Handling large workloads by profiling and clustering

Matteo Golfarelli

DEIS - University of Bologna
40136 Bologna - Italy
golfare@csr.unibo.it

Abstract. View materialization is recognized to be one of the most effective ways to increase the Data Warehouse performance; nevertheless, due to the computational complexity of the techniques aimed at choosing the best set of views to be materialized, this task is mainly carried out manually when large workloads are involved. In this paper we propose a set of statistical indicators that can be used by the designer to characterize the workload of the Data Warehouse, thus driving the logical and physical optimization tasks; furthermore we propose a clustering algorithm that allows the cardinality of the workload to be reduced and uses these indicators for measuring the quality of the reduced workload. Using the reduced workload as the input to a view materialization algorithm allows large workloads to be efficiently handled.

1 Introduction

During the design of a data warehouse (DW), the phases aimed at improving the system performance are mainly the logical and physical ones. One of the most effective ways to achieve this goal during logical design is *view materialization*. The so-called view materialization problem consists of choosing the best subset of the possible (*candidate*) views to be precomputed and stored in the database while respecting a set of system and user constraints (see [8] for a survey). Even if the most important constraint is the disk space available for storing aggregated data, the quality of the result is usually measured in terms of the number of disk pages necessary to answer a given workload.

Despite the efforts made by research in the last years, view materialization remains a task whose success depends on the experience of the designer that, adopting rules of thumb and applying the trial-and-error approach, may lead to acceptable solutions. Unlike other issues in the Data Warehouse (DW) field, understanding why the large set of techniques available in the literature have not been engineered and included in some commercial tools is fundamental to solving the problem. Of course the main reason is the computational complexity of view materialization that makes all the approaches proposed unsuitable for workloads larger than about forty queries. Unfortunately, real workloads are much larger and are not usually available during the DW design but only when the system is on-line. Nevertheless, the designer can estimate the core of the workload at design phase but such a rough approximation will lead to a largely sub-optimal solution.

We believe that the best solution is to carry out a rough optimization at design time and to refine the solution by tuning it, manually or automatically, when the system is on-line on the base of the real workload. The main difficulty with this approach is the huge size of the workload that cannot be handled by the algorithms known in the literature. In this context the contribution of the paper is twofold: firstly we propose a technique for profiling large workloads that can be obtained from the log file produced by the DBMS when the DW is on line. The statistical indicators obtained can be used by the designer to characterize the DW workload thus driving the logical and physical optimization tasks. The second contribution concerns a clustering algorithm that allows the cardinality of the workload to be reduced and that uses the indicators in order to measure the quality of the reduced workload. Using the reduced workload as the input to a view materialization algorithm allows large workloads to be efficiently handled. Since clustering is an independent way of preprocessing, all the algorithms presented in the literature can be adopted during the views selection phase. Figure 1 shows the framework we assume for our approach: OLAP applications generate SQL queries whose logs are periodically elaborated to determine the statistical indicators and a clustered workload that can be handled by a view materialization algorithm that produces new patterns to be materialized.

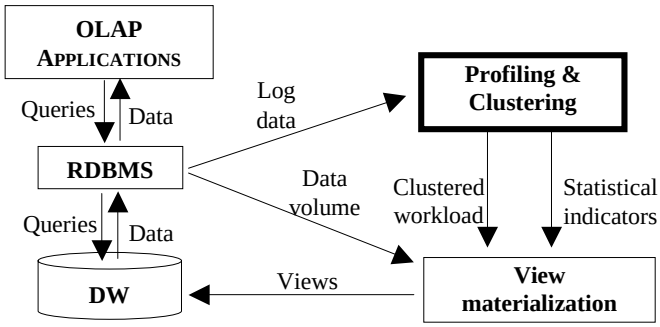

Fig. 1. Overall framework for the view materialization process

To the best of our knowledge only few works directly faced the workload size problem; in particular, in [5] the authors proposed a polynomial time algorithm that explores only a subset of the candidate views and delivers a solution whose quality is comparable with other techniques that run in exponential time. In [1] the authors propose a heuristic reduction technique that is based on the functional dependencies between attributes and excludes from the search space those views that are “similar” to other ones already considered. With respect to ours, this approach does not produce any representative workload to be used for further optimizations.

