Workload-based Heuristics for Evaluation of Physical Database Architectures

Andreas LÜBCKE [a], Martin SCHÄLER [a], Veit KÖPPEN [a] and Gunter SAAKE [a]
[a] *School of Computer Science,*
Otto-von-Guericke-University Magdeburg, Germany,
{andreas.luebcke,martin.schaeler,veit.koeppen,gunter.saake}@ovgu.de

Abstract. Database systems are widely used in different application domains. Therefore, it is difficult to decide which database management system meets the requirements of a certain application at most. This observation is also true for scientific and statistical data management, due to new application and research fields. New requirements are often implied to data management while discovering unknown research and applications areas. That is, heuristics and tools do not exist to select an optimal database management system. In previous work, we proposed a decision framework based on application workload analyses. Our framework supports application performance analyses by mapping and merging workload information to patterns. In this paper, we present heuristics for performance estimation to select an optimal database management system for a given application. We show that these heuristics improve our decision framework by complexity reduction without loss of accuracy.

Keywords. Heuristics, storage architecture, design, performance, query processing

Introduction

Database systems (DBS) are pervasively for almost each branch of business activity. Therefore, DBS have to manage different requirements for heterogeneous application domains. New data management approaches are developed (e.g., NoSQL-DBMSs [9,14], MapReduce [12,13], Cloud Computing [3,16,7], etc.) to make the growing amount of data[1] manageable for special application domains. We argue, these approaches are developed for special applications and need a high degree of expert knowledge for usage, administration, and optimization. However, we focus our observations to relational database management systems (DBMSs) in this paper. Relational DBMSs are commonly used DBS for highly diverse applications and besides relational DBMS are well-know to many IT-affine people.

Relational DBMSs[2] are developed to manage data of daily business and reduce paper trails of companies (e.g., finance institutions) [2]. This approach dominates more and more the way of data management that we today know as online transaction processing (OLTP). Nowadays, faster and more accurate forecasts for revenues and expenses are not enough anymore. A new application domain evolves that focuses on analysis of data to support business decisions. Codd et al. [8] defines this type of data analysis as

[1] Consider the data explosion problem [22,26].
[2] In the following, we use the term DBMS synonymously for relational DBMS.

online analytical processing (OLAP). Consequently, two disjunctive application domains for relational data management exist with different scopes, impacts, and limitations (cf. Section 1).

In recent years, business application have a high demand for solutions that support tasks from both OLTP and OLAP [15,21,28,31,33,34], thus coarse heuristics for typical OLTP and/or OLAP applications have become obsolete (e.g., data warehouses without updates always perform best on column-oriented DBMSs). Nevertheless, new approaches, which we mention above, also show impacts and limitations (e.g., in-Memory-DBMS only, focus on real-time or dimension updates, etc.), such that we argue there is no DBMS that fits for OLTP and OLAP in all application domains. Heuristics and current approaches for physical design and query optimization only consider a certain architecture[3] (e.g., design advisor [36] and self-tuning [10] for row-oriented DBMSs or equivalent for column-oriented DBMSs [19,30]). That is, the decision for a certain architecture has to be done beforehand. Consequently, there is no approach that neither advices physical design spanning different architectures for OLTP, OLAP, and mixed OLTP/OLAP workloads nor that estimates which architecture is optimal to process a query/database operation.

However, we refine obsolete heuristics for physical design of DBS (e.g., heuristics from classical OLTP domain). We consider OLTP, OLAP, and mixed application domains for physical design. We present heuristics that propose the usage of row-oriented DBMSs (row stores) or column-oriented DBMSs (column stores) under certain circumstances. Furthermore, we present heuristics for query execution or rather for processing (relational) database operations on column and row stores. Our heuristics show which query type and/or database operation performs better on a particular architecture[4] and how single database operations affect performance of a query or a workload. We derive our heuristics from experiences in workload analyses with the help of our decision model, presented in [23].

In the following sections, we consider the main differences between row- and column store as well as the advantages and disadvantages for column stores. Section 2 addresses our heuristics for physical design of DBMSs concerning different storage architectures (i.e., row or column store). In Section 3, we present heuristics for query processing on row and column stores. Section 4 gives an overview of related research. Finally, we summarize our discussions and give an outlook in Section 5.

