Nr.: FIN-002-2013

Dynamic Relational Data Management for Technical Applications

Andreas Lübcke, Martin Schäler, and Gunter Saake

Workgroup Databases

Fakultät für Informatik
Otto-von-Guericke-Universität Magdeburg

Nr.: FIN-002-2013

Dynamic Relational Data Management for Technical Applications

Andreas Lübcke, Martin Schäler, and Gunter Saake

Workgroup Databases

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

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: 20.03.2013

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

Dynamic Relational Data Management for Technical Applications

Andreas Lübcke, Martin Schäler, Gunter Saake
University of Magdeburg, Germany
{andreas.luebcke,martin.schaeler,gunter.saake}@ovgu.de

Abstract: Database systems are widely used in technical applications. However, it is difficult to decide which database management system fits best for a certain application. For many applications, different workload types often blend to mixed workloads that cause mixed requirements. The selection of an appropriate database management systems is more critical for mixed workloads because classical domains with complementary requirements are combined, e.g., OLTP and OLAP. A definite decision for a database management system is not possible. Hybrid database system solutions are developed to accept the challenge. However, an optimization within these systems is not supported. We develop a decision support framework to provide application-performance estimation on a certain database management system on the one hand and to provide query optimization for hybrid database systems on the other hand. In this paper, we combine heuristics to a rule-based query optimization framework for hybrid relational database systems. That is, we target on mixed requirement support for technical applications. We evaluate our framework using standard database benchmarks. We contribute with an acceleration of query execution on hybrid database systems with our approach.

1 Introduction

Database systems (`DBSs`) are pervasively used for many technical applications. Therefore, DBSs have to manage a huge amount of different requirements for heterogeneous application domains. Although, new data management approaches are developed (e.g., NoSQL-DBMSs [CDG+06, DHJ+07], MapReduce [DG04, DG08], Cloud Computing [AFG+09, BYV08, FZRL09], etc.) to make the growing amount of data[1] manageable for new application domains. However, these approaches are solutions that are developed for special applications or need a high degree of expert knowledge. Therefore, we restrict ourselves to relational database management systems (`DBMSs`). Relational DBMSs are commonly used for highly diverse applications and beside that, relational DBMS are well-known to mainstream due to standardization (e.g., SQL [MS92, GP99]).

Relational DBMSs[2] are developed to manage data of daily business and reduce paper trails of companies (e.g., financial institutions) [ABC+76]. This approach dominates the way of data management that we know as online transaction processing (`OLTP`). Nowadays, fast and accurate revenues and expenses are not enough. A new application domain has evolved that focuses on analyses of data to support business decisions. Codd et al. [CCS93]

[1]Consider the data explosion problem [KS97, NK10].
[2]In the following, we use the term DBMS synonymously for relational DBMS.

defined this type of data analysis as online analytical processing (`OLAP`). OLTP and OLAP domain are designed for different scopes, have different limitations, and require different optimization methods [AMH08, ZNB08].

In recent years, technical applications require solutions that support tasks from both domains OLTP and OLAP [Fre97, KN11, Pla09, SB08, VMRC04, ZAL08]. Our example scenario that is neither dominated by OLAP nor OLTP, is a just-in-time planned logistics network. The data for goods as well as local administration data (e.g., staff or loading dock capacity) have to be managed. We consider the management of this data as OLTP-part of a mixed workload. Due to cost optimization, moving storage area (e.g., trucks) are scheduled in these networks. That is, the same data has to be frequently analyzed locally at branches as well as globally in the network for labor efficiency (e.g., avoid idle states/bottlenecks cause by delays) and prevention of good bottlenecks/leftovers. We have to also consider the administration and analyses for the plants itself, thus the production within plants is labor efficient too. Plants are decoupled from the logistic network but the queues are small due to limited storage space/cost efficiency. We state, frequent analyses are defined by a short period of time which inhibits a complete ETL process. The efficiency/optimization analyses define the OLAP part in our mixed workload example. Consequently, simple heuristics for typical OLTP and OLAP applications become obsolete (e.g., data warehouses without updates always perform best on column-oriented DBMSs). Nevertheless, existing approaches show limitations (e.g., focus on real-time using in-memory-DBMSs or on dimension updates), such that we argue, there is no DBMS that fits for OLTP and OLAP in all application domains. Moreover, heuristics and current approaches for physical design and query optimization only consider a single architecture[3] (e.g., design advisor [ZRL$^+$04], self-tuning [CN07] for row-oriented DBMSs (`row stores`) or equivalent for column-oriented DBMSs (`column stores`) [Idr10, SAB$^+$05]). That is, two different architectures are available, that perform best on OLTP **or** OLAP, and apply (disjunctive) design and optimization approaches. Consequently, both architectures are needed for hybrid OLTP/OLAP workloads.

We need optimization approaches that consider both architectures to support hybrid workloads. To the best of our knowledge, query optimizers that are functional across both architectures do not exist. Therefore, we present our framework in Section 2 that analyzes workloads and distributes (sub-) queries to the corresponding store in hybrid stores which support row-wise and column-wise storage. In the our framework, we combine our previous work for workload analyses [LKS11b], a storage advisor [LKS11a], and heuristics [LKS12, LSKS12]) to a monolithic approach. That is, we use a) our storage-advisor approach for OLTP/OLAP workloads and b) our heuristics for performance estimations for both architectures. Furthermore, we discuss different optimization methods for hybrid stores. In Section 3, we show the feasibility of our approach with two different underlying setups. First, we use our prototype that supports row-oriented and column-oriented storage to show feasibility in a monolithic system. Second, we use one row-oriented DBMS and one column-oriented DBMS for a two system solution as reference setup. We give an overview to related topics (Section 4) and finally conclude in Section 5.

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

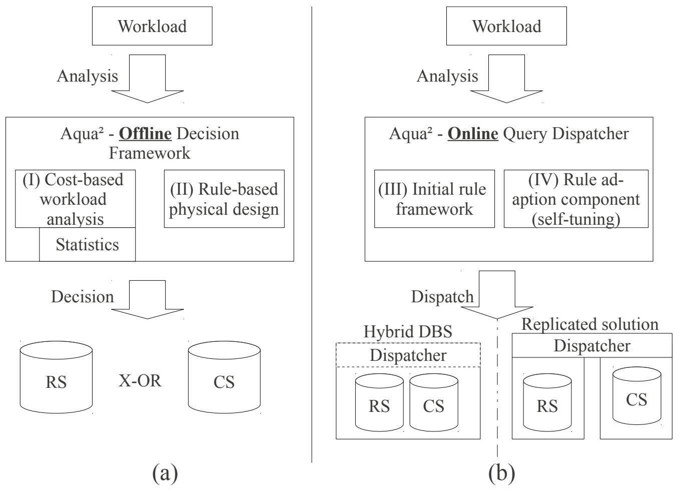
Workload
Analysis
Aqua² - Offline Decision Framework
(I) Cost-based workload analysis
(II) Rule-based physical design
Statistics
Decision
RS
X-OR
CS
(a)
Workload
Analysis
Aqua² - Online Query Dispatcher
(III) Initial rule framework
(IV) Rule adaption component (self-tuning)
Dispatch
Hybrid DBS
Dispatcher
RS
CS
Replicated solution
Dispatcher
RS
CS
(b)

Figure 1: Overview of Framework Core Components

2 Framework Overview

In this section, we first give an holistic overview of our framework and describe the ruled-based query optimization. Second, we describe the components and a rule set of our framework allowing us to dispatch queries in hybrid DBSs.