Clustering of queries in the field of DWs has been recently used to reduce the complexity of the plan selection task [2]: each cluster has a representative for whom the execution plan, as determined by the optimizer, is persistently stored. Here the concept of similarity is based on a complex set of features that it is necessary to encode when different queries can be efficiently solved using the same execution plan. This idea has been implicitly used in several previous works where a global optimization plan was obtained given a set of queries [7].

The rest of the paper is organized as follows: Section 2 presents the necessary background, Section 3 defines the statistical indicators for workload profiling; Section 4 presents the algorithm for query clustering while in Section 5 a set of experiments, aimed at proving its effectiveness, are reported. Finally in Section 6 the conclusions are drawn.

2 Background

It is recognized that DWs lean on the multidimensional model to represent data, meaning that indicators that *measure* a fact of interest are organized according a set of *dimensions of analysis*; for example, sales can be measured by the `quantity sold` and the `price` of each sale of a given product that took place in a given store and on a given day. Each dimension is usually related to a set of attributes describing it at different aggregation levels; the attributes are organized in a *hierarchy* defined according to a set of functional dependencies. For example a product can be characterized by the attributes `PName`, `Type`, `Category` and `Brand` among which the following functional dependencies are defined: `PName`→`Type`, `Type`→`Category` and `PName`→`Brand`; on the other hand, stores can be described by their geographical and commercial location: `SName`→`City`, `City`→`Country`, `SName`→`CommArea`, `CommArea`→`CommZone`.

On relational solutions, the multidimensional nature of data is implemented on the logical model by adopting the so-called *star scheme*, composed by a set of, fully denormalized, *dimension tables*, one for each dimension of analysis, and a *fact table* whose primary key is obtained by composing the foreign keys referencing the dimension tables. The most common class of queries used to extract information from a star schema are GPSJ [3] that consists of a selection, over a generalized projection over a selection over a join between the fact table and the dimension table involved.

It is easy to understand that grouping heavily contributes to the global query cost and that such a cost can be reduced precomputing (materializing) that aggregated information that is useful to answer a given workload. Unfortunately, in real applications, the size of such views never fits the constraint given by the available disk space and it is very hard to choose the best subset to be actually materialized. When working on a single fact scheme and assuming that all the measures contained in the elemental fact table are replicated a view is completely defined by its aggregation level.

Definition 1 The *pattern* of a view consists of a set of dimension table attributes such that no functional dependency exists between attributes in the same pattern.

Possible patterns for the sales fact are: P_1 = {`Month, Country, Category`}, P_2 = {`Year, Sname`}, P_3 = {`Brand`}. In the following we will use indifferently the terms pattern and view and we will refer to the query pattern as the coarsest pattern that can be used to answer the query.

Definition 2 Given two views V_i, V_j with patterns P_i, P_j respectively, we say that V_i can be *derived* from V_j ($V_i \leq V_j$) if the data in V_i can be calculated from the data in V_j.

Derivability determines a partial-order relationship between the views, and thus between patterns, of a fact scheme. Such partial-order can be represented by the so-called *multidimensional lattice* [1] whose nodes are the patterns and whose arcs show a direct derivability relationship between patterns.

Definition 3 We denote with $P_i \oplus P_j$ the least upper bound (*ancestor*) of two patterns in the multidimensional lattice.

In other words the ancestor of two patterns corresponds to the coarsest one from which both can be derived.

Given a set of queries the ancestor operator can be used to determine the set of views that are potentially useful to reduce the workload cost (candidate views). The candidate set can be obtained, starting from the workload queries, by iteratively adding to the set the ancestors of each couple of patterns until a fixed point is reached. Most of the approaches to view materialization try to determine first the candidate views, and to choose the best subset that fits the constraints later. Both problems have an exponential complexity.

3 Profiling the workload

Profiling means determining a set of indicators that captures the workload features that have an impact on the effectiveness of different optimization techniques. In particular, we are interested in those relevant to the problem of view materialization and that help the designer to answer queries like: "*How suitable to materialization is the workload ?*", "*How much space do I need to obtain good results ?*".