1. Column or Row Store: Assets and Drawbacks

In previous work, we already discussed differences between column and row stores according to different subjects [25,24,23]. We can summarize our major observations in Table 1. Of course, column stores architecture also has disadvantages. We just name them because they are mostly contradictory to row store architecture advantages. Column stores perform worse on update operations and concurrent non-read-only data access due to partitioned data, thus on frequent updates and consistency checks tuple reconstructions cause notable cost.

Finally, row stores can outperform column stores in their traditional application domain nor vice versa. Other researchers confirm our consideration that one architecture

[3] We use the term architecture synonymously for storage architecture.

[4] The term architecture refers to row- and column-oriented database architecture.

cannot sustain the other architecture in their native domain (e.g., by simulating architecture through partitioned schema [4]).

Table 1. Advantages of Column Stores

Requirement	CS Property	Comment
Disc space	Reduced	Aggressive compression (e.g., optimal compression per data type)
Data transfer	Less	More data fits in main memory
Data processing	Compressed and decompressed	Does not work for each compression nor for all operations
OLAP I/O	No	Neither for aggregations nor column operations
Parallelization	For inter- and intra-query	Not for ACID-transaction with write operations
Vector operations	Fast	Easily adaptable

2. Heuristics on Physical Design

Physical design of DBSs is important as long as DBMSs exist. We do not only restrict our considerations to a certain architecture. Further, we consider a set of heuristics that can be used to forecast which architecture is more suitable for a given application.

Some existing rules still have their validity. First, pure OLTP applications perform best on row stores. Second, classic OLAP application with an ETL (extract, transform, and load) process or (very) rare updates are satisfied with column stores[5]. In the following, we consider a more exciting question. In which situation one architecture outperforms the other one and in which case they perform nearly equivalent.

OLTP For OLTP workloads, we just recommend to use row stores as we do it for decades. A column store does not achieve competitive performance except column-store architecture will significantly change.

OLAP In this domain one might suspect a similar situation as for OLTP workloads. However, this is not true in general. We are aware that column stores outperform row stores for many applications and/or queries in this domain; that is, for aggregates and access as well as processing of a few columns. In most cases, column stores are most suitable for applications in this domain. Nevertheless, there exist complex OLAP queries where column stores lose their advantages (cf. Section 1). For these complex queries, row stores can achieve competitive results, even if they consume more memory. These queries have to be considered for architecture selection because they critically influence the physical design estimation even more if there is a significant amount of these queries in workload.

OLTP/OLAP In these scenarios, your physical design strongly depends on the ratio between updates, point queries, and analytical queries. Our experience is that column stores perform about 100-times slower on OLTP-transactions (updates, inserts, etc.) than row stores. This fact is even worse because we do not even consider concurrency (e.g., ACID); that is, we make this observation in single-user execution on transactions. Assuming transaction and analytical queries in average take the same time, we state that one transaction only occurs every 100 queries (OLAP). The fact that analytical queries

[5]Note: Not all column stores support updates just ETL.

last longer than a single iteration leads us to a smaller ratio. Our experience shows that 10 executions of analytical queries on a column store are of greater advantage than the loss by a single transaction. If you have a smaller ration than 10:1 (analyses/Tx) then we cannot give a clear statement. We recommend using a row store in this situation or you know beforehand that the ratio will change to more analytical queries. If the ratio falls under this ratio for a column store and is not a temporary change then a system change is appropriate. So, in mixed workloads it is all about the ratio of analytical queries to transactions. Note that the ratio 100:1 and 10:1 can change considering OLAP-query type. We address this issue in Section 3.

Our heuristics can be used as a guideline for architecture decisions for certain applications. That is, we select the most suitable architecture for an application and afterwards use existing approaches (like IBM's advisor [36]) to tune physical design of a certain architecture. If workload and DBSs are available for analysis, we emphasize to use our decision model [23] to calculate an optimal architecture for a defined workload. The presented heuristics for physical design extent our decision model to reduce calculation costs (i.e., solution room is pruned). Additionally, heuristics make our decision model available for scenarios where only restricted information is available.

