PERGAMON

Information Systems 28 (2003) 111–135

www.elsevier.com/locate/infosys

Efficient OLAP query processing in distributed data warehouses ☆

Michael O. Akinde[a,*], Michael H. Böhlen[a], Theodore Johnson[b], Laks V.S. Lakshmanan[c], Divesh Srivastava[b]

[a] *Department of Computer Science, Aalborg University, Fredrik Bajers Vej 7E, DK-9220 Aalborg, Denmark*
[b] *AT&T Labs–Research, P.O. Box 971, Florham Park, NJ 07932-0971, USA*
[c] *Department of Computer Science, The University of British Columbia, 2329 West Mall, Vancouver, B.C., Canada V6T 1Z4*

Received 1 May 2002; received in revised form 15 August 2002; accepted 16 August 2002

Abstract

The success of Internet applications has led to an explosive growth in the demand for bandwidth from Internet Service Providers. Managing an Internet protocol network requires collecting and analyzing network data, such as flow-level traffic statistics. Such analyses can typically be expressed as OLAP queries, e.g., correlated aggregate queries and data cubes. Current day OLAP tools for this task assume the availability of the data in a centralized data warehouse. However, the inherently distributed nature of data collection and the huge amount of data extracted at each collection point make it impractical to gather all data at a centralized site. One solution is to maintain a distributed data warehouse, consisting of local data warehouses at each collection point and a coordinator site, with most of the processing being performed at the local sites. In this paper, we consider the problem of efficient evaluation of OLAP queries over a distributed data warehouse. We have developed the Skalla system for this task. Skalla translates OLAP queries, specified as certain algebraic expressions, into distributed evaluation plans which are shipped to individual sites. A salient property of our approach is that only partial results are shipped — never parts of the detail data. We propose a variety of optimizations to minimize both the synchronization traffic and the local processing done at each site. We finally present an experimental study based on TPC-R data. Our results demonstrate the scalability of our techniques and quantify the performance benefits of the optimization techniques that have gone into the Skalla system.

1. Introduction

The success of Internet applications has led to an explosive growth in the demand for bandwidth from Internet Service Providers (ISPs), thus increasing the size and complexity of Internet Protocol (IP) networks. Managing such a network involves debugging performance problems, optimizing the configuration of routing protocols, and planning the roll-out of new capacity, to name a

☆ Recommended by X.

*Corresponding author. Now at: MHO Data Warehouse Unit, Computer Science Department, Swedish Meteorological and Hydrological Institute (SMHI), Folkborgsvägen 1, SE-601 76 Norrköping, Sweden. Tel.: +46-11-4958667; fax: +46-11-4958001.

E-mail addresses: michael.akinde@smhi.se (M.O. Akinde), boehlen@cs.auc.dk (M.H. Böhlen), johnsont@research.att.com (T. Johnson), divesh@research.att.com (D. Srivastava), laks@cs.ubc.ca (L.V.S. Lakshmanan).

0306-4379/02/$ - see front matter
PII: S0306-4379(02)00051-0

few tasks. These tasks require the ISP to take into account the topology and usage of the network; a daunting task, particularly in the face of varying traffic on the network. Effective management of a network thus requires collecting, correlating, and analyzing a variety of network trace data.

Typically, trace data such as packet headers, flow-level traffic statistics, and router statistics are collected using tools like packet sniffers, NetFlow-enabled routers, and SNMP polling of network elements. A wide variety of analyses are then performed to characterize the usage and behavior of the network (see, e.g., [1,2]). For example, using flow-level traffic statistics data one can ask questions like: "On an hourly basis, what fraction of the total number of flows is due to Web traffic?", "On an hourly basis, what fraction of the total traffic flowing into the network is from IP subnets whose total hourly traffic is within 10% of the maximum?", or "For each possible combination of source and destination (autonomous) systems, how many flows exceed the average flow duration each day?" Currently, such analyses are usually implemented by the networking community in an ad hoc manner using complex algorithms coded in procedural programming languages like Perl. They can actually be expressed as OLAP queries, including SQL grouping/aggregation, data cubes [3], using marginal distributions extracted by the `unpivot` operator [4], and multi-feature queries [5,6]. Indeed, leveraging such a well-developed technology can greatly facilitate and speed up network data analysis.

A serious impediment to the use of current-day OLAP tools for analyzing network trace data is that the tools require all the data to be available in a single, centralized data warehouse. The inherently distributed nature of data collection (e.g., flow-level statistics are gathered at network routers, spread throughout the network) and the huge amount of data extracted at each collection point (of the order of several gigabytes per day for large IP networks), make it impractical to gather all this data at a single centralized data warehouse, for example, Feldmann et al. [2] report that use of a single centralized collection server for NetFlow data resulted in a loss of up to 90% of NetFlow tuples during heavy load periods!

The natural solution to this problem is to maintain a *distributed data warehouse*, where data gathered at each collection point (e.g., router) is maintained at a local data warehouse, adjacent to the collection point, to avoid loss of collected trace data. For such a solution to work, we need a technology for *distributed processing of complex OLAP queries*—something that does not yet exist. The goal of this paper is to take the first steps in this important direction.

1.1. Outline and contributions

The rest of this paper is organized as follows. We first present related work in Section 1.2. In Section 2, we describe a motivating application and define the GMDJ operator for expressing OLAP queries. Our technical contributions are as follows:

- We present a general strategy for the distributed evaluation of relational OLAP queries, specified as GMDJ expressions, and present the Skalla system, developed by us for this task (Section 3).
- We develop and define optimization strategies for distributed OLAP that can exploit distribution knowledge, if known, as well as strategies that do not assume any such knowledge, to minimize both the synchronization traffic, and the local processing done at each site (Section 4).
- We conducted a series of experiments, based on TPC-R data, to study the performance of the Skalla approach. Our results show the effectiveness of our strategies for distributed OLAP query processing, and also quantify the performance benefits of our optimizations. This demonstrates the validity of the Skalla approach (Section 5).

While we illustrate our techniques using examples drawn from the network management application, our approach and results are more generally applicable to *distributed* data warehouses and OLAP query processing in other application domains as well (e.g., with heterogeneous data marts distributed across an enterprise). To the best

of our knowledge, ours is the first paper on this important topic.

1.2. Related work

The most closely related prior work is that of Shatdal and Naughton [7], who use a similar coordinator/sites model for the parallel evaluation of aggregates, and present various strategies where the aggregate computation is split between the sites and the coordinator, to optimize performance. Aggregates are also considered in a number of parallel database systems, such as in [8,9]. There are two main differences with our work. First, their results are tuned for a parallel computer, where communication is assumed to be very cheap, which is certainly not the case in our distributed data warehouse setting. Second, they deal only with the case of simple SQL aggregates, while we consider significantly more complex OLAP queries.

A variety of OLAP queries have been proposed in the literature, allowing a fine degree of control over both the group definition and the aggregates that are computed using operators such as `cube by` [3], `unpivot` [4], and other SQL extensions (e.g., [5,10,11]). Chatziantoniou et al. [12] proposed the MDJ operator for complex OLAP queries, which provides a clean separation between group definition and aggregate computation, allowing considerable flexibility in the expression of OLAP queries. The processing and optimization of these complex OLAP queries has received a great deal of attention in recent years (see, e.g., [3,5,6,12–16]), but it has all been in the context of a single centralized data warehouse. Our results form the basis for extending these techniques to the distributed case.

A considerable body of research has been performed for processing and optimizing queries over distributed data (see, e.g., [17–19]). However, this research has focused on distributed join processing rather than distributed aggregate computation. The approach we explore in this paper uses an extended aggregation operator to express complex aggregation queries. Some of the distributed evaluation optimizations that we have developed resemble previously proposed optimizations, e.g., exploiting data distributions [18] or local reduction [20]. However, our architecture for evaluating distributed aggregation queries allows for novel optimizations not exploited by conventional algorithms for distributed processing.

2. Preliminaries

In this section, we give an example of an application area that motivates the use of distributed data warehouse techniques. We then define the GMDJ operator and demonstrate how the GMDJ operator allows us to uniformly express a variety of OLAP queries.

2.1. Motivating example

Analysis of IP flow data is a compelling application that can tremendously benefit from distributed data warehouse technology. An IP flow is a sequence of packets transferred from a given source to a given destination (identified by an IP address, Port, and Autonomous system), using a given `Protocol`. All packets in a flow pass through a given router, which maintains summary statistics about the flow and which dumps out a tuple for each flow passing through it. Thus, all flow information is conceptually part of a single relation with the following schema:

```
Flow (RouterId, SourceIP, SourcePort,
    SourceMask, SourceAS, DestIP,
    DestPort, DestMask, DestAS,
    StartTime, EndTime, NumPackets,
    NumBytes)
```

We assume that flow tuples generated by a router are stored in a local data warehouse "adjacent" to the router, i.e., `RouterId` is a partition attribute. Each local warehouse is assumed to be fully capable of evaluating any complex OLAP query. We refer to this collection of local data warehouses (or sites), along with a coordinator site that correlates subquery results, as a *distributed data warehouse*.

2.2. GMDJ expressions

The GMDJ operator is an OLAP operator that allows for the algebraic expression of many complex OLAP queries [14]. An important feature of this operator is that it provides a clean separation between the definition of the groups and the definition of aggregates in an OLAP query, and, in contrast to the MDJ operator, allows the expression of a series of complex aggregations over different data partitions using a single expression.

Let θ be a condition, b be a tuple, and R be a relation. We write $attr(\theta)$ to denote the set of attributes used in θ. $RNG(b, R, \theta) =_{\text{def}} \{r | r \in R \wedge \theta(b, r)\}$ denotes the range of tuples in R that satisfies θ, with respect to the tuple b. E.g., $RNG(b, R, b.A = R.B)$ denotes those tuples in R whose B-value matches the A-value of b. We use $\{\{\ldots\}\}$ to denote a multiset.

Definition 1. Let $B(\mathbf{B})$ and $R(\mathbf{R})$ be relations, θ_i a condition with $attr(\theta_i) \subseteq \mathbf{B} \cup \mathbf{R}$, and l_i be a list of aggregate functions $(f_{i1}, f_{i2}, \ldots, f_{in_i})$ over attributes $c_{i1}, c_{i2}, \ldots, c_{in_i}$ in $\mathbf{R}$. The GMDJ, $MD(B, R, (l_1, \ldots, l_m), (\theta_1, \ldots, \theta_m))$, is a relation with schema:[1]

$$\mathbf{X} = (\mathbf{B}, f_{11}\_R\_c_{11}, \ldots, f_{1n}\_R\_c_{1n_1}, \ldots, f_{m1}\_R\_c_{m1}, \ldots, f_{mn}\_R\_c_{mn_m}),$$

where the instance of the relation is determined as follows. Each tuple $b \in B$ contributes to an output tuple $\mathbf{x}$, such that:

- $\mathbf{x}[A] = b[A]$, for every attribute $A \in \mathbf{B}$,
- $\mathbf{x}[f_{ij}\_R\_c_{ij}] = f_{ij}\{\{t[c_{ij}] | t \in RNG(b, R, \theta_i)\}\}$, for every attribute $f_{ij}\_R\_c_{ij}$ of $\mathbf{x}$.

We call B the base-values relation and R the detail relation.

Usually, one can determine a subset K of key attributes of the base-values relation B for each θ_i, which uniquely determine a tuple in B (K can be $\mathbf{B}$). We make use of key attributes in several of our strategies.

[1] Attributes are appropriately renamed if there are any duplicate names generated this way. We note that the renaming scheme employed in the examples will use a shorthand form.

It should be noted that conventional SQL groupwise and hash-based aggregation techniques cannot be directly applied to GMDJ expressions, since the set of tuples in the detail relation R that satisfy condition θ with respect to tuples b_1 and b_2 of the base-values relation, i.e., $RNG(b_1, R, \theta)$ and $RNG(b_2, R, \theta)$, might not be disjoint. However, see [14,12] for a discussion of how GMDJ expressions can be evaluated efficiently in a centralized system.

Using relational algebra extended with a simple aggregation operator (we use the aggregation operator $\mathscr{F}$ of Elmasri and Navathe [21]) a GMDJ can be expressed as follows.

