A Framework for Optimal Selection of a Storage Architecture in RDBMS

Andreas Lübcke and Gunter Saake

School of Computer Science,
Otto-von-Guericke-University Magdeburg, Germany
{andreas.luebcke,gunter.saake}@ovgu.de

Abstract. Self-tuning techniques are widely used in the database community especially for large analyses systems. Within this domain, a new architecture known as column store comes up to analyse and aggregate data. Column stores demonstrate good results in real world examples and benchmarks like TPC-H. In recent years, row stores dominate the data warehousing domain. To the best of our knowledge, there is no advisor for the optimal selection of storage architecture. We discuss this fact with respect to our example based on TPC-H. Afterwards, we present our research steps to develop a decision model for optimal selection of storage architecture. Finally, we motivate and discuss the development of a hybrid architecture.

1 Introduction

Database management systems (DBMSs) are pervasive in current applications. With database tuning, practitioners aim at optimal performance which is a primary objective for database applications. The administration and optimization of DBMSs is costly. Researchers and DBMS vendors are developing *self-tuning* techniques such that DBMSs continuously improve themselves [1, 2]. Chaudhuri et al. show in their summary of ten years of self-tuning [3] that almost all techniques have been investigated for *row-oriented* DBMSs (*row stores*).

In recent years, different approaches have come up to fulfil new requirements for applications, e.g., *column-oriented* DBMSs (*column stores*) [4–6]. Abadi et al. define "Column-stores, in a nutshell, store each database table column separately, with attribute values belonging to the same column stored contiguously, compressed, and densely packed, as opposed to traditional database systems that store entire records (rows) one after the other" [7]. However, column stores have been neglected so far for self-tuning techniques. In this paper, we show that self-tuning techniques are beneficial for column stores as well.

We motivate our research with a small example and discuss the results according to our goals. Afterwards, we discuss the differences among the two storage architectures and show first ideas to develop a decision model for selection of an optimal storage architecture. Therefore, such a decision model should map different application fields to the best supporting storage architectures. Such a decision model is applicable to all relational DBMSs but our following descriptions are mainly based on the data warehousing (DWH) domain. To the best of our knowledge, there is no decision model to advise one

of both architectures for a given application domain. Based on our insights, we argue that there is a high benefit for combining both architectures to a hybrid system. Such a system could use the advantages of both approaches in certain parts of application. We finally note how self-tuning techniques for row stores can be used in hybrid systems and how they can be adapted for column stores parts within the system.

2 Motivating Example

In this section, we present the results of a motivating example. In our example, we decide to use Infobright ICE[1] 3.2.2 and MySQL[2] 5.1.37. MySQL represents the row stores and ICE represents the column stores. At first view, the two DBMSs are as similar as these two because both systems are based on the MySQL kernel/management services except that they use different storage architectures. Due to their affinity, we selected these two DBMSs to compare the architectures instead of different functionality. Afterwards, we discuss the results of our example.

2.1 The TPC-H Benchmark

Our test environment is an Ubuntu 9.10 64bit system running on a Samsung X65 with a 2.2GHz dual core processor, 2GB RAM and 2GB swap partition. Furthermore, both DBMS configurations are adjusted to MySQL standard configuration and we use the standardized benchmark TPC-H (2.8.0) with scale factor 1 (1GB), i.e., both DBMSs use considerably less than 1GB of RAM (e.g., key buffer 16MB). No indexes or views are create to our environment expect indexes and views that are caused by DDL (primary key) or the workload itself. We run a series of tests with the TPC-H benchmark that represents DWH applications on our machine.

Our results are presented in Table 1. The query execution times are in the format hours, minutes and seconds (hh:mm:ss). We note that we are not able to run TPC-H Query 13 with MySQL syntax. We aborted Query 18 running on MySQL after more than 21 hours. All executions of Query 18 (MySQL) led to the same result. Some queries perform similarly on both architectures like Query 10 and Query 12. In contrast, Query 4 performs 20 times faster on MySQL (row store) than ICE (column store) as well as ICE executes some queries much faster than MySQL, e.g., Query 9 and Query 19. Our results show that neither of both architectures can outperform the other for every query.

