Andreas Bauer, Wolfgang Lehner

On Solving the View Selection Problem in Distributed Data Warehouse Architectures

Erstveröffentlichung in / First published in:

15th International Conference on Scientific and Statistical Database Management. Cambridge, 09.-11.07.2003. IEEE, S. 43-51. ISBN 0-7695-1964-4

DOI: https://doi.org/10.1109/SSDM.2003.1214953

Final edited form was published in "15th International Conference on Scientific and Statistical Database Management", Cambridge, 2003. ISBN 0-7695-1964-4.
https://doi.org/10.1109/SSDM.2003.1214953

On Solving the View Selection Problem in Distributed Data Warehouse Architectures

Andreas Bauer

T-Systems Nova GmbH
Merianstr. 32, D-90409 Nuremberg, Germany
andreas.bauer@t-systems.com

Wolfgang Lehner

Database Technology Group
Dresden University of Technology
D-01062 Dresden, Germany
lehner@inf.tu-dresden.de

Abstract

The use of materialized views in a data warehouse instal lation is a common tool to speed up mostly aggregation que ries. The problems coming along with materialized aggre gate views have triggered a huge variety of proposals, such as picking the optimal set of aggregation combinations, transparently rewriting user queries to take advantage of the summary data, or synchronizing pre computed sum mary data as soon as the base data changes. This paper focusses on the problem of view selection in the context of distributed data warehouse architectures. While much research was done with regard to the view selection problem in the central case, we are not aware to any other work dis cussing the problem of view selection in distributed data warehouse systems. The paper proposes an extension of the concept of an aggregation lattice to capture the distributed semantics. Moreover, we extend a greedy based selection algorithm based on an adequate cost model for the distrib uted case. Within a performance study, we finally compare our findings with the approach of applying a selection algo rithm locally to each node in a distributed warehouse envi ronment.

1. Introduction

Data warehouse systems are a well-known concept which provide an integrated and consolidated basis for performing organization-wide analyses that mainly support the decision making process. While research has provided a huge set of technologies for the efficient operation of centralized data warehouse systems, the architectural style of huge data warehouse installations has shifted from a centralized to a decentralized federated or distributed structure. There are various reasons for this shift: organizational units may have started data warehouse projects long before the overall enterprise discovered the idea of an integrated and historic database; long running systems behaving similarly to a data warehouse have incorporated into an organization-wide warehouse infrastructure without changing the running systems; operating a single warehouse system is simply not feasible due to technical and/or administrative restrictions. With this background, well-studied problems with well-known solutions in the centralized context are suddenly experiencing rejuvenation in a de-centralized world. Obviously, not every solution is easily and directly transferable from the centralized to the distributed situation. In this paper, we are targeting one of the most studied problems in the centralized data warehouse architecture: selecting the optimal set of materialized aggregate views to speed up incoming queries constrained by an additional storage overhead and/or maintenance costs to keep the aggregate views synchronized with the base data. In a distributed data warehouse architecture, multiple nodes are connected to each other and may freely share data or issue aggregation queries against other nodes participating in a distributed scenario. The problem of selecting the optimal aggregate view combination for materialization is now additionally constrained by storage capacities *per node*, maximum global maintenance costs, and a query mix *per node*. Furthermore, the distributed selection algorithm is able to consider *replicates of data warehouse information*, i.e. data of a specific granularity may be stored multiple times on different nodes to save communication costs and speed up local queries.

After reviewing and classifying prior work in the context of the view selection problem, section 3 proposes an extended aggregation lattice and section 4 formally introduces the cost model used within the distributed data warehouse scenario. Finally, section 5 presents the distributed view selection algorithm based on the extended aggregation lattice with support of storage, maintenance, communication, and computation costs. Section 6 gives a performance analysis comparing our distributed selection algorithm with a centralized solution. The paper closes with a summary in section 7.

2. Related Work

Solutions to the well-studied problem of view selection can be classified into several categories. The first huge category of *static selection algorithms* is based on a set of user

Final edited form was published in "15th International Conference on Scientific and Statistical Database Management", Cambridge, 2003. ISBN 0-7695-1964-4.
https://doi.org/10.1109/SSDM.2003.1214953

queries and a space or time constraint returning a set of materialization candidates. The first subclass of selection algorithms is constrained by a *maximum storage overhead*. For example [10], extended by [7] additionally considering indices, provides a simple greedy-based selection algorithm based on the maximum benefit per unit space (BPUS). The work of [4] improves this approach by reducing the size of the aggregation lattice using diverse heuristics. [15] proposes a selection process by ordering views with regard to their size. Under certain limitations, all selection algorithms provide a solution of at least *(63 x)%* of the optimal solution with *x* as the ratio of the largest materialization candidate to the overall storage capacity.

The second subclass focusses on the constraint by *maximum maintenance costs* to keep the materialized aggregate views synchronized with the base data. This problem is much harder to solve, because the relative benefit function considering the maintenance costs exhibits non-monotonic properties. [8] provides a greedy algorithm delivering robust solutions even in the context of the non-monotonic behavior. A third subset of view selection algorithms finally operates on multiple view processing plans and applies the idea of multiple query optimization ([14]). The main idea consists in building a reasonably good overall query plan for a given query scenario, where the single nodes reflect either a selection, a join, or an aggregation operator ([16]). The second main category of *dynamic selection algorithms* applies the idea of caching the result of user queries to speed up similar queries of other users ([5], [13], [11]). After successfully testing a query result for admission to the cache, a replacement strategy has to find the set of temporarily cached objects to be released from the cache in favor of the new result set.

After thoroughly examinig related work, we may conclude that none of the prior work is addressing the problem of view selection in a distributed environment, where data and queries may freely be shipped from one node to another. To the best of our knowledge, the work presented in this paper is the first to discuss the distributed case, which we think is the next major step needed to successfully operate an enterprise-wide data warehouse scenario.