Proposition 1. *Let l_i be a list of aggregate functions $(f_{i1}(R.c_{i1}), \ldots, f_{im_i}(R.c_{im_i}))$. If every tuple in B is distinct, then the GMDJ expression $MD(B(\mathbf{B}), R(\mathbf{R}), (l_1, \ldots, l_m), (\theta_1, \ldots, \theta_m))$ is equivalent to the relational algebra expression B_n, where*

$$B_0 = B,$$

$$B_i = {}_{\mathbf{B}_{i-1}}\mathscr{F}_{f_{i1}(R.c_{i1}), \ldots, f_{im}(R.c_{im})}(B_{i-1} \bowtie_\theta R) \bigcup \left((B_{i-1} - \pi_{\mathbf{B}_{i-1}}(B_{i_1} \bowtie_\theta R)) \times \mathrm{N}_{l_i}\right) \quad \text{for } 0 < i \leqslant n.$$

N_{l_i} is a one tuple n-ary relation whose attributes are the initial values for the aggregates in l_i (i.e., 0 for `SUM` *and* `COUNT`, `NULL` *for* `MAX` *and* `MIN`*).*

Even without considering the possibility of duplicates in B (which can be handled by introducing additional joins), the algebraic expressions derived using Proposition 1 are extremely unwieldy. A salient feature of the GMDJ is that it succinctly expresses the complex algebraic aggregate-join expression as given by Proposition 1.

Assume a data warehouse scenario such as the one described in Section 2.1. Then one might wish to ask queries such as:

Example 1. For each `SourceIP (SIP)` address, compute the total number of flows to each of the destination autonomous systems with `DestAS (DAS)` numbers 1 and 29 (e.g., evaluate peering arrangements). We evaluate this using the

following GMDJ expression:

$MD(\pi_{\texttt{SIP}}(\texttt{Flow}) \to B, \texttt{Flow} \to F, (l_1, l_2), (\theta_1, \theta_2))$,

where l_1 is $(cnt(*) \to cnt1)$, l_2 is $(cnt(*) \to cnt2, cnt(distinct\ SP) \to cntD)$, θ_1 is (F.SIP = B.SIP & F.DAS = 1), and θ_2 is (F.SIP = B.SIP & F.DAS = 29).

The GMDJ expression above computes for each source IP address the total number of flows to each of the destination autonomous systems with numbers 1 and 29. For DestAS 29 the number of different source ports (SP) being used is computed as well. Assume an input relation for flow as follows:

Flow

SIP	SP	DAS	ST
5	A	29	MO
5	B	29	MO
5	A	6	TU
5	A	1	TU
7	A	29	MO
5	A	29	TU

Evaluating the GMDJ given above, we then get the following result relation:

result

SIP	cnt1	cnt2	cntD
5	1	3	2
7	0	1	1

A GMDJ operator can be composed with other relational algebra operators (and other GMDJs) to create complex GMDJ expressions. While arbitrary expressions are possible, it is often the case that the result of a GMDJ expression serves as the base-values relation for another GMDJ operator. This is because the result of the GMDJ expression has exactly as many tuples as there are in the base-values relation B. In the rest of this paper, when we refer to (complex) GMDJ expressions, we mean only expressions where the result of an (inner) GMDJ is used as a base-values relation for an (outer) GMDJ.

Example 2. Given our IP Flows application, an interesting OLAP query might be to ask for the total number of flows, and the number of flows whose NumBytes (NB) value exceeds or is equal to the average value of NB, for each combination of source and destination autonomous system (e.g., to identify special traffic). This query is computed by the complex GMDJ expression given below

$MD(MD(B_0, F_0, l_1, \theta_1) \to B_1, F_1, l_2, \theta_2)$,

where B_0 is $\pi_{\texttt{SAS,DAS}}(\texttt{Flow})$, F_0 is Flow, l_1 is $(cnt(*) \to cnt1, sum(NB) \to sum1)$, θ_1 is ($\texttt{F}_0\texttt{.SAS} = \texttt{B}_0\texttt{.SAS}$ & $\texttt{F}_0\texttt{.DAS} = \texttt{B}_0\texttt{.DAS}$), F_1 is Flow, l_2 is $(cnt(*) \to cnt2)$, and θ_2 is ($\texttt{F}_1\texttt{.SAS} = \texttt{B}_1\texttt{.SAS}$ & $\texttt{F}_1\texttt{.DAS} = \texttt{B}_1\texttt{.DAS}$ & $\texttt{F}_1\texttt{.NB} \geqslant \texttt{sum1/cnt1}$).

The flow data may or may not be clustered on SourceAS or DestAS. If it is not, the distributed evaluation of this query requires correlating aggregate data at multiple sites, and alternating evaluation (in multiple passes) at the sites and at the coordinator. We develop efficient evaluation strategies for both cases.

We refer to queries such as the one in Example 2 as *correlated aggregate queries* since they involve computing aggregates with respect to a specified grouping and then computing further values (which may be aggregates) based on the previously computed aggregates. In general, there may be a chain of dependencies among the various (aggregate or otherwise) attributes computed by such a query. In Example 2, the length of this chain is two. Several examples involving correlated aggregates can be found in previous OLAP literature [4,6,10,12].

It is possible to construct many different kinds of OLAP queries and identifying distributed evaluation strategies for each would be quite tedious. The GMDJ operator allows us to express a significant variety of OLAP queries (enabled by disparate SQL extensions or SQL itself) in a uniform algebraic manner [14,12]. Thus, it suffices to consider the distributed evaluation of GMDJ expressions, to capture most of the OLAP queries proposed in the literature.

2.3. Data warehouse model

A typical data warehouse consists of dimensions and measures that are gauged as functions of the dimensions. These can be modeled using, e.g., *star schemas* or *snowflake schemas* [5,22]. In our

motivating example, `Flow` is a denormalized fact relation, with dimension attributes `RouterId`, `SourceIP`, `SourcePort`, `SourceMask`, `SourceAS`, `DestIP`, `DestPort`, `DestMask`, `DestAS`, `StartTime`, and `EndTime`, and measure attributes `NumPackets` and `NumBytes`. Our techniques are oblivious to which of these data warehouse models are used for conceptually modeling the data, and our results would hold in either model.

Recall that the distributed data warehouse consists of multiple local data warehouses, adjacent to data collection points. In terms of our conceptual model of the data warehouse, the fact relation would be the *union* of the tuples captured at each data collection point (e.g., NetFlow tuples generated by a router). Equivalently, since no tuple would be present at multiple sites, one can view the highly dynamic conceptual fact relation of the distributed data warehouse as being physically *partitioned* among the various sites. We assume that the dimension relations, which are orders of magnitudes smaller than the fact relation, are replicated at all sites. Users can pose OLAP queries against the conceptual data model of our distributed data warehouse, without regard to the location of individual tuples at the various sites.

Our data warehouse architecture supports *correlation* of the results of subqueries across the local data warehouses using a *coordinator architecture*. Each OLAP client interacts with a distinct coordinator that is responsible for interacting with each of the local sites, and for correlating and aggregating the results of subqueries sent from the local sites.

2.4. Problem statement

The problem addressed in the paper is that of efficient evaluation of OLAP queries, represented as GMDJ expressions, using the coordinator architecture for distributed data warehouses.

Since the amount of data captured at each data collection point is huge, it is infeasible to ship all the relevant data to the coordinator site to compute the query answer using centralized techniques for OLAP query evaluation. This leads us to focus on distributed techniques for OLAP query evaluation that rely on: (i) subquery shipping to the local sites, (ii) subquery evaluation at the local sites, and (iii) synchronization of subquery results at the coordinator site.

The development of techniques for the efficient evaluation of complex OLAP queries involving correlated aggregates which optimize (i) synchronization traffic, and (ii) the query processing effort at the local sites as well as the coordinator, are the focus of this paper.

3. Distributed GMDJ evaluation

In this section, we will describe the distributed GMDJ evaluation algorithm implemented in the Skalla prototype. We defer the presentation of query optimizations of the core Skalla evaluation algorithm until the next section.

3.1. Skalla: an overview

The Skalla system for distributed data warehousing is based on a coordinator architecture (i.e., strict client-server) as depicted in Fig. 1. It consists of multiple local data warehouses (Skalla sites) adjacent to data collection points, together with the Skalla coordinator (we note that the coordinator can be a single instance as in Fig. 1 or may consist of multiple instances, e.g., each client may have its own coordinator instance). Conceptually, the fact relation of our data warehouse is the *union* of the tuples captured at each data collection point. However, users can pose OLAP queries against the conceptual data model of our distributed data warehouse, without regard to the location of individual tuples at the various sites.

We define a *distributed evaluation plan* (*plan* for short) for our coordinator architecture as a sequence of rounds, where a round consists of: (i) each skalla site performing some computation and communicating the results to the coordinator, and (ii) the coordinator synchronizing the local results into a global result, and (possibly) communicating the global result back to the sites. Thus, the overall cost of a plan (in terms of response time) has several components: (i) communication (or, synchronization traffic), and (ii) computation

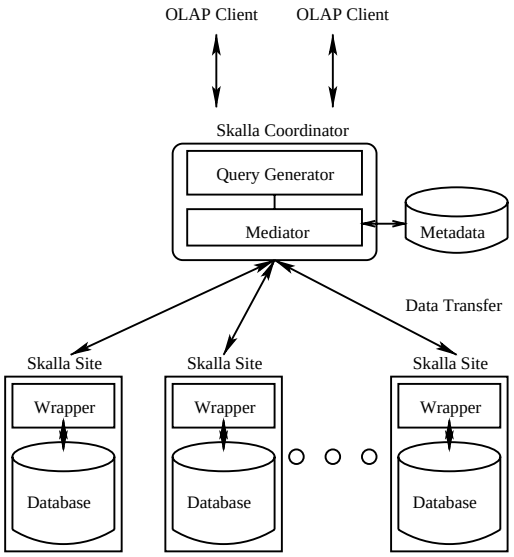


Fig. 1. The Skalla architecture.

```
Algorithm GMDJDistribEval {
   construct empty X
   transfer X and B0 to each site Si
   at each site Si ∈ SB
      compute X0 at the local sites
      transfer local X0 to coordinator
   synchronize X0
   for each GMDJ MDk (k = 1 to m) {
      transfer Xk−1 to each Si ∈ SMDk
      at each Si ∈ SMDk
         compute MDk(Xk−1, Rk, lk, θk)
         transfer Xk to coordinator
      synchronize Xk
   }
}
```

Fig. 2. Skalla evaluation algorithm.

(or, the query processing effort at the local sites as well as the coordinator).

The query generator of the Skalla system constructs query plans from the OLAP queries, which are then passed on to and executed by the mediator, using Algorithm `GMDJDistribEval` (in Fig. 2). The mediator handles the communication with the local sites, and the actual synchronization of sub-results after each round of computation at the local sites.

3.2. Algorithm description

We will now describe how Skalla works when receiving an OLAP query, using Example 2 to illustrate the Skalla evaluation.

First, the Skalla query generator uses *Egil*, a GMDJ query optimizer, to translate the OLAP query into GMDJ expressions. These GMDJ expressions are then optimized for distributed computation. We note that even simple GMDJ expressions can involve aggregation and multiple self-joins, which would be hard for a conventional centralized—let alone a distributed—query optimizer to handle. We defer a study of these optimizations to Section 4.

Algorithm `GMDJDistribEval` (Fig. 2) gives an overview of the basic query evaluation strategy of Skalla for complex GMDJ expressions. Given the GMDJ query of Example 2, the mediator will construct the empty *base-result structure* $X(\mathbf{X})$ with the schema:

$$\mathbf{X} = (\texttt{SAS}, \texttt{DAS}, \texttt{cnt1}, \texttt{sum1}, \texttt{cnt2})$$

The query $B_0 = \pi_{\texttt{SourceAS,DestAS}}(\texttt{Flow})$ is then shipped to each of the sites, executed locally, and the result shipped back to the mediator. During evaluation, the coordinator maintains the base-result structure by synchronization of the sub-results that it receives.

The term *synchronization* as used in this paper refers to the process of consolidating the results processed at the local sites into the base-results structure X. We refer to each local-processing-then-synchronization step as a *round* of processing. An evaluation of a GMDJ expression involving m GMDJ operators uses $m+1$ rounds. The notation X_k in Algorithm `GMDJDistribEval` refers to the instance of X after the computation of the kth GMDJ.

Aggregates of X are computed from the local aggregate values computed at the sites as appropriate. For example, to compute `cnt1`, we would need to compute the sum of the `COUNT(*)`s computed at the local sites. Following Gray et al. [3], we call the aggregates computed at the local sites the *sub-aggregates* and the aggregate computed at the mediator the *super-aggregate*. $\mathscr{S}_B$ is

the set of all local sites, while $\mathscr{S}_{MD_k}$ is the set of local sites that participate in the kth round.[2]

We use R_k to denote the detail relation at round k.[3] Again, depending on the query, the detail relation may or may not be the same across all rounds. This shows the considerable class of OLAP queries the basic Skalla evaluation framework is able to handle. In this paper, we give only examples where the detail relation does not change over rounds. Finally, l_k is the list of aggregate functions to be evaluated at round k and θ_k is the corresponding condition (see Definition 1).

