposition 1. Furthermore, let* $\mathbf{R}_i = \mathbf{R} \cap attr(\theta_i)$ *denote the attributes in* $\mathbf{R}$ *that occur in* θ_i *and* $\mathbf{R}_i = \mathbf{R}_j$ *for all* i, j, $1 \leq i, j \leq m$. *Then,* $\mathbf{x} = \mathcal{G}^\theta(\mathbf{b}, \mathbf{r}, F, \Theta)$ *can be computed as*

$$\mathbf{x} = \mathcal{G}^\theta(\mathbf{b}, \mathcal{G}^\theta(\pi_{\mathbf{R}_i}(\mathbf{r}), \mathbf{r}, F, \tilde{\Theta}), G, \Theta).$$

Proof. The inner $\mathcal{G}^\theta$-call computes the intermediate table $\tilde{\mathbf{x}}$, which is passed as detail table to the outer $\mathcal{G}^\theta$-call. In the outer call, the original θ_is assign to each $b \in \mathbf{b}$ the associated intermediate tuples, $\tilde{x} \in \tilde{\mathbf{x}}$. Since all aggregates use the same aggregation group attributes, $\mathbf{R}_i$, there are no duplicate aggregation groups in $\tilde{\mathbf{x}}$ for any aggregate, hence a projection as in Proposition 1 is not necessary. The super aggregates in G in combination with the original conditions Θ correctly combine the partial aggregates to the final result. □

Note that using the aggregation groups in $\mathbf{b}$ instead of $\pi_{\mathbf{R}_1 \cup \cdots \cup \mathbf{R}_m}(\mathbf{r})$ for the computation of the intermediate result table $\tilde{\mathbf{x}}$ would not be correct. This happens if $\mathbf{b}$ is sparse, i.e., $\mathbf{r}$ contains more combinations of the grouping attribute values than $\mathbf{b}$ does, i.e., $\mathbf{b} \subset \pi_{\mathbf{B}}(\mathbf{r})$. Since the $\tilde{\theta}_i$s use only equality constraints, some tuples in $\mathbf{r}$ might not be assigned to any entry in the intermediate table $\tilde{\mathbf{x}}$ although they contribute to the final aggregation result. Figure 4 shows table $\tilde{\mathbf{x}}$ for Query Q1 when using the original base table $\mathbf{b}$. In table $\tilde{\mathbf{x}}$, the pre-aggregates for the aggregation groups $(31/01/13, 1)$ and $(01/02/13, 1)$ are missing, but they are needed for the correct computation of the final result. A similar situation occurs if Θ uses a lower number of $\mathbf{B}$-attributes than $\mathbf{R}$-attributes. In this case, the intermediate table $\tilde{\mathbf{x}}$ misses attributes that are needed for the evaluation of the θ_is when producing the final result table.

4.2 Separate Intermediate Result Tables

Even if the $\tilde{\theta}_i$-conditions contain only equality constraints, a single $\mathbf{r}$-tuple might still affect several entries in $\tilde{\mathbf{x}}$. This is the case if a $\tilde{\theta}_i$ constrains only a subset of all grouping attributes, i.e., $\mathbf{R}_i \subset \mathbf{R}_1 \cup \cdots \cup \mathbf{R}_m$. For instance, condition $\tilde{\theta}_1 \equiv (r.D{=}b.D)$ of the first aggregate groups only by D. This produces duplicate aggregation groups in $\tilde{\mathbf{x}}$, such as $D = 31/01/13$ which is present in $\tilde{x}_1$, $\tilde{x}_2$, and $\tilde{x}_3$ (cf. Fig. 3). Each of the detail tuples $r_1, \ldots, r_6$ is assigned to each of these entries, yielding a total of 18 updates of

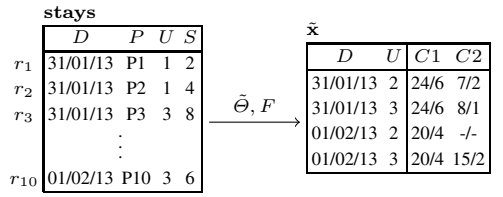


Fig. 4. Using Base Table $\mathbf{b}$ in $\tilde{\mathbf{x}}$

$C1$ instead of six. This type of *redundant* updates can be avoided by using a separate intermediate result table, $\tilde{\mathbf{x}}^i$, for each $\tilde{\theta}_i$ (f_i).