Due to the significantly different results, we have to analyse several queries in detail, e.g., Query 21 for ICE or Query 18 for MySQL. If we exclude Query 18 and 21, we receive more expressive values, i.e., 58 minutes 33 seconds for ICE and 13 minutes 06 seconds for MySQL. ICE does not outperform MySQL as one could expect.

Threats to Validity. In this study, we made some assumptions. We have chosen MySQL and ICE as representatives because both systems are based on MySQL management services. There are many other column and row stores which we do not compare in this study, i.e., our study does not make claims of being complete.

[1] http://www.infobright.org
[2] http://www.mysql.org

2.2 Discussion

One can argue that our results arise from different functionality and query optimization techniques. We mention, ICE uses query optimization techniques which MySQL does not support and ICE optimizer poorly supports nested queries at the moment. We assume that this behaviour is intensified by necessary tuple operations for certain queries, e.g., Query 21 or 22. ICE has to process the entire tuples for the (not) exist clauses in Query 21 and 22. Additionally, ICE is already designed as read-only DWH system whereas MySQL is designed for transactional processing (OLTP). Nevertheless, we note that this functional gap appears to each system comparison concerning different architectures.

In our example, ICE outperforms MySQL on queries that contain a high number of aggregation and grouping operations, e.g., Query 1 and Query 3. The tuples processed for these queries contain only few columns in the (intermediate) result sets, i.e., a small number of tuple reconstructions is necessary. For (intermediate) result sets with a higher number of columns, ICE loses its advantages, e.g., Query 10 or in the worst case Query 21/22 because a high number of tuple reconstructions increase significantly the costs of query execution.

Query no.	ICE	MySQL	Query no.	ICE	MySQL
1	00:00:25	00:00:54	12	00:00:03	00:00:04
2	00:00:44	00:02:42	13	-	-
3	00:00:02	00:00:46	14	00:00:01	00:00:37
4	00:02:33	00:00:07	15	00:00:04	00:00:17
5	00:00:03	00:01:28	16	00:00:00	00:00:10
6	00:00:00	00:00:04	17	00:24:12	00:00:01
7	00:00:03	00:00:31	18	00:00:08	>21:07:44
8	00:00:02	00:00:04	19	00:00:03	00:02:31
9	00:00:06	00:01:11	20	00:10:51	00:02:24
10	00:00:08	00:00:11	21	06:00:27	00:02:48
11	00:00:00	00:00:01	22	00:19:13	00:00:03
carryover	00:04:06	00:07:59	overall	06:59:08	>21:24:38

Table 1. TPC-H results for MySQL and ICE

Our example shows the need for new approaches in the DWH domain considering the new requirements [5, 6, 8–11] in this domain. There are queries which perform better on the row store and queries which perform better on column store architecture. The results verify our assumption that different workload parts perform better on one of the architectures. Consequently, different applications and their needs have to be evaluated to use the optimal architecture. The optimal architecture can only be figured out if we have a decision support for architecture selection. Therefore, we recommend the development of a general decision model for row- and column-oriented architectures. We assume that a prototype supporting both architectures independently leads to a more general decision model.

According to the architecture, a decision model will support design and redesign of databases, e.g., DWH, and improves the quality of design decisions.

2.3 A hybrid storage architecture

We argue that different applications, in our case the DWH domain, have common needs and we can decide between both architectures based on a decision model. However, we will have performance losses caused by the architecture because there are probably also suitable patterns for the converse architecture. Moreover, the storage architecture of a system has to be changed if the workload changes extremely. This involves a complete architectural redesign and is the worst case on operational systems. So, the decision

model cannot only be utilized to support design decisions but also to support development of new architectures, e.g., a hybrid architecture.