The following theorem establishes the basis of the synchronization in Skalla:

Theorem 1. *Let*

$$X = MD(B, R, (l_1, \ldots, l_m), (\theta_1, \ldots, \theta_m)),$$

where B has key attributes K. Let $R_1, \ldots, R_n$ *be a partition of R. Let* l'_j *and* l''_j *be the lists of sub-aggregates and super-aggregates, respectively, corresponding to the aggregates in* l_j. *Let* $H_i = MD(B, R_i, (l'_1, \ldots, l'_m), (\theta_1, \ldots, \theta_m))$ for $i = 1, \ldots, n$. Let $H = H_1 \sqcup \cdots \sqcup H_n$, *where* $\sqcup$ *indicates multiset union. Then* $X = MD(B, H, (l''_1, \ldots, l''_m), \theta_K)$ *where* θ_K *is a test for equality on the attributes in K.*

Proof. It follows from the definition of the GMDJ (Definition 1) that each aggregate f_{ij} in the lists $l_1, \ldots, l_m$ is computed for each $b \in B$ as: $f_{ij}\{\{t[c_{ij}] | t \in RNG(b, R, \theta_i)\}\}$. Let f'_{ij} be the sub-aggregate and f''_{ij} the super-aggregate of the aggregate function f_{ij} and let $R = R_1 \sqcup \cdots \sqcup R_n$. Then it follows from the definition of sub- and super-aggregates [3] that:

$$\begin{aligned} f''_{ij}\{\{f'_{ij}\{\{t_1[c_{ij}] | t_1 \in RNG(b, R_1, \theta_i)\}\} \sqcup \cdots \sqcup \\ f'_{ij}\{\{t_n[c_{ij}] | t_n \in RNG(b, R_n, \theta_i)\}\}\}\} \\ = f_{ij}\{\{t[c_{ij}] | t \in RNG(b, R, \theta_i)\}\}. \quad \square \end{aligned}$$

[2] Typically, $\mathscr{S}_{MD_k} = \mathscr{S}_B$, but it is possible that $\mathscr{S}_{MD_k} \subset \mathscr{S}_B$.

[3] Note that every site refers to its local detail relation as R_k at round k. To avoid clutter, we preferred this notation to something like R^i_k.

In practice, the scheme given by Theorem 1 is executed very efficiently. The base-results structure maintained at the mediator is indexed on K, which allows us to efficiently determine $RNG(X, t, \theta_K)$ for any tuple t in H and then update the structure accordingly, i.e., the synchronization, including the indexing, can be computed in $\mathcal{O}(|H|)$. Since the GMDJ can be horizontally partitioned [12,23], the mediator can synchronize H with those sub-results it has already received while receiving blocks of H from slower sites, rather than having to wait for all of H to be assembled before performing the synchronization. Between each computation at the local sites and synchronization, we ship the base-results structure (or fragments thereof) between the sites and the mediator.

Example 3. Consider the query in Example 2. We wish to evaluate the following complex GMDJ expression:

$$MD(MD(B_0, F_0, l_1, \theta_1) \to B_1, F_1, l_2, \theta_2),$$

where B_0 is $\pi_{\texttt{SAS,DAS}}(\texttt{Flow})$, F_0 is the `Flow`, l_1 is $(cnt(*) \to cnt1, sum(NB) \to sum1)$, θ_1 is ($\texttt{F}_0\texttt{.SAS} = \texttt{B}_0\texttt{.SAS}$ & $\texttt{F}_0\texttt{.DAS} = \texttt{B}_0\texttt{.DAS}$), F_1 is the `Flow`, l_2 is $(cnt(*) \to cnt2)$, and θ_2 is ($\texttt{F}_1\texttt{.SAS} = \texttt{B}_1\texttt{.SAS}$ & $\texttt{F}_1\texttt{.DAS} = \texttt{B}_1\texttt{.DAS}$ & $\texttt{F}_1\texttt{.NB} \geqslant \texttt{sum1/cnt1}$).

Assume a table `Flow(SAS, DAS, NB)` and that the source data is distributed over two sites, S_1 and S_2 as follows:

`Flow at site S1`

SAS	DAS	NB
18	29	200
18	29	300
18	1	2500

`Flow at site S2`

SAS	DAS	NB
16	6	700
18	29	400

The first step of the evaluation determines the schema of the result table: $\mathbf{X} = \{SAS, DAS, cnt1, sum1, cnt2\}$. We then send the initial schema ($\mathbf{X}$) and the query plan to the local sites. The base tables (groups of the query) are then computed at the local sites by evaluating $\pi_{SAS,DAS}(Flow)$. This results in the following two

tables for X_0:

X_0^{s1}

SAS	DAS
18	1
18	29

X_0^{s2}

SAS	DAS
16	6
18	29

Each local site sends its tables to the mediator, which synchronizes the results to get table X_0, the base-values (groups) of the query:

X_0

SAS	DAS
16	6
18	1
18	29

This table is then sent to the local sites, S_1 and S_2. The local sites use this table as the basis of their next *round* of computation, computing the first (inner) GMDJ at each of the sites

$MD(X_0, F_0, l_1, \theta_1)$,

where F_0 is the `Flow`, l_1 is $(cnt(*) \rightarrow cnt1, sum(NB) \rightarrow sum1)$, and θ_1 is ($\mathtt{F_0.SAS} = \mathtt{X_0.SAS}$ & $\mathtt{F_0.DAS} = \mathtt{X_0.DAS}$).

This results in the following two tables:

X_1^{s1}

SAS	DAS	cnt1	sum1
16	6	0	0
18	1	1	2500
18	29	2	500

X_1^{s2}

SAS	DAS	cnt1	sum1
16	6	1	700
18	1	0	0
18	29	1	400

These sub-result are then sent to and synchronized at the mediator:

X_1

SAS	DAS	cnt1	sum1
16	6	1	700
18	1	1	2500
18	29	3	900

This result is again passed back to the local sites, and the $cnt2$ aggregate computed at each site, making use of the average ($sum1/cnt1$) computed in the first round

$MD(X_1, F_1, l_2, \theta_2)$,

where F_1 is the `Flow`, l_2 is $(cnt(*) \rightarrow cnt2)$, and θ_2 is ($\mathtt{F_1.SAS} = \mathtt{X_1.SAS}$ & $\mathtt{F_1.DAS} = \mathtt{X_1.DAS}$ & $\mathtt{F_1.NB} \geqslant \mathtt{sum1/cnt1}$).

This results in the following tables at the local sites:

X_2^{s1}

SAS	DAS	cnt1	sum1	cnt2
16	6	1	700	0
18	1	1	2500	1
18	29	3	900	1

X_2^{s2}

SAS	DAS	cnt1	sum1	cnt2
16	6	1	700	1
18	1	1	2500	0
18	29	3	900	1

Finally, the sub-results are passed back to the mediator and synchronized to compute the final result:

X_2

SAS	DAS	cnt1	sum1	cnt2
16	6	1	700	1
18	1	1	2500	1
18	29	3	900	2

Note that only the result table and queries are shipped between coordinator and sites. This is an important feature of the Skalla processing that permits the effective minimization of the data transferred.

The following result bounds the maximum amount of data transferred during the evaluation of Algorithm `GMDJDistribEval` for distributive aggregate queries.

Theorem 2. *Let the distributed data warehouse contain n sites, and the size of the result of query Q, expressed as a GMDJ expression with m GMDJ operators be $|Q|$. Let s_0 denote the number of sites participating in the computation of the base values*

relation and s_i the number of sites participating in the computation of the ith GMDJ operator (the upper bound of s_0 and s_i is thus n). Then the maximum amount of data transferred during the evaluation of Algorithm `GMDJDistribEval` *on Q is bounded by*

$$\left(\sum_{i=1}^{m}(2 * s_i * |Q|\right) + (s_0 * |Q|).$$

Proof. Recall that Algorithm `GMDJDistribEval` only ships the base-result structure X_k. Since $X_k \subseteq Q$, it follows that the maximum size of any X_k is $|Q|$. Theorem 2 thus follows directly from Algorithm `GMDJDistribEval`. □

The significance of Theorem 2 is that it provides a bound on the maximum amount of data transferred as a function of the size of the query result, the size of the query (i.e. number of rounds), and the number of local sites in the distributed data warehouse, which is *independent of the size of the fact relation in the database*. $|Q|$ depends only on the size of the base values relation and the aggregates to be computed, not the detail relation. This is significant in that such a bound does not hold for the distributed processing of traditional SQL join queries (see, for example, [18,19]), where intermediate results can be arbitrarily larger than the final query result, even when using semijoin-style optimizations.

3.3. Non-distributive aggregates

Non-distributive aggregates are inherently problematic in distributed query processing. A subset of the non-distributive aggregates, the *algebraic aggregates* (e.g., `AVG`), can be calculated from component distributive aggregates, need therefore not concern us further. We refer to any non-distributive aggregates that cannot be calculated from a set of distributive aggregate functions as *holistic aggregates* [3] (e.g., `MEDIAN`, `RANK`). Holistic aggregates are only very rarely addressed in the OLAP literature, although they are quite significant in many OLAP applications [24].

Obviously, holistic aggregates can be handled by the naive approach of simply shipping the local detail data to the coordinator, which then evaluates the query with holistic aggregates. However, holistic aggregates can be elegantly handled within the framework of GMDJ processing [24]. As with single-site processing, our distributed evaluation algorithms can be made to support non-distributive aggregates through the use of multi-set valued attributes in the base-results structure and a function for multi-set construction. The multi-set construction `MS` is defined as follows:

Definition 2. Let $B(\mathbf{B})$ and $R(\mathbf{R})$ be relations, θ a condition, $a \in \mathbf{R}$. Then $X = MD(B, R, (\mathtt{MS}(a)), \theta)$ is a table with the schema $\mathbf{X} = (\mathbf{B}, ms\_R\_a)$, where ms_R_a is a multi-set-valued attribute computed such that:

- $\mathbf{x}[A] = b[A]$, for every attribute $A \in \mathbf{B}$, and
- $\mathbf{x}[ms\_R\_a] = \sqcup\{\{t[a] | t \in RNG(n, R, \theta_j)\}\}$.

Given any holistic aggregate $f(a)$ in a GMDJ expression, we: (1) generate expressions to be evaluated on the local sites using the function $\mathtt{MS}(a)$ as the sub-aggregate, and (2) compute the holistic aggregate itself as the super-aggregate over the multi-set construction.

In a worst-case scenario holistic aggregates substantially increase communication costs in distributed processing. This is due to the fact that, by their very nature, holistic aggregates require that the values are centralized for accurate computation. In such a case, Theorem 2 offers scant solace, as the partial sub-results of the `MS` aggregate function can be larger than the detail data itself.

We note, however, that the distributed GMDJ evaluation algorithm opens up possibilities for extensive optimizations that can be deployed to reduce communications costs.

Thus, we observe that only one `MS` function needs to be computed for any given θ_i, i.e., several

holistic aggregates in l can be computed from one sub-aggregate.

For various holistic aggregates, it is possible to apply smart compression algorithms on the attribute columns of MS. For example:

- Duplicate-insensitive aggregate functions (e.g., COUNT(DISTINCT *MS_R_A*)) permit the elimination of duplicates in the multi-set-valued attribute.
- Provided that the granularity of the attribute being computed is not too large; we can perform duplicate compression on the attribute column; annotating each individual value with its number of occurrences. For example, the multi-set $\{1,1,1,2,2,2,2,3\}$ would be compressed to $\{1(3),2(4),3(1)\}$ (the number in parentheses indicating the number of duplicates).

The latter form of duplicate compression is simple to implement; it is particularly useful since many duplicate-sensitive holistic aggregates (e.g., MEDIAN) often occur on attribute columns with low granularity.

It is obviously impossible to derive a general solution for the distributive computation of non-distributive, holistic aggregates. However, as the above examples indicate, it is possible to perform effective optimizations on the distributed evaluation algorithm for non-distributive, holistic aggregates, thereby contributing to the minimization of the amount of data that must inevitably be shipped for computation.