Proposition 2 (Separate Intermediate Result Tables). *Let* $\mathbf{b}$, $\mathbf{r}$, F, G, Θ, $\tilde{\Theta}$, *and* $\mathbf{R}_i$ *be as in Proposition 1. Then,* $\mathbf{x} = \mathcal{G}^\theta(\mathbf{b}, \mathbf{r}, F, \Theta)$ *can be computed as follows:*

1. *compute* m *intermediate result tables* $\tilde{\mathbf{x}}^i = \mathcal{G}^\theta(\pi_{\mathbf{R}_i}(\mathbf{r}), \mathbf{r}, f_i, \tilde{\theta}_i)$ *for* $i = 1, \ldots, m$*;*
2. *compute the result table* $\mathbf{x} = \{b \circ f \mid b \in \mathbf{b} \wedge f = (g_1(\tilde{\mathbf{x}}^1_{[b,\theta_1]}), \ldots, g_m(\tilde{\mathbf{x}}^m_{[b,\theta_m]}))\}$, *where* $\tilde{\mathbf{x}}^i_{[b,\theta_i]} = \{\tilde{x} \in \tilde{\mathbf{x}}^i \mid \theta_i(\tilde{x}, b)\}$.

Figure 5 shows the two intermediate result tables, $\tilde{\mathbf{x}}^1$ and $\tilde{\mathbf{x}}^2$, in our running example that replace table $\tilde{\mathbf{x}}$ from Fig. 3. Table $\tilde{\mathbf{x}}^1$ has one grouping attribute, whereas $\tilde{\mathbf{x}}^2$ has two. The detail tuples $r_1, \ldots, r_6$ require now a total of six updates of $C1$ in table $\tilde{\mathbf{x}}^1$ (one for each tuple), instead of 18 in Fig. 3.

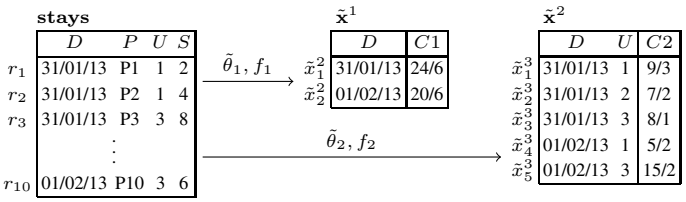


Fig. 5. Separate Intermediate Result Tables

Theorem 2. *The evaluation strategy in Proposition 2 correctly computes* θ*-MDA.*

Proof. The proof is similar as for Theorem 1 with two differences. In the first step, m intermediate result tables are constructed using the $\tilde{\theta}_i$s. Since the aggregation groups of each $\tilde{\mathbf{x}}^i$ are produced by a projection of $\mathbf{r}$ to the attributes $\mathbf{R}_i$ that are used in the corresponding θ_i-condition, duplicate aggregation groups are avoided. The second step merges the intermediate tables to produce the final result table in the same way as in Proposition 1, except the projection in $\tilde{\mathbf{x}}_{[b,\theta_i]}$, which is not needed since no duplicate aggregation groups exist. □

Corollary 1. *The evaluation of* θ*-MDA queries using separate intermediate result tables,* $\tilde{\mathbf{x}}^i$*, as in Proposition 2, requires for each* $r \in \mathbf{r}$ *exactly one update in each* $\tilde{\mathbf{x}}^i$*.*

Proof. None of the intermediate result tables, $\tilde{\mathbf{x}}^i$, contains duplicate aggregation groups and all aggregates are point queries. Thus, each condition, $\tilde{\theta}_i$, associates each detail tuple, $r \in \mathbf{r}$, to exactly one entry in $\tilde{\mathbf{x}}^i$. □

5 Algorithm TCMDA$^+$

Algorithm 1 shows a new algorithm, TCMDA$^+$, for the evaluation of θ-MDA queries that adopts the optimization techniques introduced before.