2.1 The Need for Hybrid DBS Solutions

The combination of the OLTP and OLAP domains in one application scenario raises the question of how to process such mixed workloads. Therefore, we developed a framework (cf. Figure 1) that considers performance estimation and design prediction across column- and row-oriented architecture.

In previous work, we present cost-based workload analyses to select the optimal storage architecture [LKS11a] for a given (sample) workload based on our statistic-storage and -aggregation approach [LKS11b] (cf. Figure 1(a)). On the one hand, we are able to extract the workload statistics from existing DBMSs (e.g., by explain-plan functionality). On the other hand, we can use pre-aggregated samples of workload statistics for our storage advisor. We store the statistics in our workload-representation approach [LKS11b] in either case. Our decision model [LKS11a] use the stored statistics to estimate the optimal storage architecture. We point out that the uncertainty of decision is dependent on the statistic source. However, there may be cases where statistics and sample workloads are not available. Therefore, we develop a set of heuristics for physical design to estimate the optimal storage architecture [LSKS12, LKS12]. The three above mentioned parts are summarized in Figure 1(a) as the **Offline-Decision Framework**.

We state that we cannot select an optimal architecture for mixed workloads from two predefined architectures if the workload is not dominated either by OLAP or OLTP queries (e.g., our logistic-network example). Hence, we extend a rule-based architecture-selection approach to a physical design approach for mixed workloads (cf. Figure 1(b)). For mixed workloads, we claim that hybrid DBSs and hybrid DBMSs are more efficient than a single architecture. We assume, the hybrid DBSs as well as the hybrid DBMSs support row and column store[4] functionality. Hybrid DBSs consist at least of one row and one column store; that is, both stores are managed by two different DBMSs. In contrast, a hybrid DBMS supports both architectures in one system, thus both stores are managed by one DBMS. However, we develop *two* different decision methodologies.

First, we reuse our storage-advisor functionality [LKS11a] extended with query-processing heuristics [LSKS12, LKS12] for the replicated-storage solutions. Therefore, our approach decides based on our query-optimization rules where to execute a query (allocation) or a part of it (distribution) to accelerate query execution. In short, our approach a) allocates queries to column or row store or b) distributes query parts to column and row store for parallelization or sub-query processing.

Second, we propose a self-tuning component. Therefore, we adapt the idea of self-tuning index configurations/views etc. [CN07]. To the best of our knowledge, no DBMS either supports both architectures nor have the ability to self-adapt the storage shape currently. That is, we implement a prototype for both architectures to make the first step and will gradually improve our solution. Our first prototype (introduction in Section 3.4) implements the hybrid DBMS that supports both architectures by redundant storage. That is, we currently use the rule-based query allocation/distribution approach of our offline-decision framework in our prototype. The discussed components are encapsulated in Figure 1(b) as the **Online Query Dispatcher**. In future work, we will extend our prototype to reduce redundancy by evolutionary physical design and add further approach to approximate state-of-the-art query optimization (e.g., rewrite queries in the framework).

2.2 Global vs. Local Optimization

Query optimization is ubiquitous in DBSs. As for distributed DBMS, we have to consider global and local optimization. We present a global rule-based optimization approach due to the lack of architecture-independent optimizer. To the best of our knowledge, neither rule-based nor cost-based optimizer exists for our requirements.

For each DBMS, query optimization is a basic task. Cost-based optimizers are commonly used in DBMS to achieve optimal performance. Nevertheless, cost-based optimization is very costly due to computational complexity [KPP04]. Therefore, rule-based approaches are introduced to prune the solution space for cost-based optimization [Fre87, Sel88]. In hybrid DBSs, we have to consider cost-based and rule-based optimization on two different levels. First, we perform optimization on different stores on the global level; that is, we need architecture-independent optimization. Second, the local optimization-level that is dependent on the corresponding architecture. On the local level we propose to use existing optimization approaches from row and column stores.

[4]That is, we refer to row- and column-oriented DBMSs.

Due to the lack of architecture-independent optimizer, we propose rule-based optimization for hybrid DB(M)Ss on the global level. That is, we optimize queries based on architecture-independent heuristics and rules (cf. Section 2.3) which in our case means query distribution/allocation without query rewriting. The query rewriting is locally done by either architecture. Consequently, we reduce the solution space for optimization to either architecture with global optimization. From this point, we reuse existing functionality for optimization. Local optimization is rule- or cost-based as it is common in commercial DBMSs. We state, we achieve the best performance by native optimizers (local optimization) because they are tailor-made implemented for the corresponding DBMS. Moreover, we cause minimal overhead for an additional optimization step in our approach.

Finally, an approach for global cost-based optimization is also conceivable. Such an approach can be promising for global optimization goals (e.g., minimum total time). However, we argue that spanning cost-based optimization to two different architectures significantly increases the solution space. That is, the computational cost for cost-based query plans also increases. Note, cost-based optimization can cause high computational cost with respect to one architecture, thus most commercial systems first prune the solution space by rule-based optimization. Additionally, we assume that an architecture-independent optimizer for both architectures cannot achieve competitive results compared to a tailor-made optimizer for one architecture.

2.3 Heuristics and Rule Set of Online Query Dispatcher

In previous work, we introduce a heuristic-based decision framework that dispatches queries to the optimal architecture independent of whether we apply the redundant or the hybrid solution [LSKS12, LKS12].