3. Problem Space of the Distributed View Selection Process

Before coming up with a selection process to pick an optimal set of materialized views to speed up user queries, we describe the derivability problem, i.e. the question which query can be computed by a set of (distributed) views by extending the concept of an aggregation lattice to cover the distributed case.

3.1. Basic Prerequisites

The derivability theory includes the question how and under which prerequisites a query q can be answered using an equivalent materialized view. Thus, an incoming query and a materialized view are characterized in the same way. Although the concepts of an extended aggregation lattice introduced below may be seamlessly extended to view restrictions, we omit the implication problem of selection predicates for the sake of simplicity. Furthermore, all measures of the queries are supposed to be computable from the views, so that even this perspective can be neglected (see [12] for details). These assumptions entail that a view is essentially characterized by the granularity G described by the grouping attributes $G_1,...,G_n$.

Definition 1: *View*
A view $v = (G)_N$ is characterized by its granularity G with orthogonal grouping attributes $G = \{G_1,..,G_n\}$, $\forall i,j\ 1 \le i,j \le n,\ i \ne j:\ G_i \not\rightarrow G_j$ with $\rightarrow$ being the functional dependency and the warehouse node N where the view is located.

The views of different granularity form a partial ordering denoting which view can be calculated based on which other view.

Definition 2: *Derivability*
A view v_2 with granularity G^{v_2} located at N^{v_2} is directly derivable from a view v_1 with granularity G^{v_1} located at N^{v_1} $(v_2 \ll v_1)$ if:

- A communication edge exists between the two warehouse nodes N^{v_1} and N^{v_2} where the views are located.
- The granularity of v_1 is finer than that of v_2: $\forall G_j^{v_1} \in G^{v_1} \exists G_i^{v_2} \in G^{v_1} \rightarrow G_i^{v_2} \le G_j^{v_1}$
  with $\le$ being the partial order between classification levels (i.e. grouping attributes) which is induced by the functional dependencies between the grouping attributes.

All possible views of a data warehouse, i.e. all combinations of grouping attributes ([6]), can be organized according to their derivability relationship. The result forms a directed acyclic graph for which [10] coined the term »aggregation lattice«. Within this graph, a node represents a single grouping combination (i.e. aggregation granularity). An edge represents the direct derivability relationship. For a detailed reasoning about derivability, see e.g. [9], [3].

3.2. Distributed Aggregation Lattice

The concept of an aggregation lattice has to be extended to capture the distributed case with n data warehouse nodes. Therefore, in addition to the edges representing aggregation dependencies, further edges are introduced to denote the communication channels within the distributed scenario,

Final edited form was published in "15th International Conference on Scientific and Statistical Database Management", Cambridge, 2003. ISBN 0-7695-1964-4.
https://doi.org/10.1109/SSDM.2003.1214953

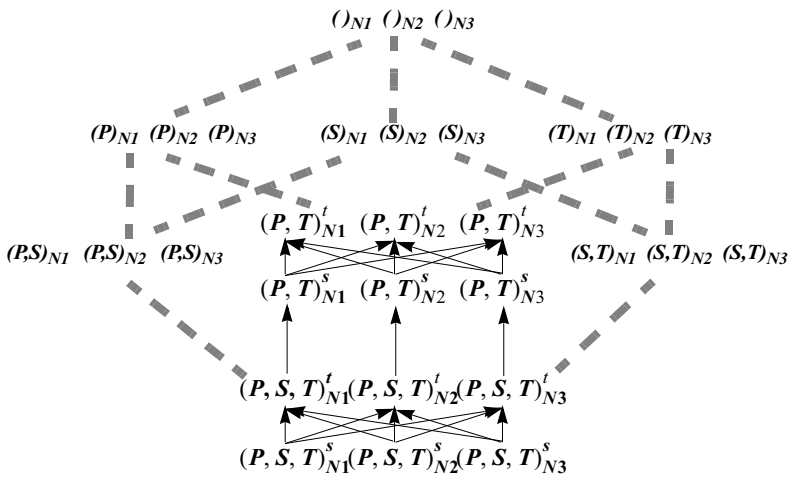


Fig. 1: Distributed Aggregation Lattice

e.g. between lattice nodes of the same granularity at different warehouse nodes. The derivation relationship between a view v_{Ni} and a dependant view v'_{Nj} on a different warehouse node may be omitted as it can be expressed by a combination of other derivation relationships (transitive closure).

Unfortunately, cycles occur in the resulting lattice with communication edges between nodes of the same granularity on different warehouse nodes $v_{Ni}, 1 \le i \le n$. In our approach, they are extended to bipartite subgraphs as illustrated in figure 1 in order to eliminate cycles in the subgraphs. The lattice assumes a network of three warehouse nodes. To keep the scenario as small as possible, only a tiny part of the full lattice is given in its full complexity. The remaining dependencies are just indicated by dashed lines. The main idea to apply the well-known notion of an aggregation lattice to the distributed case is to artificially split each node v_{Ni} into two nodes v^s_{Ni} and v^t_{Ni}. Applying this split to every node yields a subgraph which is fully connected and represents the communication network between nodes of the same granularity. As can be seen in figure 1, this separation of communication and aggregation relationships results in a lattice which again may serve as a solid foundation for the view selection process.

