Challenges in Workload Analyses for Column and Row Stores*

Andreas Lübcke
School of Computer Science
Otto-von-Guericke-University Magdeburg, Germany
andreas.luebcke@ovgu.de

ABSTRACT

Within the DWH domain, a new architecture known as column store is emerging especially to improve the performance of data analyses/aggregations. Column stores offer good results in performance benchmarks like TPC-H. In recent years, row stores dominate the data warehousing domain. We consider application fields for both architectures. We want to figure out and examine different application fields for row and column stores. To encourage our assumption, we perform a case study based on the TPC-H benchmark. To the best of our knowledge, there is no advisor for selection of storage architecture for a given application. We present an idea to overcome the workload analysis problems across different architectures in the DWH domain.

1. INTRODUCTION

Database management systems (DBMSs) are pervasive in current applications. With database tuning, practitioners aim at optimal performance especially for large-scale systems like a data warehouse (DWH). The administration and optimization of DBMSs is costly. In recent years, row-oriented DBMSs (row stores) dominate the DWH domain but a new approach known as column-oriented DBMSs (column stores) arises [2]. The column stores should increase the performance of analyses in the DWH domain [1, 9, 14, 18]. In the DWH domain, the column stores outperform the row stores lately and offer good results in performance benchmarks like TPC-H[1]. Consequently, the complexity of system design increases through the new available DBMSs because the design process also implies the selection of a suitable system architecture now. To the best of our knowledge, there is no framework available that advises the optimal architecture for a given application or workload. Heuristics and studies show that column stores perform very well on aggregations [3, 14]. In contrast, column stores perform worse than row stores on update and tuple operations, i.e., processing entire tuples. Operation types and their frequency within a workload differ from application to application, thus we have to analyze their workloads to select the most suitable architecture. In other words, we have to contrast the performance of row stores with the performance of column stores for different applications.

In this paper, we analyze the differences between column and row stores to show the increased challenge of system design within the DWH domain. We illustrate new challenges for workload analyses and introduce a solution. Afterwards, our study illustrates that there are application fields for both architectures within the DWH domain.

2. COLUMN VS. ROW STORES

This section gives an overview to the differences in storage method and functionality between column and row stores. Due to the differences, we show the increased complexity of system design and the difficulty of performance estimation across row and column stores.

First, the storage architecture differs in the type of partitioning a (relational) table. Row stores horizontally partition a table, i.e., all attribute values of one tuple are sequentially stored. In contrast, column stores sequentially store the values of an attribute (column). Hence, column stores have to reconstruct the tuples during the query execution if more than one column is affected. In the DWH domain, we mostly process aggregations over one column. Column stores reduce the overhead for aggregations, e.g., I/O costs, because the affected column is directly accessed. However, the vertical partitioning in column stores also implies worse performance on update processing. That means, caused by the column-oriented architecture, the tuples have to be reconstructed and partitioned subsequently during update processing. In contrast, row stores perform better on tuples (operations) and updates that are also important operations within workloads even in the DWH domain. Several approaches [1, 9, 12, 16] attempt to solve the update problems of column stores but these approaches do not reach the performance of row stores. Abadi et al. show that vertical partitioning of relations in row stores is not a suitable compromise, too [3].

Second, column and row stores also differ in their functionality. Row stores utilize indexes and materialized views to improve the performance. Therefore, DBMS vendors and researchers develop a number of self-tuning techniques [5] to automatically tune the DBMSs. These techniques, e.g., in-

*We thank Martin Kuhlemann and Ingolf Geist for helpful discussions and comments on earlier drafts of this paper.
[1]Refer http://www.tpc.org/tpch/results/tpch_perf_results.asp for the latest results.

GvD Workshop'10, 25.-28.05.2010, Bad Helmstedt, Germany

dexes and index self-tuning, do not exist for column stores, i.e., column stores do not utilize mature frameworks/tools for self-tuning like row stores according to shifting workloads [5]. In contrast, column stores have a better support of compression techniques than row stores [1]. Column stores offer a number of compression techniques that can be chosen for each column concerning its data type. Moreover, some column stores can directly process on compressed data [1]. Row stores have to use one compression for a tuple or tuple partition [3], i.e., the selected compression techniques is a compromise to satisfy different data types. Furthermore, column and row use different query processing techniques caused by the different storage architectures. On the one hand, column stores have to reconstruct tuples while query processing whereby the point of time for reconstruction crucially influence the query processing [4]. On the other hand, row stores can directly process on several columns (no tuple reconstruction) but they have to access entire tuples even if just one column have to be processed. Row store query processors are always tuple-oriented no matter how the data is partitioned. Column stores can utilize a row-oriented query processor as well as a column-oriented query processor [1], i.e., the performance of a column store is already affected by the selection of the query processor.