In previous work [Lüb10, LS10, LKS11b], we show column store performance as well as row store performance on OLTP transactions and OLAP queries. We figure out that it is promising to dispatch OLAP/OLTP workloads to different architectures. We conclude, a solution is to compute OLAP queries on column stores and OLTP transactions on row stores. Nevertheless, we observe promising results for load balancing, query distribution over both architecture, and real-time OLAP. As a result, we need a complex and extensible rule set to dispatch queries to the best architecture while considering more optimization parameter than only type (OLAP or OLTP) of the query, such as current CPU usage or I/O utilization. In Table 1, we summarize an excerpt of our rule set from the viewpoint of the column store. For example, we use column-store functionality for lookups on OLTP- and OLAP-workload parts with respect to transaction processing (e.g., ACID). Based on hardware monitoring, we are also able to figure out resource bottlenecks. That is, we distribute the workload rule-based to either architecture that consumes less of the corresponding resource. Our rule set is developed from experiences on different DBMSs and architectures. For a comprehensive overview on our rule set, underlying heuristics, assumptions, and limitations, we refer the reader to previous work [LSKS12, LKS12].

Feature	Column Store Behavior	Comment
Lookup	Fast response	True for OLTP & OLAP
Space consumption	Reduced by factor ~ 10 (HDD & RAM)	Aggressive compression (e.g., optimal compression per data type)
Data transfer	Less	More data fits in main memory, less swapping
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

Table 1: Advantages of Column Stores

3 Evaluation - Online Query Dispatcher

In this section, we evaluate the framework Online Query Dispatcher to validate our claim that online dispatching of queries performs better than an architecture decision. To do so, we evaluate both solutions (cf. Section 2.1) and discuss further application scenarios. Finally, we discuss approaches to reduce redundancy for hybrid stores.

3.1 Global Evaluation Settings

To ensure *validity* of our experimental results, we use the standardized benchmarks TPC-H [Tra10] and TPC-C [Fer06][5] (both with 10GB of data) to show the significance of our results. Additionally, we use the TPC-CH benchmark [CFG$^+$11] that simulates a mixed OLTP/OLAP workload. For the performance measurements we use a Dell Optiplex 980[6] whereby we measure CPU consumption and used I/O bandwidth for TPC-H and TPC-CH every 0.25 seconds; and for TPC-C every 0.01 seconds.

3.2 Evaluation - Redundant Solution

In the following, we evaluate the redundant solution for hybrid workloads (cf. Section 2.1). That is, we use a column and a row store keeping the data fully redundant and use a dispatcher to distribute the query workload. For our test scenario, we use an Oracle 11gR2.1 (row store) and a SybaseIQ 15.2 (column store) with each 1GB main memory available.

An overview of the execution times for the TPC-H, TPC-CH, and TPC-C benchmarks is given in Table 2 and 3. For query Q5 on TPC-CH, we obtain two results due to the fact that Sybase causes problems on modulo computation. That is, we abort the original query after more than 10 hours and restart it without modulo computation. However, these results show that the column store cannot significantly outperform the row store for each OLAP query. That is, we use our observations to improve the overall performance in redundant solutions (cf. Section 3.3). Concerning Table 3, we conclude that the row store

[5] A prepared TPC-C environment; Referring: http://www.TPC.org.
[6] QuadCore @3.33GHz, 8GB RAM running an Ubuntu 10.04LTS (2.6.32-41).

is much more efficient for OLTP transactions. We present monitoring results (CPU and I/O) for interesting (selected) queries (cf. Figures 2 to 7) that substantiate our observations concerning hybrid query-processing heuristics. Our measurements show that in general the column store consumes more CPU; whereas the row store consumes more I/O bandwidth. The exception is the OLTP computation as we discussed before (cf. Table 3).

	TPC-H		TPC-CH			TPC-H		TPC-CH	
Query	**Oracle**	**Sybase**	**Oracle**	**Sybase**	**Query**	**Oracle**	**Sybase**	**Oracle**	**Sybase**
Q1	01:40	00:58	00:24	00:55	**Q12**	**01:41**	**01:26**	02:08	00:22
Q2	01:21	00:28	00:48	00:39	**Q13**	**00:52**	**00:42**	00:02	00:02
Q3	01:52	00:47	00:28	00:38	Q14	01:24	02:16	00:28	00:09
Q4	01:38	00:35	<00:01	<00:01	Q15	01:22	00:20	02:06	00:37
Q5	**03:03**	**00:25**	06:10	00:12 (>10h)	Q16	00:09	00:07	01:33	00:20
Q6	**01:21**	**00:05**	00:23	00:03	Q17	01:22	01:14	00:46	00:06
Q7	02:12	00:06	00:25	00:02	Q18	03:51	01:05	01:28	00:45
Q8	01:47	00:21	00:01	00:06	**Q19**	**01:23**	**02:05**	00:04	00:02
Q9	03:42	02:30	00:41	00:16	Q20	01:33	00:50	00:58	01:03
Q10	02:00	00:15	01:10	00:41	Q21	04:08	02:22	02:14	01:27
Q11	**00:13**	**00:10**	01:05	00:17	Q22	00:20	00:11	00:12	00:02

Table 2: Execution Time of TPC-H & TPC-CH Queries (in mm:ss)

TPC-H query Q6 is a typical aggregation query that performs best on columns stores. Only a few columns of a large fact table have to be accessed and aggregated. In contrast, row stores access a lot of unnecessary data. We can observe that the system reach the limit of I/O bandwidth frequently (cf. Figure 3). The selectivity of selection predicates is high enough for this query that intermediate results fit into main memory. In contrast, the selectivity of selection predicates is low for TPC-H query Q5 (cf. Figure 2). In this case, data size is too large for main memory, thus both systems start swapping to disk. The swapping decreases the performance in such an extent that the row store can never be competitive even the column store has to swap some data, too. Both queries are typical representatives for worse OLAP performance of row stores. For different reasons, TPC-H query Q11, Q12, and

	Oracle			Sybase		
Tx	Time	CPU	I/O	Time	CPU	I/O
2.4 (NewOrder)	<1	73	2.99	5	65	18.15
2.5 (Payment)	<1	62	29.35	4	75	12.07
2.7 (OrderState)	**<1**	68	4.65	**40**	139	9.05
2.8 (Delivery)	<1	63	8.20	<1	198	11.51

Table 3: Execution Time (in Sec.), Ø CPU in %, and Ø I/O (in MB/s) of TPC-C Transactions

Q13 fit more to row store functionality. The row store can achieve competitive results. For query Q11 many columns have to be accessed as well as Q11 contains a sub-query. That is, the column store is slowed down by tuple reconstruction. However, the column store achieves competitive performance due to parallel processing on sub-query and top-query for predicate selection. Query Q13 shows the effect of dependent predicate selection on a fact table. Even the I/O and CPU consumption most time is low; there is no parallelization opportunity. Query 13 has a very low predicate selectivity. Both relations have to be read completely. We observe I/O waits for intermediate results from the order table; the column store computes large intermediate results which are caused by tuple reconstruction on low predicate selectivity. Finally, the performance to that query is not such bad due to the fact that the computational cost for the sub-query is not too high and no further bottlenecks occur for the column store.