The decision on which node to allocate the base table can be included into the algorithm as presented in the following. A new imaginary base node has to be introduced in the lattice as seen in figure 2. This node, in the running example $(P,S,T)_\perp$, serves as a data provider holding the complete data and distributing it to the base tables conceptually located at the different warehouse nodes. The communication costs $C_{comm(\perp,Ni)}$ between the newly introduced node and the warehouse nodes are set at infinity. This enforces an allocation or replication, respectively, of the base data at the optimal node(s) by the view selection algorithm. In the same

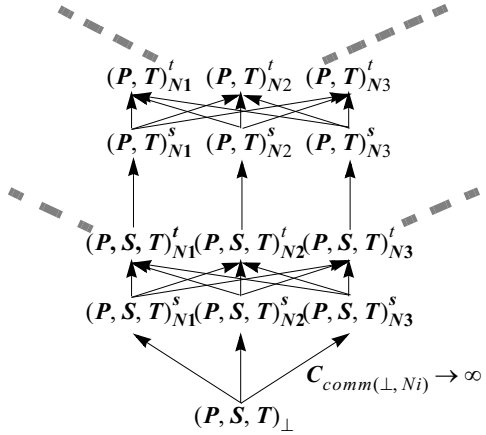


Fig. 2: Base Table Allocation

way, a new top node is introduced to close the lattice with a single node. This node will never be queried and has therefore no influence on the decision algorithm.

Before defining the distributed lattice formally, the definition of *derivability* has to be extended by two different node types *s* and *t*:

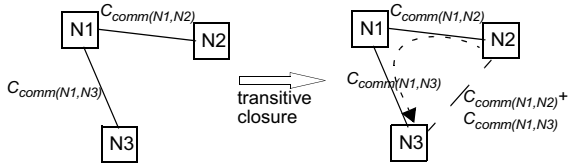
- The node type of v_1 is an *s*-node and that of v_2 a *t*-node ($type(v_1) = s \wedge type(v_2) = t$).

Definition 3: *Distributed Aggregation Lattice*
Let $\mathcal{G} = \mathcal{G}^s \cup \mathcal{G}^t$ with $\mathcal{G}^s, \mathcal{G}^t \subseteq 2^D$ denote the set of all granularity combinations, once annotated as s-node and once as t-node, with the orthogonal dimensions $D = \{d_1, ..., d_n\}$. $\mathcal{G}$ together with the partial order « defines a lattice $\mathcal{L}(\mathcal{G},\ll)$.

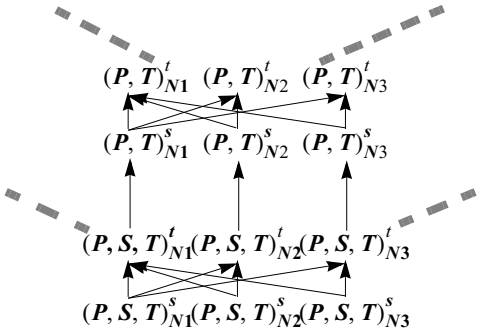
This definition based on the partially ordered set may be equivalently expressed by the algebra $\mathcal{L}(\mathcal{G},\nabla,\Delta)$ with ∇ as the coarsest common predecessor and Δ as the finest common descendant. The edges of the lattice are tagged with costs (see section 4). Edges between lattice nodes of differ-

Final edited form was published in "15th International Conference on Scientific and Statistical Database Management", Cambridge, 2003. ISBN 0-7695-1964-4.
https://doi.org/10.1109/SSDM.2003.1214953

ent granularity are denoted with computation costs for aggregating the data; edges between an *s*-node and a *t*-node, i.e. nodes of same granularity, show the communication costs for transferring data from one warehouse node to another. The above lattice assumes a fully connected communication network (clique), i.e. every warehouse node may query views from any other warehouse node with given communication costs. Obviously, variations of this assumption are possible. First, a communication network has to be considered in which not all nodes are connected to each other, but can exchange data by way of a third node (e.g. coordinator network). A communication graph like this can easily be extended to a fully connected one by building the transitive closure (figure 3a). If each node can directly communicate with only a few other nodes (sparse network), this may be reflected by the lattice. In the communication part between views of the same granularity, edges exist only between lattice nodes v^s_{Ni} and v^t_{Ni} if an edge exists in the communication network (figure 3b).



a) Transformation of the Communication Graph



b) Mapping the Network to the Lattice

Fig. 3: Alternative Communication Networks

4 Cost Model

The selection of a set of materialized views must be based on a criterion such as the benefit gained by the materialization of a view. This section focusses on a cost model for the distributed view selection problem. Given a set of queries representing the characteristic user query behavior, a set of materialized views has to be found to reduce execution costs for the queries. The cardinalities of the materialized views on each node have to satisfy an additional size restriction.

A query $q \in Q$ is always computed from the materialized view producing the minimal cost, so that the effort answering query q amounts to $C(q, M) = min_{v \in M} C(q, v)$ with $C(q,v)$ as the costs to evaluate q from the view v. This query cost is composed of actual evaluation costs to answer query q from view v denoted by $C_{eval}(q,v)$ and the communication costs if the query is stated at a different warehouse node than the view is located given by $C_{comm}(q,v)$. In our framework we consider a linear cost model and assume that both cost functions are dependent on the cardinality of the view $|v|$ and the size of the query result $|q|$. Both cost factors are weighted by an additional factor w comprising the ratio of computation and communication costs so that the overall cost function yields to: $C(q, M) = C_{eval}(q,M) + w \cdot C_{comm}(q, M) = |v| + w \cdot |q|$

This assumption wrt the cost model is justified because $|v|$-many tuples have to be read to compute q and $|q|$-many tuples have to be transferred from the computing node to the shipping node. A complete user query behavior is represented by a query set Q together with the query frequencies f_i expressing how often query q_i is stated. Thus, the overall query costs accumulate to:

$$C(Q, M) = \sum_{q_i \in Q} f_i \cdot C(q_i, M)$$

Since the complexity of the view selection problem to compute the optimal solution is NP-hard ([9]), approximations mainly based on a greedy approach are used to get a suitable solution. The greedy algorithms use a benefit function in order to decide which view to select in each step.

