Nr.: FIN-011-2010

Towards Selection of Optimal Storage Architecture for Relational Databases

Andreas Lübcke, Veit Köppen, and Gunter Saake

Workgroup Databases

Fakultät für Informatik
Otto-von-Guericke-Universität Magdeburg

Technical report

Nr.: FIN-011-2010

Towards Selection of Optimal Storage Architecture for Relational Databases

Andreas Lübcke, Veit Köppen, and Gunter Saake

Workgroup Databases

Technical report (Internet)
Elektronische Zeitschriftenreihe
der Fakultät für Informatik
der Otto-von-Guericke-Universität Magdeburg
ISSN 1869-5078

Fakultät für Informatik
Otto-von-Guericke-Universität Magdeburg

Impressum (§ 5 TMG)

Herausgeber:
Otto-von-Guericke-Universität Magdeburg
Fakultät für Informatik
Der Dekan

Verantwortlich für diese Ausgabe:
Otto-von-Guericke-Universität Magdeburg
Fakultät für Informatik
Andreas Lübcke
Postfach 4120
39016 Magdeburg
E-Mail: andreas.luebcke@ovgu.de

http://www.cs.uni-magdeburg.de/Technical_reports.html

Technical report (Internet)
ISSN 1869-5078

Redaktionsschluss: 17.12.2010

Bezug: Otto-von-Guericke-Universität Magdeburg
Fakultät für Informatik
Dekanat

Towards Selection of Optimal Storage Architecture for Relational Databases

Andreas Lübcke, Veit Köppen, and Gunter Saake
School of Computer Science,
Otto-von-Guericke-University Magdeburg, Germany
{andreas.luebcke,veit.koeppen,gunter.saake}@ovgu.de

Abstract: Requirements for database systems differ from small-scale stripped minimal database programs for embedded devices with minimal footprint to large-scale on-line analytical processing applications. For relational database management systems, two storage architectures have been introduced: a) row-oriented architecture and b) column-oriented architecture. In this paper, we present a query decomposition approach to evaluate database operations with respect to their performance according to the storage architecture. We map decomposed queries to workload patterns which contain aggregated database statistics. Further, we develop our decision models which advise the selection of the optimal storage architecture for a given application domain. We develop complementary decision models to select the storage architecture. The first decision model improves the performance of running systems (on-line). The second and third decision model advises an efficient database design or decides which architecture is more suitable for a given application domain (off-line).

1 Introduction

Database management systems (*DBMSs*) are pervasive in current applications. Consequently, practitioners aim at optimal performance by database tuning. But, administrating and optimizing DBMSs is a costly task [WKKS99]. Therefore, DBMS vendors and researchers develop self-tuning techniques to continuously and automatically tune DBMSs [IBM06, WHMZ94]. However, Chaudhuri et al. summarize the last ten years of self-tuning [CN07] and show that almost all approaches have been investigated for row-oriented DBMSs (*row stores*).

In recent years, various approaches came up to fulfill new requirements for database applications, e.g., column-oriented DBMSs to improve analysis performance (*column stores*) [Aba08, ABH09, Pla09, ZBNH05]. Column stores are well-suited for the on-line analytical processing (*OLAP*) domain whereas row stores are originally designed for the on-line transaction processing (*OLTP*) domain [ABC^{+}76]. However, the above mentioned requirements [SB08, VMRC04, ZAL08], e.g., real-time updates in DWHs, soften the borders between OLAP and OLTP, i.e., the decision process to find the suitable DBMS is more complex because we have to figure out which domain (OLTP or OLAP) is more costly within workloads. We show that query-based workload analyses as in [CN98] are not suitable to select the optimal storage architecture. Therefore, we define workload patterns that represent decomposed workloads to compare the performance of a certain operation for column and row stores. Based on our workload patterns, we present our decision model which advises the optimal storage architecture under certain evaluation constraints, e.g, minimal I/O costs.

2 Challenges for Physical Design Process

In this section, we illustrate the increased complexity of physical design process in the data warehouse (*DWH*) domain. Formerly, the only architecture for (large-scale) relational DWH was row store which provide similar algorithms. Row stores only had slight differences in functionality, thus they are comparable and the design process decision is only based on different optimization and implementation techniques. Nowadays, we have to decide between column or row stores in the DWH domain. Additionally, column stores are faster than row stores[1] for DWH workloads [AMH08, SAB$^+$05]. In contrast to row stores, column stores partition data column-wise, i.e., values of a column are stored one after the other. This vertical partitioning is advantageous for aggregations that frequently appear in DWH workloads and mostly process on single columns. But, column stores perform worse on tuple and update operations because after vertical partitioning, column stores have to reconstruct tuples for these operations. Compared to column stores, row stores perform better on tuple operations, thus we assume that there are still application fields for row and column stores in the DWH domain with respect to the above mentioned new requirements.

	Standard TPC-H	
Query #	**MySQL**	**ICE**
TPC-H Q15	00:00:08	00:00:01
TPC-H Q16	00:00:09	00:00:01
	Adjusted TPC-H	
Query #	**MySQL**	**ICE**
TPC-H Q15	00:00:08	00:00:02
TPC-H Q16	00:00:12	00:00:24

Table 1: Influence of operations to DBMS performance.

To select the optimal storage architecture for a use case, e.g., real-time DWH, we have to compare row and column architectures and estimate their performance for a given workload. We have to consider different optimization techniques, e.g., different index structures. Unfortunately, several optimization techniques cannot be used for both architectures, e.g., self-tuning and vector based operations. Self-tuning approaches [CN07] (indexes etc.) are well investigated for row stores but not for column stores. Column stores support a number of compression techniques and vector based operations [Aba08] which are not supported by row stores. Moreover, column stores themselves increase decision complexity because there is an amount of different approaches, e.g., column stores utilize either tuple-oriented or column-oriented query processors. In contrast, row stores process nearly standardized.