In conclusion, we state that the complexity of database design (and also tuning) has been increased by the appearance of column stores within the DWH domain. In recent years, we estimate the performance of row stores for a given workload and tune these systems concerning the given workload. We are able to compare several systems (row stores) because their core functionality only differs very slightly. Today, we have to estimate the performance of DBMSs across two architectures, i.e., we have to choose the most suitable architecture for a given workload. One can argue that column stores are more suitable for DWH applications in general because column stores perform better on essential operations for the DWH domain, e.g., aggregations. We argue there are application fields for column and row stores because none of the two architectures is suitable for every workload. Current approaches [1, 9, 11, 12, 16] confirm our position, e.g., updates in real-time DWHs [17].

3. NEW CHALLENGES FOR WORKLOAD ANALYSES

New applications demand for performant aggregations in column stores as well as performant tuple operations in row stores or at least row store functionality. Hence, we need new approaches for workload analyses because we have to compare different DBMSs (row vs. column store) that differ significantly in their storage, functionality, and query processing techniques.

In recent years, we compare different tuning and optimization techniques of DBMSs to figure out the most suitable DBMSs for a given workload. Row stores have only minor differences in functionality and they often only differ in their implementation. Therefore, workload analyses based on entire queries are suitable because the queries are processed in the same way and the comparison of query execution times is sufficiently.

To compare (systems with) different architectures, we have to estimate the performance of a given workload for both architectures. Due to the different performance of both architectures on certain operations, it seems to be obvious to analyze the operations itself. Consequently, current workload analysis approaches and estimation tools have to be adapted because these approaches process on entire queries and their structure and do not offer the opportunity to analyze operations of a query. We argue that this step is necessary because certain operations, e.g., tuple reconstructions in column stores, are not appraisable from a given query and its structure. Furthermore, a certain operation can directly influence the overall performance of a query, e.g., tuple operations. Due to heuristics, column stores will be chosen for workloads that contain an amount of aggregations. This assumption is often applicable but we cannot generalize it. If the workload contains a huge number of tuple operations besides the amount of aggregations then column stores perform poorly because they have reconstruct a lot of tuples. Tuple reconstructions are not necessary for row stores, thus we have to analyze the operations of a query to estimate the overall performance of a query correctly. With help of workload analyses based on operations, we can obtain weighted comparable estimations according to different operations even if certain operations only exist for one of the both architectures, e.g., tuple reconstruction for column stores.

4. CASE STUDY: TPC-H ON DIFFERENT ARCHITECTURES

In this section, we present our case study according to the TPC-H benchmark on a column store and a row store. We describe our study environment and introduce our assumptions for this study. Afterwards, we discuss the results of the benchmark runs.

4.1 Environment & Assumptions

Our test environment is an Ubuntu 9.10 64bit system running on Samsung X65 with a 2.2GHz dual core processor, 2GB RAM, and 2GB swap partition. We decide to use Infobright ICE[2] 3.2.2 and MySQL[3] 5.1.37 for our study.

Thereby, ICE represents the column stores and MySQL represents the row stores. Our decision is based on two main reasons. First, both DBMSs are freely available, and second both DBMSs are relatively similar. Both systems use the common MySQL kernel/management services except that they utilize different storage architectures. Of course, Infobright adds functionality to the underlying MySQL, e.g., another storage manager, but there are no other DBMSs that utilize different storage architectures and are as similar as these two DBMSs. To the best of our knowledge, there are no DBMSs more suitable to compare impacts on column and row store even if ICE is already a DBMS for DWH applications and MySQL is implemented for transactional processing (OLTP). We adjust both DBMSs configurations to the MySQL standard configuration to guarantee the comparability of the results. For both systems, no additional indexes or views are created except indexes and views that are caused by the DDL (primary key) or by the workload itself.

To exclude impacts from a poor chosen workload, we use the standardized TPC-H (2.8.0) benchmark with scale factor 1 (1GB). This benchmark is representative for the DWH

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

domain. The data (1GB) does not fit completely into RAM for MySQL standard configuration, e.g., 16MB key buffer size is much less than 1GB. We run two series of tests with the TPC-H benchmark to show that there are still application fields for row stores in the DWH domain, thus column stores do not outperform row stores at each query.