Based on the cost model, the benefit B of an additionally materialized view or set of views $V = \{v_1,...,v_k\}$ can be expressed by the reduction of the total query costs if these views are materialized in addition to an already given set of materialized views M. A necessary prerequisite to deliver a reasonably good solution using a greedy based algorithm is the property of monotonicity with regard to the benefit function. As shown in [8], a greedy approach just considering a single view per iteration may deliver in an arbitrary bad solution, if the monotonicity property is not satisfied. In general, a benefit function B is monotonic with respect to the disjunctive sets of views $V_1,...,V_m$ if the following holds:

$$B(V_1 \cup \ldots \cup V_m, M) \leq \sum_{i=1} B(V_i, M)$$

4.1. A Monotonic Cost Model

The benefit B_S is normalized per unit of space consumption. Otherwise, a single big view would be preferred instead of several smaller views with the same total size, each of those having a smaller benefit than the big view, but gaining a higher benefit in total.

Final edited form was published in "15th International Conference on Scientific and Statistical Database Management", Cambridge, 2003. ISBN 0-7695-1964-4.
https://doi.org/10.1109/SSDM.2003.1214953

$$B_{\boldsymbol{S}}(\{v_1, \ldots, v_k\}, M) = \frac{C(Q, M) - C(Q, M \cup \{v_1, \ldots, v_k\})}{\left|M \cup \{v_1, \ldots, v_k\}\right| - |M|}$$

This benefit definition satisfies the inequation of the monotonicity property stated above. Furthermore, the size of a set of materialized views $|M|$ rises completely monotonically with the number of element views.

4.2. A Non-Monotonic Cost Model

If the limiting quantity is not the size of the views as mentioned in the definition of the distributed view selection problem but the time slot to update the views or the bandwidth of the communication network between the warehouse nodes, this has to be reflected in the benefit function. However, as soon as the benefit function considers update (B_U) or communication costs (B_C) instead of space consumption as the limiting factor, the resulting benefit function is no longer monotonic. An example for the benefit per unit update costs and the benefit per unit communication costs is shown below.

$$B_{\boldsymbol{U}}(\{v_1, \ldots, v_k\}, M) = \frac{C(Q, M) - C(Q, M \cup \{v_1, \ldots, v_k\})}{U(Q, M \cup \{v_1, \ldots, v_k\}) - U(Q, M)}$$

$$B_{\boldsymbol{C}}(\{u_1, \ldots, u_k\}, M) = \frac{C(Q, M) - C(Q, M \cup \{v_1, \ldots, v_k\})}{1 + C_{comm}(Q, M \cup \{v_1, \ldots, v_k\}) - C_{comm}(Q, M)}$$

The reason for the non-monotonicity of the benefit function normalized by update costs is that updating several views individually derived from the base view may cause higher costs than a gradual update of the views. Instead of updating from the base view, a view can be derived from one that has been updated before, which will lower the costs. A detailed example for the non-monotonic behavior of the benefit per unit update costs can be found in [8].

5. Distributed View Selection Algorithms

In this section, the distributed view selection algorithm based on the distributed aggregation lattice will be stated. An example comparing our distributed approach with the standard greedy algorithm applied to each node locally closes this section. First of all, the distributed view selection problem may be stated formally:

Definition 4: *Distributed View Selection Problem*
Let $Q = \{q_1,..,q_n\}$ be a set of queries with access frequencies $\{f_1,..,f_n\}$ and let S_{Ni} be a supplied amount of space for materialized views per warehouse node. A solution to the view selection problem is a set of views $M = \{v_1,..,v_m\}$ with $\sum_j |v_{jNi}| \le S_{Ni}$ so that the total costs to answer all queries $C(Q,M) = \sum_i f_i\, C(q_i,M)$ are minimal.

The benefit function per unit space used in this paper is monotonic as it relates just to the size of evaluated views. Using a monotonic benefit function, [10], [9], and [8] show that a greedy-based algorithm provides a solution of at least (63-x)% of the optimal solution with x as the ratio of the largest materialization candidate to the overall storage capacity. Therefore, it is of tremendous importance to deal with a monotonic benefit function if a lower bound of the quality of the approximate solution has to be satisfied. The lower bound of (63-x)% of the optimal solution of the greedy approach holds if a single limit for the amount of space for materialized views is given. In the distributed environment an individual space limit for each warehouse node has to be stated and therefore the lower bound cannot be guaranteed. The n-dimensional space constraint for n warehouse nodes implies a multiple knapsack problem so that the bound of (63-x)% of the optimal solution drops.

5.1. Distributed Node Set Greedy

The distributed node set greedy algorithm shown in figure 4 may be seen as an seamless extension to the single node greedy algorithm. One major difference is the underlying distributed aggregation lattice introduced in section 3. The algorithm uses the benefit per unit storage space in the case of materialization as a local decision criterion for the selection of a view and is sketched in the following.