3.4. Summary

Let B_0 be the base-values relation, R_1, R_2, …, R_m be detail relations, l_1, …, l_m be lists of aggregate functions, and θ_1, …, θ_m be lists of conditions.[4] Let B_k denote the result of the GMDJ MD_k. Let $\mathscr{S}_B$ be the set of sites required for the computation of B_0, and let $\mathscr{S}_{MD_i}$ be the set of sites required for the computation of the GMDJ MD_i. Algorithm GMDJDistribEval is a simple and efficient algorithm for distributed OLAP processing that does not transfer any detailed data between the sites.

Theorem 3. *Given a GMDJ expression*:

$$Q = MD_n(\cdots(MD_1(B_0, R_1, (l_{11}, \ldots, l_{1k}), (\theta_{11}, \ldots, \theta_{1k}))\cdots) \to B_{n-1}, R_n, (l_{n1}, \ldots, l_{nm}), (\theta_{n1}, \ldots, \theta_{nm}))$$

the set of sites $\mathscr{S}_B$ required for computing B_0, and the sets of sites $\mathscr{S}_{MD_i}, i \leqslant n$, required for computing GMDJ MD_i, then Algorithm GMDJDistribEval *correctly computes the result of Q.*

Proof. Theorem 1 gives the basis for the synchronization and computation of a single GMDJ in Algorithm GMDJDistribEval. Thus, Theorem 1 proves the correctness of the algorithm for a single GMDJ given that each site has the correct base-values table for the computation.

For each GMDJ in the complex GMDJ expression Q, Algorithm GMDJDistribEval distributes the entire base-values table B_{i-1} (i.e., the original base-values table B_0 or the result of the prior GMDJ MD_{i-1}) for each GMDJ MD_i to each of the sites involved in the computation of the GMDJ. It then follows by induction that if MD_i is computed correctly, then MD_{i+1} will also be computed correctly. Hence the correctness of Algorithm GMDJDistribEval. □

While Algorithm GMDJDistribEval is efficient in relation to the size of the detail relations, the amount of data transfer and the computation can still be objectionably large for very large distributed data warehouses and thus these resources need to be optimized substantially. This is the subject of Section 4.

4. Distributed evaluation optimizations

Clearly, query optimization techniques used for centralized evaluation of GMDJ expressions (e.g., indexing, coalescing), which have been previously studied [14,12] apply in an orthogonal way to their

[4] l_i is a list of aggregates and θ_i is a condition for its respective l_i.

distributed processing. Similarly, classical distributed query optimization techniques developed for SQL queries such as row blocking, optimization of multi-casts, horizontal partitioning of data, or semijoin programs [18] also apply to the distributed processing of GMDJs. We do not dwell on these any further. In the following sections, we consider distributed evaluation optimizations as they pertain specifically to the case of distributed GMDJs. In Sections 4.1 and 4.2 we generalize local reductions to GMDJ expressions. Sections 4.3 and 4.4 describe optimizations specific to distributed GMDJ query processing.

It should be stressed that in each case, we present what are best characterized as optimization schemes. Depending on the specific situation (i.e. the amount of distribution knowledge available), one may be able to come up with specific optimization strategies, which are instances of these optimization schemes. In this sense, our optimization schemes can be used for determining whether a strategy is correct in the sense that it computes the right result. Throughout this section, we give examples which illustrate this point.

4.1. Attribute reduction

We can generalize local reduction strategies [20] to GMDJ expressions, by using attribute reduction to prune attributes from the base-results structure. Attribute reductions can significantly reduce the data transfer cost, and thereby improve overall query processing costs.

Theorem 4. *Let Q be a GMDJ operator $MD(B, R, (l_1, \ldots, l_m), (\theta_1, \ldots, \theta_m))$. The required attributes of Q are: $\alpha = attr(l_1) \cup \cdots \cup attr(l_m) \cup attr(\theta_1) \cup \cdots \cup attr(\theta_m)$; (i.e., the attributes of B and R referenced in $(l_1, \ldots, l_m)$ and $(\theta_1, \ldots, \theta_m)$). Let $\mathbf{R}$ be the set of aggregates computed the GMDJ expression. Let K be the key attributes of B and θ_K be an equality condition on the attributes of K. Then*

$$MD(B, R, (l_1, \ldots, l_m), (\theta_1, \ldots, \theta_m)) = B \bowtie_{\theta_K} (\pi_{K \cup \mathbf{F}}(MD(\pi_{\alpha \cup K}(B), R, (l_1, \ldots, l_m), (\theta_1, \ldots, \theta_m)))).$$

Proof. Given that K is a key of B, it follows from the definition of a standard join that

$$MD(B, R, (l_1, \ldots, l_m), (\theta_1, \ldots, \theta_m)) = B \bowtie_{\theta_K} (MD(B, R, (l_1, \ldots, l_m), (\theta_1, \ldots, \theta_m))).$$

Let $\mathbf{X}$ be the schema of the GMDJ $MD(B, R, (l_1, \ldots, l_m), (\theta_1, \ldots, \theta_m))$. Since $\mathbf{B} \supseteq (\mathbf{X} \backslash l)$, it follows that:

$$MD(B, R, (l_1, \ldots, l_m), (\theta_1, \ldots, \theta_m)) = B \bowtie_{\theta_K} (\pi_{K \cup l}(MD(B, R, (l_1, \ldots, l_m), (\theta_1, \ldots, \theta_m)))).$$

Let A be a set of attributes, and $A' = (A \cup attr(\theta_1) \cup \ldots \cup attr(\theta_m)) \backslash \mathbf{F}$. Then we can apply the following general transformation rule for GMDJs [24] in the above expression

$$\pi_A MD(B, R, (l_1, \ldots, l_m), (\theta_1, \ldots, \theta_m)) \Rightarrow \pi[A] MD(\pi_{A'}(B), R, (l_1, \ldots, l_m), (\theta_1, \ldots, \theta_m)). \quad \square$$

The core importance of this theorem is that it allows us to prune the synchronized attributes of the base-results structure that are no longer required in the distributed computation, as well as pruning the base relations of the query.

Example 4. We return to the query and tables of Example 3. Assume that in addition to the total number of flows and the number of flows whose `NumBytes` value exceeds or is equal to the average value, we also want to identify the maximum and minimum size of `NumBytes` for each combination of source and destination systems. This query can be expressed very easily by adding the new aggregate functions to the inner GMDJ. Thus, we get the following GMDJ expression:

$$MD(MD(B_0, F_0, l_1, \theta_1) \rightarrow B_1, F_1, l_2, \theta_2),$$

where B_0 is $\pi_{\texttt{SAS,DAS}}(\texttt{Flow})$, F_0 is the `Flow`, l_1 is $(cnt(*) \rightarrow cnt1, sum(NB) \rightarrow sum1, \quad max(NB) \rightarrow max1, min(NB) \rightarrow min1)$, θ_1 is $(\texttt{F}_0\texttt{.SAS} = \texttt{B}_0\texttt{.SAS} \ \& \ \texttt{F}_0\texttt{.DAS} = \texttt{B}_0\texttt{.DAS})$, F_1 is the `Flow`, l_2 is $(cnt(*) \rightarrow cnt2)$, and θ_2 is $(\texttt{F}_1\texttt{.SAS} = B_1.SAS \ \& \ \texttt{F}_1\texttt{.DAS} = \texttt{B}_1\texttt{.DAS} \ \& \ \texttt{F}_1\texttt{.NB} \geqslant \texttt{sum1/cnt})$.

This results in the following sub-result (X_1) being computed at the mediator as the result of the first GMDJ:

X_1

SAS	DAS	cnt1	sum1	max1	min1
16	6	1	700	700	700
18	1	1	2500	2500	2500
18	29	3	900	400	200

Following the basic algorithm, this table is to be shipped to each of the local sites. By applying the principles of attribute reduction, however, we can determine that *max*1 and *min*1 are not required for the computation of the second (outer) GMDJ. We can thus limit the table sent to the local sites to $\pi_{SAS,DAS,cnt1,sum1}$, resulting in 30% less data being transmitted with the distribution of data to the sites in this round.

With the return of the tables from site S_1 and S_2, we can then synchronize the two tables of the sub-result, to return the same table X_2 as in Example 3. We then compute the final sub-result ($X_2' = X_1 \bowtie_{X_1.SAS=X_2.SAS \& X_1.DAS=X_2.DAS} X_2$) to end up with the following result:

X_2'

SAS	DAS	cnt1	sum1	max1	min1	cnt2
16	6	1	700	700	700	1
18	1	1	2500	2500	2500	1
18	29	3	900	400	200	2

In fact, it will typically be possible to compute the union, GMDJ, and join of the synchronization above in a single merging operation. Thus the only additional cost caused by the attribute reductions would be the cost of determining and pruning the superfluous attributes during each round of evaluation. □

4.2. Distribution-aware group reduction

Recall the definition of the function, $RNG(b, R, \theta) = \{r \in R | \theta(b, r) \text{ is true}\}$. Given information about the partitioning of tuples in the distributed data warehouse, we may be able reduce the size of the base-results structure transferred between the sites and the coordinator. The following theorem provides the basis for this optimization.

Theorem 5. *Consider the GMDJ expression* $Q = MD(B, R, (l_1, \ldots, l_m), (\theta_1, \ldots, \theta_m))$. *Let* $R_1 \cup \cdots \cup R_n$ *be a partitioning of the detail relation R. For each* R_i, let ϕ_i *be a predicate such that for each* $r \in R_i$, $\phi_i(r)$ *is true. Let* $\psi_i(b)$ *be the formula* $\forall_r \phi_i(r) \Rightarrow \neg(\theta_1 \vee \cdots \vee \theta_m)(b, r)$. *Let* RNG_i *be* $RNG(b, R_i, \theta_1 \vee \cdots \vee \theta_m)$. *Let* C *be* $|RNG_i| > 0$. *Then, we have*

$$\sigma_C(MD(B, R_i, (l_1, \ldots, l_m), (\theta_1, \ldots, \theta_m))) = \sigma_C(MD(\sigma_{\neg\psi_i}(B), R_i, (l_1, \ldots, l_m), (\theta_1, \ldots, \theta_m))).$$

Proof. Theorem 5 is a straightforward specialization to the distributed case of the following general transformation rule for GMDJs:

$$\sigma_{|RNG|>0} MD(B, R, (l_1, \ldots, l_m), (\theta_1, \ldots, \theta_m)) = \sigma_{|RNG|>0} MD(\sigma_{\theta_{1B} \vee \ldots \vee \theta_{mB}} B, R, (l_1, \ldots, l_m), (\theta_1, \ldots, \theta_m)).$$

For details of the proof, see Appendix A.2. □

Using Algorithm `GMDJDistribEval`, a local site will compute

$$H_i = MD(B, R_i, (l_1', \ldots, l_m'), (\theta_1, \ldots, \theta_m)).$$

Let $\overline{B_i} = \{b \in B | \psi_i(b)\}$. Theorem 5 states that if we have a-priori knowledge about whether $RNG(b, R_i, \theta)$ is empty for any given b, we need to send to site S_i only $B - \overline{B_i}$. Given knowledge about the data distribution at the individual sites, group reductions can be performed by restricting B using the $\neg\psi_i$ condition. □

Example 5. Assume that each of the packets for a specific `SourceAS` passes through a router with a specific `RouterId`. For example, site S_1 handles all and only autonomous systems with `SourceAS` between 1 and 25. The condition θ in the query of Example 2 contains the condition `Flow.SourceAS = B.SourceAS`. We can deduce that at S_1, $\psi_i(b)$ is true when b.`SourceAS` $\notin$ [13,7]. Therefore, $\neg\psi_i(b)$ is the condition b.`SourceAS` $\in$ [13,7].

Example 5 gives a simple example of the kind of optimization possible using distribution-aware group reductions. The analysis is easy to perform if ψ_i and θ are conjunctive and the atoms of the predicates involve tests for equality.

In fact, far more complex constraints can be handled. For example, assume the condition θ in Example 5 is revised to be $B.DestAS + B.SourceAS < Flow.SourceAS * 2$. Then condition $\neg\psi_i(b)$ becomes $B.DestAS + B.SourceAS < 50$. The significance of Theorem 5 is that we can use it to determine the correctness of the optimizer.

Other uses of Theorem 5 are also possible. For example, `SourceAS` might not be partitioned among the sites, but any given value of `SourceAS` might occur in the `Flow` relation at only a few sites. Even in such cases, we would be able to further reduce the number of groups sent to the sites.

4.3. Distribution-independent group reduction

A significant feature of the GMDJ processing, compared to traditional distributed algorithms, is the possibility of performing distribution-independent group reduction.