We run one series of tests with the standard TPC-H benchmark to obtain reference values. The second series of tests run with an adjusted TPC-H benchmark. We adjusted the TPC-H benchmark in the following way. We want to show that a storage architecture decision can be easily shifted by changing workloads. We modify the TPC-H benchmark, i.e., we change the number of returned attributes for each query. Hence, each query returns the result without projection (in the SELECT-statement). Listing 1 shows an exemplary adjusted TPC-H query. We have to add a GROUP BY-statements to the queries `Q6`, `Q14`, `Q17`, and `Q19` to create valid SQL statements because more than one attribute has to be processed now. We decide to group all four queries by the same attribute (l_shipdate), i.e., each query is extended by GROUP BY l_shipdate. We also apply these changes to the series of tests with standard TPC-H benchmark to guarantee the comparability.

Finally, we state that we exclude three queries from our test series. First, `Q13` is not executable on MySQL-syntax. Second, we remove `Q18` from our test series because MySQL is not able to finish the query. We abort the execution after over 21 hours. In contrast, the execution time on ICE is only 8 seconds. Third, `Q21` has an extreme execution time of 6 hours on ICE that indicates optimizer problems for this query. MySQL executes Q21 in only 2 minutes and 48 seconds.

Our results of the two test series are shown in Table 1 and will be discussed in the following section.

	Standard TPC-H		Adjusted TPC-H	
Query #	ICE	MySQL	ICE	MySQL
TPC-H Q1	00:00:25	00:00:26	00:01:18	00:00:28
TPC-H Q2	00:00:45	**00:01:31**	00:01:09	**00:01:34**
TPC-H Q3	**00:00:03**	00:00:28	**00:01:11**	00:00:27
TPC-H Q4	00:02:32	00:00:05	00:02:42	00:00:05
TPC-H Q5	00:00:03	**00:01:25**	00:01:06	**00:01:31**
TPC-H Q6	00:00:00	00:00:03	00:00:40	00:00:04
TPC-H Q7	**00:00:03**	**00:00:30**	**00:00:04**	**00:00:30**
TPC-H Q8	00:00:02	00:00:05	00:00:02	00:00:05
TPC-H Q9	00:00:05	00:00:50	00:01:09	00:00:48
TPC-H Q10	00:00:08	00:00:10	00:02:06	00:00:12
TPC-H Q11	00:00:01	00:00:00	00:00:22	00:00:01
TPC-H Q12	00:00:02	00:00:04	00:01:00	00:00:04
TPC-H Q14	00:00:01	00:00:32	00:00:43	00:00:31
TPC-H Q15	**00:00:01**	**00:00:08**	**00:00:02**	**00:00:08**
TPC-H Q16	00:00:01	00:00:09	00:00:24	00:00:12
TPC-H Q17	00:24:15	00:00:01	00:24:41	00:00:01
TPC-H Q19	**00:00:03**	00:00:00	**00:00:31**	00:00:00
TPC-H Q20	00:10:48	00:00:01	00:10:51	00:00:00
TPC-H Q22	00:19:21	00:00:01	00:19:23	00:00:01

Table 1: Comparison of Query Execution Times for ICE and MySQL on TPC-H and adjusted TPC-H

4.2 Discussion

Our study shows different impacts on the query execution time of the queries. We determine *three* different impacts within our study.

First, we cannot figure out an impact of our adjustments on the query execution of MySQL. There are only some queries that show a negligible impact, e.g., the query execution times of `Q2` and `Q5` are only increased by 3 seconds with respect to the overall query execution time above 1 minute. We expect this behavior because row stores do not have drawbacks while processing entire tuples. We assume, the final projection within a query plan has no impact on query execution time for row stores because unnecessary attribute will only be cropped from the result sets. We assume, this fact does not refer to projections on intermediate result sets during the optimization process.

Second, ICE shows an obvious impact on several queries, e.g., `Q3` and `Q19`. The increased costs for these queries show the influence of the size of processed tuples to the performance of column stores. We assume, these costs are caused by the necessary tuple reconstruction during the query execution. Hence, analyses on larger tuples within result sets can corrupt the performance of column stores. We state that large tuple sizes within a result set cannot disregarded for storage architecture decision, especially not for analyzing and reporting tools that process on huge data sets.

Third, some queries do not show an impact according to our adjustments of the TPC-H benchmark e.g., `Q7` or `Q15`. We assume that the costs for tuple reconstructions for these queries do not have a major share of total costs. The result sets and the interim results are comparatively small. Hence, there are other operations within these queries that cause the major part of the costs. Consequently, we cannot figure out general decision rules. We have to analyze the influence of single operations to the total costs of a query.

Queries `Q4`, `Q17`, `Q19`, and `Q22` have to be separately considered. The long query execution times for ICE indicate the same issue with respect to the query structure. We assume that the ICE optimizer or the ICE query processor has an issue while processing nested queries. However, our results also show that there is only a negligible impact for both systems by our adjustments for these queries.