We see two strong intercessory aspects to develop a hybrid architecture and/or prototype. First, a hybrid system should be more suitable for applications which have mixed workload patterns, thus both storage architectures are required. The performance losses can be reduced compared with the other two architectures by a mixed workload scenario (tuple operations and aggregations). Second, the redesign of a system with one of both architectures can be prevented by a hybrid architecture because a hybrid architecture contains both. The hybrid architecture should reduce the performance losses caused by changing workload in contrast to the other both architectures. Additionally, it is conceivable that a hybrid architecture will be able to adapt constantly to a changing workload with some extensions. The costs of dynamical adaption have to be limited by thresholds in a cost function. These thresholds prevent the growth of the adaption costs beyond the benefit of adaption. Hence, self-tuning in an architectural manner will be possible.

We have to consider several aspects to develop a hybrid system, e.g., compression techniques, query processing. Some aspects are mutually exclusive to each other like different query processing techniques. Nevertheless, some conflicts can be solved by adapting current techniques. These aspects, e.g., cooperative query processing, can be integrated into a hybrid system and into the decision model in further steps. A decision model for the selection of the optimal architecture has to be extended and provides also the basis for self-tuning in hybrid system. Thereby, analyses for the storage architecture selection have to be broken down from whole workloads to parts of it (decomposition), e.g., storage architecture will be advised for each relation.

3 Development Steps for a Decision Model

The database community has mainly considered that row stores can solve all data management tasks [12, 13] in the past. Other storage architectures lead to niche solutions [14–16]. Thereby, column stores are most suitable for DWH and business intelligence [7]. Some TPC-H benchmark results emphasize this estimation but one cannot exactly say which application or workload advances the usage of a column store (cf. Section 2). To the best of our knowledge, there is no study or model which determines the use of column stores for a DWH. In this section, we present the first ideas to develop a decision model for storage architectures selection. Afterwards, we discuss the main influence factors for a decision model according to storage architectures[3] and how we can extend this model.

3.1 Workload Patterns

We have examined the general assumption that the typical column store application is DWH. Aggregation and grouping are typically used in the DWH domain but computations based on tuples occur in these workloads, too. As we already mentioned column stores perform worse on tuple operations, thus column stores are not suitable for each

[3] In the remaining work, we use column and row store as synonyms for the corresponding database storage architectures.

application in the DWH domain. Our considerations should help to overcome the main problem in this domain according to the selection of storage architectures. We assume, row and column stores have their own application fields within the DWH domain especially with respect to the new requirements [5, 6, 8–11]. Hence, we have to analyse workloads and derive workload patterns to estimate or select the most suitable architecture. Additionally, we need to analyse computation types of DBMS operations for the DWH domain.

Based on our analyses, any workload is composed out of three workload patterns that crucially influence the performance on both architectures. First, aggregation and grouping form one pattern because in an abstract manner, the computations proceed the same way. The second pattern covers join operations as essential part of nearly every DBMS. Third, tuple operations/reconstructions are composed in a separate pattern.

Aggregation and grouping are very important in the DWH domain, e.g., cube building or on-the-fly analysis. The aggregation and grouping pattern has to be analysed for row and column stores even if the operations themselves are the same because they are processed in different ways. We need to know how different both architectures perform on this pattern. The comparison of two systems provides system-specific results which we have to generalise. This pattern points out the strengths of columns stores.

The behaviour of join pattern is similar to the aggregation and grouping pattern. There are several join techniques in DBMSs, e.g., merge join, hash join, or nested loop join, which are common in both architectures. Hence, we do not analyse different join techniques themselves but the different computation types in both architectures, e.g., vector and non-vector based. Join operations crucially influence the size of intermediate results, e.g., selectivity join over primary key values or full table join that imply different amount of tuple reconstructions in column stores. Thus, the performance can be changed critically.

The number of tuple operations and/or reconstructions is a critical factor. In contrast to row stores, each tuple operations/reconstruction causes computational costs in column stores because column stores recompose tuples with all necessary columns. Row stores access tuples directly and cause only costs directly from the access path, i.e., this pattern advantages row stores and also represents several new mentioned requirements.