We extend Theorem 1 for distribution-independent group reduction:

Proposition 2. *Consider the GMDJ* $Q = MD(B, R, (l_1, \ldots, l_m), (\theta_1, \ldots, \theta_m))$ *where* B *has key attributes* K. *Let* $R_1, \ldots, R_n$ *be a partition of* R. *Let* l'_i *and* l''_i *be the lists of sub-aggregates and super-aggregates, respectively, corresponding to the aggregates in* l_i. *Then*

$$\begin{aligned} &MD(B, R, (l_1, \ldots, l_m), (\theta_1, \ldots, \theta_m)) \\ &= MD(B, \sigma_{|RNG|>0}(MD(B, R_1, (l'_1, \ldots, l'_m), \\ &\quad (\theta_1, \ldots, \theta_m))) \sqcup \cdots \sqcup \sigma_{|RNG|>0}(MD \\ &\quad (B, R_n, (l'_1, \ldots, l'_m), (\theta_1, \ldots, \theta_m))), \\ &\quad (l''_1, \ldots, l''_m), \theta_K), \end{aligned}$$

where $\sqcup$ *indicates multiset union, and* θ_K *is a test for equality on the attributes in* K.

Let $H_1, H_2, \ldots, H_n$ be the results of processing the GMDJ expressions at the local sites. Then Proposition 2 states that the only tuples of H_i required for synchronization of the results are those tuples t such that $|RNG(t, R_i, (\theta_1 \vee \ldots \vee \theta_m))| > 0$, as otherwise the tuple does not contribute any information to the global aggregate. A simple way of detecting $|RNG| > 0$ with respect to tuples in H_i is to compute an additional aggregate $l_{m+1} = \texttt{COUNT(*)}$ on H_i such that $\theta_{m+1} = (\theta_1 \vee \cdots \vee \theta_m)$. The only overhead to this optimization then becomes the additional computing time for the extra `COUNT(*)`, and to perform the selection `COUNT(*)` > 0 at the sites.

Example 6. We revisit Example 3. The computation of the first (inner) GMDJ results in the following two tables:

X_1^{s1}

SAS	DAS	cnt1	sum1
16	6	0	0
18	1	1	2500
18	29	2	500

X_1^{s2}

SAS	DAS	cnt1	sum1
16	6	1	700
18	1	0	0
18	29	1	400

Applying Proposition 2, however, allows the reduction of these tables at the local sites. Only the reduced sub-results (given below) are transferred to the mediator:

$X_1^{s1\prime}$

SAS	DAS	cnt1	sum1
18	1	1	2500
18	29	2	500

$X_1^{s2\prime}$

SAS	DAS	cnt1	sum1
16	6	1	700
18	29	1	400

Similarly, we can reduce the tables passed from the sites to the mediator for the second round of computation to the following tables:

X_2^{s1}

SAS	DAS	cnt1	sum1	cnt2
18	1	1	2500	1
18	29	3	900	1

X_2^{s2}

SAS	DAS	cnt1	sum1	cnt2
16	6	1	700	1
18	29	3	900	1

In this scenario, we succeed in reducing the communication from the local sites to the

coordinator during the evaluation by 1/3. Assuming n sites, and that the size of the GMDJ is $|B|$, we transmit $n*|B|$ data. If each site, on average, computes aggregates for $1/k$ tuples in B, then distribution-independent group reduction will reduce the amount of data transmitted by each site to $|B|/k$ and the total data transmission to $n/k*|B|$.

An advantage of distribution-independent group reduction is that it improves performance even without semantic information about the distribution of R (which might not be available).

4.4. Synchronization reduction

Synchronization reduction is concerned with reducing data transfer between the local sites and the coordinator by reducing the rounds of computation. One of the algebraic transformations possible on GMDJ operators is to coalesce two GMDJs into a single GMDJ. More precisely

$$\begin{aligned} &MD_2(MD_1(B, R, (l_{11}, \ldots, l_{1l}), (\theta_{11}, \ldots, \theta_{1l})), R, \\ &\quad (l_{21}, \ldots, l_{2m}), (\theta_{21}, \ldots, \theta_{2m})) \\ &= MD(B, R, (l_{11}, \ldots, l_{1l}, l_{21}, \ldots, l_{2m}), \\ &\quad (\theta_{11}, \ldots, \theta_{1l}, \theta_{21}, \ldots, \theta_{2m})) \end{aligned}$$

if the conditions $\theta_{21}, \ldots, \theta_{2m}$ do not refer to attributes generated by MD_1 [12].

However, in many instances the OLAP query may consist of only one or two simple GMDJ expressions. In this case, the advantage of the coalescing is limited, because we may still have to synchronize the base-results structure at the coordinator after its construction. We present two results specific to the distributed query processing of GMDJs, that permit synchronization reduction.

Proposition 3. *Consider a GMDJ query* $Q = MD(B, R, (l_1, \ldots, l_m), (\theta_1, \ldots, \theta_m))$. *Let* B *be the result of evaluating a relational expression* $\mathscr{B}$ *on* R, *let* $R_1, \ldots, R_n$ *be a partition of* R, *and let* B_i *be the result of evaluating* $\mathscr{B}$ *on* R_i. *Suppose that* $B = \bigsqcup_i B_i$. *Let* B *have key attributes* K. *If* θ_j *entails* θ_K, *the test for equality on the attributes in* K, $\forall j | 1 \leqslant j \leqslant m$, *then*

$$\begin{aligned} &MD(B, R, (l_1, \ldots, l_m), (\theta_1, \ldots, \theta_m)) \\ &\quad = MD(\pi_{\mathbf{B}} H, H, (l''_1, \ldots, l''_m), (\theta_K, \ldots, \theta_K)), \end{aligned}$$

where $H = \bigsqcup_i H_i$ *and*

$$H_i = MD(B_i, R_i, (l'_1, \ldots, l'_m), (\theta_1, \ldots, \theta_m)).$$

Proposition 3 states that, if $\mathscr{B}$ is evaluated over the relation R (e.g., $\pi_K(R)$) and each condition tests for equality on the key attributes K, then we can omit the synchronization of the base-values relation.

Example 7. Consider again Example 2. Following Proposition 3, we can compute B_0 and the first GMDJ B_1 directly, instead of synchronizing in between the two computations as would otherwise be the case. Thus, the number of synchronizations can be cut down from three to two, with a potential 40% reduction in the amount of data transferred.

In the instance of Example 3, we are able to avoid the need to transfer X_0^{s1} and X_0^{s2} from the sites to the mediator, and X_0 to the sites.

Theorem 6. *Consider the GMDJ* $Q = MD_2(MD_1(B, R, (l_{11}, \ldots, l_{1k}), (\theta_{11}, \ldots, \theta_{1k})), R, (l_{21}, \ldots, l_{2m}), (\theta_{21}, \ldots, \theta_{2m}))$. *Let* $R_1 \cup \cdots \cup R_n$ *be a partitioning of the detail relation* R. *For each* R_i, *let* ϕ_i *be a predicate such that for each* $r \in R_i$, $\phi_i(r)$ *is true. Let* $\psi_i^1(b)$ *be the formula* $\forall_r \phi_i(r) \Rightarrow \neg(\theta_{11} \vee \cdots \vee \theta_{1l})(b, r)$, *and let* $\psi_i^2(b)$ *be the formula* $\forall_r \phi_i(r) \Rightarrow \neg(\theta_{21} \vee \cdots \vee \theta_{2m})(b, r)$. *Suppose that* $\forall_j (j \neq i) \Rightarrow (\psi_j^1(b)\ \&\ \psi_j^2(b))$. *Then site* i *does not need to synchronize tuple* b *between the evaluation of* MD_1 *and* MD_2.

Proof. Theorem 6 extends Theorems 1 and 5, given appropriate information about the distribution of R among the sites.

Let $\bar{l}_1 = (l_{11}, \ldots, l_{1k})$, $\bar{l}_2 = (l_{21}, \ldots, l_{2m})$, $\bar{\theta}_1 = (\theta_{11}, \ldots, \theta_{1l})$, and $\bar{\theta}_2 = (\theta_{21}, \ldots, \theta_{2m})$. Let $\bar{l}'_j$ and $\bar{l}''_j$ be the lists of sub-aggregates and super-aggregates, respectively, corresponding to the aggregates in $\bar{l}_j$. Let θ_K be an test for equality on

the attributes of K. Then:

$$MD_2(MD_1(B, R, \bar{l}_1, \bar{\theta}_1), R, \bar{l}_2, \bar{\theta}_2) = MD(B, H, (\bar{l}''_1, \bar{l}''_2), \theta_K),$$

where $H = H_1 \sqcup \ldots \sqcup H_n$, and

$$H_i = MD_2(MD_1(B, R_i, \bar{l}'_1, \bar{\theta}_1), R_i, \bar{l}'_2, \bar{\theta}_2).$$

It follows from the conditions of Theorem 6, that for any tuple $b \in B$, if $RNG(b, R_i, (\theta_1 1 \vee \cdots \vee \theta_1 k)) \neq \emptyset$, then there does not exist an $R_j, j \neq i$ for which $RNG(b, R_j, (\theta_1 1 \vee \cdots \vee \theta_1 k)) \neq \emptyset$ or $RNG(b, R_j, (\theta_2 1 \vee \cdots \vee \theta_2 k)) \neq \emptyset$. It then follows that if $RNG(b, R_i, (\theta_1 1 \vee \cdots \vee \theta_1 k)) \neq \emptyset$, then

$$MD_2(MD_1(B, R, \bar{l}_1, \bar{\theta}_1), R, \bar{l}_2, \bar{\theta}_2) = MD(B, H, (\bar{l}'_1, \bar{l}'_2), \theta_K),$$

where $H = H_1 \sqcup \cdots \sqcup H_n$, and $H_i = MD_2(MD_1(B, R_i, \bar{l}'_1, \bar{\theta}_1), R_i, \bar{l}'_2, \bar{\theta}_2)$, and $H_j = MD_2(MD_1((B/b), R_j, \bar{l}'_1, \bar{\theta}_1), R_i, \bar{l}'_2, \bar{\theta}_2)$, where $j \neq i$. □

Theorem 6 states that it is not necessary to synchronize a tuple $b \in B$ if we know that the only site which updates b's aggregates during MD_1 and MD_2 is site i. If we have strong information about the distribution of R among the sites, we can avoid synchronizing between the evaluation of MD_1 and MD_2 altogether.

For a distributed setting, we may often be able to identify a partition attribute.

Definition 3. *Let $R_1 \cup \cdots \cup R_n$ be a partitioning of the detail relation R. For each R_i, let ϕ_i be a predicate such that for each $r \in R_i$, $\phi_i(r)$ is true. An attribute A is a partition attribute iff*: $\forall_{i \neq j} \pi_A(\sigma_{\phi_i}(R)) \cap \pi_A(\sigma_{\phi_j}(R)) = \emptyset$.

Given metadata identifying a partition attribute for the distributed detail data, we can then determine whether or not synchronization reduction is possible as follows:

Corollary 1. *Consider the GMDJ $Q = MD_2(MD_1(B, R, (l_{11}, \ldots, l_{1l}), (\theta_{11}, \ldots, \theta_{1l})), R, (l_{21}, \ldots, l_{2m}), (\theta_{21}, \ldots, \theta_{2m}))$. If $\theta_{11}, \ldots, \theta_{1l}, \theta_{21}, \ldots, \theta_{2m}$ all entail condition $R.A = f(B.A)$, where $f(B.A)$ is a bijective function on A, and A is a partition attribute, then MD_2 can be computed after MD_1 without synchronizing between the GMDJs.*

Thus, by performing a simple analysis of ϕ_i and θ, we are able to identify a significant subset of queries where synchronization reduction is possible. We note that more than one attribute can be a partition attribute, e.g., if a partition attribute is functionally determined by another attribute.

Example 8. Consider the query of Example 2.

$$MD(MD(B_0, F_0, l_1, \theta_1) \rightarrow B_1, F_1, l_2, \theta_2),$$

where B_0 is $\pi_{\texttt{SAS,DAS}}(\texttt{Flow})$, F_0 is the `Flow`, l_1 is $(cnt(*) \rightarrow cnt1, sum(NB) \rightarrow sum1)$, θ_1 is ($\texttt{F}_0\texttt{.SAS} = \texttt{B}_0\texttt{.SAS}$ & $\texttt{F}_0\texttt{.DAS} = \texttt{B}_0\texttt{.DAS}$), F_1 is `Flow`, l_2 is $(cnt(*) \rightarrow cnt2)$, and θ_2 is ($\texttt{F}_1\texttt{.SAS} = \texttt{B}_1\texttt{.SAS}$ & $\texttt{F}_1\texttt{.DAS} = \texttt{B}_1\texttt{.DAS}$ & $\texttt{F}_1\texttt{.NB} \geqslant \texttt{sum1/cnt1}$).