To show the complexity of decisions, we introduce our example based on the TPC-H

[1] http://www.tpc.org/tpch/results/tpch_perf_results.asp

benchmark [Tra10]. We use the DBMSs MySQL[2] 5.1.37 (row store) and Infobright ICE[3] 3.2.2 (column store) to show this complexity. Because of limited space, we only present an excerpt of our study. To demonstrate influences of a single operation to the query performance, we modify the number of returned attributes of the TPC-H queries `Q15` and `Q16`[4] (cf. Listing 1 and 2), i.e., the size of processed tuples is increased. We choose the naive approach to return all attributes (like `SELECT * FROM ...`). The results[5] (cf. Table 1) show that there is only a neglectable influence by our changes to `Q15`, i.e., the mutual performance of both DBMS is not affected. In contrast, the mutual performance of both DBMS alters for `Q16`. The differences are not obvious from the query structure or syntax (cf. Listing 1 and 2). We suggest that the change of projection differentially alters the size of intermediate and final result of both queries. Hence, the performance differences are caused by different number of involved columns[6]. In other words, modifications to a single operation have different impacts on different queries. The full study can be found in [Lüb10]. Hence, we state that a general decision regarding the storage architecture is not possible based on the query structure. We have to analyze single operations of a query, e.g., join operation or tuple selection, to select the optimal storage architecture for a given workload (at least a query).

```sql
CREATE VIEW revenue0
(supplier_no,total_revenue) AS
SELECT l_suppkey,SUM(
   l_extendedprice*(1-l_discount))
FROM lineitem
WHERE l_shipdate>=date '1993-05-01'
AND l_shipdate<date '1993-05-01'
   + interval '3' month
GROUP BY l_suppkey;

SELECT s_suppkey,s_name,s_address,
   s_phone,total_revenue
FROM supplier, revenue0
WHERE s_suppkey=supplier_no
AND total_revenue=(
   SELECT MAX(total_revenue)
   FROM revenue0)
ORDER BY s_suppkey;
DROP VIEW revenue0;
```

Listing 1: Tpc-h query Q15.

```sql
SELECT p_brand,p_type,p_size,COUNT(
   DISTINCT ps_suppkey) AS supplier_cnt
FROM partsupp,part
WHERE p_partkey=ps_partkey
AND p_brand<>'Brand#51'
AND p_type NOT LIKE 'SMALL PLATED%'
AND p_size IN (
   3, 12, 14, 45, 42, 21, 13, 37)
AND ps_suppkey NOT IN (
   SELECT s_suppkey
   FROM supplier
   WHERE s_comment LIKE
      '%Customer%Complaints%')
GROUP BY p_brand,p_type,p_size
ORDER BY supplier_cnt DESC,p_brand,
   p_type, p_size;
```

Listing 2: Tpc-h query Q16.

3 Workload Patterns

We have to analyze a given workload to select the optimal storage architecture, thus we have to analyze a workload with respect to pattern. Due to the influence of single oper-

[2]http://www.mysql.com/

[3]http://www.infobright.com/

[4]These two queries illustrate typical results from our study, thus they are representative.

[5]Query execution times are in format hours, minutes and seconds (hh:mm:ss).

[6]Projection does not change number of tuples but number of columns per tuple.

ations on the performance (e.g., tuple selection, average calculation (cf. Section 2)), we have to map operations of a workload (at least of a query) and their statistics to evaluable patterns. Therefore, we introduce a pattern framework which stores all necessary statistics for subsequent performance analyses (within a decision model). Figure 1 outlines an overview of our decision process regarding the storage architecture. In the sections, we outline the design of our pattern framework.

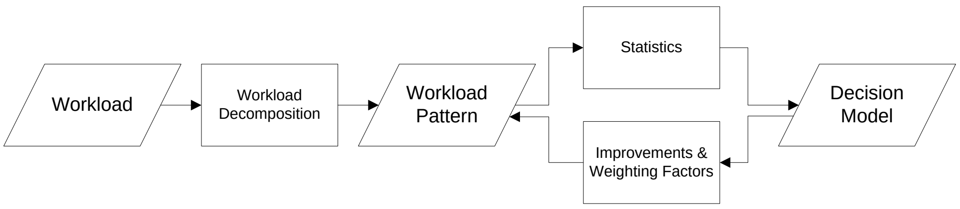

Figure 1: Workflow of our storage architecture decision process.

3.1 Pattern Detection

To analyze the influence of single operations (cf. Section 2), we identify three patterns concerning operations of queries in workloads. The three workload patterns are *tuple operations*, *aggregations and groupings*, and *join operations*. To characterize particular operations more precisely within the patterns, we define a number of sub-patterns for each of those three. In this way, we enable analyses based on the three patterns and additionally fine granular analyses based on sub-patterns, i.e., we can determine where the majority of costs emerge within a pattern.

First, the ***tuple operation pattern*** covers all operations that process or modify tuples, e.g., selection, sort and order operations. We select this pattern for performance analyses because row store process directly on tuples in contrast to column stores which need costly tuple reconstructions to process on tuples. Therefore, we identify the following sub-patterns:

Sort/order operation: Sort and order operations create certain sequences of tuples and affect all attribute values of a tuple. We assume that duplicate elimination is also a kind of sort operations, e.g., DISTINCT-statement, because an internal sort is necessary to find duplicates.

Access data and tuple reconstruction: Row stores always access tuples and column stores need tuple reconstruction to access more than one column.

Projection: Projection returns a subset of tuple attribute values and causes (normally) no additional costs.

Filtering: Tuple selection of tables or intermediate results based on a selection predicate, e.g., selection in WHERE-clause and HAVING-clause.

Second, we group all column processing operations in the ***aggregation and grouping pattern***, e.g., COUNT and MIN/MAX. We determine this pattern as counterpart to the tuple operation pattern. The operations of this pattern process only on single columns except for grouping operations which can also process several columns, e.g., GROUP BY or CUBE. Due to single column processing, column stores perform well on aggregations (cf. Section 2). For this pattern, we identified the following sub-patterns:

Min/Max operation: `MIN/MAX` operation provides the minimum/maximum value of a single attribute (column).

Sum operation: This operator computes the sum of all values according to one column.

Count operation: The `COUNT` operator counts the number of attribute values in a column as well as `COUNT(*)` counts only the number of key values, thus it process a single column.

Average computation: The average computation processes all values of a single column as well as the sum operation, but it can have different characteristics, e.g., mean (avg) or median.

Group by operation: This operation merges equal values according to a certain column and determines a subset of tuples. Grouping across a number of columns is also possible.

Cube operations: The Cube operator computes all feasible combination of aggregates for selected dimensions. This generation requires the power set of aggregating columns, i.e., n attributes are computed by 2^n `GROUP BY` clauses.

Standard deviation: The standard deviation (or variance) is a statistical measure for the variability of a data set and is computed by a two pass algorithm which means two cycles.

Third, we propose the***join pattern*** to contain all join operations of a workload. Join operations are basic but costly tasks for DBMS which significantly affect any relational DBMS. We determine this pattern to highlight the differences according to join techniques of column and row stores, e.g., processing directly on compressed columns or bitmaps. Within this pattern, we differentiate the different techniques against each other. Consequently, we identify the following sub-patterns:

Vector based: The column oriented architecture inherently supports vector based join techniques while row stores have to maintain/create costly structures, e.g., bitmap (join) indexes [Lüb08].

Non-vector based: This pattern covers classic join techniques to differentiate the performance between vector based and non-vector based join. As a result, we can estimate effects on the join behavior by architecture.

