Materialized View Selection: A Survey

Xiang Li
Informatik 5, RWTH Aachen University
lixiang@dbis.rwth-aachen.de

Abstract
This is a draft of my contribution to a book chapter (doi: 10.4018/978-1-60566-816-1.ch005).
We mainly categorize and review the existing approaches to materialized view selection.

View selection addresses the problem of choosing materialized views in the design process of
data warehouses. Informally, it can be stated as, given an estimated query workload and possibly also
sets of candidate views, to select a set of views, such that some operation goals (e.g., average query
response time, maintenance costs, or both) are optimized and meanwhile some resource constraints
are satisfied. View selection, view maintenance, and answering queries using views [Halevy, 2001]
comprise a relatively complete framework for query processing in data warehousing.

In this section, we discuss the challenges of and solutions to the view selection problem. We first
state the formal framework of the view selection problem and some theoretical aspects based on the
framework (Section 1). We then describe a taxonomy that we use to classify view selection techniques
(Section 2). The main body of view selection approaches in the static setting is then discussed (Section
3), where the query workload is assumed to be known in advance. Finally, dynamic view selection
approaches are introduced (Section 4).

1 Theoretical Perspectives
We focus on static view selection in this section and start by introducing a model treating the view
selection problem as an optimization problem.

View selection is handled as an optimization problem, which can be described as a tuple \((S, V, M, Q)\),
where \(S\) is the schema together with some size estimation model, \(V\) is the set of all the possible views
to choose from, \(M\) is a quantity denoting the space available for materializing views, and \(Q\) is a set
of queries. The problem is to find a subset of \(V\) that optimizes the queries against \(Q\) while the storage
requirement is no larger than \(M\). The overall evaluation cost is formulated as a weighted sum over \(Q\):

\[
E(V) = \sum_{q \in Q} E(q, V) \times f_q
\]

where \(f_q\) denotes the frequency of query \(q\), and \(E(q, V)\) is the minimal possible evaluation cost of
\(q\) using the selected views \(V\). Besides the cost of queries, view maintenance can be considered in a
similar way:

\[
U(V) = \sum_{v \in V} U(v) \times f_v
\]
where $f_v$ denotes the frequency of updating view $v$, and $U(v)$ is the cost of updating the view. Thus, the task is a multi-goal optimization problem:

$$C(V) = E(V) + \alpha U(V)$$

where $\alpha$ is a positive weight parameter adjusting these two goals. Normally, these two goals lead to different solutions if considered separately. For example, given a large enough storage, minimizing query evaluation requires all queries in the workload to be materialized as views, while update cost is minimized when only base relations are stored.

The view selection problem is known to be \textit{NP-hard}, since even the simplest form in a hypercube has a reduction from the \textit{set cover} problem \cite{Harinarayan1996}. Chirkova et al. \cite{Chirkova2001} prove that view selection is decidable under equality-selection, projection and join. It is in the same paper \cite{Chirkova2001} that a triple-exponential upper bound is established. It is also known that the view selection problem has an exponential lower bound under either product cost model \cite{Chirkova2001} or sum cost model \cite{Chirkova2002} for the join operator. Furthermore, it is shown that regarding query response time \textit{no} polynomial approximation algorithm can guarantee a result better than $n^{1-\varepsilon}$ \forall $\varepsilon > 0$, if $P \neq NP$ \cite{Karloff1999}. Since $n$ is the worst possible performance ratio, the result defeats any efforts attempting to guarantee performance using approximation algorithms.

2 A Taxonomy of View Selection Approaches

The taxonomy which we used for classifying view selection techniques is illustrated in Figure 1.
Configuration Evolution

One most characteristic dimension of view selection techniques is whether the selected views evolve over time. Most of the techniques are offline/static and will not change once the selection is done. In contrast, online/dynamic approaches, such as WATCHMAN \cite{Scheuermann1996} and Dyna-Mat \cite{Kotidis1999}, adjust the view selection following similar ideas as semantic caching (cf. Section 4). Inside the scope of dynamic view selection, there are two significant issues: namely the caching policy and the reuse policy. The caching policy is the way how the view cache admits/evicts views, while the reuse policy describes how the cached views are used to speed up later queries.