Without any synchronization reduction, the evaluation of this GMDJ expression would require multiple passes over the `Flow` relation, and three synchronizations, one for the base-values relation and one each for the results of the two GMDJs.

Let the detail table of our query be `Flow(RID, SAS, DAS, NB)`, where `RouterId` (RID) comprises a partition attribute. Furthermore, let us assume that all packets from any given `SourceAS` only pass through a router with a particular `RouterId` (RID). Then we might have the source distributed over two sites, S_1 and S_2 as follows (this data is different from that used in previous examples to illustrate the use of this optimization):

`Flow at site S1`

RID	SAS	DAS	NB
1	18	29	200
1	18	29	300
1	18	1	2500

`Flow at site S2`

RID	SAS	DAS	NB
2	16	6	700
2	16	29	400
2	16	6	900

The mediator still sends the initial schema and query plan to the local sites, where the base tables are computed by evaluating $\pi_{SAS,DAS}(Flow)$ to

derive the following two tables for X_0:

X_0^{s1}

SAS	DAS
18	1
18	29

X_0^{s2}

SAS	DAS
16	6
16	29

Since (SourceAS, DestAS) form a key of the base-values relation, Proposition 3 is applicable and no synchronization of the base-values relation is needed. Thus, we directly evaluate the first (inner) GMDJ at the local sites, without the intermediate synchronization step. This results in the following tables:

X_1^{s1}

SAS	DAS	cnt1	sum1
18	1	1	2500
18	29	2	500

X_1^{s2}

SAS	DAS	cnt1	sum1
16	6	2	1600
16	29	1	400

Since RouterId is a partition attribute, SourceAS is a partition attribute as well. Since the θ condition of the inner and outer GMDJs both entails $F_1.SAS = B_1.SAS$, it follows from Corollary 1 that no synchronization is required between these two GMDJs. Thus we can also evaluate the second (outer) GMDJ at the local sites without the intermediate synchronization step. This results in the following tables at the local sites:

X_2^{s1}

SAS	DAS	cnt1	sum1	cnt2
18	1	1	2500	1
18	29	2	500	1

X_2^{s2}

SAS	DAS	cnt1	sum1	cnt2
16	6	2	1600	1
16	29	1	400	1

Finally, the sub-results are passed back to the mediator and synchronized to compute the final result:

X_2

SAS	DAS	cnt1	sum1	cnt2
16	6	2	1600	1
16	29	1	400	1
18	1	1	2500	1
18	29	2	500	1

Essentially, the entire query can be evaluated against the distributed data warehouse with *the entire query being evaluated locally, and with a single synchronization at the coordinator*.

Synchronization reduction is a distinctive feature of distributed GMDJ processing, which is difficult or impossible to duplicate using traditional distributed query optimizations methods. In addition, it is a key factor in keeping the distributed processing of OLAP queries scalable, as we shall show in Section 5.

5. Experimental evaluation

In this section, we describe a set of experiments to study the performance of Skalla. We show the scalability of our strategies and also quantify the performance benefits of our optimizations.

5.1. Setup and data

We used Daytona [25] as the target DBMS for GMDJ expression evaluation in Skalla, both for the local data warehouse sites and the coordinator site. We derived a test database from the TPC-R dbgen program, creating a denormalized 900 Mbyte data set with 6 million tuples (named TPCR). We partitioned the data set on the NationKey attribute (and therefore also on the CustKey attribute). The partitions were then distributed among eight sites.

In each of our test queries, we compute a COUNT and an AVG aggregate on each GMDJ operator. We ran two different experiments with different attributes of the TPCR relation as the grouping attribute. The first set of experiments (high-cardinality) use the Customer.Name attribute, which has 100,000 unique values partitioned

among eight sites. The second set of experiments (low-cardinality) uses attributes between 2000 and 4000 unique values.

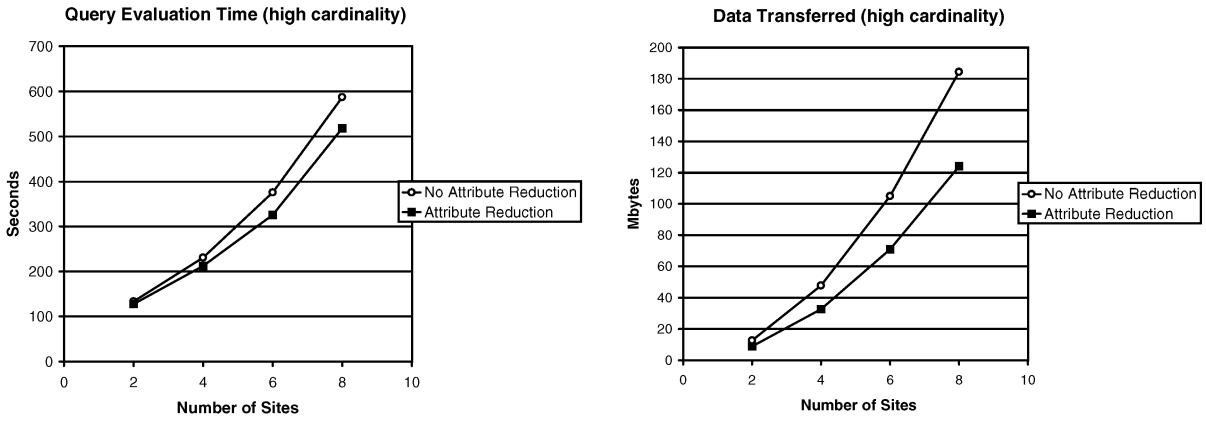
We performed five queries for the experiments documented in this paper, examining the effects of attribute reductions, group reductions, synchronization reductions, and a combination of all the optimizations proposed. We ran two types of experiments. For the first four experimental setups, we fixed the amount of data at each site (dividing the 6 million tuples of the TPCR relation equally among eight sites) and varied the number of sites participating in the query. We use this experimental setup to evaluate the impact of the various individual optimizations. For the final experiment on the combined optimizations, we used four sites and varied the amount of data on each site. For each data point, we measured five executions of a query and used the average.

5.2. Attribute reduction

To test the effect of attribute reduction, we used a query that contains two GMDJs, MD_1 and MD_2. The low-cardinality experiment uses the `Type` attribute, which has 2500 unique values all present at each site. MD_1 depends on the `AVG` computed in the base-values relation, and MD_2 depends on the `AVG` computed by MD_1. Because of these dependencies, no coalescing is possible. We do not apply group reduction or synchronization reduction in this experiment.

In the first set of experiments, we test the effect of attribute reduction. Fig. 3(a) shows the execution time of the high-cardinality attribute reduction query as the number of sites is varied. With eight sites, attribute reduction reduces query evaluation time by 12%. This improvement is largely due to a reduction in the volume of data transferred. As shown in Fig. 3(b), attribute reduction reduces the volume of data transferred by 30%. We note that attribute reduction achieves this improvement in spite of the fact that a large portion of computed aggregates are used in subsequent evaluation rounds.

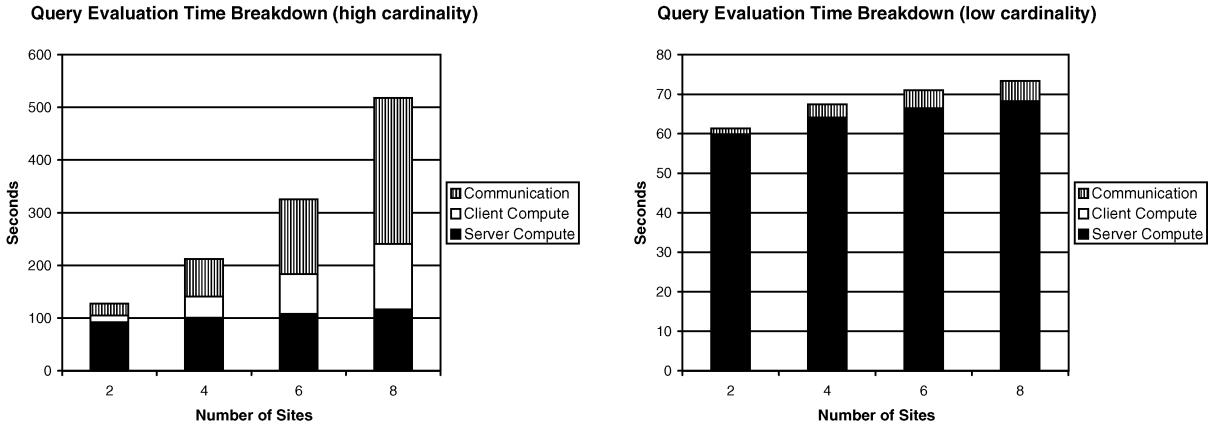
We can break the query evaluation time into three components: site computation time (evaluating the GMDJ queries), coordinator computation time (evaluating the synchronization query), and data communication time. In Fig. 4(a), we show this breakdown for the attribute-reduced high-cardinality query. While the site computation time remains nearly constant, the coordinator computation time and the communication time increase quadratically. This trend is due to the extreme setup of the experiment: the number of groups is large and increases linearly with number of sites (as the groups are partitioned among the sites). The communication overhead increases quadratically because the linearly increasing number of groups must be communicated with a linearly



(a) Query Time

(b) Data Transfer

Fig. 3. Attribute reduction query: (a) query time and (b) data transfer.



(a) High Cardinality

(b) Low Cardinality

Fig. 4. Query evaluation time breakdown for attribute reduction: (a) high-cardinality and (b) low-cardinality.

increasing number of sites. The sites return a quadratic number of groups causing a quadratic increase in coordinator computation time. The low-cardinality experiment (Fig. 4(b)) shows a different trend, as the number of groups is moderate and does not increase much with an increasing number of sites. The coordinator computation is negligible, the site computation is nearly constant, and the communication overhead is small and linearly increasing.

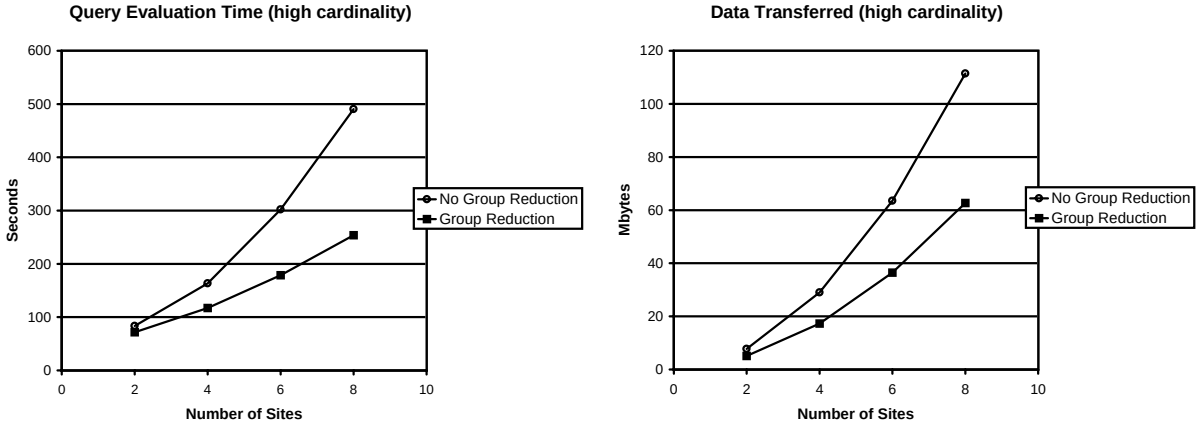
5.3. Group reduction

To test the effect of group reduction, we used a query with two GMDJs, MD_1 and MD_2. MD_1 depends on an aggregate of the group, and MD_2 depends on an aggregate of MD_1. We set the predicates of MD_1 and MD_2 so that they have zero counts in 45% of the groups. We use `Customer.Name` as the grouping attribute. We apply attribute reduction but not synchronization reduction.

In the current implementation of Skalla, the metadata indicates only which attributes of a relation are used for partitioning, but it does not specify the predicate used for partitioning. Therefore, we performed distribution-independent group reduction but not distribution-aware group reduction.