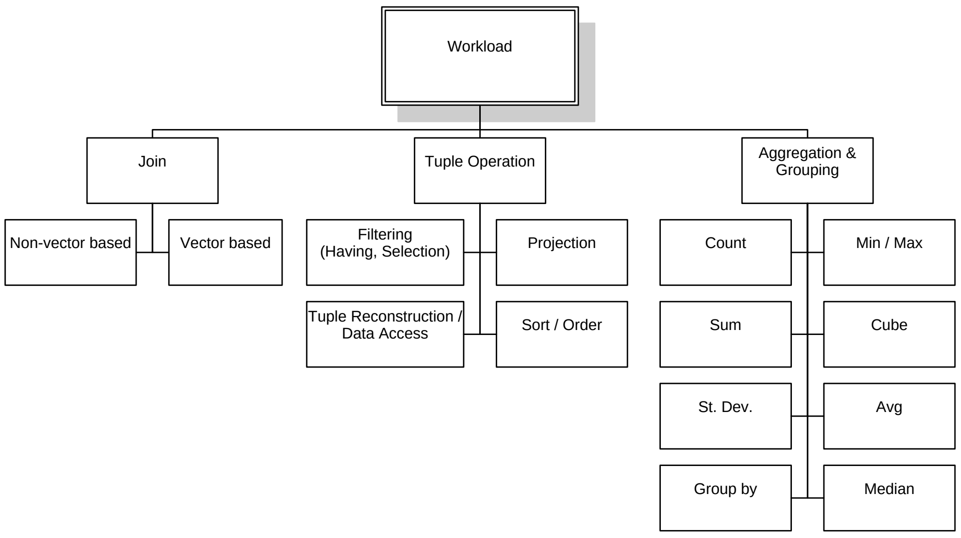

Figure 2: Workload patterns based on operations.

We identify these two sub-patterns because the join concepts, e.g., merge or nested loop join, are implemented for both architectures, thus, we do not have to distinguish between these join concepts. Hence, we assume that there is no necessity to map each join concept into its own sub-pattern. As a result, Figure 2 shows all introduced patterns and their relation to each other.

3.2 Dependencies between Patterns

There are dependencies between the following patterns: join, filtering, sort/order, group/ cube, and data access pattern.

First, join operations innately imply tuple selections (filtering pattern). However, the tuple selection itself is part of the join operation by definition, thus we assume that an additional decomposition of join operations is not necessary. Moreover, new techniques would have been implemented to further decompose join operations and gather the necessary statistics. Hence, the administrative cost for tuning will be noticeably increased. To a side-effect, the comparison of join techniques belonging to different architectures is no longer possible because of system-specific decomposition.

Second, we note that two different types of sort/order operation can occur, i.e., implicit and explicit sort. The explicit sort is evoked by workload or user (commands), thus we consider this operation in the sort/order pattern. In contrast, we do not consider the implicit sort operation in the sort/order pattern because this sort operation is caused by the optimizer, e.g., for sort-merge join or duplicate elimination. To sustain comparability, we assign all costs of grouping to the `GROUP BY` (or cube) pattern including the sort costs.

Third, tuple reconstruction is part of several operations (which return tuples) in case of column stores. As we already mentioned for the other operations, we add these costs to the tuple operation pattern, thus we separate operations costs and tuple reconstruction costs. Consequently, we sustain the comparability of operations beyond the architectures because row stores are not affected by tuple reconstructions.

4 Decision Model

We integrate our workload patterns and the statistics within them[7] in a decision model. This model allows us to compute recommendations about the optimal storage architecture based on our workload pattern approach. However, some assumptions for the decision model have to be made. In this paper, we derive three decision models at an abstract level and describe them in an abstract and implementation independent way, e.g., cost functions can modularly replaced or extended as well as (sub-) pattern can be added or refined. Hence, these decision models can be further refined, e.g., a more fine granular approach or in future research the search space can be extended by further architectures.

Before we construct a decision model, we have to determine which aspect has to be optimized with respect to given constraints. In the context of database tuning, constraints can be for instance minimum response time for each query, maximum throughput, or average query response time for the overall workload, an optimized load balance or user-specified requirements such as optimizing certain queries to a given time threshold. A weighted approach of all these properties is also imaginable and therefore, a methodology of ranking system alternatives has to pay attention on this as well. According to given constraints, an important point for ranking alternatives is the evaluation function. For the comparison of database storage architectures a cost function model can be used that measures user-specified as well as standard measures/statistics of database system performance like minimum, average, or maximum query execution time. In Fig. 3, we depict a selection of different cost function structures concerning query execution time without claiming generality.

Linearity in a cost function results from algorithms with linear complexity without further restrictions. This is also true for square root and quadratic functions. A staircase or stepwise linear function comprises response times where involved data has to be loaded and reallocated. Mostly, we assume that mixed or piecewise defined cost functions are common in practice. For small query sizes a quite different cost function behavior is appropriate than for larger query sizes. This is respected by limited exclusive resources like RAM. For this paper, we assume that the evaluation criteria are defined and according to the cost function a separation is plausible made. Therefore, we do not pay further attention on this detail.

In our decision model, we differentiate between on-line and off-line decisions. Whereas the information space in on-line decisions is assumed to be without uncertainty, i.e., the database system, its optimizer estimations, statistics, etc. are available. We develop our

[7]Directly from optimizer or estimated workload statistics.

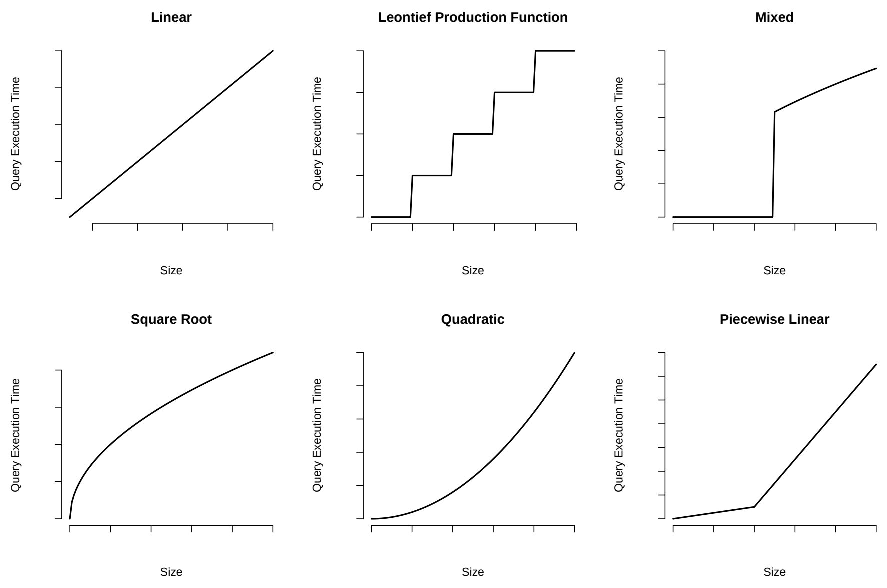

Figure 3: Different cost functions for the query Size and the corresponding execution time.