Resource Constraints

Due to the vast volume of data in OLAP systems, storage space is usually the first issue to be considered. However, regarding materialized views, the limit on maintenance time also becomes vital. Too much materialization can make the maintenance time prohibitively long and hence decrease the availability of the OLAP systems.

Optimization Goals

Ideally, view selection techniques are aimed at query response time. However, this goal is usually too expensive to achieve. Many existing approaches opt for some other relevant measures such as the benefit of a unit storage space. It is noted in \cite{Ross1996,Labio1997} that materializing additional views may reduce the overall maintenance costs in presence of a set of pre-determined materialized views. It has been shown that materializing additional views may reduce the overall maintenance costs in presence of a set of pre-determined materialized views \cite{Ross1996,Labio1997}. With the costs of extra storage decreasing dramatically, the update window, however, keeps shrinking. Therefore, many approaches (e.g., \cite{Gupta2005,Sells1997,Mistry2001}) aim at optimizing maintenance costs instead of query response. It is also easy to incorporate both query evaluation costs and materialized view maintenance costs in the goal function (cf. Section 1).

Candidate Views

In general, there are prohibitively many views that can be materialized. Even restricted to possible combinations of selections and group-bys, the number of candidate views is already exponential in terms of the size of the queries. Therefore, the scope of candidate views has to be confined to allow feasible tools. Most of the static view selection techniques (cf. 3) restrict themselves to a given set of candidate views. For example, a natural scope is defined by all the aggregation possibilities in the data cube \cite{Gray1997}, which can be modeled as nodes in a lattice. Dynamic view selection approaches (cf. section 4) make use of a windowed query history of users as a candidate domain. Another type of candidates are the views that are common subexpressions or ancestors of queries. They are not optimal for a single query, but they can benefit more than one query and hence be a globally optimal choice. Moreover, indexes can be also deemed as a special type of views.
Interrelationship Modeling

Candidate views have interactions. Selection of one view may render materialization of another view useless, while several views may have overlapping information and therefore duplications. Dependencies between aggregations in a data cube are modeled using a lattice. Analytic queries involving selection and aggregation can be modeled using hyperplanes covering. Chunks in view selection approaches for MOLAP (Zhao et al., 1997) usually have more intricate relationships such as part-of. Modern tuning tools shipped with commercial database management systems rely on query optimizers to explore implicit interrelationships between candidate views (Zilio et al., 2004; Dageville et al., 2004; Agrawal et al., 2004).

3 Static View Selection

Most of the view selection techniques follow the paradigm of static view selection (or data warehouse configuration) (Theodoratos and Sellis, 1997), which selects views from a given input candidate view set under storage and/or maintenance constraints. The materialized views, once determined offline, will not change over time. Therefore, this line of work is good for cases where the queries are relatively fixed or similar. When the query patterns of the users change dramatically, view selection has to be redone.

In the seminal work of Harinarayan et al. (1996), both the query workload and the candidate views are taken from the nodes in the data cube, which are various group-bys for aggregation. The interrelationship between candidates is then represented by a lattice. For example, the database in Example ?? gives rise to the lattice depicted in Figure 2.

![Lattice for the TPC-D database](image)

Figure 2: The lattice for the TPC-D database

The edges in the lattice depict dependency relationships. One view is dependent on another, if it can be computed using solely data in the latter view. Moreover, the attribute hierarchies inside dimensions can be taken into account by computing a composite lattice of the group-bys and the
attribute hierarchies. An aggregation query can be answered using any of its ancestors, i.e., the views it transitively depends on. With the lattice model at hand, the benefit of materializing a view \( v \) wrt. \( S \), the set of views already selected, is defined as the gain in terms of costs:

\[
B(v, S) = \sum_{w \leq v} \max(0, E(w, S) - E(w, v))
\]

where \( \leq \) is the partial order represented in the lattice, \( E(w, v) \) is the evaluation cost of query \( w \) using view \( v \), and \( E(w, S) \) is the minimal evaluation cost of \( w \) using \( S \), i.e., \( E(w, S) = \min_{u \in S} \{E(w, u)\} \).