Typically, insert, delete, and update operations are not considered in the DWH domain, so we do not discuss these operations for now. For the new requirements in the DWH domain our patterns should be extended in this direction at a future date. Equally, the projection will not covered in a separate pattern because each system uses projections on intermediate and final results.

3.2 The Decision Model

We mentioned, different workload patterns are more suitable for either row or column store architecture (cf. Section 3.1). None of both architectures, row- and column-oriented, support the whole DWH domain optimally, i.e., we have to assign different workload patterns to one of the two storage architectures. Hence, we have to identify the influence of different workload patterns on the architectures and the performance of these patterns on a certain architecture. The workload patterns are used in our decision model that gives an advice for the optimal storage architecture. The input parameters

are a given workload, statistics that we map into our workload patterns, and derived or user-defined heuristics/rules. The rough functionality is covered below and given in Fig. 1.

A given workload will be analysed and parts of the decomposed workload are mapped to the described patterns. As we mentioned before, the different patterns advantage a certain architecture. We compute the costs of each pattern for the respective architecture and aggregate these costs for instance. In this case, the most qualified architecture for a given workload causes less total costs. Hence, we can recommend a certain architecture for a given workload. Therefore, we can derive heuristics and rules from our workload patterns which contain the workload statistics. The heuristics and rules can be iteratively refined. The different optimization techniques and/or rules of a system have to be hidden for a general model because we want to evaluate the architectures and not the different (query) optimization techniques. This reduces the initial complexity the decision model development.

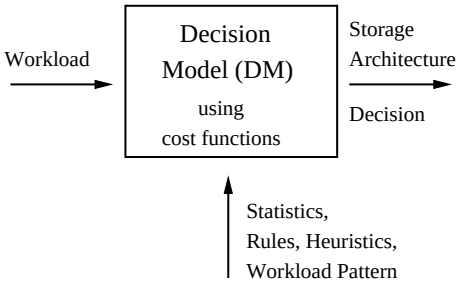

Fig. 1. Scheme of our Decision Model

In detail, we analyse the workload based on used operations, e.g., we count the frequency of operations and the participated tuples and columns for each operation. The computation cost for each operation, e.g., average calculation operation (aggregation/grouping pattern); will be estimated by statistics of corresponding DBMS or empirical determined factors. We state, these factors weight costs of column and row stores for a certain operation to each other. In other words, a heuristic could state that column stores cause costs by factor `X` per tuple for a certain column and row stores cause costs by factor `Y` per tuple for a certain column in our average calculation example. This procedure will be repeated for each operation belonging to a pattern. From the heuristics, we can derive decision rules. For the total costs calculation, it is not decisively if the costs are completely computed or some costs are estimated by heuristics or rules. The usage of heuristics or rules will only reduce the computation complexity, i.e., these decision rules replace cost computations in our decision model for pre-defined events.

Due to a fine-granular approach, the complexity of storage architecture decision based on statistics is very high. As we mentioned, we can use rules to reduce the complexity, e.g., column stores perform always better on average calculation for a column than row stores. To derive correctly heuristics and rules, we need to use real statistics from operating DBMSs. Thus, we have to accept the computation costs and start with the fine granular approach to constitute facts for optimization of decision process. Afterwards, we are able to reduce the complexity by our weighted heuristics/rules, e.g., we expect that a certain part of the workload belongs to average calculation operations that perform better on column store by factor `X`. The complexity will crucially lowered by each heuristic for a certain operation. As a consequence of coarse granularity, we

only have to compute the costs for operations where the performance approximation is not decidable by heuristics and rules.

In the design process, we cannot build on given workloads and have to use an estimated workload behaviour. Again, we use weighting factors to estimate the performance of workload parts. Our coarse decision model approach using heuristics and rules will satisfy these demands. Therefore, we use weighting factors which we obtained by complexity reduction process of fine-granular approach. In the most extreme case, the decision rules stored in the decision model can describe entire patterns and replace the complete cost calculation after workload decomposition.