first decision model for this scenario in Section 4.1 and we call this *on-line decision model*. For the off-line decisions, uncertainty has to be taken into account. The uncertainty results from unknown workload and/or estimation of the corresponding cost. Therefore, we develop *two off-line decision models* that first, support the design/redraft of systems (cf. Section 4.2), and second the benchmarking of different systems (cf. Section 4.3). Due to the distinction between both architectures, we regard decision problems as a ranking of architectural designs. Challenges of rankings under uncertainty are addressed in [BK10] which we will not discuss in this paper.

We develop the *on-line decision model* based on linear programming in Section 4.1. We develop the *two off-line decision models* that include uncertainty of a workload in Section 4.2 and 4.3.

4.1 On-line Analysis with Statistics from DB

For an *on-line decision model*, we use statistics that are directly accessible through the DBMSs. This enables us to derive an optimization problem for the storage architecture decision. As discussed before, we have to decide what the cost criteria for optimization are. We note that an optimal storage architecture has to satisfy all or only some of optimization criteria. Another challenge is that there is not a single architecture outperforming the other for each database operation. Therefore, the overall optimum of all database tasks (user-specified or related to optimization criteria) has to be taken into account. We assume that statistics are available for both architectural designs (row and column stores). In the following, we define a (database) task as a part of a query, e.g., `ID 15` for TPC-H query `Q15` (cf. Section 2 and 5).

The architectural decision is built upon a linear program that is comparable to the assignment problem, cf. [Chv83, pp. 341]. We use this approach at a first glance to decide on a cost basis between both architectures, where only one performs all tasks. However, this model can be easily adapted by changing the constraints and therefore used for hybrid architectures in the future. Furthermore, by applying a sensitivity analysis we can identify tasks that are very different for the architectures. In the following formulae, column stores are abbreviated as CS and row stores as RS. The general form of our class of decision for one of both architecture types is given in Eq. 1.

$$min \sum_{i\in\{CS;RS\}} \sum_{j\in T} C(i,j)x_{ij} \quad \text{with the constraints:}$$

$$\begin{aligned}
I: \sum_{j\in T} x_{ij} &= \begin{cases} |T| \\ 0 \end{cases} && \forall i \in \{CS;RS\} \\
II: \sum_{i\in\{CS;RS\}} x_{ij} &= 1 \;\; \forall j \in T \\
x_{ij} &\in \{0,1\} \;\; \forall i \in \{CS;RS\}, \forall j \in T
\end{aligned} \tag{1}$$

The assignment x_{ij} is built up by the set of system tasks T and the storage architecture $\{CS;RS\}$. According to a task j and storage architecture i, cost function values $C(i,j)$ exist. Given an existing database system and a workload, we access and extract the statistics on operational costs as discussed in Section 3 and in [LKS10]. The first constraint (I in Eq. 1) ensures either all or none of the tasks are performed by an architecture. The second constraint (II in Eq. 1) guarantees that all tasks T are executed exactly ones.

We assume, that the cost functions and their corresponding query sizes are already partitioned in such a way that all values $C(i,j)$ describe the cost structure sufficiently. The costs $C(i,j)$ have to be selected according the optimization plan, e.g., time, size, or performance measures. The result of the linear program is the optimal storage architecture for a given workload. The usage of the above model for decision making on the optimal architecture assumes that the system does not change its query structure, which builds upon all database queries and their corresponding execution information. For this structure, we obtain the optimal storage architecture. Afterwards, a sensitivity analysis can be performed, where an evaluation of the restrictions and especially the cost structure is under a more fine granular examination.

In practice, the amount of queries and workload tasks is not useful or even unavailable. Therefore we use the workload pattern approach to restrict the number of tasks at an appropriate level and group similar database tasks together. Consequently, the linear program remains manageable for practical use.

Uncertainty according to query structure is not respected in our on-line decision model. Each query has already been performed and used for the derivation of the overall cost and query structure.

4.2 Off-line Design Prediction

Uncertainty of a multi-dimensional decision problem has to be considered when the query structure is not known but can be estimated. The multi-dimensionality appears due to different tasks within query plans.

In this section, we introduce our first *off-line decision model* on design prediction that incorporates future workload structure, associates the query structure to our workload patterns, and includes fraction on the overall workload. Our off-line decision model supports architectural decisions of a DBMS. We use probability theory in this section to represent future workload and changes in the DBMS behavior. The estimation of both aspects can be combined into a cost function. Therefore, the described assignment problem from Eq. 1 has to use an adapted cost function. $C(i, j)$ changes to:

$$\begin{aligned} C^*(i,j) &= p(i,j) \cdot C(i,j) \\ wrt. \sum_{j \in T_{WL}} p(i,j) &= 1 \;\; \forall i \in \{CS; RS\}, \end{aligned} \tag{2}$$

where $p(i, j)$ is the probability that task j is performed by architecture $i \in \{CS; RS\}$. Due to the high number of queries, a partition for $C(i, j)$ has to be done. We partition the tasks according to our workload structure T_{WL}, see Section 3. As a result, we use the task set $T_{WL} = \{Join,\ Tuple\ Operations,\ Aggregation\ \&\ Grouping\}$ where the elements are further refined, e.g., join consists of non-vector based and vector based joins. This restricts the information space inappropriately. Therefore, an estimation of average costs (for a certain level of the cost function) must be made, too. This results in a heuristic design of the decision model which is adaptable or can be iteratively improved. Note, that we use all available queries in our on-line decision model, in contrast, we use in this off-line model only adequate candidates from our workload structure.

This off-line decision model enables us to estimate the cost of unknown workloads and also reports us where cost information of one architecture is available with constraints or it has to be estimated. For quality of our decision model according to design predictions, we assume that the estimated cost structure is sufficient. Therefore, knowledge of domain experts is required. However, the partitioning enables a more sophisticated approach than guessed decisions. Consequently, our approach supports the development of sufficient design rules/heuristics. We obtain an optimal expectation value and cost plan under the given probabilities of the workload tasks. This can again be used for a sensitivity analysis where more restriction values are considered.