The algorithm takes a distributed aggregation lattice and storage limit per warehouse node as input. The set of views to be materialized is initialized with the base table. As long as storage capacity is left ($\exists S_i > 0$), the view with the highest benefit relating to the current materialization configuration is chosen and added to the current set of views to be materialized. Therefore, all nodes of the lattice which are not yet chosen for materialization ($\forall v \in \{L\ M\}$) are taken into account. For each of these lattice nodes, the benefit in case of materialization is calculated ($B(\{v\}, M)$), which results from the sum of cost savings for the computation of all descendant nodes. The node with the highest benefit (v_{opt}) after one complete iteration is added to set of materialization candidates. Moreover, to solve ties when evaluating lattice nodes of the same granularity at different warehouse nodes with identical benefit values (V), a strategy to pick the one at the network node which has the highest storage space left ($max(S_i)$) is applied. This approximation of the included allocation knapsack problem does not guarantee an optimalization assurance in favor of computation performance, as otherwise the independence of the greedy iterations is violated. When allocating a view at a warehouse node, future decisions should be taken into account to get

Final edited form was published in "15th International Conference on Scientific and Statistical Database Management", Cambridge, 2003. ISBN 0-7695-1964-4.
https://doi.org/10.1109/SSDM.2003.1214953

the solution at optimal cost. As the major characteristic of a greedy approach are the isolated decisions in each step of the algorithm, the global optimum must be given up and this selection mechanism has to be introduced as a further heuristic.

```
Input:  L              // distributed lattice with all granularity combinations
        S  (S1,...,Sn) // maximum amount of storage space for
                       // materialized views per node
Output: M              // set of views to be materialized

Begin
    // initialize with lattice node with finest granularity (base table)
    M :  { v0 }
    While (∃ Si > 0)
        vopt :  ∅;  vopt,i :  ∅, ∀1 ≤ i ≤ n;  B({vopt}, M) :  0
        // search view v with maximal benefit per unit space
        Foreach (v ∈ {L-M})  // all not yet materialized lattice nodes
            // compute cost savings for each desendant node
            Foreach (q ∈ {descendants(v)})
                B({v}, M) :  B({v}, M) + (C(q, M) - C(q, M ∪ {v}))
            End Foreach
            // check whether it is new optimal node
            If (B({v}, M) > B({vopt}, M))
                V :  {v' | gran(v)  gran(v') ∧ B({v}, M)  B({v'}, M)}
                vopt :  v
            End If
        End Foreach
        // add optimal view to result set if enough space is left,
        // if necessary choose network node
        If ((max (Si | i ∈ {node(v'), ∀ v' ∈ V}) - SIZE({vopt})) > 0)
            M :  M ∪ {(vopt)i | i ∈ {node(v'), ∀ v' ∈ V, max(Si)}}
            Si :  Si - SIZE({vopt})
        Else
            Si :  0, ∀1 ≤ i ≤ n
        End If
    End While
    Return(M)
End
```

Fig. 4: Distributed Node Set Greedy Algorithm

The complexity of the single node greedy selection algorithm is $O(m \cdot k^2)$ with m as the number of views to be chosen for materialization and k as the number nodes of the aggregation lattice. Given a distributed view selection problem with n warehouse nodes, a multiple usage of the standard approach - one for each node locally - exhibits a complexity of $O(n \cdot m \cdot k^2)$. The distributed view selection algorithm described in this paper yields to $O(m \cdot (n \cdot k)^2)$ as it is comparable to the standard approach with $n \cdot k$ lattice nodes. The distributed approach has the same polynomial class of complexity like the local greedy one. A reduction of the lattice can be done analogous to [4] if a representative set of queries is given.

To improve the above-stated simple greedy algorithm which considers the next iteration solely based on the current configuration, it can be extended to a greedy which looks one step ahead. In each iteration i, all materialization combinations of the ith and $(i+1)$th view are considered. The view combination with the highest benefit is chosen. In the next step, the $(i+1)$th and $(i+2)$th views are chosen for materialization. The complexity rises to $O(m \cdot (n \cdot k)^3)$, but potentially wrong decisions done within two subsequent steps are omitted.

A different approach is the exhaustive search in subsets of size q. All possible subsets with q elements are built and the according benefit in case of materialization is computed. The subset with the highest benefit is used as the starting point for a standard greedy algorithm, which searches the rest of the views until the storage restrictions are reached.

Further extensions, like minimization of the unused storage space due to clipping by including the amount of wasted space into the objective function, are conceivable. However, these are also approximation approaches for which no bound with regard to the quality of the solution can be given. On the positive side, the probability to select a very bad solution is dramatlically reduced.

5.2. Example of the Local versus Distributed View Selection

This section illustrates the single node and node set greedy algorithm with a simple example. The scenario reflects a distributed warehouse architecture with three nodes *N1*, *N2* and *N3*. Computation and communication costs are assumed to be equally weighted ($w = 1$). The query set Q is supposed to be equally distributed on all lattice nodes. The standard lattice forms the base of the selection algorithm for the first run. The distributed aggregation lattice (figure 1) is used for the node set greedy algorithm. The query frequencies f_i for all queries $q_i \in Q$ are set to 1. Each node has a storage space limit for materialized views of $S_{Ni} = 120$. The base table (P,S,T) is initially just materialized at node N.

The first iteration of the standard greedy algorithm at each warehouse node selects the top node $( )$ because it has the highest benefit per unit space (BPUS), as can be seen in figure 5. In the second iteration, node (T) is the most beneficial. In the third iteration finally, node (S) is added to the selected views to be materialized. The sum of the cardinalities of views selected by now is 63. As the storage limit is 120 units, no further view can be materialized, because the next smallest view exhibits a size of 100 storage space units. As a result, these three nodes are selected on each of the three warehouse nodes. The total query costs $C(Q,M)$ with $M = \{()_{N1},()_{N2},()_{N3},(T)_{N1},(T)_{N2},(T)_{N3},(S)_{N1},(S)_{N2},(S)_{N3}\}$ amount to 112295 units. This value is composed of $5 \cdot 6000+50+12+1 = 30063$ units for node *N1* and $6000 \cdot w+3161 \cdot w+1192 \cdot w+600 \cdot w+100 \cdot w+5 \cdot 6000+50+12+1 = 41116$ units for *N2* and *N3*. Nodes *N2* and *N3* have additional communication costs for transferring data from the bottom view at *N1*.