Algorithm 1. TCMDA$^+(\mathbf{b}, \mathbf{r}, F, \Theta)$

input : base table $\mathbf{b}$, detail table $\mathbf{r}$, aggregate functions $F = (f_1, \ldots, f_m)$, conditions $\Theta = (\theta_1, \ldots, \theta_m)$
output : result relation $\mathbf{x}$

```
// Initialize intermediate result tables
```
Let $\mathbf{R}_i \leftarrow \mathbf{R} \cap attr(\theta_i)$ for $i = 1, \ldots, m$;
Let $(\mathbf{R}_{j_1}, F_{j_1}), \ldots, (\mathbf{R}_{j_k}, F_{j_k})$, $k \leq m$, be a partitioning of F according to $\mathbf{R}_i$;
foreach *partition* $(\mathbf{R}_j, F_j)$ **do**
  $\tilde{\mathbf{x}}^j \leftarrow$ empty table with schema $(\mathbf{R}_j, C_{j_1}, \ldots, C_{j_{k_j}})$;
  Create an index on $\tilde{\mathbf{x}}^j$ over the attributes $\mathbf{R}_j$;
  $\tilde{\theta}_j(r, b) = \bigwedge_{A \in \mathbf{R}_j} r.A = b.A$;

```
// Scan detail table r and update intermediate result tables
```
foreach *tuple* $r \in \mathbf{r}$ **do**
  foreach *partition* $(\mathbf{R}_j, F_j)$ **do**
    if $\exists \tilde{x} \in \tilde{\mathbf{x}}^j$ *such that* $\tilde{\theta}_j(r, \tilde{x})$ **then**
      $\tilde{x}.C_{j_i} \leftarrow g_{j_i}(\tilde{x}.C_{j_i}, f_{j_i}(\{r\}))$ for $i = 1, \ldots, k_j$;
    else
      $\tilde{\mathbf{x}}^j \leftarrow \tilde{\mathbf{x}}^j \cup \{r.\mathbf{R}_j \circ (f_{j_1}(\{r\}), \ldots, f_{j_{k_j}}(\{r\}))\}$;

```
// Build final result table x
```
$\mathbf{x} = \mathbf{b} \times \{(v_1, \ldots, v_m)\}$;
Create index on $\mathbf{x}$ over attributes $\mathbf{B}$;
for $i = 1$ **to** m **do**
  foreach $\tilde{x} \in \tilde{\mathbf{x}}^i$ **do**
    foreach $x \in \mathbf{x}$ *such that* $\theta_i(\tilde{x}, x)$ **do**
      $x.C_i \leftarrow g_i(x.C_i, \tilde{x}.C_i)$;

return $\mathbf{x}$;

The algorithm starts with the initialization of empty intermediate result tables. According to Proposition 2, for each $\tilde{\theta}_i$ a separate table is created. This leads to tables with identical grouping attributes if different θ_is reference the same attributes in $\mathbf{R}$, i.e., $\mathbf{R}_i = \mathbf{R}_j$ for $i \neq j$. Therefore, in the algorithm we apply a further optimization and merge tables with identical grouping attributes to a single table with one column for each aggregate function. For each intermediate result table, $\tilde{\mathbf{x}}^j$, constructed in this way we create an index over the grouping attributes. The conditions $\tilde{\theta}_j$ are generated as described in Proposition 1. Next, the detail table is scanned, and for each $r \in \mathbf{r}$ the aggregates in the intermediate result tables, $\tilde{\mathbf{x}}^j$, are updated. If an entry in $\tilde{\mathbf{x}}^j$ exists that matches tuple r, the aggregates are incrementally updated. Otherwise, a new entry is created and the aggregate values are initialized to the functions evaluated over r. Finally, the result table $\mathbf{x}$ is initialized to $\mathbf{b}$ with the aggregates initialized to the neutral

values v_i. The final result table is computed by combining the partial aggregates from the intermediate result tables $\tilde{\mathbf{x}}^j$ using the super aggregates as described in Proposition 2. For that the intermediate result tables are scanned and the aggregate values in the final result table are incrementally updated.

Figure 6 illustrates a few steps of the computation of Query Q1. Empty intermediate result tables $\tilde{\mathbf{x}}^1$ and $\tilde{\mathbf{x}}^2$ are created for the partitions $(\{D\}, \{f_1\})$ and $(\{D, U\}, \{f_2\})$. The first tuple r_1 creates a new entry in both tables. Tuple r_2 creates no new entries in any of the intermediate tables, it only updates aggregates. Tuple r_3 updates $C1$ in $\tilde{\mathbf{x}}^1$ and creates a new entry in $\tilde{\mathbf{x}}^2$. After processing r_{10}, the intermediate tables contain the same partial aggregate values as in Fig. 5.