A greedy algorithm is then adopted to select the most beneficial view per storage space (Benefit-Per-Unit-Space) up to the storage limit. It is shown that the view selection problem modeled in the above approach is NP-complete by a reduction from minimum set cover (Harinarayan et al., 1996). Even in a simplified linear cost model in which evaluation cost using a view is proportional to the size of the view, the view selection problem can be shown to be NP-complete.

Gupta et al. (1997) extend the framework to accommodate index selection while the query workload is extended to allow both aggregation and selection (also called slice (Gray et al., 1997)). An index, regarded as a special view, is selected only after the view it is defined over is selected. A greedy algorithm is employed to choose at each step a physical structure, either a view or an index, to maximize the benefit per unit space.

In the above case when slice queries are considered, the lattice is no longer sufficient to express the interrelationship of candidate views. A bipartite query-view graph is used in (Gupta et al., 1997), in which there are a set of edges between a query and a view, each labeled with the evaluation of the query against the view using a different index.

There is a trade-off between distribution of storage space between views and indices if there are global space constraints. Bellatreche et al. (2000) introduce a procedure that iteratively approximates a solution and in the dynamic case maintains it as query workloads change and data is added or removed.

The interrelationship modeling is further extended to so-called AND-OR view graphs in (Gupta, 1997). A lattice is then a special case called OR view graph, in which each AND-arc consists of only one edge. Maintenance costs can be incorporated into the framework either as constraints (Gupta and Mumick, 2005), or in the optimization goal (Gupta, 1997). Gupta and Mumick (2005) present a relatively complete summary of various aspects of the general framework.

It is shown that the benefit measure of the result of the greedy algorithm is at least 63% of the benefit of the optimal solution without index selection (Harinarayan et al., 1996), and 46% with index selection (Gupta et al., 1997). Interestingly, Shukla et al. (1998) show that the Benefit-Per-Unit-Space (BPUS) greedy strategy is equivalent to a simpler heuristic algorithm which picks the view with the smallest size at each iteration step. Furthermore, the Pick-By-Size (PBS) algorithm is in \( O(n \log n) \) while BPUS is in \( O(kn^2) \), where \( n \) is the size of the candidate view set and \( k \) is the number of views selected. Karloff and Mihail (1999) argue that the benefit metric does not provide a guarantee for query response time, which is the complementary problem. In fact, if we take \( M \) for the trivial cost without materialized views, \( C^* \) for the optimal cost, and \( C \) for the cost of the result of the greedy algorithm, we can see the query response time ratio is \( C^*/C \leq \alpha + (1-\alpha)M/C \), given a benefit metric guarantee \( \alpha \). It follows immediately that maximizing benefit does not guarantee performance of query
response time. Interestingly, the AND-OR view graph in (Gupta and Mumick, 2005, 1999) is of the same structure of the multiple query processing graph, though greedy algorithm is performed over it.

Another line of work (Theodoratos and Sellis, 1997, 1999; Theodoratos et al., 2001; Yang et al., 1997, Baralis et al., 1997) is influenced by multiple query optimization techniques (Sellis, 1988; Roy et al., 2000), with the aim of finding reusable common sub-expressions of queries. Theodoratos and Sellis (1997) model the problem as a state space search problem, where each state is a multi-query graph specifying Select-Join queries. State transitions achieved by cutting edges or merging nodes represent local modifications of the candidate views selected. Though the exhaustive search gives the optimal view set regarding both query evaluation and maintenance costs, it is exponential and a storage limit is not taken into account. The work is extended in (Theodoratos and Sellis, 1999) to incorporate storage limit, and in (Theodoratos et al., 2001) to allow for projection in queries.