TPC-H Query Q19 contains a high number of predicates that are concatenated in multiple form, thus we observe for the column store a scattered computation on low CPU load (cf. Figure 7). The I/O waits consume a lot of time that in the row store even full table scans on both tables are finally faster. We also observe this behavior for transaction 2.7. Transaction 2.7 contains correlated operations, thus a column store cannot parallelize operations on different columns due to consistency issues (e.g., update/insert cause tuple reconstruction for next lookup). We see a scattered CPU and I/O patterns (cf. Figure 8); that is, limited resources does not cause the worse performance on column stores. The peaks occur while processing lookups and some inserts on small tables. The performance of column stores decreases instantly in case a set of updates occurs that is not independent from other operations.

We conclude that certain OLAP queries are worth to dispatch them to row or column store according to environment and query. Furthermore, OLTP transactions may not dispatched to single column-store environments. We state that mixed OLTP/OLAP-workload processing is not competitive on single architecture because row stores does not achieve competitive overall performance on OLAP queries as well as OLAP-query performance significantly decreases on column stores due to consistency issues by updates and inserts. Nevertheless, we want to figure out the behavior for mixed workloads for (real) hybrid stores.

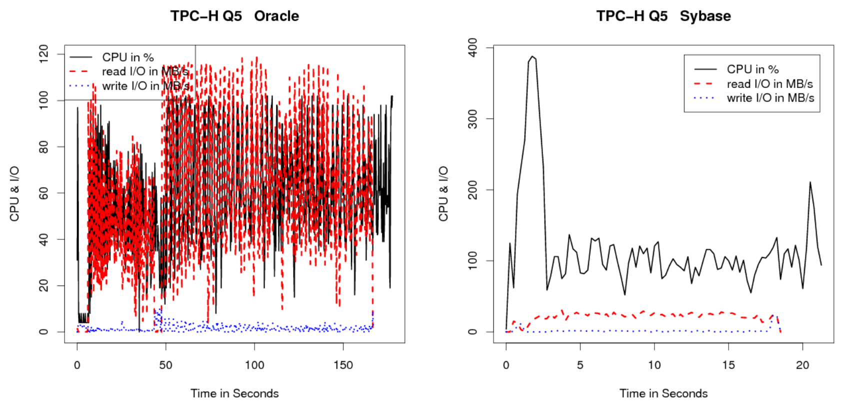

Figure 2: CPU and I/O for TPC-H Q5 on Oracle and Sybase

3.3 Additional Results for Load Balancing & Analyses on Real-time data

Beside query distribution and allocation, we also use our approach for load-balancing in hybrid DBSs with full data redundancy. Full data redundancy (in both column and row store) supports the execution of all OLAP queries and OLTP transactions on either architecture. Therefore, load balancing promises an improvement of the overall system throughput. However, we show real-time updates in column store of a hybrid DBSs are inefficient before (cf. Subsections above), thus data synchronization slow down the performance. We

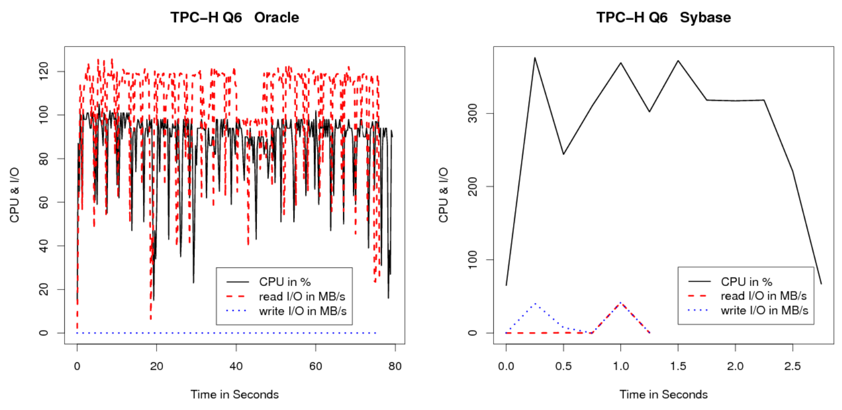

Figure 3: CPU and I/O for TPC-H Q6 on Oracle and Sybase

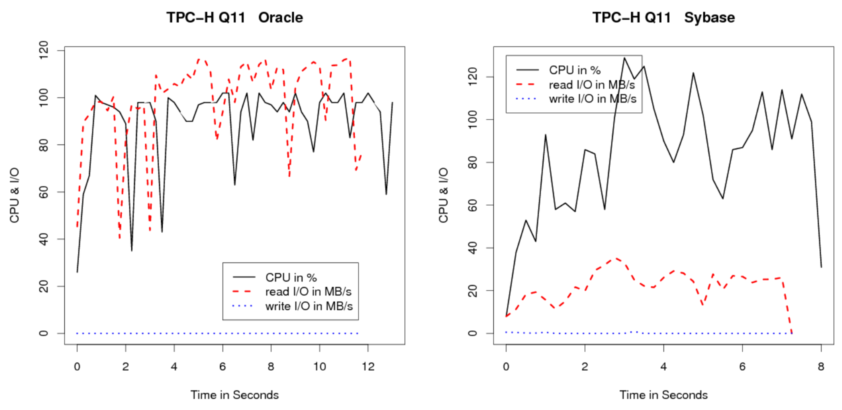

Figure 4: CPU and I/O for TPC-H Q11 on Oracle and Sybase

assume the same behavior in a hybrid DBMS but we cannot prove this observation due to the lack of implementation. Column stores show competetive performance on non-concurrent data access. Consequently, OLTP transactions only should be dispatched to column stores if they only contain lookups that do not depend on most up-to-date data or do not involve concurrent access. We propose load balancing of OLAP queries on hybrid DBSs; especially queries that perform (nearly) equivalent on both system parts (cf. Sections 2.3 and 3.2). That is, we allocate OLAP queries to the row store if the column store is overloaded. For a hybrid DBMS, the load balancing is only useful if the DBMS is set up on a distributed environment and does not share hardware for column and row store. To observe load peaks, we use monitoring functionality from the DBMS if available or from the operating system, alternatively.