3.3 Development Strategies

In this section, we will present development strategies for a decision model to select the optimal storage architecture. We assume at least two development strategies.

First, we develop a DBMS which supports independently both architectures to obtain and evaluate heuristics and rules for the model, i.e., we can switch the storage engines within the DBMS. Thus, we can hide the optimization techniques for decision model development because both storage engines use the same DBMS functionalities. Finally, we use a set of cost functions to estimate the performance differences between the architectures in our decision model. With the assistance of such a decision model, we can select an architecture in the design process or if necessary the redesign for a certain application.

Second, we develop a decision model that does not hide system-specific optimization techniques. This strategy compares two systems, i.e., a column and a row store, and weights the different workload patterns for both systems. Hence, we can develop an architectural decision model for the two systems. Such a model can give advice which architecture we should use for a given workload. However, this strategy implies new parameter sets for the decision model concerning each pairs of DBMSs.

We argue, one system that supports both architectures is needed to obtain a general model. Hence, our model based on heuristics/rules can be proven and refined using such a system, i.e., we can decide which architecture is more suitable for a given workload.

Independently from the development strategy, the general functionality of the decision model is equally for design tasks (weighted comparison) and self-tuning (hybrid system). The set of rules derived from the workload patterns and heuristics are integrated as cost functions into the decision model. Finally, the given workload will be processed by the decision model based on cost functions.

3.4 Interactions with other DBMS parts

Many aspects influence the storage model decision. We introduce the estimated main factors of influence to support an open and objective discussion and introduce them into the model in later stages. We assume at least four main interactions: the SQL optimizer, the query processor, the query processing, and compression techniques.

First, the SQL optimizer is integrated into nearly every DBMS. The functionality, optimization rules, and implementation techniques vary from system to system and make a comparable evaluation very difficult. At the same time, no current DBMS supports both, column-oriented and row-oriented, architectures. Even if a DBMS supports

both architectures, it is still a vague expectation that its SQL optimizer does not affect the query execution in different ways. By hiding the optimization step, we can guarantee that the SQL optimizer does not perform better for one of both storage architectures.

Second, the query processor of a system is an important aspect for the performance of a system. Thereby, the quality of the query processor depends on the functionality, quality of implementation etc. Even worse, column stores can use a row-oriented or a column-oriented query processor. In case of column stores, query optimization depends on the query processor and point of time for tuple reconstruction.

Third, the query processing (and optimization) can be based on tuples like all row stores and most column stores or are based on columns like C-Store/Vertica [5]. By using a column-oriented query processor, advantages of compression techniques can be utilized more efficiently because columns can be processed for a query without decompression [5].

The last aspect regarding different compression techniques is directly linked to query processing on compressed data. Several compression techniques are used in column stores in miscellaneous variants, e.g., dependently on data type. Thereby, it is nearly impossible to observe every aspect of compression across different systems. Some systems use a huge set of compression techniques (nearly for each data type) and other use compression only for specific columns. Adding the compression techniques of row stores will corrupt these observations because row stores process compression tuple-wise or partition-wise with different data types (mostly based on relations).

To hide the mentioned interactions, we propose the extension of a row store, e.g., BerkeleyDB [17] or FAME DBMS [18], with column-oriented architecture. This strategy promises the best results according to the model development. Hence, we guarantee the same environment for both architectures and develop a general model for them.

3.5 Perspective

Nevertheless, we have to recover the interactions and optimization techniques which we had hidden (cf. Section 3.4). These interactions have to be integrated in a general manner into our decision model because we also consider distinct systems in our general decision model besides a prototype. In this way, we can develop a general decision model for column and row stores. But as already mentioned, neither of both architectures can perform efficiently on each workload. We can only overcome this gap between column and row stores with aid of a hybrid architecture and develop a hybrid system based on this architecture.