In the following we propose four indicators that have proved to properly capture all the relevant aspects and that can be used as guidance by the designer that manually tunes the DW or as input to an optimization algorithm for a materialized view section. All the indicators are based on the concept of cardinality of the views associated to a given pattern that can be estimated knowing the data volume of the fact scheme that we assume to contain the cardinality of the base fact table and the number of distinct values of each attribute in the dimension tables. The cardinality of an aggregate view can be estimated using Cardenas' formula. In our case the objects are the tuples in the elemental fact table with pattern P_0 (whose number $|P_0|$ is assumed to be known) while the number of buckets is the maximum number of tuples, $|P|_{Max}$, that can be stored in a view with pattern P and that can be easily calculated given the cardinalities of the attributes belonging to the pattern, thus

$$Card(P) = \Phi(|P|_{Max}, |I_0|) \quad (1)$$

3.1 Aggregation level of the workload

The aggregation level of a pattern P is calculated as:

$$Agg(P) = 1 - \frac{Card(P)}{|P_0|} \quad (2)$$

Agg(*P*) ranges between [0,1[, the higher the values the coarser the pattern. The average aggregation level (*AAL*) of the full workload $W = \{Q_1, \ldots Q_n\}$ can be calculated as

$$AAL = \frac{1}{n} \sum_{i=1}^{n} Agg(P_i) \tag{3}$$

where P_i is the pattern of query Q_i.
In order to partially capture how the queries are distributed at different aggregation levels we also include the aggregation level standard deviation (*ALSD*), which is the standard deviation for *AAL*:

$$ALSD = \sqrt{\frac{\sum_{i=1}^{n} (Agg(P_i) - AAL)^2}{n}} \tag{4}$$

AAL and *ALSD* characterize to what extent the information required by the users is aggregated and express the willingness of the workload to be optimized using materialized views. Intuitively, workloads with high values of *AAL* will be efficiently optimized using materialized views since they determine a strong reduction of the number of tuples to be read. Furthermore, the limited size of such tables allows a higher number of views to be materialized. On the other hand, a low value for *ALSD* denotes that most of the views share the same aggregation level further improving the usefulness of view materialization.

3.2 Skewness of the workload

Measuring the aggregation level is not sufficient to characterize the workload; in fact workloads with similar values of *AAL* and *ALSD* can behave differently, with respect to materialization, depending on the attributes involved in the queries. Consider for example two workloads $W_1 = \{Q_1, Q_2\}$ and $W_2 = \{Q_3, Q_4\}$ formulated on the Sales fact and the pattern of their queries:

- P_1 = {`Category`, `City`} $Card(P_1) = 2100$
- P_2 = {`Type`, `Country`} $Card(P_2) = 1450$
- P_3 = {`Category`, `Country`} $Card(P_3) = 380$
- P_4 = {`Brand`, `CommZone`} $Card(P_4) = 680$

Materializing a single view to answer both the queries in the workload is much more useful for W_1, than for W_2 since in the first case the ancestor is very "close" to the queries ($P_1 \oplus P_2$={`Type`, `City`}) and still coarse, while in the second case it is "far" and fine ($P_3 \oplus P_4$={`SName`, `PName`}).

This difference is captured by the distance between the two patterns that we calculate as:

$$Dist(P_i, P_j) = Agg(P_i) + Agg(P_j) - 2\, Agg(P_i \oplus P_j) \tag{5}$$

Dist(P_i, P_j) is calculated in terms of distance of P_i and P_j from their ancestor that is the point of the multidimensional lattice closest to both the views. Figure 2 shows two different situations on the same multidimensional lattice: even if the aggregation level of the patterns is similar, the distance between each couple change significantly.
The average skewness (*ASK*) of the full workload $W = \{Q_1, \ldots Q_n\}$ can be calculated as

$$ASK = \frac{2}{n \cdot (n-1)} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} Dist(P_i, P_j) \tag{6}$$

where P_z is the pattern of query Q_z. *ASK* ranges in [0,2[[1] . Also for the skewness indicator it is useful to calculate the standard deviation (Skewness Standard Deviation, *SKSD*) in order to evaluate how the distances between queries are distributed with respect to their mean value:

$$SKSD = \sqrt{\frac{2}{n \cdot (n-1)} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \left(Dist(P_i, P_j) - ASK\right)^2} \tag{7}$$

Intuitively, workloads with low values for *ASK* will be efficiently optimized using materialized views since the similarity of the query patterns makes it possible to materialize few views to optimize several queries.

