Icarus: Towards a Multistore Database System

Marco Vogt   Alexander Stiemer   Heiko Schuldt
Department of Mathematics and Computer Science
University of Basel, Switzerland
{*firstname.lastname*}*@unibas.ch*

Abstract—The last years have seen a vast diversification on the database market. In contrast to the "one-size-fits-all" paradigm according to which systems have been designed in the past, today's database management systems (DBMS) are tuned for particular workloads. This has led to DBMSs optimized for high performance, high throughput read/write workloads in online transaction processing (OLTP) and systems optimized for complex analytical queries (OLAP). However, this approach reaches a limit when systems have to deal with mixed workloads that are neither pure OLAP nor pure OLTP workloads. In such cases, multistores are increasingly gaining popularity. Rather than supporting one single database paradigm and addressing one particular workload, multistores encompass several DBMSs that store data in different schemas and allow to route requests on a per-query level to the most appropriate system. In this paper, we introduce the multistore ICARUS. In our evaluation based on a workload that combines OLTP and OLAP elements, we show that ICARUS is able to speed-up queries up to a factor of three by properly routing queries to the best underlying DBMS.

Keywords-Multistore databases, mixed OLTP and OLAP workload, query routing.

I. INTRODUCTION

Since the introduction of relational database management systems (DBMSs) in the late 1960's, they have served – with few exceptions of niche products – as commonly accepted, generic platforms for data management in a large variety of applications. However, with the advent of earmarked DBMSs and NoSQL data stores a few years ago, the landscape has changed. Nowadays, DBMSs are either optimized for complex analytical queries in *Online Analytical Processing* (OLAP) applications or for short read and write transactions with high throughput in *Online Transaction Processing* (OLTP) settings. These earmarked DBMSs are helpful if applications either come with a pure OLTP or a pure OLAP workload, but they turn out to be suboptimal for mixed workloads, containing aspects of both worlds. Moreover, in applications demanding a large degree of flexibility, schemaless databases such as key-value stores have become popular. Other applications rather demand for graph databases as they need to store and analyze large graph data structures. As a consequence, the "one-size-fits-all" approach of traditional DBMSs is no longer suitable [1]. With the advent of new storage technologies and DBMS architectures such as solid state disks or main memory databases, the situation has become even more complex.

Different storage technologies also come with different properties regarding memory bandwidth, capacity, latency, and throughput. At the same time, there is a significant trade-off between bandwidth, latency, throughput, and capacity on one side and price per gigabyte on the other side.

The Cloud with its nearly unlimited capacity and rather moderate resource costs has led to a new momentum in the discussion of this trade-off. Rather than optimizing a DBMS for one of these properties and focusing only on either OLAP or OLTP, systems tend to support several different storage technologies and different DBMS architectures at the same time. This has led to so-called *multistores* or *polystores*. These systems operate different technologies and DBMSs with different architectures in parallel. When requests come in, they route them on a per-query basis to the best suited DBMS, i.e., to the DBMS that provides the highest throughput and/or the smallest response time.

So far, the selection of an appropriate storage medium strongly had to consider the data to be stored and the expected workload, especially the access characteristics (e.g., read-only, mostly reads, balanced read/write ratio, high update frequencies), the frequency in which a data tuple is accessed, and also the size and amount of data stored in the database itself. Failure in properly assessing these criteria has led to deployments that are far from optimal. With the concept of multistore database systems, the selection of storage media and database architecture is no longer necessary at design time. Rather, several different systems are used jointly, and the optimization (query routing) is done at run-time by the multistore system. However, the major challenge in multistores is to have proper characterizations of the expected workload and, at the same time, a mapping from the expected queries to the best suited data store.