Final edited form was published in "15th International Conference on Scientific and Statistical Database Management", Cambridge, 2003. ISBN 0-7695-1964-4.
https://doi.org/10.1109/SSDM.2003.1214953

	abs. benefit	BPUS
(P,S)	11356	3,59
(P,T)	19232	16,13
(S,T)	21600	36
(P)	11800	118
(S)	**11900**	**238**
(T)	**11976**	**998**
( )	**5999**	**5999**

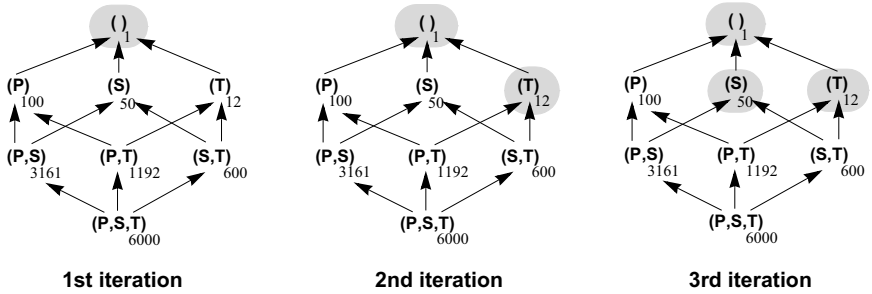
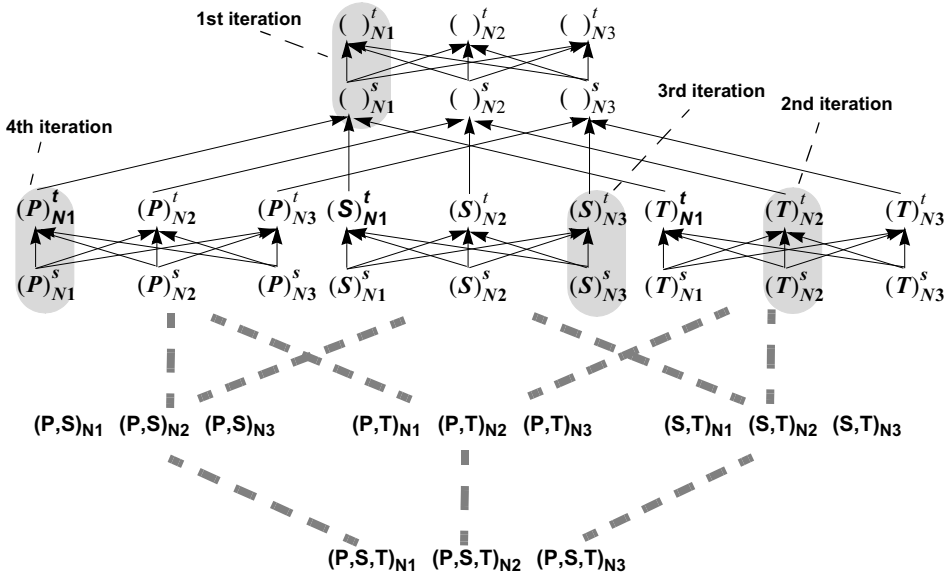


Fig. 5: View Selection with Single Node Greedy Algorithm



1st iteration	benefit	BPUS
$(P,S,T)_{N2/N3}$	11116	1,85
$(P,S)_{N1/N2/N3}$	34068	10,78
$(P,T)_{N1/N2/N3}$	57696	48,40
$(S,T)_{N1/N2/N3}$	64800	108
$(P)_{N1/N2/N3}$	35400	354
$(S)_{N1/N2/N3}$	35700	714
$(T)_{N1/N2/N3}$	35928	2994
$(\ )_{\mathbf{N1}/N2/N3}$	**17997**	**17997**

2nd iteration	benefit	BPUS
$(P,S,T)_{N2/N3}$	11115	1,85
$(P,S)_{N1/N2/N3}$	25551	8,08
$(P,T)_{N1/N2/N3}$	43272	36,30
$(S,T)_{N1/N2/N3}$	48600	81
$(P)_{N1/N2/N3}$	17700	177
$(S)_{N1/N2/N3}$	17850	357
$(T)_{N1/\mathbf{N2}/N3}$	**17964**	**1497**
$(\ )_{N2/N3}$	1	1

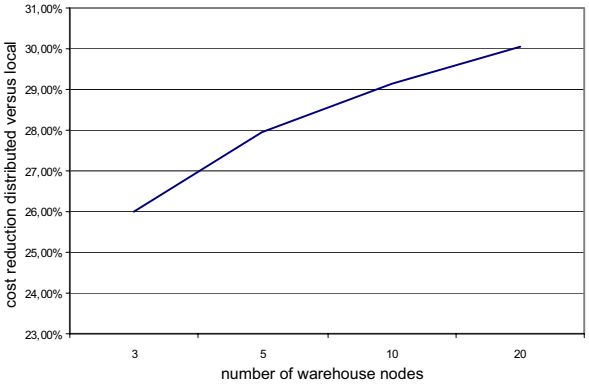
3rd iteration	benefit	BPUS
$(P,S,T)_{N2/N3}$	11103	1,85
$(P,S)_{N1/N2/N3}$	25551	8,08
$(P,T)_{N1/N2/N3}$	28848	24,20
$(S,T)_{N1/N2/N3}$	32400	54
$(P)_{N1/N2/N3}$	17700	177
$(S)_{N1/N2/\mathbf{N3}}$	**17850**	**357**
$(T)_{N1/N3}$	12	1
$(\ )_{N2/N3}$	1	1