To develop a hybrid system or architecture, we do not need to integrate the interactions of DBMS parts. On the contrary, a hybrid system should utilize and consider architectural strengths and weaknesses before considering optimization techniques. We argue, these optimization techniques have to be adapted for a hybrid system anyway, thus we do not need to consider them. The reimplementation or adaption of current optimization techniques is also a chance to overcome existing barriers, e.g., the static storage allocation [19], OLTP and/or real-time DWH [6] because they strongly depend on the current architectures. The reached degree of freedom involves administrative overhead which has to be lower than the benefit guaranteed by cost functions.

In contrast to our proposed prototypical implementation for model development, we mention that it could be more advantageous to develop the hybrid architecture based on

a column store (or complete reimplementation). We reason this approach by functional gaps between column and row stores, e.g., update-in-place strategies cannot be applied in column stores or concurrency control in column stores is very different (mostly delta based).

Update operations are another new challenge in the DWH domain and new DWH applications. Updates can critically affect the performance of a DBMS. We note that updates can be implemented by delete and insert operations. So, we determine all three operations as update behaviour. Formerly, the database community estimated, update processing does not influence DWH applications because the DWH works on time-invariant data and executes the updates in an ETL process. Such process requires a time of minor workload or offline time of the DWH but most DWH should be available 24 hours a day. However, new approaches [9, 20, 21] show that updates processing will be crucially increased for DWH. Updates are an architectural problem in column stores due to tuple partitioning in the column-oriented architecture. To support the tuple reconstruction after partitioning, each column value has to refer to unique tuple identifier or have to keep the columns in physical sequential order especially for sorted columns. Hence, several column store implementations try to reduce drawbacks by updates, e.g., C-Store/Vertica [5]. However, we want to evaluate a decision model for architectural decision between column and row store for a certain application and not specific implementations even if there are already good approaches available.

Finally, a hybrid architecture can help to overcome the update problem of column stores and could open new application fields because it contains both architectures that can be adapted to workloads, e.g., storage architecture will be advised for each relation. A hybrid architecture that utilizes both architecture enables self-tuning techniques in an architectural manner, i.e., the storage architecture of a certain part of the database can be changed and adapted to changing workloads. Additionally, the adaptation of self-tuning techniques from row stores are possible and will increase the overall performance, e.g., self-adapting compressions (techniques) for shifting data distribution or materialized views that store the tuple reconstruction results for frequently accessed partitioned tuples etc. These self-tuning approaches can reduce I/O costs for data access even for changing data sets or reduce tuple reconstruction costs within query execution. These self-tuning techniques can certainly be utilized in column stores, too.

4 Related Work

In recent years, several open- and closed-source column stores have been released [4, 5, 15, 22]. There are well-known column stores but all systems are pure column stores and do not support any row store functionality. Abadi et al. [23] 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 weakness of both architectures to encourage our work of combining them in a hybrid system.

Regarding the solutions for architectural problems, there are systems available which are not a hybrid system in our sense (between a column and a row store). Neverthe-

less, they contain very interesting approaches for development of a hybrid system. C-Store [5] 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 online transactional processing (OLTP) data [9]. 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 are not a hybrid system in our sense because we want to combine two architectures to a hybrid storage without replication.

Another interesting approach is Ingres/Vectorwise which applies the Vectorwise (formerly MonetDB/X100) architecture into the Ingres product family [24]. In cooperation with Vectorwise, Ingres is developing a new storage manager ColumnBM for the new Ingres/Vectorwise. From [24] and product documentations, the integration of the new architecture into the existing environment remains unclear.

BigTable [16] or HadoopDB [25] bring together different architectures or competing approaches. However, these systems are developed for quite different application fields (niche solutions) and do not support the relational data model. Thereby, the development of a hybrid system is supported by our decision model which is our main goal in our work. We compare the architectures to obtain a general model and not certain DBMSs which is nearly impossible by the amount of DBMSs in the community. Moreover, a decision model and its resulting knowledge are the basis to develop a hybrid architecture. This strategy is the main distinction of our work to the other approaches.