Figs. 5(a) and (b) depict graphs showing the query evaluation time (Fig. 5(a)) and the amount of data transferred (Fig. 5(b)) for distribution-independent group-reduced and the non-group-reduced versions of the group reduction query. The set of non-group-reduced curves shows a quadratic increase in query evaluation time and in the number of bytes transferred. This behavior is due to a linearly increasing number of groups being sent to a linearly increasing number of sites; thus the communication overhead and synchronization overhead increases quadratically. When group reduction is applied, the curves are still quadratic, but to a lesser degree. The distribution-independent (i.e., site side) group reduction solves half of the inefficiency, as the sites send a linear amount of data to the coordinator, but the coordinator sends a quadratic amount of data to the sites. Distribution-aware (i.e., coordinator side) group reduction would make the curves linear.

To see this, we perform an analysis of the number of bytes transferred. Let the number of groups residing on a single site be g, the number of sites be n, and the fraction of sites' group aggregates updated during the evaluation of a grouping variable be c. In the first round, ng groups are sent from the sites to the coordinator. Without group reduction, n^2g groups are sent from the coordinator to the sites, and n^2g groups are sent back. With group reduction, only cng groups are returned. Therefore, the proportion of groups transferred with group reduction versus without



(a) Query Time

(b) Data Transfer

Fig. 5. Group reductions: (a) query time and (b) data transfer.

group reduction is $(ng(2c+1+2n))/(ng(4n+1)) = (2c+2n+1)/(4n+1)$. The number of bytes transferred is roughly proportional to the number of groups transferred, and in fact this formula matches the experimental results to within 5%.

5.4. Coalescing

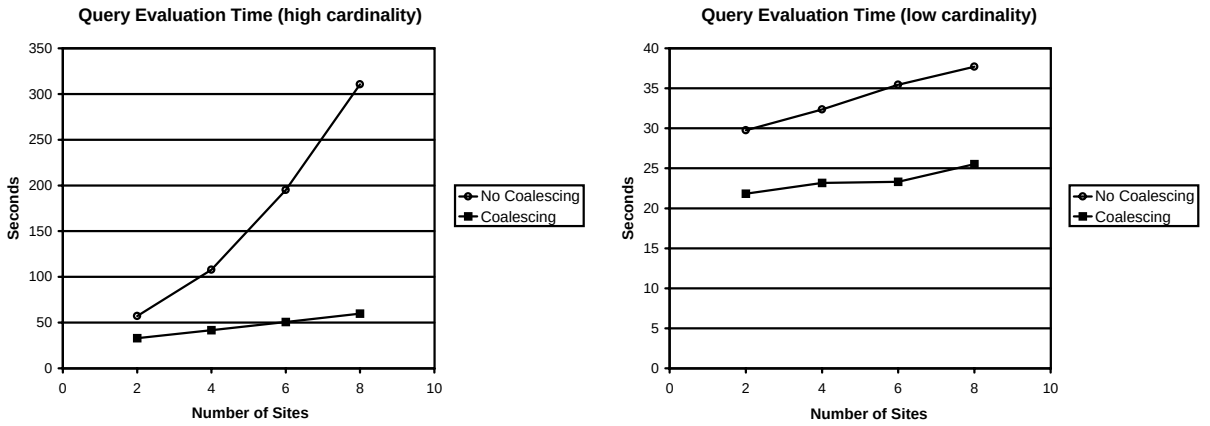
To test the effect of coalescing, we use a query with a single GMDJ MD_1 and compute a `COUNT` and an `AVG` average in MD_1 and also for the group. Therefore, coalescing reduces the query from two rounds to one round. The low-cardinality experiment uses the `Type` attribute. We used attribute reduction in these experiments, but no synchronization reduction or group reduction.

Fig. 6(a) shows the evaluation time of the coalesced and non-coalesced query for high-cardinality data, while Fig. 6(b) shows the results of the same queries over low-cardinality of data. The non coalesced curve in the high-cardinality query shows a quadratic increase in execution time. The coalesced GMDJ curve is linear. There is only one evaluation round, at the end of which the sites send their results to the coordinator, so the volume of data transferred increases linearly with the number of sites. For the low-cardinality query the difference is less dramatic. Even though the amount of data transferred is small, coalescing reduces query evaluation time by 30%, primarily due to a reduction in the site computation time.

5.5. Synchronization reduction

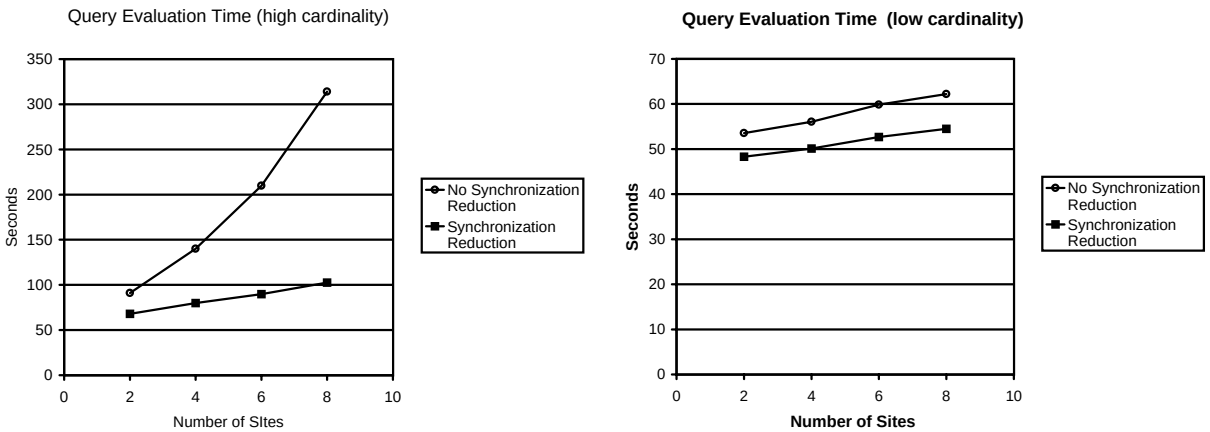
To test the effect of synchronization reduction, we use a query with a single GMDJ MD_1 For the low-cardinality experiment we used `Nation` and `Brand` as the grouping attributes (3750 unique value pairs). Without synchronization, two rounds are required, but with synchronization reduction only one round is required. We used attribute reduction but not group reduction for these experiments.

We test the effect of synchronization reduction without coalescing. Fig. 7(a) shows the query evaluation time of the OLAP query evaluated with and without synchronization reduction for the high-cardinality version, while Fig. 7(b) gives the low-cardinality version of the query. Without synchronization reduction in the high-cardinality query, the query evaluation time is quadratic with an increasing number of sites. With synchronization reduction, the query is evaluated in a single round, and shows a linear growth in evaluation time (due to the linearly increasing size of the output). Thus, synchronization reduction removes the inefficiencies (due to attribute partitioning) seen in the previous experiments. For the low-cardinality query, synchronization reduction without



(a) High Cardinality (b) Low Cardinality

Fig. 6. Coalescing query: (a) high-cardinality and (b) low-cardinality.



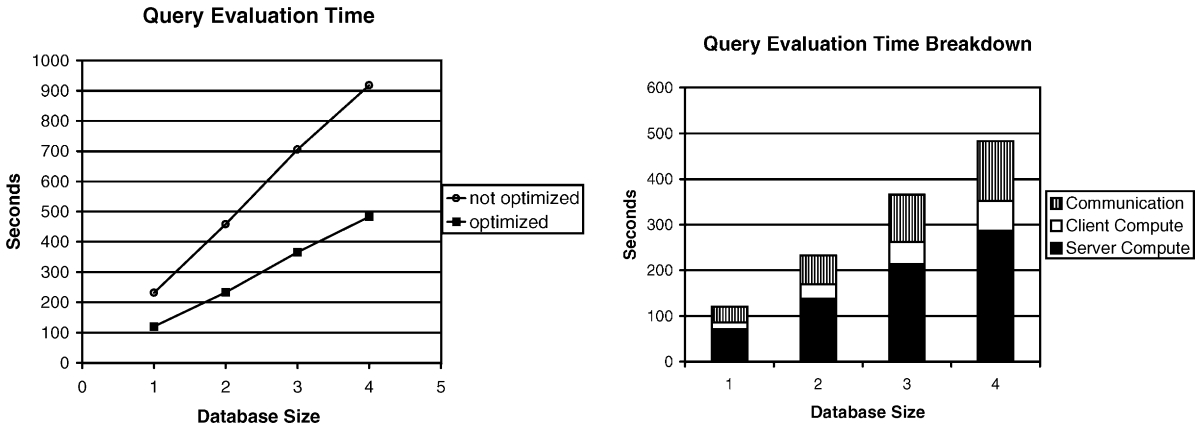
(a) High Cardinality (b) Low Cardinality

Fig. 7. Synchronization reduction query: (a) high-cardinality and (b) low-cardinality.

coalescing reduces the query evaluation time, but not to the same degree achieved with coalescing of GMDJs on the high-cardinality query. This is because coalescing improves computation time as well as reducing communication; thus the work performed by the sites is nearly the same, and the difference in query evaluation time only represents the reduction in synchronization overhead.

5.6. Combined reductions

For this experiment, we fix the number of sites at four, and vary the data set size at each of these sites. We start with the data set used in the previous experiments and increase its size by up to a factor of four. We developed a query for testing the aggregate effect of all reductions. This query has three grouping variables, MD_1, MD_2, and MD_3. MD_1 is pass-reducible to the group-forming pass. MD_2 depends on an aggregate of MD_1, but is synchronization-reducible to MD_1. MD_3 depends on an aggregate of MD_2 but is not synchronization-reducible to MD_2. We designed the predicate of MD_3 so that at any site, it has a non-zero count in 45% of the groups.



(a) Query Evaluation Time

(b) Query Evaluation Time Breakdown

Fig. 8. Combined reductions query: (a) Query evaluation time (b) Query evaluation time breakdown.

We used the attributes `Nation` and `ShipDate` as the grouping attributes (with 62,000 unique values of the pairs). These groups are partitioned among the sites, but the attribute `ShipDate` alone is not a partition attribute.

In our first set of experiments, the number of groups increased linearly with the data set size. The graph in Fig. 8(a) shows the query evaluation time when the optimizations are turned on or off. In both cases there is a linear increase in query evaluation time with increasing database size. Using the optimizations improved the query evaluation time by nearly half. The bar graph of Fig. 8(b) breaks down the evaluation time of the optimized query into the site computation time, coordinator computation time, and communication overhead, showing linear growth in each component. We ran the same set of experiments using a data set in which the number of groups remains constant with an increasing database size, and obtained comparable results.

5.7. Summary

For many queries (e.g., with a moderate number of groups), distributed OLAP evaluation is effective and scalable (see, e.g., Fig. 6(a)). The optimizations discussed in this paper are effective in reducing query evaluation time by a large fraction (see, e.g., Fig. 8(a)).

Distributed OLAP faces a scalability problem when a partition attribute is used as one of the grouping attributes, leading to a quadratic increase in query evaluation time with a linearly increasing number of sites. Most of the work is wasted as the sites do not have tuples for most of the groups sent to them. Two of the optimizations we have considered in this paper are effective in eliminating this inefficiency. Group reduction (both at the coordinator and at the sites) reduces the data traffic (and thus the query evaluation time) From quadratic to linear. Synchronization reduction further takes advantage of the partition attribute by eliminating much of the data traffic altogether.

6. Conclusions

In this paper, we have developed a framework for evaluating complex OLAP queries on distributed data warehouses. We build efficient query plans using GMDJ expressions, which allow the succinct expression of a large class of complex multi-round OLAP queries. In our distributed OLAP architecture, a coordinator manages, collects and correlates aggregate results from the distributed warehouse sites.

The use of GMDJ expressions allows us to avoid complex distributed join optimization

problems. However, query plans involving GMDJ operators also require optimization for best performance. We show a collection of novel GMDJ transformations which allow us to minimize the cost of computation and of communication between the sites and the coordinator.

We built Skalla, a distributed OLAP system which implements the distributed OLAP architecture. Skalla also implements most of the optimizations discussed in this paper. We ran a collection of experiments, and found that the optimizations lead to a scalable distributed OLAP system.

The results we present in this paper are the first steps in the exploration of research issues in the important area of distributed OLAP. Future research topics include:

Multi-tier coordinator architecture: The coordinator architecture used in this paper works well for a small to moderate number of data warehouse sites. As the number of sites increases, various options can be examined to parallelize the operation of the coordinator instances. One option is to run the coordinator on a multiprocessor server. Other options are to use a distributed coordinator, arranged in a hierarchy or a mesh.