4th iteration	benefit	BPUS
$(P,S,T)_{N2/N3}$	11053	1,84
$(P,S)_{N1/N2/N3}$	17034	5,39
$(P,T)_{N1/N2/N3}$	28848	24,20
$(S,T)_{N1/N2/N3}$	16200	27
$(P)_{\mathbf{N1}/N2/N3}$	**17700**	**177**
$(S)_{N1/N2}$	50	1
$(T)_{N1/N3}$	12	1
$(\ )_{N2/N3}$	1	1

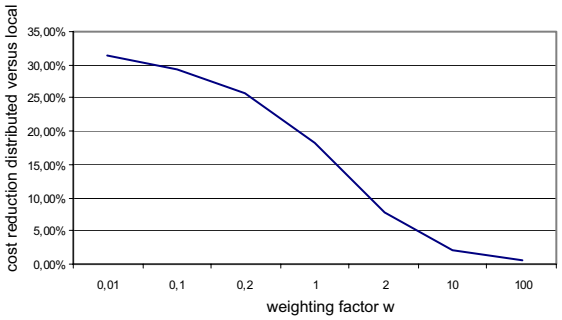
Fig. 6: View Selection with Node Set Greedy Algorithm

The comparative scenario is based on the single distributed lattice (figure 6) for all three warehouse nodes. The first iteration determines the top node with the highest benefit per unit space. If several nodes with the same benefit arise, the one at the warehouse node with most free space for materialized views is selected. Here, view $( )_{N1}$ is chosen. In the second iteration, the top nodes of *N2* and *N3* are not selected, because now they can easily be retrieved from node *N1* and their benefit is just the costs for the saved network transfer. The algorithm takes four iterations until the space constraint of 120 units per node is reached. The total query costs $C(Q,M)$ with $M = \{()_{N1},(T)_{N2},(S)_{N3},(P)_{N1}\}$ amounts to 94721 units. This means a cost reduction of 17574 ($\hat{}$ *15,6 %*) units compared to the single node greedy approach. The used storage space with *1+12+50+100 = 163* units is even smaller than *(1+12+50)·3 = 189* units in the first case. In bigger scenarios where the dedicated query is set bigger, the savings of query costs are higher.

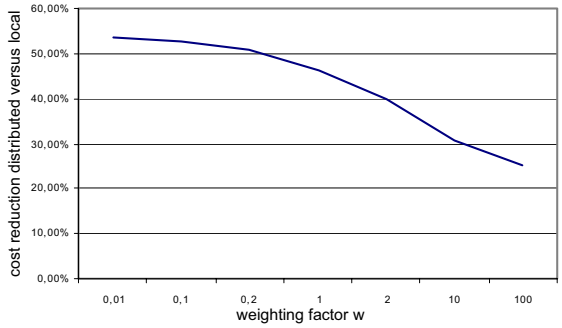
Final edited form was published in "15th International Conference on Scientific and Statistical Database Management", Cambridge, 2003. ISBN 0-7695-1964-4.
https://doi.org/10.1109/SSDM.2003.1214953



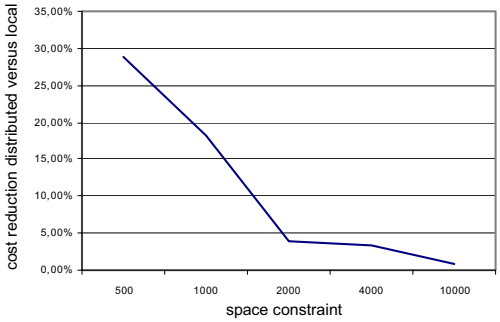
a) Influence of the Number of Warehouse Nodes



b) Influence of the Weighting Factor (Equally Distributed Queries)



c) Influence of the Weighting Factor (Query Set)



d) Influence of the Space Constraint

Fig. 7: Cost Reduction of the Distributed versus the Local Algorithm

6. Performance Studies

To illustrate the advantage of our distributed node set greedy algorithm with regard to the view selection problem, it will be compared with the single node greedy algorithm applied locally to each warehouse node in a more complex scenario. The scenario comprises of a schema with three dimensions with four and two times five aggregation levels per dimension. The query load for the first set of runs is equally distributed, i.e. every possible grouping combination is queried once. The second query set consists of ten randomly generated queries per warehouse node with query frequencies between 10 and 30.

Figure 7 a) compares the distributed with the single node set greedy algorithm for different numbers of warehouse nodes in the network. The performance gain increases with the number of network nodes, as the full potential can only be tapped with more complex systems. If a view doesn't have to be stored locally, but can instead be queried from a remote node, there is more storage space left on the local node to materialize further views, which of course contribute to a higher total benefit. In the above tests, a weighting factor $w = 1$ is assumed, so that computation and communication exhibits equal weights. Figure 7 b) shows the result of our distributed node set greedy algorithm compared to the conservative greedy approach with a variation in the weighting factors. We can easily see that, the higher the communication costs compared to computation costs are, the smaller is the benefit of the distributed approach in comparison to applying the algorithms locally to each warehouse node. The support for the queries by the materialized views in the distributed case is more efficient when a smaller set of queries (second query set) is given as more queries can be answered by some preaggregate located somewhere in the system (figure 7 c).

As illustrated in figure 7 d) the advantage of the distributed node set greedy decreases with larger space constraints for materialized views, i.e. more views can be materialized per warehouse node. The abscissa shows different amounts of storage for aggregates. The fact table is assumed to have 1000 storage units. With a large amount of memory, the same complete set of materialized views per node can be stored in the non-distributed case. Thus, all required data is stored locally and no network access is necessary. With small amounts of storage for materialized views in a range of 50-100% of the fact table size, there is a good advantage

Final edited form was published in "15th International Conference on Scientific and Statistical Database Management", Cambridge, 2003. ISBN 0-7695-1964-4.
https://doi.org/10.1109/SSDM.2003.1214953

of the distributed approach compared to the local one. At a storage size of 2000 units the graph bends sharply, due to the possibility to store further big fine granular views which support many queries on the warehouse nodes. Thus, the distributed approach, by answering queries remotely, cannot show to advantage.