To develop our decision model, we need to analyse any workloads and derive workload patterns therefrom. Therefore, we can utilize, adapt and extend existing approaches such as Turbyfill's approach [26] that considers mixed database workloads concerning disk access. We can map this approach to the strengths and weaknesses of different architectures. In contrast, the approach by Raatikainen [27] considers workload classes and how to find them based on cluster analysis. We do not want to cluster workloads into a certain number of classes but we want to classify operations. We can use these and other approaches to find and evaluate our workload patterns. The approaches of Favre et al. [28] and Holze et al. [29] step into the direction of 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 is very interesting according to our proposed hybrid system. The related approach of Holze et al. concerns clustering of workloads based on distance functions but is developed for lightweight and stream-based operations. Thereby, the development of a hybrid system is supported by the workload pattern development.

The current research reflects new approaches to solve the update problems of OLAP applications, e.g., dimension updates [10]. Moreover, the update problem is increased according to new demands like real-time DWH [8, 11]. These new demands call for new decision models in the DWH domain and databases at all.

5 Conclusion

In this paper, we report on a study that we performed on column store and row store systems. In this study, we observed the necessity of a decision model for architectural design in the data warehousing domain. To map a workload to an architecture (column

or row store), we identified three workload patterns which will be refined into fine-grained sub-patterns. The patterns have to be evaluated concerning their performance on different architectures, i.e., the recommended architecture depends on workload or found pattern. Based on these patterns, we presented our ideas to develop our decision model. In future work, we want to advise the architectural (re-) design decisions of DWH systems. We assume that the development of a decision model for the choice of storage architecture (row or column store) is the precondition for new self-tuning techniques in hybrid systems. Furthermore, the generality of a decision model should be verified in a number of case studies. The interactions of DBMS parts have to be discussed afterwards, e.g., SQL optimizer or query processor.

Moreover, we conclude the development of a hybrid architecture to overcome the gap between column and row stores as consequence of our study. This step is an explicit conclusion of our considerations concerning a decision model because we can map the workload pattern to the strengths of either architecture. So, our model will be the base for a hybrid system and we can utilize it for column, row, and hybrid architectures. We argue that our model will foster the development of a hybrid architecture by architectural design decision support and rules/heuristics.

A hybrid system will enable self-tuning approaches in an architectural manner. The monitored workloads will be analysed with the aid of our decision model. The self-tuning approach enables continuous adaption of architectures in small steps for a hybrid system instead of periodical (re-) design of complete systems. The adaptive architecture of a hybrid system will save the costly redesigns of a system and will endorse changing workloads.

In the future, we will determine the necessary workload patterns using a prototypical implementation and evaluate their performance on the different architectures. These patterns will be integrated in our model and refined by evaluation on different existing environments/systems.

Acknowledgments. We thank Christian Kästner, Martin Kuhlemann, Ingolf Geist, and Veit Köppen for helpful discussions and comments on earlier drafts of this paper.

References

1. Weikum, G., Hasse, C., Moenkeberg, A., Zabback, P.: The COMFORT automatic tuning project, invited project review. Information Systems **19**(5) (1994) 381–432
2. IBM: An architectural blueprint for autonomic computing. White Paper (June 2006) Fourth Edition, IBM Corporation.
3. Chaudhuri, S., Narasayya, V.: Self-tuning database systems: A decade of progress. In: VLDB '07, VLDB Endowment (2007) 3–14
4. Zukowski, M., Boncz, P.A., Nes, N., Heman, S.: MonetDB/X100 - a DBMS in the CPU cache. IEEE Data Engineering Bulletin **28**(2) (June 2005) 17–22
5. Abadi, D.J.: Query execution in column-oriented database systems. PhD thesis, Cambridge, MA, USA (2008) Adviser: Madden, Samuel.
6. Plattner, H.: A common database approach for OLTP and OLAP using an in-memory column database. In: SIGMOD '09, New York, NY, USA, ACM (2009) 1–2
7. Abadi, D.J., Boncz, P.A., Harizopoulos, S.: Column oriented database systems. PVLDB '09 **2**(2) (2009) 1664–1665