3. Heuristics on Query Execution

In the following, we present heuristics for query execution on complex or hybrid DBS. We assume a DBS that supports column- and row-store functionality or a complex environment with at least two DBMSs containing data redundant whereby at least one DBMS is setup for each OLTP and OLAP. In the following, we only discuss query processing heuristics for OLAP and OLTP/OLAP workloads due to our assumptions above and the fact that there is no competitive alternative to row stores for OLTP.

OLAP In this domain, we face a huge amount of data that generally is not frequently updated. Column stores are able to significantly reduce the amount of data due to aggressive compression. That is, more data can be loaded into main memory and I/O between storage and main memory is reduced. We state that this I/O reduction is the major benefit of column stores. We made the experience that row stores perform worse on many OLAP queries because row stores drop performance due to fact that CPUs often are idle while waiting for I/O from storage. Moreover, row stores read data that is not required due to the physical design of data. In our example from TPC-H benchmark [32], one can see that only a few columns of the lineitem relation have to be accessed (cf. Listing 1). In contrast to column stores, row stores have to access the complete relation to answer this query. We state, most OLAP query fit into this pattern. We recommend using column store functionality to answer this type of OLAP queries as long as they only access a minority of columns from relation for aggregation and predicate selection.

```
1 from     lineitem
2 where    l_shipdate >= date '1994-01-01'
3     and  l_shipdate < date '1994-01-01' + interval '1' year
4     and  l_discount between .06 - 0.01 and .06 + 0.01
5     and  l_quantity < 24;
```

Listing 1. TPC-H query Q6

Complex Queries OLAP is often used for complex analyses, thus queries become more complex too. These queries describe complex issues and/or produce large business reports. Our example (Listing 2) from the TPC-H benchmark could be part of a report (or a more complex query). Complex queries access and/or aggregate many tuples, that is, nearly the complete relation has to be read. This implies a number of tuple reconstructions that significantly reduce the performance of column stores. Hence, row stores can achieve competitive performance because nearly the complete relations have to be accessed. Our example shows another reason for a number of tuple reconstructions: group operations. Tuples have to be reconstructed before aggregating groups. Other reasons for significant performance reduction by tuple reconstructions can be a large number of predicate selections on different columns as well as complex joins. We argue, this complex OLAP query type can be executed on both architectures. This fact can be used to load balance queries in complex environments and hybrid DBMSs.

```
6     select c_custkey, count(o_orderkey) from
7         customer left outer join orders on c_custkey = o_custkey
8         and o_comment not like '%special%request%'
9         group by c_custkey) as c_orders (c_custkey, c_count)
10 group by c_count
11 order by custdist desc, c_count desc;
```

Listing 2. TPC-H query Q13

Mixed Workloads In mixed workload environments, our first recommendation is to split the workload into two parts OLTP and OLAP. Both parts can be allocated to the corresponding DBS with row- or column-store functionality. As we mentioned above, this split methodology can also be used for load balancing if one DBS is too busy. With this approach, we achieve competitive performance for both OLAP and OLTP because, according to our assumptions, complex systems have to have at least one DBMS of each architecture. Further, we state that a future hybrid system has to satisfy both architecture, too. Processing mixed workloads with our split behavior has two additional advantages. First, we can reduce issues with complex OLAP queries by correct allocation or dividing over both DBS. Second, we can consider time-bound parameters for queries.

As mentioned above, two integration methods for the query-processing heuristics are available. First, we propose the integration into a hybrid DBMSto decide where to optimally execute a query. That is, heuristics enable rule-based query optimization for hybrid DBMS as we know from row-store optimizer [17]. Second, we propose a global manager on top of complex environments to optimally distribute queries (as known from distributed DBMSs [27]). Our decision model analyzes queries and decides where to distribute queries to. Afterwards, queries are locally optimized by DBMS itself. Note, time-bound requirements change over time and system design determines how up-to-date data is in OLAP system (e.g., real-time load [31] is available or not). We state that such time-bound requirements can be passed as parameters to our decision model. That is, even OLAP queries can be distributed to OLTP part of the complex environment if data in OLAP part is insufficiently updated or vice versa we have to ensure analysis on most up-to-date data. Additionally, such parameters can be used to alternatively allocate high-priority queries to another DBS part if the query has to wait otherwise.