7. Summary

This paper discusses the problem of picking the optimal set of views for materialization in a distributed data warehouse environment. A single node and node-set greedy algorithms based on an extended aggregation lattice to adequately reflect the communication costs are compared. The performance test illustrates the different distributed greedy algorithms with the centralized solution locally applied to each node. The study shows that our proposed approach yields results significantly better than the greedy-based solution directly applied locally to each node. The approach also seamlessly delivers a replication schema by returning views that should be multiply materialized. Future work will examine the possibility of load balancing by introducing CPU constraints and a normalization by computation costs, or constraining network transmission in the benefit function as well as a more fine granular view selection by a partitioning of the lattice nodes ([2]). The objective function used is the total query costs for the complete distributed warehouse system. Different optimization goals could be the response time per single query or an optimal load balancing.

A more sophisticated approach addressing load balancing will also distinguish between storage and calculation. This means that the node where a query is stated, the node where a materialized view is located and the node where the query is computed can be three different nodes. Computation may even be split for a distributed query execution. The partitioning approach introduced in [2] allows to consider query hot spots. Combined with the approach described in this paper, it is possible to support the distributed query load and find a partitioning scheme at optimal cost.

In conclusion, we believe that the area of distributed data warehousing is the next major step in building huge data warehouse systems. Proposed solutions for the centralized scenario should be adequately adapted to this context. In providing a view selection algorithm for the distributed case we have opened that door for many more interesting research opportunities in this area.

References

[1] Albrecht, J.; Huemmer, W.; Lehner, W.; Schlesinger, L.: Using Semantics for Query Derivability in Data Warehouse Applications. In: *Proceedings of the 4th International Conference on Flexible Query Answering Systems* (FQAS'00, Warsaw, Poland, Oktober 25 28), 2000

[2] Albrecht, J.; Bauer, A.; Redert, M.: Supporting Hot Spots with Materialized Views. In: *Proceedings of the 2nd International Conference on Data Warehousing and Knowledge Discovery* (DAWAK 2000, London, UK, September 4 6), 2000

[3] Albrecht, J.; Guenzel, H.; Lehner, L.: Set Derivability of Multidimensional Aggregates. In: *Proceedings of the 1st International Conference on Data Warehousing and Knowledge Discovery* (DAWAK'99, Florence, Italy, August 30 September 1), 1999

[4] Baralis, E.; Paraboschi, S.; Teniente, E.: Materialized Views Selection in a Multidimensional Database, in: *23rd International Conference on Very Large Data Bases* (VLDB'97, Athens, Greece, August 25 29), 1997

[5] Deshpande, P.; Ramasamy, K.; Shukla, A.; Naughton, J.: Caching Multidimensional Queries Using Chunks, in: *27th International Conference on the Management of Data* (SIGMOD '98, Seattle, USA, June 2 4), 1998

[6] Gray, J.; Bosworth, A.; Layman, A.; Pirahesh, H.: Data Cube: A Relational Aggregation Operator Generalizing Group By, Cross Tab, and Sub Total. In: *Proceedings of the 12th International Conference on Data Engineering* (ICDE'96, New Orleans (LA), U.S.A., February 26 March 1), 1996

[7] Gupta, H.; Harinarayan, V.; Rajaraman, A.; Ullman, J.D.: Index Selection for OLAP, in: *13th International Conference on Data Engineering* (ICDE'97, Birmingham, U.K., April 7 11), 1997

[8] Gupta, H.; Mumick, I.: Selection of Views to Materialize Under a Maintenance Cost Constraint. In: *Proceedings of the 7th International Conference on Database Theory* (ICDT'99, Jerusalem, Israel, January 10 12), 1999

[9] Gupta, H.: Selection of Views to Materialize in a Data Warehouse. In: *Proceedings of the 6th International Conference on Database Theory* (ICDT'97, Delphi, January 8 10), 1997

[10] Harinarayan, V.; Rajaraman, A.; Ullman, J.D.: Implementing Data Cubes Efficiently. In: *25th International Conference on Management of Data*, (SIGMOD'96, Montreal, Canada, June 4 6), 1996

[11] Kotidis, Y.; Roussopoulos, N.: DynaMat: A Dynamic View Management System for Data Warehouses. In: *Proceedings ACM International Conference on Management of Data* (SIGMOD'99, Philadephia (PA), U.S.A., June 1 3), 1999

[12] Lenz, H. J.; Shoshani, A.: Summarizability in OLAP and Statistical Data Bases. In: *Proceedings of the 9th International Conference on Statistical and Scientific Database Management* (SSDBM'97, Olympia (WA), USA, August 11 13), 1997

[13] Scheuermann, P.; Shim, J.; Vingralek, R.: WATCHMAN: A Data Warehouse Intelligent Cache Manager, in: *22nd International Conference on Very Large Data Bases* (VLDB '96, Bombay, India, September 3 6), 1996

[14] Sellis, T.: Multiple Query Optimization. In: *ACM Transactions on Database Systems, 13(1)*, 1988

[15] Shukla, A.; Deshpande, P.; Naughton, J.: Materialized View Selection for Multidimensional Datasets. In: *Proceedings of the 24th International Conference on Very Large Data Bases* (VLDB'98, New York City, August 24 27), 1998

[16] Yang, J.; Karlapalem, K.; Li, Q.: Algorithms for Materialized View Design in Data Warehousing Environment, in: *23rd International Conference on Very Large Data Bases* (VLDB'97, Athens, Greece, August 25 29), 1997