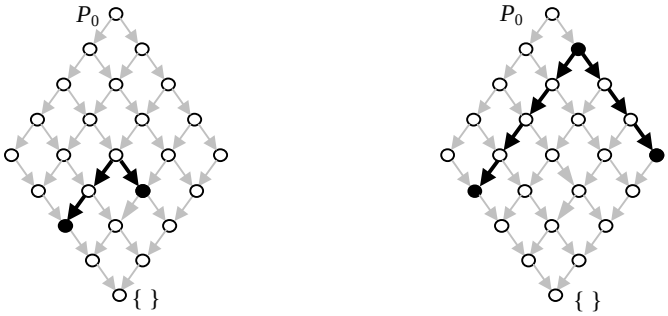

Fig. 2. Distance between close and far patterns

4 Clustering of queries

Clustering is one of the most common techniques for classification of features into groups. Several algorithms have been proposed in the literature (see [4] for a survey) each suitable for a specific class of problems. In this paper we adopted the hierarchical approach that recursively agglomerates the two most similar clusters forming a dendogram whose creation can be stopped at different levels to yield different clustering of data, each related to a different level of similarity that will be evaluated using the statistical indicators introduced so far. The initial clusters contain a single query of the workload that represent them. At each step the algorithm looks

[1] The maximum value for *ASK* depends on the cardinalities of the attributes and on the functional dependencies defined on the hierarchies, thus it cannot be defined without considering the specific star schema.

for the two most similar clusters that are collapsed forming a new one that is represented by the query whose pattern is the ancestor of their representative. Figure 3 shows the output of this process. With a little abuse of terminology we write $q_x \oplus q_y$ meaning that the ancestor operator is applied to the pattern of the queries.

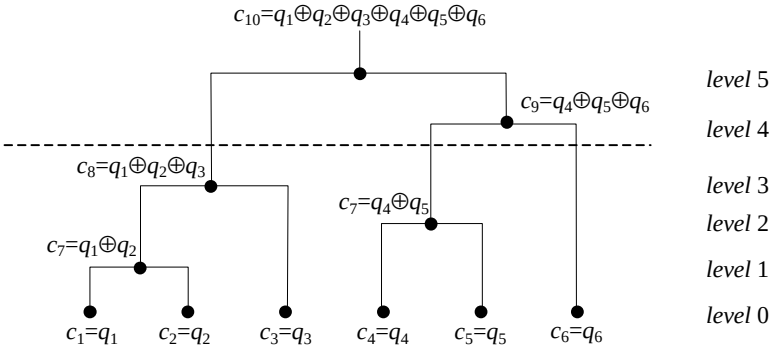

Fig. 3. A possible dendogram for a workload with 6 queries

Similarity between clusters is expressed in terms of the distance, as defined in Section 3.2, between the patterns of their representatives. Each cluster is represented by the ancestor of all the queries belonging to it and is labeled with the sum of the frequencies of its queries. This simple, but effective, solution reflects the criteria adopted by the view materialization algorithms that rely on the ancestor concept when choosing one view to answer several queries. The main drawback here is that the value of *AAL* tends to decrease when the initial workload is strongly aggregated. Nevertheless the ancestor solution is the only one ensuring that the cluster representative effectively characterizes its queries with respect to materialization (i.e. all the queries in the cluster can be answered on a view on which the representative can also be answered). Adding new queries to a cluster inevitably induces heterogeneity in the aggregation level of its queries thus reducing its capability to represent all of them. Given a clustering $Wc = \{C_1,\ldots C_m\}$, we measure the compactness of the clusters in terms of similarity of the aggregation levels of the queries in each cluster as:

$$IntraALSD = \frac{1}{m}\sum_{i=1}^{m} ALSD_i \tag{8}$$

where $ALSD_i$ is the standard deviation of the aggregation level for queries in the cluster C_i. The lower *IntraALSD* the closer the queries in the clusters.

As to the behavior of *ASK*, it tends to increase when the number of clusters decreases since the closer queries are collapsed earlier than the far ones. While this is an obvious effect of clustering a second relevant measure of the compactness of the clusters in $Wc = \{C_1,\ldots C_m\}$ can be expressed in terms of internal skewness:

$$IntraASK = \frac{1}{m}\sum_{i=1}^{m} ASK_i \tag{9}$$

where ASK_i is the skewness of the queries in the cluster C_i. The lower *IntraASK* the closer the queries in the clusters.