Query optimization: We have shown a collection of simple optimization strategies for GMDJs that almost always result in improved performance. Future work is needed to expand the range of optimization strategies and to develop a query optimizer.

Holistic aggregates: We have outlined a number of optimization schemes that could be used to make practicable and optimize distributed evaluation of non-distributive, holistic aggregates. Further work is needed to detail and expand upon the range of optimizations.

Appendix A. Proofs

A.1. The Monoid Comprehension Calculus

For the purposes of the proof in the following section, we will be making use of the *monoid comprehension calculus*. In the following, we provide a brief overview of the monoid comprehension calculus, omitting those aspects not used in this paper. We shall base our presentation of the calculus on Fegaras and Maier [26].

Queries in the monoid comprehension calculus are expressed in terms of monoid comprehension. Informally, a monoid comprehension over the monoid $\oplus$ takes the form $\oplus\{e\|q_1, \ldots, q_n\}$. he merge function $\oplus$ is called the *accumulator* of the comprehension, the expression e is called the *head* of the comprehension. Each term in $q_1, \ldots, q_n$ (where $n \geqslant 0$) is called a *qualifier*, and can be either a *generator* on the form $v \leftarrow e'$, where v is a *range variable* and e' is an expression that constructs a collection, or a *filter* P, where P is a predicate. Examples of monoids would be $+, \times, \cup, \uplus$. Monoids like $+$ and $\times$ are called *primitive monoids* because they construct values of a primitive type. $\cup$ and $\uplus$ are called collection monoids; $\cup$ collects values into a set, whereas $\uplus$ collects values into a bag.

More formally, the monoid comprehension calculus consists of a number of syntactic forms that operate over monoid types. These syntactic forms can be composed to construct yet more complicated terms.

Definition 4. The monoid comprehension calculus consists of the syntactic forms in Table 1, where $\oplus$ is a monoid, $e, e_1, \ldots, e_n$ are terms in the monoid calculus, v is a variable, t is a monoid type, and $q_1, \ldots, q_n$ are qualifiers of the form $v \leftarrow e$ or e.

In the monoid comprehension calculus, relations and tables correspond to sets and bag expressions, and can be collected using the $\cup$ and $\uplus$ monoids respectively. Thus we can write a

Table 1
Monoid comprehension calculus elements

NULL	null value
c	constant
v	variable
$e.A$	record projection
$\langle A_1 = e_1, \ldots, A_n = e_n \rangle$	record construction
e_1 op e_2	where op is a primitive binary function, such as $+, =, <, >$
$\oplus e\|q_1, \ldots, q_n$	comprehension

projection $\pi[A]R$ as

$$\uplus\{r.A \| r \leftarrow R\}.$$

The bag collection monoid collects the tuples generated by $r \leftarrow R$, and conforms them to record projection $r.A$. If we wanted to utilize a duplicate-eliminating projection, we would instead have used the set collection monoid, $\cup$.

Similarly, a simple selection $\sigma[P](R)$ can be expressed as

$$\uplus\{r \| r \leftarrow R, P\}.$$

The monoid comprehension calculus can be put into a canonical form by a number of efficient rewrite rules. The following is the primary rule used in our proof:

$$\oplus\{e \| \bar{q}, v \leftarrow \oplus\{e' \| \bar{r}\}, \bar{s}\} \rightarrow \oplus\{e \| \bar{q}, \bar{r}, v \equiv e', \bar{s}\}. \quad \text{(A.1)}$$

This normalization rule is meaning preserving [26].

The shorthand notation $x \equiv u$ is written to represent the binding of the variable x to the value u. The meaning of this construct is given by the following rule:

$$\oplus\{e \| \bar{r}, x \equiv u, \bar{s}\} \rightarrow \oplus\{e[u/x] \| \bar{r}, \bar{s}[u/x]\}, \quad \text{(A.2)}$$

where $e[u/x]$ is the expression e with u substituted for all the free occurrences of x.

A.2. Proof of Theorem 5

Proof. Theorem 5 is a straightforward specialization to the distributed case of the following general transformation rule for GMDJs:

$$\sigma_{|RNG|>0} MD(B, R, (l_1, \ldots, l_m), (\theta_1, \ldots, \theta_m)) = \sigma_{|RNG|>0} MD(\sigma_{\theta_{1B} \vee \ldots \vee \theta_{mB}} B, R, (l_1, \ldots, l_m), (\theta_1, \ldots, \theta_m)).$$

The $|RNG| > 0$ query assumes that there exists a l_i containing a `COUNT` and that $\theta_i \in \{\theta_1, \ldots, \theta_m\}$ such that $\theta_i \theta_i \wedge (\theta_1 \vee \ldots \vee \theta_m)$. Intuitively, if $\theta_{iB}(b) = \texttt{false}$ for a tuple $b \in B$, then it follows that no aggregate functions of the tuple b will be computed for the aggregate list l_i, i.e., all aggregates in l_i of b will have their initial values. If a tuple b neither fulfills θ_{1B} or θ_{2B} or … or θ_{mB}, then it follows that the tuple b will not have any aggregates computed at all; i.e., $MD(b, R, (l_1, \ldots, l_m), (\theta_1, \ldots, \theta_m)) = b \times N_l$.

Consider the theorem for a single tuple $b \in B$. The equivalence of the theorem then corresponds to stating that this equation:

$$\{s.A, s.cnt \| s \leftarrow \cup\{b.A, cnt : +\{r.c_{11} \| r \leftarrow R, \theta_i\} \| b \leftarrow B\}, cnt > 0\} \quad \text{(A.3)}$$

is equivalent to the following equation:

$$\{s.A, s.cnt \| s \leftarrow \cup\{x.A, cnt : +\{r.c_{11} \| r \leftarrow R, \theta_i\} \| x \leftarrow \cup\{b \| b \leftarrow B, \theta_{iB}\}\}, cnt > 0\}. \quad \text{(A.4)}$$

Eq. (A.4) can then be reduced as follows:

$$\{s.A, s.cnt \| s \leftarrow \cup\{x.A, cnt : +\{r.c_{11} \| r \leftarrow R, \theta_i\} \| b \leftarrow B, x \equiv b, \theta_{iB}\}\}, cnt > 0\} \quad \text{(A.5)}$$

from (A.1)

$$\{s.A, s.cnt \| s \leftarrow \cup\{b.A, cnt : +\{r.c_{11} \| r \leftarrow R, theta_i\} \| b \leftarrow B, \theta_{iB}\}\}, cnt > 0\} \quad \text{(A.6)}$$

from (A.2)

$$\{s.A, s.cnt \| s \leftarrow \cup\{b.A, cnt : +\{r.c_{11} \| r \leftarrow R, \theta'_i, \theta_{iB}\} \| b \leftarrow B, \theta_{iB}\}\}, cnt > 0\} \quad \text{(A.7)}$$

from $\theta_i = \theta'_i \wedge \theta_{iB}$.

Consider the two expressions

$$\cup\{b.A, cnt : +\{r.c_{11} \| r \leftarrow R, \theta'_i, \theta_{iB}\} \| b \leftarrow B, \theta_{iB}\}\}$$

and

$$\cup\{b.A, cnt : +\{r.c_{11} \| r \leftarrow R, \theta'_i, \theta_{iB}\} \| b \leftarrow B\}\}.$$

Recall that $attr(\theta_{iB}) \subseteq \mathbf{B}$, which implies that $\forall b \in B$, $cnt = 0$ if $\theta_{iB} = \texttt{false}$. It thus follows that the predicate θ_{iB} discards a tuple $b \in B$, iff that same tuple would also be discarded by $cnt > 0$. □

References

[1] R. Cáceres, N. Duffield, A. Feldmann, J. Friedmann, A. Greenberg, R. Greer, T. Johnson, C. Kalmanek, B. Krishnamurthy, D. Lavelle, P. Mishra, K. K. Ramakrishnan, J. Rexford, F. True, J. van der Merwe, Measurement and analysis of IP network usage and behavior, IEEE Communications Magazine 38 (5) (2000) 144–151.

[2] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, J. Rexford, F. True, Deriving traffic demands for operational IP networks: methodology and experience, IEEE/ACM Transactions on Networking 9 (3) (2001) 265–279.

[3] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, H. Pirahesh, Datacube: a relational aggregation operator generalizing group-by, cross-tab, and sub-totals, Data Mining and Knowledge Discovery 1 (1) (1997) 29–53.
[4] G. Graefe, U. Fayyad, S. Chaudhuri, On the efficient gathering of sufficient statistics for classification from large SQL databases, in: Proceedings of the International Conference on Knowledge Discovery and Data Mining, New York, 1998, pp. 204–208.
[5] D. Chatziantoniou, K.A. Ross. Querying multiple features of groups in relational databases, in: Proceedings of the International Conference on Very Large Databases, Mumbai, India, 1996, pp. 295–306.
[6] K.A. Ross, D. Srivastava, D. Chatziantoniou, Complex aggregation at multiple granularities, in: Proceeding of the International Conference on Extending Database Technology, Valencia, Spain, 1998, pp. 263–277.
[7] A. Shatdal, J.F. Naughton. Adaptive parallel aggregation algorithms, in: Proceedings of the ACM SIGMOD Conference on Management of Data, San Jose CA, 1995, pp. 104–114.
[8] D. Bitton, H. Boral, D.J. DeWitt, W.K. Wilkinson, Parallel algorithms for the executions of relational database operations, ACM 8 (3) (1983) 324–353.
[9] H. Boral, W. Alexander, L. Clay, G. Copeland, S. Danforth, M. Franklin, B. Hart, M. Smith, P. Valduriez, Prototyping Bubba, a highly parallel database system. IEEE TKDE 2(1) (1990).
[10] D. Chatziantoniou, Ad hoc OLAP: expression and evaluation. in: Proceedings of the IEEE International Conference on Data Engineering, Sydney, 1999.
[11] T. Johnson, D. Chatziantoniou, Extending complex ad hoc OLAP, in: Proceedings of the Conference on Information and Knowledge Management, Kansas City, MS, 1999.
[12] D. Chatziantoniou, M.O. Akinde, T. Johnson, S. Kim, The MD-join: an operator for complex OLAP, in: Proceedings of the IEEE International Conference on Data Engineering, Heidelberg, Germany, 2001.
[13] S. Agarwal, R. Agrawal, P.M. Deshpande, A. Gupta, J.F. Naughton, R. Ramakrishnan, S. Sarawagi, On the computation of multidimensional aggregates, in Proceedings of the International Conference on Very Large Databases, Mumbai, India, 1996, pp. 506–521.
[14] M.O. Akinde, M.H. Böhlen, Generalized MD-joins: evaluation and reduction to SQL, in: Databases in Telecommunications II, Rome, September 2001, pp. 52–67.
[15] S. Chaudhuri, U. Dayal, An overview of data warehousing and OLAP technology, SIGMOD Record 26 (1) (1997) 65–74.
[16] K.A. Ross, D. Srivastava, Fast computation of sparse datacubes, in: Proceedings of the International Conference on Very Large Databases, Athens, 1997, pp. 116–125.
[17] S. Ceri, G. Pelagatti, Principles of Distributed Databases, McGraw-Hill, New York, 1984.
[18] D. Kossman, The state of the art in distributed query processing, ACM Computing Surveys 32 (4) (2000) 422–469.
[19] M. T. Özsu, P. Valduriez, Principles of Distributed Database Systems, Prentice-Hall, Englewood Cliffs, 1991.
[20] C.T. Yu, K.C. Guh, A.L.P. Chen, An integrated algorithm for distributed query processing, in: Proceedings of the IFIP Conference on Distributed Processing, Amsterdam, 1987.
[21] R. Elmasri, S.B. Navathe, Fundamentals of Database Systems, 2nd Edition, Benjamin/Cummings, Merlo Park, CA, 1994.
[22] R. Kimball, The Data Warehouse Toolkit, Wiley, New York, 1996.
[23] M.O. Akinde, Complex and distributed on-line analytical processing, Ph.D. Thesis, Aalborg University, Denmark, 2002, to be published.
[24] R. Kimball, K. Strehlo, Why decision support fails and how to fix it, SIGMOD Record 24 (3) (1995) 92–97.
[25] R. Greer, Daytona and the fourth-generation language Cymbal, in: Proceedings of the ACM SIGMOD Conference on Management of Data, Philadelphia, PA, 1999, pp. 525–526.
[26] L. Fegaras, D. Maier, Optimizing object queries using an effective calculus, ACM 26 (4) (2000) 457–516.