4.3 Off-line Benchmarking of Different Systems

Our second *off-line decision model* adopts and uses ideas of our approaches in Section 4.1 and 4.2. In contrast to the on-line approach, we cannot access the statistics through DBMS, thus we have to estimate the workloads, i.e., the statistics. Furthermore, we use a multi-criteria approach for evaluating the system requirements.

An important aspect in the context of multi-criteria decision analysis (MCDA) and uncertainty is the representation of the ranking function. Schneeweiss classifies MCDA methods according the preference function [BK10, Sch91]. For further applications of MCDA see [FGE05]. Due to the usage of cost, we can use multi-attribute utility theory (MAUT), cf. [vE86], where a utility function is available. MAUT is used in recommendation systems where the estimation of user interests is achieved. To decide between different architectural designs, a recommendation can be made for a multi-attribute scenario, if weighting between different dimensions is possible. In our case, the dimensions are defined by the workload differentiation depicted in Figure 2. The overall value function is given by:

$$\begin{aligned} value(ALT) &= \sum_{j=1}^{n} weight_j \cdot value_j(ALT) \quad \text{with} \\ \sum_{j=1}^{n} weight_j &= 1, \end{aligned} \tag{3}$$

where an alternative (ALT) of a set of possibilities (in our case CS and RS and different database systems) are evaluated by a function $value$ that takes the cost structure into account and weights all function values according to the workload structure. This enables us to differentiate between column store and row store architecture, and additionally, it enables us to benchmark different database systems.

Another advantage of MCDA respective MAUT is the derivation of user's workload preferences. Hence, we are able to develop a model of user preferences with respect to our workload hierarchy and to estimate the desired workload structure. Using the MAUT methodology, we obtain a recommendation or ranking between the set of alternatives. This helps us to perform a decision on this multi-dimensional problem. For future research, it also enables us to use this decision model to find an optimal design for databases.

5 Evaluation

In this section, we present our case study based on the TPC-H benchmark [Tra10] with scale factor 1 to evaluate our decision model approach. We focus on the first decision model, although the case study can be adapted to our both off-line models as well. The evaluation process would be equally, i.e., only the granularity of used statistics from our workload patterns differs for the both off-line models. Our three decision models support inter alia an efficient database design in the context of an optimal architecture (row or

	Oracle		
Workload Pattern	**Q6**	**Q15**	**Q16**
Data Access	*ID2*:155900	*ID8*:10000;*ID6*:218657	*ID9*:800000;*ID8*:30515;*ID6*:500
Non-vector		*ID1*:20000	*ID7*:830515:*ID5*:121371
Group By		*ID5*:218657	*ID4*:114828;*ID2*:114828
Sort		*ID7*:10000;*ID2*:10000	*ID1*:15000
Sum	*ID1*:155900		
Projection	*ID0*:1	*ID3*:10000;*ID0*:10000	*ID3*:114828;*ID0*:15000
	ICE		
Workload Pattern	**Q6**	**Q15**	**Q16**
Data Access	*ID2*:3*6.029.312	*ID6*:6029312;*ID4*:6029312*ID2*:65536	*ID6*:65536;*ID5*:849628;*ID4*:261424
Non-vector		*ID1*:131072(75536)	*ID3*:1111052
Group By		*ID5*:225954;*ID3*:225954	*ID2*:118274
Sort			*ID1*:18314
Sum	*ID1*:114160		
Projection	*ID0*:1	*ID0*:1	*ID0*:18314

Table 2: Accessed rows resp. number of values for a column for tpc-h queries.

column store). For reasons of clearness, we select only two systems, Infobright ICE 3.3.1 and Oracle 11gR2. Both systems were installed with standard parameters except that we restrict the available main memory to 250MB. Additionally, we restrict our workload to three queries from the TPC-H benchmark to save comprehensibility of our considerations. We select the above discussed queries `Q15` and `Q16` as well as query `Q6` to show different query types in our evaluation.

5.1 Gathered Statistics from Workload Patterns

In the following, we use statistics gathered from the corresponding optimizer as described in [LKS10]. The cost information are stored in our workload pattern (cf. Section 3). Nevertheless, we extracted the cost information to Table 2 and 3 to ensure readability. Furthermore, we use two cost measures because as we discussed before, one cost measure is not sufficient to select the optimal storage architecture. Additionally, we can show with these two measures that measures do not correlate across both architectures even the measures are interdependently as discussed in Section 2. Hence, we select the accessed number of rows and the I/O cost to compare Oracle and ICE. Note, ICE does not access single values of a column but so-called data packs which are compressed storage units. These data packs contain 65536 values, thus in Table 2, the values for ICE are multiple of the data pack size.

The operations of a query are identified by `IDs` from the optimizer which we also apply in our approach. We can read the processing order from these `IDs`, i.e., the highest `ID` of a query represent first operation and `ID0` the last operation in query execution plan. We can also read the assigned workload pattern from Table 2 and 3. Table 2 shows the number of accessed rows resp. number of accessed values of a column for the TPC-H queries `Q6`, `Q15`, and `Q16` in Oracle and ICE. Note, Oracle is a row store, thus Oracle access the entire tuples. In contrast, ICE only accesses the required columns and their values. In Table 2, we can see this effect for query `Q6`. Oracle access the required columns in one step,thus,

	Oracle		
Workload Pattern	**Q6**	**Q15**	**Q16**
Data Access	*ID2*:3118	*ID7*:86925;*ID6*:750	*ID9*:7200;*ID8*:1252;*ID6*:34
Non-vector		*ID5*:87675	*ID7*:8452;*ID5*:6078
Group By		*ID4*:94050;*ID2*:1310	*ID4*:13550;*ID2*:5627
Sort		*ID1*:1310	*ID1*:735
Sum	*ID1*:3118		
Projection	*ID0*:1	*ID3*:6647;*ID0*:1310	*ID3*:13550;*ID0*:735
	ICE		
Workload Pattern	**Q6**	**Q15**	**Q16**
Data Access	*ID2*:14471	*ID6*:4824;*ID4*:4824;*ID2*:2583	*ID6*:1320;*ID5*:682;*ID4*:2045
Non-vector		*ID1*:2763(2591)	*ID3*:889
Group By		*ID5*:181;*ID3*:181	*ID2*:1018
Sort			*ID1*:158
Sum	*ID1*:183		
Projection	*ID0*:1	*ID0*:1	*ID0*:158

Table 3: Accessed data of tpc-h queries in Kbytes.

it only scans the `LINEITEM` table one time. ICE scans the `LINEITEM` table three times because three different columns[8] have to be accessed in query `Q6`. Furthermore, ICE does not reconstruct tuples before join execution. In other words, ICE only reconstructs tuples to process final result, e.g., `GROUP BY`. Nevertheless, ICE access more values than Oracle in our test setup, e.g., in query `Q6` Oracle reads 2.494.400 values (155900 rows with 16 columns) and ICE reads 18.087.936 values.