We state that our global distribution approach is suitable to support (near) real-time OLAP

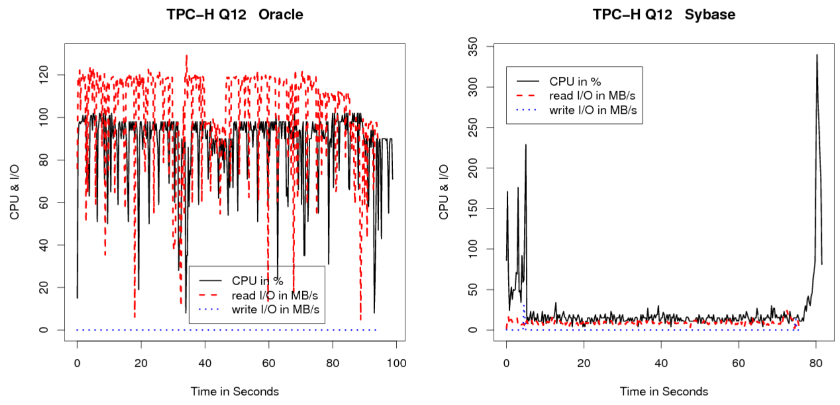

Figure 5: CPU and I/O for TPC-H Q12 on Oracle and Sybase

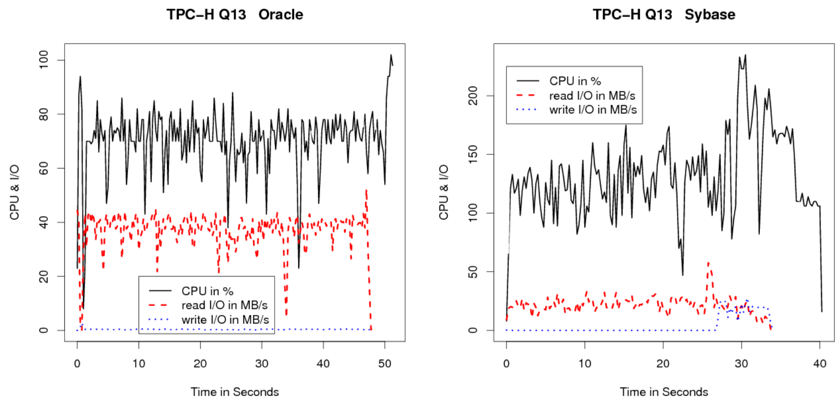

Figure 6: CPU and I/O for TPC-H Q13 on Oracle and Sybase

queries. We propose two approaches to process (near) real-time OLAP queries. First, we compute queries, which request the real-time feature, in the row-store part of hybrid DB(M)Ss. Second, we use approaches from distributed DBMSs [ÖV11] to compute real-time queries in a distributed way. Therefore, we need a) information from the application if a query requires real-time data (i.e., a real-time flag) and b) a timestamp as separator between current time and last load. We have to extend distributed-processing approaches. Current approaches distribute queries with respect to load balancing or data availability. For time-bound data, we have to adapt these approaches to use timestamps instead of data locality or monitoring information. That means, we process the major amount of data in the column store and only process the data that is not yet loaded to the column store in the row store. We assume that we achieve better performance on real-time queries with the second distributed approach.

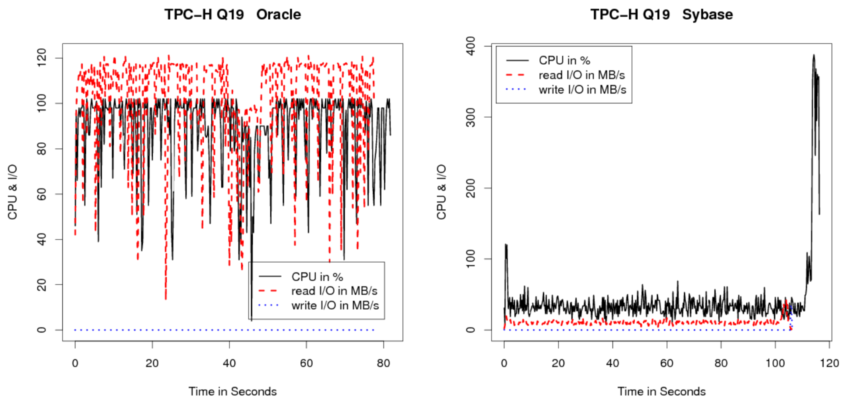

Figure 7: CPU and I/O for TPC-H Q19 on Oracle and Sybase

3.4 A Hybrid DBMS Prototype - An Implementation

In the following, we describe and evaluate our hybrid DBMS solution to dispatch mixed query workloads. For proof of concept and evaluation purposes, we implemented a hybrid DBMS based on HSQLDB[7]. HSQLDB is an open source row-oriented DBMS implemented in JAVA that is widely used, such as in Open Office[8]. We implement column-store functionality for HSQLDB based on version 2.2.5, which we briefly present subsequently.

We integrate the column store minimal-invasive to guarantee comparability of row and column store in HSQLDB. That is, we reuse existing functionality (methods and classes)

[7]http://www.hsqldb.org/

[8]http://hsqldb.org/web/openoffice.html

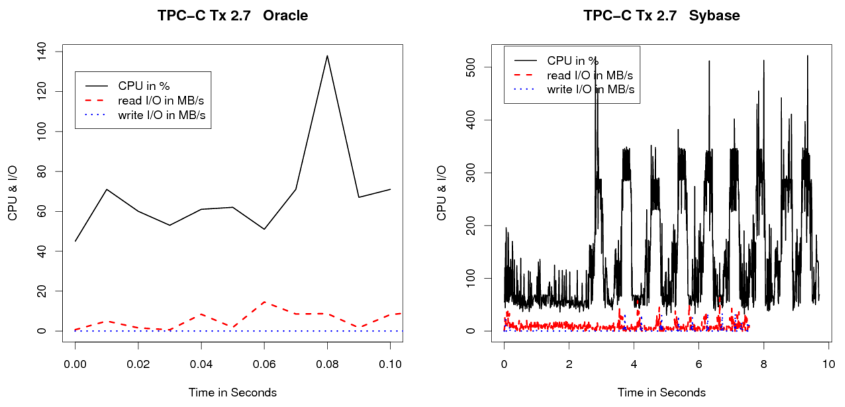

Figure 8: CPU and I/O for TPC-C Transaction 2.7 on Oracle and Sybase

as far as possible. Furthermore, we apply methods from software engineering (similar to preprocessors), allowing us to automatically remove our modifications from the compiled HSQLDB variant [Käs10] (i.e., we can, for instance, create the original row-store) used for evaluating variants of our prototype. The modifications are in detail (cf. Figure 9):