The ratio between the statistical indicators and the corresponding intra cluster ones can be used to evaluate how well the clustering models the original workload; in particular we adopted this technique to define when the clustering process must be stopped; the stop rule we adopt is as follows:

$$\text{Stop if } \frac{AAL}{IntraAAL} > T_{AL} \vee \frac{ASK}{IntraASK} > T_{SK}$$

In our tests both T_{AL} and T_{SK} have been set to 5.

5 Tests and discussion

In this section we present four different tests aimed at proving the effectiveness of both profiling and clustering. The tests have been carried out on the LINEITEM fact scheme described in the TPC-H/R benchmark [9] using a set of generated workloads. Since selections are rarely take into account by view materialization algorithms our queries do not present any selection clause. As to the materialization algorithm, we adopted the classic one in the literature proposed by Baralis et al. [1]; the algorithm first determines the set of candidate views and then heuristically chooses the best subset that fits given space constraints. Splitting the process into two phases allows us to estimate both the difficulty of the problem, that we measure in terms of the number of candidate views, and the effectiveness of materialization that is calculated in terms of the number of disk pages saved by materialization. The cost function we adopted computes the cost of a query Q on a star schema S composed by a fact table FT and a set $\{DT_1,\ldots, DT_n\}$ of dimension tables as

$$Cost(Q,S) = Size(FT) + \sum_{i \in Dim(Q)} \left(Size(DT_i) + Size\left(PK_i\right)\right) \tag{10}$$

where *Size*( ) function returns the size of a table/index expressed in disk pages, *Dim*(Q) returns the indexes of the dimension tables involved in Q and PK_i is the primary index on DT_i. This cost function assumes the execution plan that is adopted by Redbrick 6.0 when no select conditions are present in a query on a star schema.

5.1 Workload features fitting

The first test shows that the statistical indicators proposed in Section 3 effectively summarize the features of a workload. Four workloads, each made up of 20 queries, have been generated with different values for the indicators. Table 1 reports the value of the parameters and the resulting number of candidate views that confirms the considerations made in Section 3. The complexity of the problem mainly depends on the value of the *ASK* and is more slightly influenced by *AAL*. The simplest workloads

to be elaborated will be those with highly aggregated queries with similar patterns, while the most complex will be those with very different patterns with a low aggregation level. It should be noted that on increasing the size of the worklfoads, those with a "nice" profile still perform well, while the others quickly become too complex. For example workloads WKL5, WKL6, whose profile follows those of WKL1 and WKL4 respectively, in Table 1 contains 30 queries: while the number of candidate views remains low for WKL5, it explodes for WKL6. Actually, we stopped the algorithm after two days of computation on a PENTIUM IV CPU (1GHz). The profile is also useful to evaluate how well the workload will behave with respect to view materialization. Figure 4.a shows that, regardless of the difficulty of the problems, workloads with high values of *AAL* are strongly optimized even when a limited disk space is available for storing materialized views. This behavior is induced by the dimension, and thus by the number, of the materialized views that fits the space constraint as it can be verified in Figure 4.b.

Table 1. Number of candidate views for workloads with different profiles

Name	*AAL*	*ALSD*	*ASK*	*SKSD*	N. Candidate views
WKL1	0.835	0.307	0.348	0.393	97
WKL2	0.186	0.245	0.327	0.269	124
WKL3	0.790	0.278	0.810	0.391	596
WKL4	0.384	0.153	0.751	0.216	868
WKL5	0.884	0.297	0.316	0.371	99
WKL6	0.352	0.276	0.668	0.354	> 36158

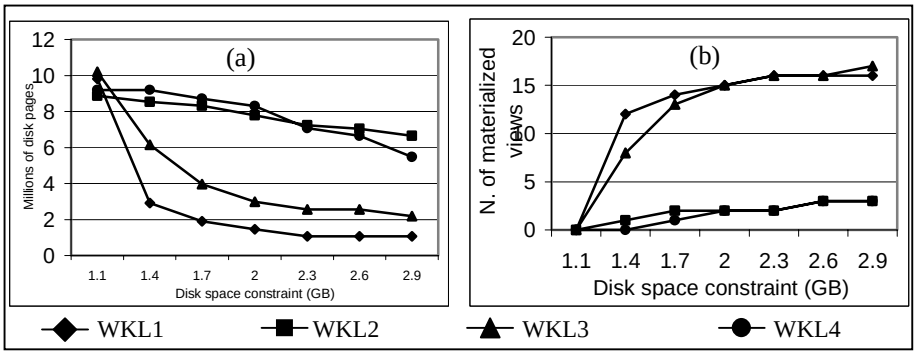

Fig. 4. Cost of the workloads (a) and number of materialized views (b) on varying the disk space constraint for the workloads in Table 1