Yang et al. (1997) consider a slightly broader range of queries including aggregation. Their approach proceeds in two steps. The first step is to generate multiple view processing plans (MVPPs), which are query processing plans for multiple queries using intermediate views. With an emphasis on shared join computation, they map the MVPP generation problem into a 0-1 integer programming problem. In the second step, given an optimal MVPP, cost driven optimization with heuristics is performed to select views under storage constraints.

Because of the high complexity in the techniques above, heuristics pruning the search space are necessary. Baralis et al. (1997) explore some heuristics to reduce the search space and promote reuse of views. A view is worth considering if it corresponds to a user query or it is the least common ancestor (in the lattice) of two worthy views. They also observe that it is often advantageous to group by key attributes instead of non-key attributes in terms of reuse.

Mistry et al. (2001) propose a framework tightly integrated with a query optimizer to select materialized views with the goal of reducing overall maintenance costs. The query optimizer is used to select an optimal maintenance plan out of a search space consisting of recomputation plans and incremental plans for materialized views. Given an optimal maintenance plan, a cost driven greedy algorithm is employed to select appropriate additional views.

The lattice based techniques assume that the candidate views are given as input, i.e., nodes in the lattice. Approaches based on multiple query optimization explore a broader scope, including the original queries in the workload and their common subexpressions. Therefore, sub-optimality may result from their restricts over the candidate views.

AutoAdmin (Chaudhuri and Narasayya, 2007), http://research.microsoft.com/DMX/autoadmin/) is an industrial strength view selection tool emphasizing practical SQL queries and scalability for commercial settings. The project started dealing with automatic index selection (Chaudhuri and Narasayya, 1997), and was extended (Agrawal et al., 2000) to allow for selection of materialized views.

The general framework is illustrated in Figure 3.

The first step is the selection of relevant syntactic structures, namely, sets of tables or columns, according to queries in the workload. A variant of frequent itemsets (Agrawal and Srikant, 1994) is employed in this step. Based on the pruned table/column sets, a second step of candidate physical structure selection uses effective heuristics to reduce the search space. An interesting observation is that a materialized view is likely to be beneficial for the whole workload only if it is part of the
best solution for at least one single query in the workload (Agrawal et al. 2000). The merging step generates additional structures by merging the optimal simpler structures obtained so far in the hope of reducing overall cost by reusing structures. In the last step of enumeration, a compromise of an exhaustive search and a greedy algorithm is used: Greedy$(m, k)$. The first $m$ structures are selected by the exhaustive search, while the remaining $k - m$ structures are selected by the greedy algorithm. During the whole process, the cost estimations and query planning are done by the query optimizer using an API creating hypothetical indexes/materialized views (Chaudhuri and Narasayya 1998).

The approach has several advantages. First, it is tightly integrated with the query optimizer, and hence the selected structures are surely to be made use of during query evaluation. Second, neither a restricted set of candidate views has to be provided in advance nor the interrelationships between candidate views have to be modeled explicitly. Third, a query language as rich as SQL is considered with effective heuristics to ensure (near-)optimality.

4 Dynamic View Selection

Static view selection techniques, though very effective, still suffer from a variety of problems. First, they rely on a pre-compiled query workload, and may not perform well for ad hoc queries. Second, the resource constraints such as space and maintenance time may change over time, while the materialized views are fixed once selected. Third, static view selection is usually unable to meet both the space bound and the maintenance bound at the same time. For example, if space is sufficiently large, we can materialize transient views, which can be dropped before the update window. Finally, in order to adapt to the evolution in reality, administrating efforts such as monitoring and reconfiguration cannot be avoided. Based on the above observations, another paradigm of view selection techniques, called dynamic view selection or view caching, is proposed to remedy such problems.
Caching can be roughly divided into two categories: physical caching and semantic caching. Physical caching refers to the mechanism employed in operating systems and traditional relational databases, where some physical storage unit such as a page or a tuple is kept in cache. Semantic caching takes advantage of high level knowledge about the data being cached, and keeps track of the semantic description of the cached data (Dar et al., 1996). In particular, caching of views or queries is semantic caching, since the cache manager knows both the data and their query expressions. Dar et al. (1996) focus on selection queries and the cache space is divided into disjoint semantic regions bounded by boolean constraints. Thus, a query $q$ with selection $Q$ can be decomposed into two parts: $Q \land C$ and $Q \land \neg C$, where $C$ is a boolean expression for the cache. The missing part is then computed from the base relations, while the in-cache part can be directly fetched. Chunk caching is a kind of semantic caching specific to chunk based organization (Zhao et al., 1997). Chunks have finer granularity than views or tables and are thus more flexible and may be more efficient in answering overlapping queries mainly involving aggregations (Deshpande and Naughton, 2000, Deshpande et al., 1998).