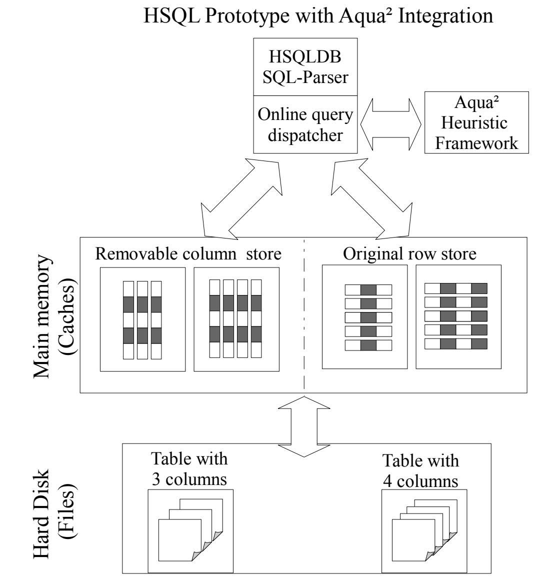

Figure 9: Hybrid HSQLDB Prototype with Framework Integration

1. **Persistence**: Column-wise storage of tables (instead of row-wise) and optimization of read-access to benefit from column-wise storage on HDD.
2. **Caching**: Redundant full in-memory caching of row and column representation for each table, used to dispatch the query workload.
3. **Dispatcher**: Online query dispatcher based on (a subset) of our heuristic decision framework for hybrid DBS solutions.

Persistence. The first modification changes the way how HSQLDB stores tables. Originally, the HSQLDB-CSV-table manager stores tables row-wise. That is, there is one file per table in which tuples are organized row-wise. With an adaptation, there is one file per column of a table. However, since tuples are read sequentially, we needed to modify the way how HSQLDB reads data from HDD, to prevent unnecessary tuple reconstructions. Our prototype reads chunks of each column and then reconstructs tuples resulting in better cache utilization and a reduced CPU overhead.

Caching. As long as there is sufficient main memory, HSQLDB tries to keep tables cached. Additionally, to the row store cache, we implement a cache for the columnar representation. Note, for evaluation purposes, we need to exclude effects caused by HDD access (especially when only one of the representations is swapped to disk) and therefore, we permit swapping of (parts of) the caches to HDD. As a result, query processing of our prototype, used for the following evaluation, works fully in *main memory*. Nevertheless, for initial loading and persistent storage of a table, we need the first addressed modification.

Dispatcher. We integrate an online query dispatcher to demonstrate the benefits of hybrid DBS solutions. It currently utilizes a subset of our heuristic decision rules. For general rule of thumb, it processes OLAP queries on the column store and OLTP queries on the row store.

3.5 Evaluation - Hybrid-Store-Prototype

To prove our claims that hybrid stores can have better performance for mixed workloads, we evaluate the previously introduced HSQLDB prototype. For this evaluation, we use the TPC-CH benchmark [CFG+11], which basically is a mixture of the analytical TPC-H and the transactional TPC-C benchmarks. Since, HSQLDB is not designed for large amount of data, we utilize a small version of TPC-CH with 100MB of data. Nevertheless, we argue that this amount of data is sufficient for proof of concept.

Evaluation setup. In our evaluation, we use the row-store and the hybrid-store of our HSQLDB prototype. All tests are on the same computer[9] (Intel Core i7, 8GB RAM, Windows 7, Oracle Java Virtual Machine JRE 1.6, maximum heap size 1.5GB). For each variant, there are two workload queues querying the database in parallel. The first workload queue performs the read (OLAP) queries, the second one issues the transactional load of the benchmark using the *read committed* isolation level. We perform all tests 120 times. We compute a robust mean value using a γ-trimming approach ($\gamma = 10\%$) to compensate outliers and ensure statistical soundness. Furthermore, we limited the queries to those that probably produce interesting insights. As a result, the test program uses TPC-CH query Q1, Q4, Q6, Q12, Q16, and Q19. Finally, we remove the changes of the benchmark after each pass of experiment from the table space by resetting it to ensure comparability of single runs.

Test results. In Table 4, we summarize the results of the average execution times for the respective TPC-CH queries for our hybrid prototype and the row-store variant of HSQLDB. Our results indicate that, except for query Q6, the hybrid prototype is faster using its column-store part. For instance, for query Q19 our hybrid prototype is more than ten times faster than the original variant. That is, our hybrid prototype significantly achieves the better overall performance for mixed workloads.

To gain further insights and explain these observations, we make additional analysis to assign execution times to single high-level operations. For our analysis, we applied *VisualVM* a Java-profiler, which is part of Oracles Java Development Kit. The results of this

[9]Note, it is another computer than we use for the TPC benchmarks.

TPC-CH	Query Q6		Query Q19	
Operations	**Row store**	**Hybrid store**	**Row store**	**Hybrid store**
Selection	72.2	54.9	13.1	16.3
Aggregation	21.1	4.7	-	-
Join	-	-	72.2	77.8
Tuple construction	-	40.1	-	<5%

Table 5: Operations with more than 5% avg. CPU Consumption

analysis for query Q6 (where row-store is faster) and query Q19 (largest benefit of hybrid) are depicted in Table 5.

Query	Row store	Hybrid store
Q1	7,183.85	6,780.90
Q4	15.11	14.62
Q6	3,134.69	5,763.29
Q12	1,027.07	29.21
Q16	33,223.67	15,591.78
Q19	24,842.66	1,967.64

Table 4: Ø Execution time (in ms) of TPC-CH queries

For query Q6, the major problem of the hybrid store (utilizing the column-store part) is tuple reconstruction to evaluate the selection predicate. Therefore, the main reason is that query processing in HSQLDB is optimized for row stores. Hence, evaluation of selection predicates is performed tuples-wise. The tuple is reconstructed and the respective attributes are accessed. Consequently, cache hierarchies can be used efficiently, since all values are stored in a tuple and thus close to each other. However, when we apply the column-store part of the hybrid, Java cannot use caches as efficient to reconstruct tuples. We are able to show the additional overhead for tuple reconstruction using the Java-profiler (cf. Table 5). Nevertheless, the difference in execution time between both variants is not too high (cf. Table 4), because aggregation itself is very efficient since Java uses the caches here efficiently (21% CPU consumption for the row store and only 4.7% for the hybrid).

For query Q19, we observe the largest benefit for our hybrid compared to the row-store. For this query, there is no tuple reconstruction necessary for two reasons. First, HSQLDB utilizes an index and does *not* scan the whole relation to find the join partners of both tables. Second, the result of the query is one aggregated attribute. As a result, the disadvantage of tuple reconstruction can be neglected here. Furthermore, the column-store part benefits from the implementation of indexes in HSQLDB. In the original row-store variant the index contains pre-computed row iterators for fast iteration over the whole indexed data that need to be updated because of TPC-CH write load. In the column store cache these iterators only contain information on how to reconstruct the rows, thus there is less computational overhead which leads to faster execution times.