Table 3 shows our measurements of the I/O costs, our second cost criterion, in Kbytes. Note, the compression ratio of ICE is approximately 5:1 in our test setup, i.e., the 1GB TPC-H data set in ICE is 182MB. For convenience, we do not examine compression ratios for each column. The results show that the number of read values does not directly correlate with the access data in Kbytes. For example, ICE access 12.124.160 values for query `Q15` in contrast to 1.690.599 values accessed by Oracle. Considering the accessed data in Kbytes, the ratio between ICE and Oracle changes. Oracle reads 87.675Kbytes while ICE only reads 12.231Kbytes. Nevertheless, ICE reads less data (in Kbytes) than Oracle even if we subtract out the compression ratio.

Our results show that physical design based on basic heuristics is not sufficient for complex workloads thus; we need decision support to select the optimal storage architecture.

5.2 Linear Program

The above given query decomposition information on rows, cf. Table 2, and I/O cost, cf. Table 3, can be aggregated for each DBMS and workload pattern. In this case study, we assume that all three selected queries (`Q6`, `Q15`, and `Q16`) are executed in the same ratio. Otherwise, we had to adjust the cost structure by a weighting function with respect to the query frequency.

[8] `L_SHIPDATE`, `L_DISCOUNT`, and `L_QUANTITY`

	Oracle		ICE	
Workload Pattern	**Rows**	**I/O cost**	**Rows**	**I/O cost**
Data Access	*1215572*	*99279*	*31388684*	*30749*
Non-vector	*971886*	*102205*	*1242124*	*3652*
Group By	*114537*	*448313*	*570182*	*1380*
Sort	*35000*	*2045*	*18314*	*158*
Sum	*155900*	*3118*	*183*	*114160*
Projection	*149829*	*22243*	*18316*	*160*

Table 4: Summary of accessed data in Oracle and ICE for tpc-h Q6, Q15, and Q16.

Taken the data from Table 4 the corresponding model can be build from Eq. 1. As linear programming language we use AMPL formulation [FGK02]. Due to the fact, that we are interested in the selected or optimal assigned database management system to a specific cost structure the problem is a mixed integer problem. In Listing 3 we present the AMPL source code of the model.

According to the object of optimization a selection of the cost values has to be done. In our example, two possible cost information are available. Either an optimization on the accessed rows or the I/O cost can be performed. In practice, this is not always an opportunity, often all cost influence structures have to be addressed. With the help of your model it is possible to adapt the cost function, as for instances depicted in Eq. 2 where uncertainty is introduced. In our case study we present two different optimization targets. In the first case, the row access should be minimized. In our second example we want to minimize the I/O cost. The complete workload is defined by `Q6`, `Q15`, and `Q16` from the TPC-H benchmark. Note, this benchmark is designed for OLAP queries. Therefore, we assume that ICE outperforms the row store based Oracle DBMS. Furthermore, our example is clearly arranged. In practice, the workload decomposition is much more complex and additional DBMSs might be ranked. Without loss of generality, we present our decision model on ICE (column store architecture) and Oracle.

Using the row access data from Table 4 in our AMPL model, see Listing 3, results in the assignment of the Oracle DBMS. Note, that this can be easily seen by comparing the overall sum of row access per DBMS. As we assumed, the I/O cost are much lower in ICE and therefore our model delivers ICE as the optimal solution for I/O cost. However, using our model enables us to run a sensitivity analysis, which identifies important cost drivers. Furthermore, it is possible to add easily more workload information, which increases complexity for the decision makers. This complexity increases also by introducing more DBMSs.

6 Related Work

In recent years, several column stores have been proposed [Aba08, LLR06, SWES08, ZBNH05]. There are well-known column stores but all systems are pure column stores

```
set DBMS;                        # set of DBMSs for ranking
set WorkloadPattern;             # set of Workload Patterns

param cost{i in DBMS, j in WorkloadPattern};   # cost
var assign{i in DBMS, j in WorkloadPattern}
      binary;                    # = 1 if DBMS i is used, 0 otherwise
var use {i in DBMS} binary;   # assignment that exactly one DBMS is used

minimize Cost:
  sum{i in DBMS, j in WorkloadPattern} cost[i,j]*assign[i,j];

subject to USAGE: sum{i in DBMS} use[i] = 1;
                                 # restriction that exactly one DBMS is in use
subject to Multi_Architecture {i in DBMS}:
  sum {j in WorkloadPattern} assign[i,j] = 6 * use[i];
                                 # this DBMS has to do all 6 Workload Pattern Tasks

subject to Tasks{j in WorkloadPattern}:
   sum{i in DBMS} assign[i,j] = 1;  # restriction that all tasks are performed
```

Listing 3: AMPL model for on-line decision.

and do not support any row store functionality. Abadi et al. [AMH08] compare row and column stores performance on the star schema benchmark. They simulate the column store architecture by indexing every single column or vertical partitioning of the schema. They show that using column store architecture in a row store is possible but the performance in a row store is poor. In this paper, we do not directly compare optimization techniques of DBMSs. Instead, we consider strengths and weaknesses of both architectures to recommend one of both architectures for a given workload. We do not discuss earlier approaches like DSM [CK85], hybrid NSM/DSM schemes [CY90], or PAX [ADHS01] because the differences to state of the art column stores have been already discussed, e.g., Harizopoulus et al. [HLAM06].

With respect to the solutions for architectural problems, there are systems available which attempt to fill the gap (between a column and a row store). They apply very interesting approaches for the development of a hybrid system. C-Store [Aba08] uses two different storages to overcome the update problems of column stores. A related approach brings together a column store approach and the typical row store domain of on-line transactional processing (OLTP) data [SBKZ08]. In contrast to our work, they focus on near real-time processing of OLTP data in a DWH and the necessary ETL components. They hold replicates of all OLTP data which is needed for reporting mechanism (OLAP/DWH). These approaches aim at another direction because we want to recommend the optimal architecture for a certain application with a decision model which is not supported by these approaches.

A number of design advisors exist which are related to our work, e.g., IBM DB2 Configuration Advisor [KLS$^+$03]. The IBM Configuration Advisor advises pre-configurations for databases which is similar to our off-line decision. Zilio et al. [ZRL$^+$04, ZZL$^+$04] introduce an approach which is related to our on-line decision model. They gather statistics directly from DBMS and utilize them to advise index and materialized view configurations. Two similar approaches are presented by Chaudhuri et al. [BC06, BC07] which illustrate the whole tuning process using constraints such as space threshold. However, these approaches operate on single systems instead of comparing two or more systems according to their architecture. Additionally, our approach aims at architectural decisions as opposed to the mentioned approaches which tune configurations, indexes, etc.