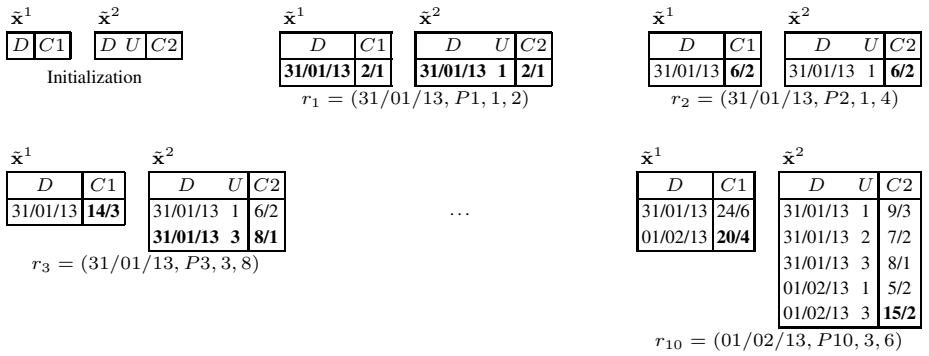


Fig. 6. Processing of Detail Tuples and Computation of Intermediate Result Tables in TCMDA$^+$

Complexity Analysis. We analyze the complexity of the TCMDA$^+$ algorithm in terms of incremental updates to the aggregate values and compare it to the original TCMDA algorithm [1]. As parameters we consider the two input relations, $\mathbf{b}$ and $\mathbf{r}$, while the number of aggregate functions and θ-conditions are considered to be constant.

The complexity of TCMDA is $C_{\texttt{TCMDA}} = |\mathbf{b}| + |\mathbf{r}| \cdot u$, where u is the average number of updates in the result table $\mathbf{x}$. The number of updates depends on the constraints in the θ_is and ranges between 0 and $|\mathbf{b}|$. For range aggregates, u is much higher than for point aggregates. The complexity of TCMDA$^+$ is $C_{\texttt{TCMDA}^+} = |\mathbf{b}| + |\mathbf{r}| + |\tilde{\mathbf{x}}| \cdot u$, where $|\tilde{\mathbf{x}}|$ is the size of the largest intermediate result table and u is the average number of updates in the result table $\mathbf{x}$. The number of updates for each $r \in \mathbf{r}$ in the intermediate tables $\tilde{\mathbf{x}}^i$ is always one due to the reduction to point aggregates and the use of separate intermediate tables. The computation of the final result table $\mathbf{x}$ requires on average u updates for each $\tilde{x} \in \tilde{\mathbf{x}}$, where $|\tilde{\mathbf{x}}| \leq |\mathbf{r}|$ and u ranges between 1 and $|\mathbf{b}|$.

The worst case complexity of TCMDA is $\mathcal{O}(|\mathbf{r}| \cdot |\mathbf{b}|)$. For the TCMDA$^+$ algorithm, we distinguish three cases. First, for $|\mathbf{b}| < \sqrt{|\mathbf{r}|}$ and $|\tilde{\mathbf{x}}| \approx |\mathbf{b}|$, which is common for OLAP, we get a worst case complexity of $\mathcal{O}(\mathbf{r})$, and thus a significant improvement over TCMDA. Second, for $|\mathbf{b}| > \sqrt{|\mathbf{r}|}$ and $|\tilde{\mathbf{x}}| \approx |\mathbf{b}|$ we have $\mathcal{O}(|\mathbf{b}|^2)$. Third, if $|\tilde{\mathbf{x}}| \gg |\mathbf{b}|$ (or almost as large as $|\mathbf{r}|$) the algorithm degrades to $\mathcal{O}(|\mathbf{r}| \cdot |\mathbf{b}|)$ and has the same complexity as TCMDA.

To summarize, for typical applications of θ-MDA, where $|\mathbf{b}| < \sqrt{|\mathbf{r}|}$ and $|\tilde{\mathbf{x}}| \approx |\mathbf{b}|$, the proposed optimization reduces the complexity of the θ-MDA evaluation from $\mathcal{O}(|\mathbf{r}| \cdot |\mathbf{b}|)$ to $\mathcal{O}(|\mathbf{r}|)$ for both range and point aggregates.