4. Related Work

Several approaches are developed to analyze and classify workloads (e.g., [18,29]). These approaches recommend tuning and design of DBS, try to merge similar tasks, etc. to improve performance of DBSs. To the present, workload-analysis approaches are limited to classification of queries to execution pattern or design estimations for a certain architecture; that is, the solution space is beforehand pruned analysis and performance estimations are done. This results in information loss due to an inappropriate reduction. With our decision model and heuristics, we propose an approach that is independent from architectural issues.

Due to success in the analytical domain, researchers devote more attention on column stores whereby focused on analysis performance [1,19] or to overcome update-problems with separate storages [15,1]. However, a hybrid system in an architectural manner does not exist. In-memory-DBMSs are developed in recent years (e.g., HyPer [21]) that satisfy requirements for mixed OLTP/OLAP workloads. Nevertheless, we state that even today not all DBSs can run in-memory (due to monetary or environmental constraints), thus we propose a more general approach.

Recommending indexes [5,10], materialized views [10], configurations [20], or physical design in general [6,37,36] are in focus of researchers. However, all approaches are limited to certain DBMSs or at least one architecture to the best of our knowledge. Our approach is situated on top of these design and tuning approaches. That is, we support a first coarse granular design and tuning for a global view on hybrid systems and utilize existing approaches to optimize locally.

In the literature, researchers compare the performance of column and row stores considering certain scenarios. Cornell and Yu [11] focus on disc-access minimization for transaction to improve query execution time. Abadi et al. [4] compare different approaches for column-oriented storage concerning analytical queries. We state, current applications demand for a combined observations of OLTP and OLAP scenarios.

In line with Zukowski et al. [35], we observe benefits to convert from NSM to DSM and vice versa during query processing. In contrast to Zukowski et al., we do not only focus on CPU trade-offs caused by tuple reconstructions. Our approach is used for both scenarios and considers overall benefit of global and local tuning.

5. Conclusion

In recent years, many new approaches as well as new requirements encourage performance of DBMSs in many application domains. Nevertheless, new approaches and requirements also increase complexity of DBMS selection, DBS design, and tuning for a certain application. We focus our considerations on OLTP, OLAP, and OLTP/OLAP workloads in general. That is, we considered which DBMS is most suitable.

Therefore, we present heuristics on design estimations and query execution for both relational storage architectures row and column stores. We use our decision model [23] to observe the performance of several applications. Consequently, we derive heuristics from our experiences while evaluating our decision model. We present these heuristics for DBS design to a priori select the most suitable DBMS and to tune this system afterwards. Our approach avoids misleading tuning if the architecture selection is wrong. Furthermore, we present heuristics on query execution for both storage architectures. On the one hand, we want to emphasize our heuristics for physical design. On the other hand,

we propose these heuristics for integration in hybrid DBS whether it is a real hybrid DBMS or it is a complex system that consists of different DBMSs (at least one row and one column store). Summarizing, there is no alternative in OLTP environments to row stores, for OLAP applications we recommend to use column stores taking into account that a number of complex OLAP queries can change this recommendation, and finally in mixed OLTP/OLAP environments the focus is on the ratio of OLAP queries to OLTP transactions (again taking very complex OLAP queries into account). Our heuristics are a first step to rule-based query optimization in hybrid systems/architectures.

In future work, we will evaluate our heuristics considering standard benchmark to achieve meaningful results. After evaluation, we will implement the heuristics in our decision model, thus we achieve a design advisor for both relational storage architectures. Finally, we plan to implement a hybrid DBMSs using our heuristics for ruled-based query optimization.

Acknowledgements

This paper has been funded in part by the German Federal Ministry of Education and Science (BMBF) through the Research Program under Contract FKZ: 13N10817.

References

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

[2] 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 Trans. Database Syst.* **1**(2) (1976), 97–137.

[3] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. Above the Clouds: A Berkeley View of Cloud Computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009.

[4] Daniel J. Abadi, Samuel R. Madden, and Nabil Hachem. Column-stores vs. row-stores: How different are they really? In *SIGMOD'08* (2008), 967–980.