3.6 Discussion on Redundancy Reduction

We show results of our approach in Table 4 and 5. In the following, we will discuss the quality of our approach. We show the usability of our approach on redundant data storage. Furthermore, we are aware that fully redundant storage cause several problems (e.g., storage space consumption, consistency). We assume that our approach is also feasible for non-redundant hybrid stores.

Reducing redundancy in hybrid stores means that we have to observe which part of database schema performs with respect to given workloads best. Therefore, we allocate parts of the schema to different stores. We introduce storage-adviser solutions for relational DBMS in previous work [LS10, LKS11a]. To reduce redundancy in our hybrid-store implementation, we utilize our adviser for performance evaluation. On a static schema design, we use our existing storage adviser following other advisers [ZRL$^+$04, BC07] for row stores. Therefore, we only replace our current heuristics on query dispatching by information on data distribution to stores. For dynamic behavior, we have to combine both the dynamic storage adviser [LS10, LKS11a] and the presented query dispatcher. The discussed approach is feasible for different granularities of distributed storage. Nevertheless, fine-grained distribution to stores cause high reorganization (on changes) and administration cost. We propose an approach on table level to observe ratio between benefit and administrative overhead.

4 Related Work

Several approaches are developed to analyze and classify workloads, e.g., [HGR09, Raa93]. These approaches recommend tuning and design of DBS, try to merge similar tasks, etc. to improve performance of DBSs. Currently, 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 pruned before analysis and performance estimations are done. With our decision model and heuristics, we propose an approach that is independent from architectural issues.

Our decision model analyzes workloads and derives workload patterns therefrom. Therefore, we analyze workload with respect to performance of database operations. However, our decision model is adaptable for existing approaches, such as Turbyfill's approach [Tur88], that considers mixed database workloads concerning disk access, to enlarge the optimization capabilities further. In contrast, the approach by Raatikainen [Raa93] considers workload classes and how to identify them based on cluster analysis. We do not aim at workload clustering into a certain number of classes instead we classify operations. We can use this and other approaches to further optimize our approach, e.g., to extend our approach to non-relational solutions. The approaches of Favre et al. [FBB07] and Holze et al. [HGR09] 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, that is we want to use evolutionary approaches to reduce redundancy in our hybrid DBMS.

For decades, rule-based query optimization is an important research field. Freytag, Lohmann

[Fre87], and Sellis [Sel88] lay the groundwork for most today's optimizers. Proof of rules and rule optimization is in research [ENR09] to the present. However, ruled-based query optimization only considers one architecture. A number of design advisors exist that are related to our work, e.g., IBM DB2 Configuration Advisor [KLS$^+$03], Microsoft's design advisor [BC07]. The IBM Configuration Advisor advises pre-configurations for databases. Zilio et al. [ZRL$^+$04, ZZL$^+$04] introduce an approach that gathers statistics directly from DBMS and utilize them to advise index and materialized view configurations. Bruno and Chaudhuri present two similar approaches [BC06, BC07] which illustrate the complete tuning process using constraints such as (storage) space threshold. However, these approaches operate on single architectures instead of hybrid DBS. Additionally, these approaches cannot be used for a hybrid DBMS because they are not independent from the underlying storage architecture.

Due to success in the analytical domain, researchers devote greater attention on column stores whereby they focus on analysis performance [Aba08, Idr10] on the one hand and on the other hand to overcome update-problems with separate storages [Aba08, Fre97]. Unfortunately, these separate storages are not fully functional for data processing; that is, they hold updates/inserts off from the analytical storage to avoid significant decrease of analyses performance. Such approaches need data synchronization between both stores sooner or later. However, a hybrid system in an architectural manner does not exist. In-memory-DBMSs are developed in recent years (e.g., Hana [Pla09], HyPer [KN11]) 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 that is also suitable for disk-based DBSs. Finally, Rösch et al. introduced a storage adviser for hybrid stores [RDHF12] that is similar to our previous work [LKS11a, LKS11b]. To the best of our knowledge, the used cost estimation approach is dependent on a specific hybrid store (e.g., SAP HANA). Furthermore, we assume that this approach is limited to hybrid DBMS; that is, one DBMS have to support row and column functionality. In contrast, our approach is also feasible for hybrid DBS (multiple DBMS used).

A related approach brings together a column store approach and the typical row store domain of 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 only hold replicates of a predefined subset of the OLTP data that is a requirement for reporting (OLAP/DWH). The row-store component gathers changes in queues, which affect the OLAP subset of data, before the columns store is updated. This approach extends the idea of separate storages (e.g., in C-Store) by ACID support. That is, the column store process OLAP queries and the row store process OLTP transactions. These approaches are not a hybrid system in our sense because they do not support different workload types, but operational reporting on real-time data (e.g., for predefined reports). Consequently, Schaffner et al. combine ACID transactions and data warehousing in one architecture. However, the proposed architecture can neither be used for load balancing between the stores nor for dynamic physical design (self-tuning) and intra-query dispatching.

5 Conclusion & Future Work

We introduce our framework to dispatch (sub-) queries to the optimal storage in hybrid stores. The framework permits to satisfy different requirements of data management for technical applications. Therefore, we combine approaches from previous work: a) a set of rules for rule-based query optimization on hybrid DB(M)Ss and b) storage-advisor functionality to select the optimal storage architecture for given workloads. Based on the given approach; we discuss the feasibility of global and local cost-based optimization concerning computational cost and existing approaches. Afterwards, we show results for a hybrid DBSs (one row- and one column-oriented DBMS) based on standardized benchmarks (TPC-C, TPC-H) and the TPC-CH benchmark and discuss further applications for our framework. To show the feasibility for hybrid DBMSs, we introduce a prototype that supports row- and column-oriented storage. Furthermore, we integrate a first prototypical implementation of our framework into a prototype. We evaluate our prototype using the TPC-CH benchmark and discuss benefits and impacts of the hybrid prototype. We show that our prototype achieve competitive results on OLTP/OLAP workloads with respect to the corresponding row-store implementation. That is, the hybrid store does not reduce the query-execution time for each query. Nevertheless, our hybrid-store implementation shows better overall performance on OLTP/OLAP workloads. We conclude that the hybrid DBS- as well as the hybrid DBMS-approach show promising results for heterogeneous technical applications. However, we figure out the following impacts. First, we observe that tuple reconstructions have a major impact to the overall performance. Second, join processing have to take advantage of architecture-specific storage, thus we reduce the overhead for both storages. Third, we discuss redundancy-reduction approaches for hybrid stores due to the fact that due to consistency requirements redundancy induces major impacts on current implementations. Consequently, current concurrency-control approaches are not sufficient for analytical queries on frequently updated data (i.e., hybrid stores).