6 Experiments

Setup and Data. We implemented the algorithms TCMDA from [1] and TCMDA$^+$ described in this paper in C using Oracle 11g for storing the data. The experiments run on a machine with two AMD Opteron processors (1.8 GHz and 2.6 GHz), 16 GB of main memory, and Ubuntu 10.04. For the experiments we used the *Orders* table of the TPC-H benchmark[1]. We generated tables of different size and ran queries over them using the aggregate function COUNT.

Varying the Size of the Detail Table. Figure 7 presents the runtime by varying the size of the detail table between 2 and 10 million tuples. The θ-conditions use the operators $\leq$ and $\neq$ (range aggregates). In all experiments, TCMDA$^+$ clearly outperforms TCMDA, and the runtime of TCMDA grows faster than for TCMDA$^+$. This improvement of up to a factor of five can be attributed to the reduction to point aggregates, which reduces the number of updates for each detail tuple to one. As expected, the less selective the θ-constraints are, the bigger the performance improvement since for less selective conditions TCMDA needs to update comparably more aggregates for each $r \in \mathbf{r}$.

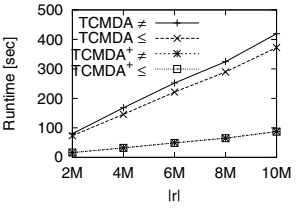
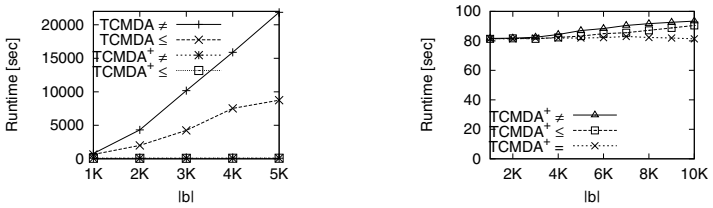


Fig. 7. Varying $|\mathbf{r}|$ ($|\mathbf{b}| = 500$, $|\Theta| = 1$, $|\theta| = 1$)

Varying the Size of the Base Table. The experimental results with a varying size of the base table are shown in Fig. 8 and exhibit an even better performance improvement than for varying $|\mathbf{r}|$. The growing size of $\mathbf{b}$ affects the runtime of TCMDA drastically, as shown in Fig. 8(a). For $|\mathbf{b}| = 5000$ the runtime is about 13 times larger for $\leq$ and about 27 larger for $\neq$ with respect to $|\mathbf{b}| = 1000$. In contrast, the runtime of TCMDA$^+$ is not affected by the growing base table. Figure 8(b) shows the runtime of TCMDA$^+$ with a larger base table varying between 2000 and 1000 tuples. The experiment confirms our analytical results that the runtime of TCMDA$^+$ is growing slowly for $\leq$ and $\neq$ queries when $|\mathbf{b}| > \sqrt{|\mathbf{r}|} \approx 3000$; for $=$ (point queries) the runtime remains constant.

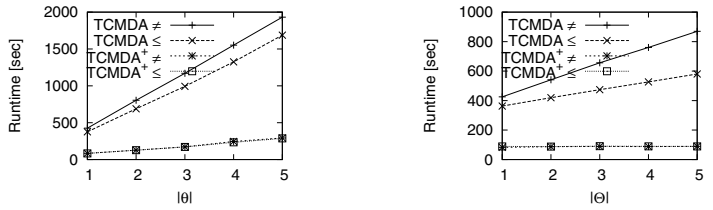
[1] TPC-H benchmark framework: http://www.tpc.org/tpch/



(a) Both algorithms (b) Only TCMDA$^+$

Fig. 8. Varying $|\mathbf{b}|$ ($|\mathbf{r}| = 10M$, $|\Theta| = 1$, $|\theta| = 1$)

Varying the Conditions. In Fig. 9, the number of constraints in the conditions θ_i and the number of θ-conditions in Θ are varied, respectively. Again, TCMDA$^+$ clearly outperforms TCMDA. The runtime for θ-conditions with a number of constraints that varies between 1 and 5 is shown in Fig. 9(a). The increase in the runtime is due to the more expensive evaluation of the θ-conditions containing more constraints. Fig. 9(b) illustrates the experimental results for a growing number of conditions in Θ. A higher number of conditions results in more aggregates to be computed, hence more incremental updates are required. For TCMDA, additional θ-conditions result in significantly more aggregates to be updated for each $\mathbf{r}$-tuple. TCMDA$^+$ is less affected because each additional θ-condition means only one more aggregate update for each $\mathbf{r}$-tuple.