5.2 Clustering suboptimality

The second test is aimed at proving that clustering produces a good approximation of the input workload, meaning that applying view materialization to the original and clustered workload does not induce a too heavy suboptimality. With reference to the workloads in Table 1, Table 2 shows how change the behavior and the effectiveness of the view materialization algorithm changes for an increasing level of clustering. It

should be noted that the number of candidate views can be strongly reduced inducing, in most cases, a limited suboptimality. By comparing the suboptimality percentages with the statistical indicator trends presented in Figure 5, it is clear that suboptimality arises earlier for workloads where *IntraASDL* and *IntraASK* increase earlier.

5.3 Handling large workloads

When workloads with hundred of queries are considered it is not possible to measure the suboptimality induced by the clustered solution since the original workloads cannot be directly optimized. On the other hand, it is still possible to compare the increase of the performance with respect to the case with no materialized views and it is also interesting to show how the workload costs change depending on the number of queries included in the clustered workload and how the cost is related to the statistical indicators.

Table 2. Effects of clustering on the view materialization algorithm applied to workload in Table 1

WKL	#. Cluster	# Cand.Views	#. Mat.Views	% SubOpt	Stop rule at
WKL1	15	90	12	0.001	3
	10	68	7	0.308	
	5	25	3	40.511	
WKL2	15	79	2	0.000	6
	10	38	2	2.561	
	5	6	2	4.564	
WKL3	15	549	10	1.186	7
	10	156	7	22.146	
	5	16	4	65.407	
WKL4	15	321	2	0.0	4
	10	129	2	0.0	
	5	17	2	0.0	

Table 3 reports the view materialization results for two workloads, WKL 7 (AAL:0.915, ALSD:0.266, ASK: 0.209, SKSD: 0.398) - WKL 8 (AAL: 0.377, ALSD: 0.250, ASK: 0.738, SKSD: 0.345), containing 200 queries. The data in the table and the graphs in Figure 6 confirm the behaviors deduced from previous tests: the effectiveness of view materialization is higher for workloads with high value of *AAL* and low value of *ASK*. Also the capability of the clustering algorithm to capture the features of the original workload depends on its profile, in fact workloads with higher values of ASK require more queries (7 for WKL7 vs. 20 for WKL8) in the clustered workload to effectively model the original one. On the other hand it is not useful to excessively increase the clustered workload cardinality since the performance improvement is much lower than the increase of the computation time.

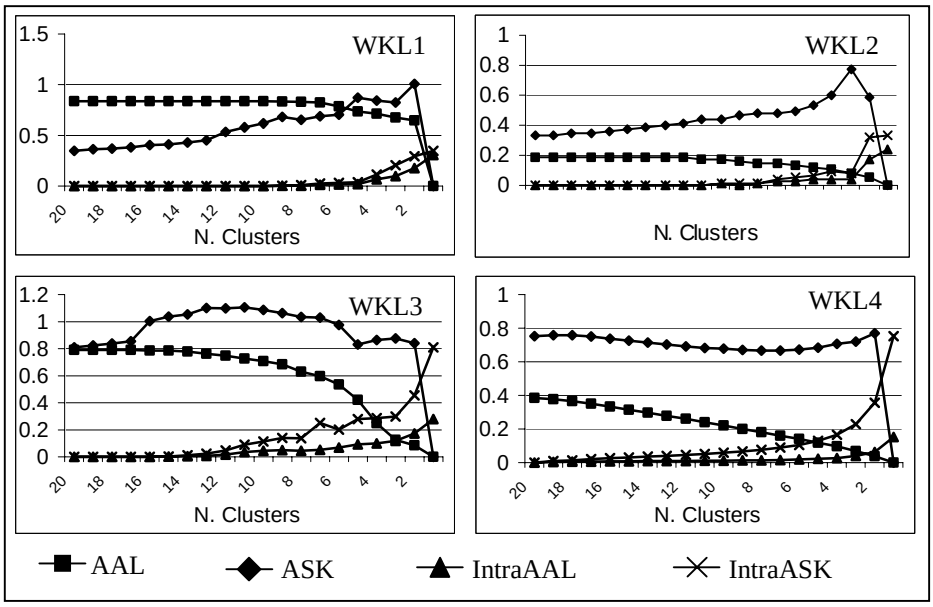

Fig. 5. Trends of the statistical indicators for increasing levels of clustering and for different workloads.