Ingres/Vectorwise applies the Vectorwise (formerly MonetDB/X100) architecture into the Ingres product family [Ing09]. In cooperation with Vectorwise, Ingres is developing a new storage manager ColumnBM for the new Ingres/Vectorwise. However, the integration of the new architecture into the existing environment remains unclear [Ing09].

To develop our decision model, we need to analyze workloads and derive workload patterns therefrom. For that, we can utilize, adapt, and extend existing approaches such as Turbyfill [Tur88] which consider mixed database workloads concerning disk access. We can adopt these ideas to our approach that map the workload behavior of different architectures. By contrast, the approach of Raatikainen [Raa93] considers workload classes and how to find them based on cluster analysis. We rather aim at classification of operations according to predefined patterns instead of clustering workloads into a number of classes like Raatikainen. We can use these and other approaches to find and evaluate our workload patterns. The approaches of Favre et al. [FBB07] and Holze et al. [HGR09] focus on self-tuning databases. Favre et al. consider the evolutionary changes within a DWH workload. They also consider the interactions between schema evolution and workload evolution. This approach should be considered with respect to the development of a hybrid system. The related approach of Holze et al. examine clustering of workloads based on distance functions but is developed for lightweight and stream-based operations.

7 Conclusion

In recent years, column stores showed good results for DWH applications, thus column stores (mostly) outperform established row stores. But, new requirements arise in the DHW domain that cannot only be satisfied by column store. New requirements in the DWH domain demand also for row store functionality, e.g., sufficient update processing. Thereby, the complexity of design processes increased because we have to choose the optimal architecture for given applications. We introduced a decision approach based on workload patterns to overcome the increased complexity of the design process concerning different storage architectures. The workload patterns contain all workload information, e.g., statistics and operation cost. Furthermore, we presented a workload decomposition approach based on operations which maps operations of a given workload to our workload patterns. We utilize this structure to estimate an optimal architecture for a given workload.

To select the optimal storage architecture, we introduced three decision models. Our models are similar to existing tuning and self-tuning approaches with respect to the decision making process. Our models are based on cost functions (tuning objective) and constraints, too but are developed on an abstract level, i.e., we can modularly refine or extend our models. The first model covers on-line prediction for existing systems which uses statistics of the existing systems. Hence, we do not consider uncertainty in our prediction for this model. It enables us to compute the optimal storage architecture in a running system. The second and third model provide off-line prediction and process with partially estimated values, thus we can deal with uncertainty in our prediction. The derived workload pattern is the basis for both models. These two models allow performing predictions for unknown and/or future workloads. The first decision model recommends on the predicted optimal storage architecture and requires cost information of the competing architectures within a database system. Our third developed decision model is applicable to different database systems and storage architectures. We use user preferences for evaluating the uncertainty and come up in our third decision model with a ranking that takes different architectures and database systems into account.

In future work, we will consider two strategies to implement our workload patterns in a prototype. First, we implement a separate DBMS to export periodically statistics and operation costs which are mapped into the workload patterns. In this way, we do not affect performance of analyzed systems by prediction computation in our on-line model. Additionally, we are able to store and estimated values in off-line prediction because we do not have a database and/or statistics for future workloads. Second, we adapt existing approaches [BC06, LGB09] to automatically gather statistics, e.g., mapping statistics and workload patterns directly into a graph structure (query graph model). To evaluate our decision models, we will perform detailed studies. These studies will be performed on existing systems to obtain expressive values for predictions. Finally, we will use our decision models to support the development of a hybrid architecture.

Acknowledgment

Veit Köppen is funded by the German Ministry of Education and Science (BMBF), project 01IM08003C. The presented work is part of the ViERforES[9] project. We thank Martin Kuhlemann and Ingolf Geist for helpful discussions and comments on earlier drafts of this paper.

References

[Aba08] Daniel J. Abadi. *Query execution in column-oriented database systems*. PhD thesis, Cambridge, MA, USA, 2008. Adviser: Madden, Samuel.

[ABC$^+$76] Morton M. Astrahan, Mike W. Blasgen, Donald D. Chamberlin, Kapali P. Eswaran, Jim Gray, Patricia P. Griffiths, W. Frank King III, Raymond A. Lorie, Paul R. McJones, James W. Mehl, Gianfranco R. Putzolu, Irving L. Traiger, Bradford W. Wade, and Vera Watson. System R: Relational approach to database management. *ACM TODS*, 1(2):97–137, 1976.

[ABH09] Daniel J. Abadi, Peter A. Boncz, and Stavros Harizopoulos. Column oriented database systems. *PVLDB*, 2(2):1664–1665, 2009.

[9]http://vierfores.de

[ADHS01] Anastassia Ailamaki, David J. DeWitt, Mark D. Hill, and Marios Skounakis. Weaving relations for cache performance. In *VLDB '01*, pages 169–180, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[AMH08] Daniel J. Abadi, Samuel R. Madden, and Nabil Hachem. Column-stores vs. row-stores: How different are they really? In *SIGMOD '08*, pages 967–980, New York, NY, USA, 2008. ACM.

[BC06] Nicolas Bruno and Surajit Chaudhuri. To tune or not to tune? A lightweight physical design alerter. In *VLDB '06*, pages 499–510. VLDB Endowment, 2006.

[BC07] Nicolas Bruno and Surajit Chaudhuri. An online approach to physical design tuning. In *ICDE '07*, pages 826–835, 2007.

[BK10] Bettina Berendt and Veit Köppen. Improving ranking by respecting the multidimensionality and uncertainty of user rreferences. In G. Armano, M. de Gemmis, G. Semeraro, and E. Vargiu, editors, *Intelligent Information Access*, Studies in Computational Intelligence. Springer, Berlin, 2010.

[Chv83] Vasek Chvatal. *Linear programming*. W. H. Freeman and Company, September 1983.

[CK85] George P. Copeland and Setrag N. Khoshafian. A decomposition storage model. In *SIGMOD '85*, pages 268–279, New York, NY, USA, 1985. ACM.

[CN98] Surajit Chaudhuri and Vivek Narasayya. AutoAdmin "what-if" index analysis utility. In *SIGMOD '98*, pages 367–378, New York, NY, USA, 1998. ACM.

[CN07] Surajit Chaudhuri and Vivek Narasayya. Self-tuning database systems: A decade of progress. In *VLDB '07*, pages 3–14. VLDB Endowment, 2007.

[CY90] Douglas W. Cornell and Philip S. Yu. An effective approach to vertical partitioning for physical design of relational databases. *IEEE Trans. Softw. Eng.*, 16(2):248–258, 1990.