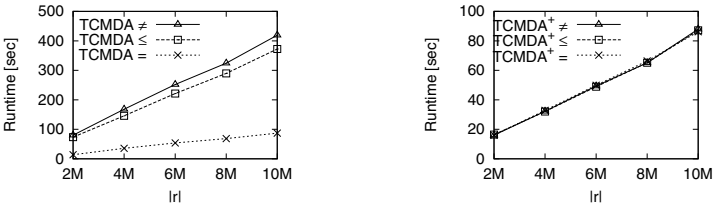


(a) Varying Size of θs (b) Varying Number of θs

Fig. 9. Varying Conditions ($|\mathbf{r}| = 10M$, $|\mathbf{b}| = 500$, $|\Theta| = 1$)

Comparing Different Constraint Types. In Fig. 10, we analyze how the constraint operators $=$, $\leq$, and $\neq$ affect the runtime (i.e., range vs. point aggregates). TCMDA is very sensitive to the type of constraint operators that are used, yielding runtimes that for $\leq$ and $\neq$ are much longer than for $=$ (see Fig. 10(a)). This is due to the higher number of aggregate updates for each $\mathbf{r}$-tuple in range aggregates (i.e., lower selectivity of the operator). In contrast, TCMDA$^+$ in Fig 10(b) is robust and independent of the type of constraint operators. Range aggregates are computed with the same efficiency as point aggregates.

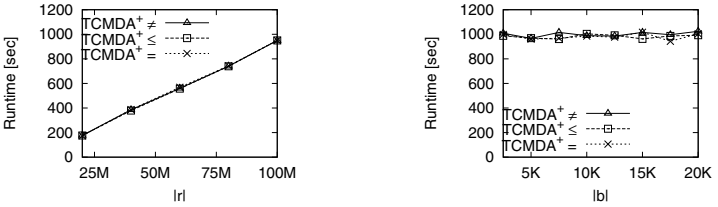
Figure 11 analyzes TCMDA$^+$ and different types of constraints for larger detail and base tables. Both graphs confirm the analytical results, i.e., the runtime of TCMDA$^+$ is linear in $|\mathbf{r}|$ when $|\mathbf{b}| \leq \sqrt{|\mathbf{r}|} = 10000$. In Fig. 11(a), the detail table varies between



(a) TCMDA (b) TCMDA$^+$

Fig. 10. Varying $|\mathbf{r}|$ ($|\mathbf{b}| = 500$, $|\Theta| = 1$, and $|\theta| = 1$)

25 and 100 million tuples, with a base table size of 1000 and one θ-condition with one constraint. The runtime shows a linear growth and is not affected by the type of the query. In Fig. 11(b), the size of the base table varies between 2500 and 20000 tuples, with a detail table of 100 million tuples and one θ-condition with one constraint. The runtime is constant for $|\mathbf{b}| \leq 10000$. For larger base tables $\mathbf{b}$, the runtime is slowly increasing, though the increase is less evident than in Fig. 8(b), where the base table is comparably larger.



(a) Varying $|\mathbf{r}|$ ($|\mathbf{b}|$=1000) (b) Varying $|\mathbf{b}|$ ($|\mathbf{r}|$=100M)

Fig. 11. TCMDA$^+$ Scalability Experiments

7 Conclusions

In this paper, we studied the efficient evaluation of complex ad-hoc multidimensional aggregation queries with the θ-MDA operator. We proposed a solution to reduce the evaluation of θ-MDA range aggregates to θ-MDA point aggregates. Point aggregates require significantly less incremental updates. This optimization has been integrated into a new evaluation algorithm, termed TCMDA$^+$, which reduces the runtime complexity from $\mathcal{O}(|\mathbf{r}| \cdot |\mathbf{b}|)$ to $\mathcal{O}(|\mathbf{r}|)$ if $|\mathbf{b}| < |\sqrt{\mathbf{r}}|$ and $\tilde{\mathbf{x}} \approx \mathbf{b}$, which is common in OLAP queries, such as for the TPC-H benchmark. Extensive experiments have shown performance improvements of more than an order of magnitude for TCMDA$^+$, and range aggregates can be computed with almost the same performance as point aggregates.