Table 3. Effects of clustering on the view materialization algorithm applied to workload in Table 1

WKL	#. Cluster	# Cand.Views	#. Mat.Views	%Cost Reduction	Comp. Time (sec.)	Stop rule at
WKL7	30	12506	17	90.6	43984	6
	20	4744	15	89.0	439	
	10	384	9	83.3	39	
	7	64	6	38.9	24	
WKL8	30	17579	5	19.1	78427	19
	20	2125	5	17.8	304	
	10	129	2	2.4	25	

6 Conclusions

In this paper we have discussed two techniques that make it possible to carry out view materialization when the high cardinality of the workload does not allow the problem to be faced directly. In particular, the set of statistical indicators proposed have proved to capture those workload features that are relevant to the view materialization problem, thus driving the designer choices. The clustering algorithm allows large workloads to be handled by automatic techniques for view materialization since it reduces its cardinality slightly corrupting the original characteristics. We believe that the use of the information carried by the statistical indicators we proposed can be

profitably used to increase the effectiveness of the optimization algorithms used in both logical and physical design. For example, in [6] the authors propose a technique for splitting a given quantity of disk space into two parts used for creating views and indexes respectively. Since the technique takes account of only information relative to a single query our indicators can improve the solution by providing the bent of the workload to be optimized by indexing or view materializing.

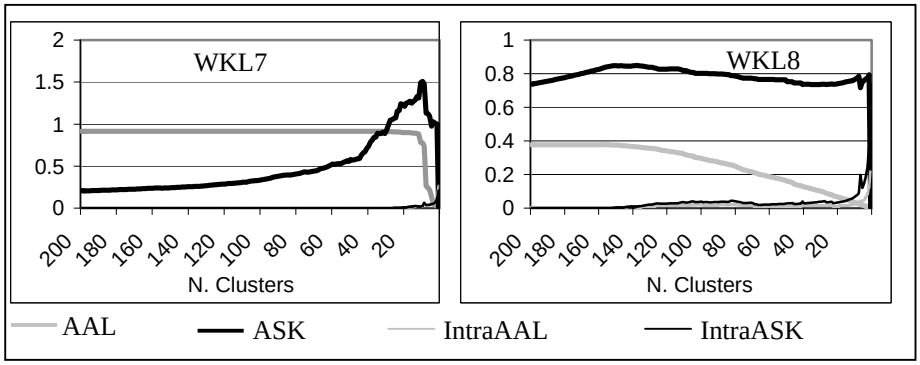

Fig. 6. Trends of the statistical indicators for increasing levels of clustering and for different workloads.

References

[1] E. Baralis, S. Paraboschi and E. Teniente. Materialized view selection in a multidimensional database. In Proc. 23rd VLDB, Greece, 1997.

[2] A. Ghosh, J. Parikh, V.S. Sengar and J. R. Haritsa. Plan Selection Based on Query Clustering, In Proc. 28th VLDB, Hong Kong, China, 2002.

[3] A. Gupta, V. Harinarayan and D. Quass. Aggregate-query processing in data-warehousing environments. In Proc. 21st VLDB, Switzerland, 1995.

[4] A.K. Jain, M.N. Murty and P.J. Flynn. Data Clustering A Review. ACM Computing Surveys, Vol. 31, N. 3, September 1999.

[5] T. P. Nadeau and T. J. Teorey. Achieving scalability in OLAP materialized view selection. In Proc. DOLAP'02, Virginia USA, 2002.

[6] S. Rizzi and E. Saltarelli. View materialization vs. Indexing: balancing space constraints in Data Warehouse Design. To appear in Proc. CAISE'03, Austria, 2003.

[7] T. K. Sellis. Global query Optimization. In Proc. SIGMOD Conference Washington D.C. 1986, pp. 191-205.

[8] D. Theodoratos, M. Bouzeghoub. A General Framework for the View Selection Problem for Data Warehouse Design and Evolution. In Proc. DOLAP'00, Washington D.C. USA, 2000.

[9] Transaction Processing Performance Council. TPC Benchmark H (Decision Support) Standard Specification, Revision 1.1.0, 1998, http://www.tpc.org.