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

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

[7] Rajkumar Buyya, Chee Shin Yeo, and Srikumar Venugopal. Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities. In *HPCC* (2008), 5–13.

[8] Edgar F. Codd, Sally B. Codd, and Clynch T. Salley. Providing OLAP to User-Analysts: An IT Mandate. *Ann ArborMichigan* (1993), 24.

[9] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Michael Burrows, Tushar Chandra, Andrew Fikes, and Robert Gruber. Bigtable: A Distributed Storage System for Structured Data. In *OSDI* (2006), 205–218.

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

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

[12] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *OSDI* (2004), 137–150.

[13] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Commun. ACM* **51**(1) (2008), 107–113.

[14] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: amazon's highly available key-value store. In *SOSP* (2007), 205–220.

[15] Clark D. French. Teaching an OLTP database kernel advanced datawarehousing techniques. In *ICDE'97* (1997), 194–198.
[16] Ian T. Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. Cloud Computing and Grid Computing 360-Degree Compared. *CoRR*, abs/0901.0131, 2009.
[17] Goetz Graefe and David J. DeWitt. The EXODUS Optimizer Generator. In *SIGMOD'87* (1987), 160–172.
[18] Marc Holze, Claas Gaidies, and Norbert Ritter. Consistent on-line classification of DBS workload events. In *CIKM'09* (2009), 1641–1644.
[19] Stratos Idreos. Database Cracking: Torwards Auto-tuning Database Kernels. PhD thesis, 2010.
[20] 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*, GI (2003), 620–629.
[21] Alfons Kemper and Thomas Neumann. HyPer: A hybrid OLTP&OLAP main memory database system based on virtual memory snapshots. In *ICDE'11* (2011), 195–206.
[22] Henry F. Korth and Abraham Silberschatz. Database Research Faces the Information Explosion. *Commun. ACM* **40**(2) (1997), 139–142.
[23] Andreas Lübcke, Veit Köppen, and Gunter Saake. A Decision Model to Select the Optimal Storage Architecture for Relational Databases. In *Proceedings of the Fifth IEEE International Conference on Research Challenges in Information Science, RCIS* (2011), 74–84.
[24] Andreas Lübcke and Gunter Saake. A Framework for Optimal Selection of a Storage Architecture in RDBMS. In *DB&IS* (2010), 65–76.
[25] Andreas Lübcke. Challenges in Workload Analyses for Column and Row Storess. In *Grundlagen von Datenbanken* (2010).
[26] Ina Naydenova and Kalinka Kaloyanova. Sparsity Handling and Data Explosion in OLAP Systems. In *MCIS* (2010), 62–70.
[27] M. Tamer Özsu and Patrick Valdurie. *Principles of Distributed Database Systems*. Springer, 3rd edition, 2011.
[28] Hasso Plattner. A common database approach for OLTP and OLAP using an in-memory column database. In *SIGMOD'09*, ACM (2009), 1–2.
[29] Kimmo E. E. Raatikainen. Cluster Analysis and Workload Classification. *SIGMETRICS Performance Evaluation Review* **20**(4) (1993), 24–30.
[30] 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*, VLDB Endowment (2005), 553–564.
[31] Ricardo Jorge Santos and Jorge Bernardino. Real-time data warehouse loading methodology. In *IDEAS'08* (2008), 49–58.
[32] Transaction Processing Performance Council. TPC BENCHMARK^{TM} H. White Paper, April 2010. Decision Support Standard Specification, Revision 2.11.0.
[33] Alejandro A. Vaisman, Alberto O. Mendelzon, Walter Ruaro, and Sergio G. Cymerman. Supporting dimension updates in an OLAP server. *Information Systems* **29**(2) (2004), 165–185.
[34] Youchan Zhu, Lei An, and Shuangxi Liu. Data Updating and Query in Real-Time Data Warehouse System. In *CSSE'08* (2008), 1295–1297.
[35] Marcin Zukowski, Niels Nes, and Peter Boncz. DSM vs. NSM: CPU performance tradeoffs in block-oriented query processing. In *Proceedings of the 4th international workshop on Data management on new hardware*, DaMoN'08, ACM (2008), 47–54.
[36] 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*, VLDB Endowment (2004), 1087–1097.
[37] 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* (2004), 180–188.