WATCHMAN (Scheuermann et al., 1996) is a cache manager targeted at OLAP. It employs a simple hit-or-miss strategy and relies on temporal locality of queries to gain benefits. That is, each query is taken as a unit for admission and replacement. No explicit modeling of interrelationship between queries and views are needed.

DynaMat is a more advanced system introduced in (Kotidis and Roussopoulos, 1999, 2001), which considers queries using group-bys and equality based selections. Each query is represented by a hyperplane in the hypercube of multidimensional data. A cover-or-miss strategy is employed, that is, a cached view can benefit any query computable from it, which is reduced to coverage of hyperplanes. Due to the restricted query type, an R-tree like view index can be built over the cached views to search in the cache to answer queries. Though DynaMat is more flexible than WATCHMAN, it does not allow combinations of cached views to answer queries. For instance, one query may be covered by the union of two cached views. Deshpande et al. (1998) propose using fine granularity of cache units, namely chunks, which are organized in a hierarchy of aggregation levels. A multidimensional query is then decomposed to chunks at the same aggregation level, with missing chunks computed from raw data. The work is further extended by Deshpande and Naughton (2000) to allow aggregation from lower level cached chunks. Due to decomposability, chunk caching is proved to be useful in distributed caching (Kalnis et al., 2002). In contrast to DynaMat, chunk caching searches for chunks covered by the query instead of making use of a materialized view covering the query.

Dynamic view selection techniques enjoy the benefit that no candidate views need to be specified in advance because the candidates are just history queries or their fragments. All the techniques restrict the query language to a relatively simple class so that the interrelationship modeling between candidate views or between views and queries is usually analogous to some geometric relationship that is efficient to reason about. In principle, the relationship between views and queries can be handled using the techniques of answering queries using views (Srivastava et al., 1996, Halevy 2001).

Another most significant issue in view caching is the admission and replacement control, i.e., how to decide which view is admitted and which view is replaced. If space allows, caching data is in general beneficial. Hence, when a new view is considered for admission and there is enough free space for the view, it is always admitted. The situation is more complicated, if the free space is not sufficient for the new view. The caching policy is usually determined by two factors: a benefit metric
and a recency metric. The benefit metric represents an estimation of the benefits of caching a query. Recency of the queries is also important because of the temporal locality.

Whenever there is not enough free space for caching a new query, a candidate replacement subset of cached queries is chosen and the benefit of the candidate subset is compared to the benefit of the new query. If the new query is more beneficial then the candidate subset is replaced by the new query, otherwise admission is refused. Scheuermann et al. (1996) adopt the cost save ratio (CSR), which measures the percentage of total query costs saved due to hits in the cache. Because of the chunking organization in Deshpande et al. (1998), a simple metric of coverage of base tables is adopted for measuring benefits of cached chunks.

Recency metrics of caching are well studied in the literature. One well known strategy is Least Recently Used (LRU), which discards the oldest data cached. The strategy is extended to LRU-K by O’Neil et al. (1993) to take advantage of recent access patterns. Deshpande et al. (1998) utilize the CLOCK algorithm (Silberschatz et al. 2002), an efficient approximation of LRU.

There are two ways to use the benefit metric and the recency metric simultaneously. One way is to consider the recency and benefit in parallel: first use LRU to select a candidate set and then use benefit to decide on replacement (Scheuermann et al. 1996). The other way is to use an aging strategy to obtain benefit in the recent time window and then use the windowed benefit for both candidate set selection and replacement decision (Deshpande et al. 1998).

References