In future work, we want to focus on completion of our rule set. Additionally, we will further integrate the framework into our prototype and further improve the column store implementation. That is, we want to improve tuple-reconstruction performance, implement join processing based on vector operations, and adapt concurrency control for hybrid stores. We will compare our prototype to hybrid DBSs based on commercial systems to figure out further optimization approaches. Finally, we want to evaluate feasibility of global cost-based optimization on hybrid DBSs. In the long view, we consider implementation self-tuning techniques into hybrid DBS solutions (e.g., evolutionary schema partitioning). Therefore, we suggest that a global concurrency control is needed which observe both stores together for conflicts.

Acknowledgements

The authors would like to thank to Ingolf Geist and Veit Köppen for reviewing the draft of this paper. 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

[Aba08] Daniel J. Abadi. *Query execution in column-oriented database systems*. PhD thesis, 2008.

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

[AFG+09] 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.

[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, 2008.

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

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

[BYV08] 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 '08*, pages 5–13, 2008.

[CCS93] Edgar F. Codd, Sally B. Codd, and Clynch T. Salley. Providing OLAP to User-Analysts: An IT Mandate. *Ann ArborMichigan*, page 24, 1993.

[CDG+06] 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 '06*, pages 205–218, 2006.

[CFG+11] Richard Cole, Florian Funke, Leo Giakoumakis, Wey Guy, Alfons Kemper, Stefan Krompass, Harumi Kuno, Raghunath Nambiar, Thomas Neumann, Meikel Poess, Kai-Uwe Sattler, Michael Seibold, Eric Simon, and Florian Waas. The mixed workload CH-benCHmark. In *DBTest '11*, pages 1–6, 2011. Article 8.

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

[DG04] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *OSDI '04*, pages 137–150, 2004.

[DG08] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *CACM*, 51(1):107–113, 2008.

[DHJ+07] 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 '07*, pages 205–220, 2007.

[ENR09] Hicham G. Elmongui, Vivek Narasayya, and Ravishankar Ramamurthy. A framework for testing query transformation rules. In *SIGMOD '09*, pages 257–268, 2009.

[FBB07] Cécile Favre, Fadila Bentayeb, and Omar Boussaid. Evolution of Data Warehouses' Optimization: A Workload Perspective. In *DaWaK '07*, pages 13–22, 2007.

[Fer06] Diego R. Llanos Ferraris. TPCC-UVa: an open-source TPC-C implementation for global performance measurement of computer systems. *SIGMOD Record*, 35(4):6–15, 2006.

[Fre87] Johann Christoph Freytag. A rule-based view of query optimization. *SIGMOD Rec.*, 16(3):173–180, December 1987.

[Fre97] Clark D. French. Teaching an OLTP database kernel advanced datawarehousing techniques. In *ICDE '97*, pages 194–198, 1997.

[FZRL09] Ian T. Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. Cloud Computing and Grid Computing 360-Degree Compared. *CoRR '09*, abs/0901.0131, 2009.

[GP99] Peter Gulutzan and Trudy Pelzer. *SQL-99 complete, really*. CMP Books, Lawrence, USA, 1st edition, 1999.

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

[Idr10] Stratos Idreos. *Database Cracking: Torwards Auto-tuning Database Kernels*. PhD thesis, 2010. Adviser: Martin L. Kersten.

[Käs10] Christian Kästner. *Virtual Separation of Concerns: Toward Preprocessors 2.0*. Phd thesis, 2010.

[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.

[KN11] Alfons Kemper and Thomas Neumann. HyPer: A hybrid OLTP&OLAP main memory database system based on virtual memory snapshots. In *ICDE '11*, pages 195–206, 2011.

[KPP04] Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack Problems*. Springer-Verlag, 2004.

[KS97] Henry F. Korth and Abraham Silberschatz. Database Research Faces the Information Explosion. *CACM*, 40(2):139–142, 1997.

[LKS11a] Andreas Lübcke, Veit Köppen, and Gunter Saake. A decision model to select the optimal storage architecture for relational databases. In *RCIS*, pages 1 –11, 2011.

[LKS11b] Andreas Lübcke, Veit Köppen, and Gunter Saake. Workload representation across different storage architectures for relational DBMS. In *Proceedings of the GI-Workshop on Foundations of Databases*, pages 79–84, 2011.

[LKS12] Andreas Lübcke, Veit Köppen, and Gunter Saake. Heuristics-based Workload Analysis for Relational DBMSs. In *UNISCON*, 2012.

[LS10] Andreas Lübcke and Gunter Saake. A Framework for Optimal Selection of a Storage Architecture in RDBMS. In *DB&IS*, pages 65–76, 2010.

[LSKS12] Andreas Lübcke, Martin Schäler, Veit Köppen, and Gunter Saake. Workload-based heuristics for evaluation of physical database architectures. In *DB&IS*, pages 3–10, 2012.

[Lüb10] Andreas Lübcke. Challenges in workload analyses for column and row stores. In *Proceedings of the GI-Workshop on Foundations of Databases*, volume 581, 2010.

[MS92] Jim Melton and Alan R. Simon. *Understanding the new SQL: A complete guide*. Morgan Kaufmann, San Francisco, USA, 1st edition, 1992.

[NK10] Ina Naydenova and Kalinka Kaloyanova. Sparsity Handling and Data Explosion in OLAP Systems. In *MCIS '10*, pages 62–70, 2010.

[ÖV11] M. Tamer Özsu and Patrick Valduriez. *Principles of Distributed Database Systems*. Springer, 3rd edition, 2011.

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

[Raa93] Kimmo E. E. Raatikainen. Cluster Analysis and Workload Classification. *SIGMETRICS PER*, 20(4):24–30, 1993.

[RDHF12] Philipp Rösch, Lars Dannecker, Gregor Hackenbroich, and Franz Färber. A Storage Advisor for Hybrid-Store Databases. *PVLDB*, 5(12):1748–1758, 2012.

[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, 2005.

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

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

[Sel88] Timos K. Sellis. Multiple-query optimization. *ACM Trans. Database Syst.*, 13(1):23–52, March 1988.

[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.

[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.

[ZAL08] Youchan Zhu, Lei An, and Shuangxi Liu. Data Updating and Query in Real-Time Data Warehouse System. In *CSSE '08*, pages 1295–1297, 2008.

[ZNB08] Marcin Zukowski, Niels Nes, and Peter A. Boncz. DSM vs. NSM: CPU performance tradeoffs in block-oriented query processing. In *DaMoN '08*, pages 47–54, 2008.

[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, 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.