Our assumption is confirmed that column stores cannot outperform row stores at each query, i.e., there are application fields for row and column stores in the DWH domain.

5. RELATED WORK

Several open- and closed-source column stores have been released [1, 8, 13, 18] but all systems are pure column stores and do not support any row store functionality. We state that all application fields in the DWH domain cannot be satisfied by systems that support only one architecture.

Regarding the solutions for architectural problems, there are several approaches [1, 12] available which try to reduce the drawbacks caused by the architecture, i.e., these approaches replicate data to overcome the drawbacks. We want to figure out the most suitable architecture for a given application/workload without replication mechanisms.

We need to analyze all workloads to figure out the applications fields for row and column stores. Therefore, we can utilize, adapt and extend existing approaches such as Turbyfill [15] who considers mixed workloads or Raatikainen [10] considers workload clustering analysis. In contrast, our approach needs to classify and analyze database operations itself. The approaches of Favre et al. [6] and Holze et al. [7] step in the direction of self-tuning databases, i.e., they con-

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

Listing 1: Adjusted TPC-H Query Q16

sider analyses on changing workload. We could adopt these approaches to develop an alerter in architectural manner to advice the redesign of a system.

The current research reflects new approaches to solve the update problems of OLAP applications, e.g., dimension updates [16]. Moreover, the update problem is increased according to new demands like real-time DWH [11, 17]. Thus, solutions are needed to overcome update processing problems.

6. CONCLUSION

We illustrated the major differences in storage form and functionality between column and row stores. Therefrom, we showed the difficulty to compare the performances of the two architectures or systems with different architectures. Consequently, we discussed the increased complexity of the physical design process for DWH applications, e.g., distinction between two architectures. We mentioned several approaches that show the necessity of column stores as well as row stores or at least row store functionality within the DWH domain. Afterwards, we analyzed the impact of certain operations on both architectures and figured out that current workload analysis approaches are not sufficient. The current analysis approaches cannot analyze workloads for both architectures sufficiently because we proposed that workload analysis should analyze workloads based on operations instead of entire queries. We argued that this approach can satisfy the necessity to evaluate performance of systems across different architectures.

Our study with ICE and MySQL shows that column stores cannot outperform row stores for every workload even ICE is implemented for DWH applications and vice versa MySQL is not. Hence, we confirm our assumption that there are application fields for column and row stores within the DWH domain. Thus, a framework for the selection of optimal architecture is necessary. Our study shows the necessity to adopt current approaches for more significant performance estimations. Finally, we discuss the impact of certain operations on the overall performance of a query.

7. REFERENCES

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

[2] D. J. Abadi, P. A. Boncz, and S. Harizopoulos. Column oriented database systems. *PVLDB '09*, 2(2):1664–1665, 2009.

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

[4] D. J. Abadi, D. S. Myers, D. J. DeWitt, and S. Madden. Materialization strategies in a column-oriented DBMS. In *ICDE*, pages 466–475, 2007.

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

[6] C. Favre, F. Bentayeb, and O. Boussaid. Evolution of data warehouses' optimization: A workload perspective. In *DaWaK '07*, pages 13–22, 2007.

[7] M. Holze, C. Gaidies, and N. Ritter. Consistent on-line classification of DBS workload events. In *CIKM '09*, pages 1641–1644, 2009.

[8] T. Legler, W. Lehner, and A. Ross. Data mining with the SAP NetWeaver BI Accelerator. In *VLDB '06*, pages 1059–1068. VLDB Endowment, 2006.

[9] H. 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.

[10] K. E. E. Raatikainen. Cluster analysis and workload classification. *SIGMETRICS Performance Evaluation Review*, 20(4):24–30, 1993.

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

[12] J. Schaffner, A. Bog, J. Krüger, and A. Zeier. A hybrid row-column OLTP database architecture for operational reporting. In *BIRTE '08*, 2008.

[13] D. Ślęzak, J. Wróblewski, V. Eastwood, and P. Synak. Brighthouse: an analytic data warehouse for ad-hoc queries. *PVLDB '08*, 1(2):1337–1345, 2008.

[14] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. J. O'Neil, P. E. O'Neil, A. Rasin, N. Tran, and S. B. Zdonik. C-Store: A column-oriented dbms. In *VLDB '05*, pages 553–564, 2005.

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

[16] A. A. Vaisman, A. O. Mendelzon, W. Ruaro, and S. G. Cymerman. Supporting dimension updates in an OLAP server. *Information Systems*, 29(2):165–185, 2004.

[17] Y. Zhu, L. An, and S. Liu. Data updating and query in real-time data warehouse system. In *CSSE '08*, pages 1295–1297, 2008.

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