8. Santos, R.J., Bernardino, J.: Real-time data warehouse loading methodology. In: IDEAS '08, New York, NY, USA, ACM (2008) 49–58
9. Schaffner, J., Bog, A., Krüger, J., Zeier, A.: A hybrid row-column OLTP database architecture for operational reporting. In: BIRTE '08. (2008)
10. Vaisman, A.A., Mendelzon, A.O., Ruaro, W., Cymerman, S.G.: Supporting dimension updates in an OLAP server. Information Systems **29**(2) (2004) 165–185
11. Zhu, Y., An, L., Liu, S.: Data updating and query in real-time data warehouse system. In: CSSE '08, Washington, DC, USA, IEEE Computer Society (2008) 1295–1297
12. Stonebraker, M., Çetintemel, U.: One size fits all: An idea whose time has come and gone. In: ICDE. (2005) 2–11
13. Stonebraker, M., Bear, C., Çetintemel, U., Cherniack, M., Ge, T., Hachem, N., Harizopoulos, S., Lifter, J., Rogers, J., Zdonik, S.B.: One size fits all? Part 2: Benchmarking studies. In: CIDR. (2007) 173–184
14. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Communications of the ACM **51**(1) (2008) 107–113
15. Legler, T., Lehner, W., Ross, A.: Data mining with the SAP NetWeaver BI Accelerator. In: VLDB '06, VLDB Endowment (2006) 1059–1068
16. Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A., Gruber, R.E.: Bigtable: A distributed storage system for structured data. ACM Transactions on Computer Systems **26**(2) (2008) 1–26
17. Yadava, H.: The Berkeley DB Book. Apress, Berkely, CA, USA (2007)
18. Rosenmüller, M., Siegmund, N., Schirmeier, H., Sincero, J., Apel, S., Leich, T., Spinczyk, O., Saake, G.: FAME-DBMS: Tailor-made data management solutions for embedded systems. In: SETMDM '08, New York, NY, USA, ACM (2008) 1–6
19. Lübcke, A.: Self-tuning of data allocation and storage management: Advantages and implications. In: GvD '09: Proceedings of the 21. GI-Workshop on Foundations of Databases. (2009) 21–25
20. Langseth, J.: Real-time data warehousing: Challenges and solutions. White Paper (February 2004) Claraview (Teradata Corporation).
21. Vandermay, J.: Considerations for building a real-time data warehouse. White Paper (November 2001) DataMirror Corporation.
22. Ślęzak, D., Wróblewski, J., Eastwood, V., Synak, P.: Brighthouse: an analytic data warehouse for ad-hoc queries. PVLDB '08 **1**(2) (2008) 1337–1345
23. Abadi, D.J., Madden, S.R., Hachem, N.: Column-stores vs. row-stores: how different are they really? In: SIGMOD '08, New York, NY, USA, ACM (2008) 967–980
24. Ingres/Vectorwise: Ingres/VectorWise sneak preview on the Intel Xeon processor 5500 series-based platform. White Paper (September 2009)
25. Abouzeid, A., Bajda-Pawlikowski, K., Abadi, D.J., Rasin, A., Silberschatz, A.: HadoopDB: An architectural hybrid of MapReduce and DBMS technologies for analytical workloads. PVLDB '09 **2**(1) (2009) 922–933
26. Turbyfill, C.: Disk performance and access patterns for mixed database workloads. IEEE Data Engineering Bulletin **11**(1) (1988) 48–54
27. Raatikainen, K.E.E.: Cluster analysis and workload classification. SIGMETRICS Performance Evaluation Review **20**(4) (1993) 24–30
28. Favre, C., Bentayeb, F., Boussaid, O.: Evolution of data warehouses' optimization: A workload perspective. In: DaWaK '07. (2007) 13–22
29. Holze, M., Gaidies, C., Ritter, N.: Consistent on-line classification of DBS workload events. In: CIKM '09. (2009) 1641–1644