[FBB07] Cécile Favre, Fadila Bentayeb, and Omar Boussaid. Evolution of data Warehouses' optimization: A workload perspective. In *DaWaK '07*, pages 13–22, 2007.

[FGE05] José Figueira, Salvatore Greco, and Matthias Ehrgott, editors. *Multiple criteria decision analysis: State of the art surveys*. Springer Science and Business Media, Boston, 2005.

[FGK02] Robert Fourer, David M. Gay, and Brian W. Kernighan. *AMPL: A modeling language for mathematical programming*. Duxbury Press, 2 edition, 2002.

[HGR09] Marc Holze, Claas Gaidies, and Norbert Ritter. Consistent on-line classification of DBS workload events. In *CIKM '09*, pages 1641–1644, 2009.

[HLAM06] Stavros Harizopoulos, Velen Liang, Daniel J. Abadi, and Samuel Madden. Performance tradeoffs in read-optimized databases. In *VLDB '06*, pages 487–498. VLDB Endowment, 2006.

[IBM06] IBM. An architectural blueprint for autonomic computing. White Paper, June 2006. Fourth Edition, IBM Corporation.

[Ing09] Ingres/Vectorwise. Ingres/VectorWise sneak preview on the Intel Xeon Processor 5500 series-based platform. White Paper, September 2009.

[KLS+03] Eva Kwan, Sam Lightstone, K. Bernhard Schiefer, Adam J. Storm, and Leanne Wu. Automatic database configuration for DB2 Universal Database: Compressing years of performance expertise into seconds of execution. In *BTW '03*, pages 620–629, 2003.

[LGB09] Andreas Lübcke, Ingolf Geist, and Ronny Bubke. Dynamic construction and administration of the workload graph for materialized views selection. *Int. Journal of Information Studies*, 1(3):172–181, 2009.

[LKS10] Andreas Lübcke, Veit Köppen, and Gunter Saake. A query decomposition approach for relational DBMS using different storage architectures. Technical report, School of Computer Science, University Magdeburg, 2010. Technical Report, Submitted for Publication.

[LLR06] Thomas Legler, Wolfgang Lehner, and Andrew Ross. Data mining with the SAP NetWeaver BI Accelerator. In *VLDB '06*, pages 1059–1068. VLDB Endowment, 2006.

[Lüb08] Andreas Lübcke. Cost-effective usage of bitmap-indexes in DS-Systems. In *20th Workshop "Grundlagen von Datenbanken"*, pages 96–100. School of Information Technology, International University in Germany, 2008.

[Lüb10] Andreas Lübcke. Challenges in workload analyses for column and row stores. In *22nd Workshop "Grundlagen von Datenbanken"*, volume 581. CEUR-WS.org, 2010.

[Pla09] Hasso Plattner. A common database approach for OLTP and OLAP using an in-memory column database. In *SIGMOD '09*, pages 1–2, New York, NY, USA, 2009. ACM.

[Raa93] Kimmo E. E. Raatikainen. Cluster analysis and workload classification. *ACM PER*, 20(4):24–30, 1993.

[SAB+05] Michael Stonebraker, Daniel J. Abadi, Adam Batkin, Xuedong Chen, Mitch Cherniack, Miguel Ferreira, Edmond Lau, Amerson Lin, Samuel Madden, Elizabeth J. O'Neil, Patrick E. O'Neil, Alex Rasin, Nga Tran, and Stanley B. Zdonik. C-Store: A column-oriented DBMS. In *VLDB '05*, pages 553–564. VLDB Endowment, 2005.

[SB08] Ricardo Jorge Santos and Jorge Bernardino. Real-time data warehouse loading methodology. In *IDEAS '08*, pages 49–58, New York, NY, USA, 2008. ACM.

[SBKZ08] Jan Schaffner, Anja Bog, Jens Krüger, and Alexander Zeier. A hybrid row-column OLTP database architecture for operational reporting. In *BIRTE '08*, 2008.

[Sch91] Christoph Schneeweiß. *Planung 1, Systemanalytische und entscheidungstheoretische Grundlagen*. Springer, Berlin, 1991.

[SWES08] Dominik Ślęzak, Jakub Wróblewski, Victoria Eastwood, and Piotr Synak. Brighthouse: An analytic data warehouse for ad-hoc queries. *PVLDB*, 1(2):1337–1345, 2008.

[Tra10] Transaction Processing Performance Council. TPC BENCHMARK^{TM} H. White Paper, April 2010. Decision Support Standard Specification,Revision 2.11.0.

[Tur88] Carolyn Turbyfill. Disk performance and access patterns for mixed database workloads. *IEEE Data Eng. Bulletin*, 11(1):48–54, 1988.

[vE86] Detlef von Winterfeldt and Ward Edwards. *Decision analysis and behavior research*. Cambridge University Press, 1986.

[VMRC04] Alejandro A. Vaisman, Alberto O. Mendelzon, Walter Ruaro, and Sergio G. Cymerman. Supporting dimension updates in an OLAP server. *Information Systems*, 29(2):165–185, 2004.

[WHMZ94] G. Weikum, C. Hasse, A. Moenkeberg, and P. Zabback. The COMFORT automatic tuning project, invited project review. *Information Systems*, 19(5):381–432, 1994.

[WKKS99] Gerhard Weikum, Arnd Christian Knig, Achim Kraiss, and Markus Sinnwell. Towards self-tuning memory management for data servers. *IEEE Data Eng. Bulletin*, 22:3–11, 1999.

[ZAL08] Youchan Zhu, Lei An, and Shuangxi Liu. Data updating and query in real-time data warehouse system. In *CSSE '08*, pages 1295–1297, Washington, DC, USA, 2008. IEEE Computer Society.

[ZBNH05] M. Zukowski, P. A. Boncz, N. Nes, and S. Heman. MonetDB/X100 - A DBMS in the CPU cache. *IEEE Data Eng. Bulletin*, 28(2):17–22, June 2005.

[ZRL$^+$04] Daniel C. Zilio, Jun Rao, Sam Lightstone, Guy M. Lohman, Adam J. Storm, Christian Garcia-Arellano, and Scott Fadden. DB2 Design Advisor: Integrated automatic physical database design. In *VLDB '04*, pages 1087–1097. VLDB Endowment, 2004.

[ZZL$^+$04] Daniel C. Zilio, Calisto Zuzarte, Sam Lightstone, Wenbin Ma, Guy M. Lohman, Roberta Cochrane, Hamid Pirahesh, Latha S. Colby, Jarek Gryz, Eric Alton, Dongming Liang, and Gary Valentin. Recommending materialized views and indexes with IBM DB2 Design Advisor. In *ICAC '04*, pages 180–188, 2004.