Future work points in several directions. First, we will investigate the integration of θ-MDA as an algebraic operator into the kernel of PostgreSQL. Second, we plan to adapt the evaluation strategy for θ-MDA queries and leverage MapReduce techniques for distributed query processing. Finally, it could be interesting to identify applications with very large base tables and develop optimization strategies for such settings.

Acknowledgments. We are indebted to Andreas Heinisch for providing us the C implementation of the algorithms and his support to run the experiments.

This work was funded in part by the Swiss National Science Foundation (SNSF) through the Tameus project (proposal no 200021_135361) and the Autonomous Province of Bozen-Bolzano through the AQUIST project.

References

1. Akinde, M., Böhlen, M.H., Chatziantoniou, D., Gamper, J.: θ-constrained multi-dimensional aggregation. Information Systems 36, 341–358 (2011)
2. Akinde, M., Chatziantoniou, D., Johnson, T., Kim, S.: The MD-join: An operator for complex OLAP. In: Proceedings of ICDE, Washington, DC, USA, pp. 524–533 (2001)
3. Akinde, M.O., Böhlen, M.H.: Generalized MD-joins: Evaluation and reduction to SQL. In: Jonker, W. (ed.) Databases in Telecommunications II. LNCS, vol. 2209, pp. 52–67. Springer, Heidelberg (2001)
4. Akinde, M.O., Böhlen, M.H., Johnson, T., Lakshmanan, L.V.S., Srivastava, D.: Efficient OLAP query processing in distributed data warehouses. Information Systems 28, 111–135 (2003)
5. Chun, S.-J., Chung, C.-W., Lee, J.-H., Lee, S.-L.: Dynamic update cube for range-sum queries. In: VLDB, pp. 521–530 (2001)
6. Geffner, S., Agrawal, D., Abbadi, A.E., Smith, T.R.: Relative prefix sums: An efficient approach for querying dynamic olap data cubes. In: ICDE, pp. 328–335 (1999)
7. Gray, J., Bosworth, A., Layman, A., Reichart, D., Pirahesh, H.: Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. In: Proceedings of ICDE, Washington, DC, USA, pp. 152–159 (1996)
8. Gupta, A., Harinarayan, V., Quass, D.: Aggregate-query processing in data warehousing environments. In: VLDB, pp. 358–369 (1995)
9. Harinarayan, V., Rajaraman, A., Ullman, J.D.: Implementing data cubes efficiently. In: Proceedings of SIGMOD, New York, NY, USA, pp. 205–216 (1996)
10. Ho, C.-T., Agrawal, R., Megiddo, N., Srikant, R.: Range queries in OLAP data cubes. In: Proceedings of SIGMOD, Tucson, Arizona, USA, May 13-15, pp. 73–88 (1997)
11. Hurtado, C.A., Mendelzon, A.O., Vaisman, A.A.: Maintaining data cubes under dimension updates. In: ICDE, pp. 346–355. IEEE Computer Society (1999)
12. Lee, K.Y., Kim, M.H.: Efficient incremental maintenance of data cubes. In: Proceedings of the VLDB Conference, pp. 823–833 (2006)
13. Lehner, W., Sidle, R., Pirahesh, H., Cochrane, R.: Maintenance of automatic summary tables. In: Proceedings of SIGMOD, pp. 512–513 (2000)
14. Liang, W., Wang, H., Orlowska, M.E.: Range queries in dynamic OLAP data cubes. Data Knowl. Eng. 34(1), 21–38 (2000)
15. Mumick, B.S., Quass, D., Mumick, B.S.: Maintenance of data cubes and summary tables in a warehouse. In: Proceedings of SIGMOD, New York, NY, USA, pp. 100–111 (1997)
16. Sridhar, R., Ravindra, P., Anyanwu, K.: RAPID: Enabling scalable ad-hoc analytics on the semantic web. In: Bernstein, A., Karger, D.R., Heath, T., Feigenbaum, L., Maynard, D., Motta, E., Thirunarayan, K. (eds.) ISWC 2009. LNCS, vol. 5823, pp. 715–730. Springer, Heidelberg (2009)