In this paper, we introduce ICARUS, a prototype of such a multistore system. ICARUS classifies queries according to their access pattern into query classes and compiles routing tables that map query types to data stores. For this, it probes the underlying stores in order to update the routing information. We present the design and implementation of ICARUS. Moreover, we show in an evaluation using *semi-analytical query* loads, i.e., query loads that combine OLAP and OLTP, that it outperforms the individual DBMSs by a factor of up to three as the latter are optimized for either workload but cannot cope well with such mixed workloads.
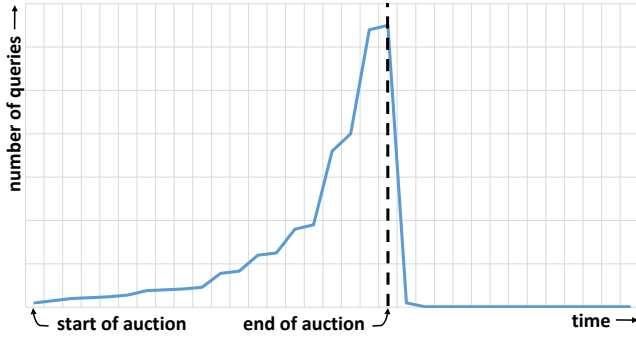
Figure 1: Average number of DB queries per auction.

Figure 2: Average number of auctions in Gavel.

The contribution of this paper is threefold: Firstly, we present the multistore database system ICARUS which automatically routes queries to the appropriate data store. Secondly, we propose a benchmark designed for mixed OLTP and OLAP workloads, based on the concrete use case of an auctioning company. Thirdly, we show the performance of ICARUS based on this benchmark and the speedup gained in comparison with the underlying data stores.

The remainder of this paper is structured as follows: Section II provides a motivating example for an application that comes with mixed workloads. In Section III we introduce the workload characterization required for query routing. The system model and architecture of ICARUS is outlined in Section IV. Our new benchmark for mixed workloads is introduced in Section V and Section VI presents details of the evaluation of ICARUS. Section VII discusses related work and Section VIII concludes.

II. Motivating Scenario

Assume, as an example, "Gavel", an online auction house which runs millions of auctions per day. Because Gavel gets 1 % of the price of each item sold, their profit depends on the number and prices of items sold. In order to maximize their profit, Gavel invests in advertising their auctions which, in turn, requires sophisticated analyzes of their users' behavior and other statistics (e.g., price of the auctions, the users' biding history, or the users' history of visited auctions).

Such analytical database queries need to run on the latest data making an offline database for the analytics infeasible. This is due to the characteristic feature of auctions being most interesting right before their end and which leads to "Number of Queries per Time" charts like Figure 1. If we further assume that Gavel is currently only available in Europe, the majority of customers visit their website between 4 p.m. and 11 p.m. This correlates to the number of database queries shown in Figure 2.

In contrast to these analyzes, the other queries ($\sim 80\%$) are mostly transactional and belong to one of these three groups: i.) select details of an auction, ii.) create a list of auctions, iii.) bidding. All three are typically short running
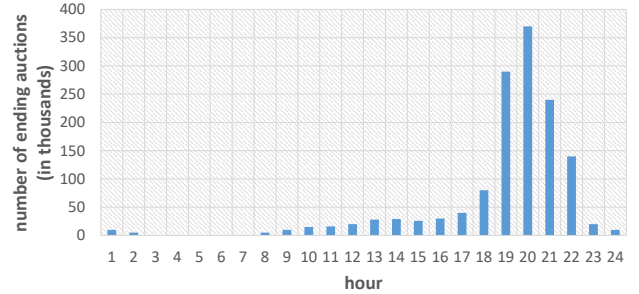
transactions and are slowed down by the analytical queries required for the advertisement, if submitted to a row-store DBMS. If, however, a column-oriented DBMS was used, this would lead to a significant improvement of the analytical queries at the prize of decelerated transactions. Hence, the workload can be considered *semi-analytical* as it mixes OLAP and OLTP elements and is neither well supported as a whole by only row-store nor by only column-store DBMSs.

Gavel's management is now wondering whether the newly proposed concepts of *multistores* can help to combine the advantages of both DBMS types to gain fast execution times for the transactional as well as for the analytical queries – by using several systems and by deciding per query where to route it to.

III. Workload Characterization

DBMSs can face various types of workloads with different characteristics. Depending on these characteristics, a query can be time-consuming when executed on one DBMS while being very efficiently on another DBMS. Query execution time can thus vary between several orders of magnitude.

Database workloads can either be characterized as OLTP or as OLAP. The transition between these two classes is fluid. While OLTP workloads are normally dominated by simple, short running transactions only reading and modifying few records in the database, OLAP workloads are associated with long running read-only queries which process huge amounts of data.

The drawback of this classification is that there is a huge amount of queries which do not fit in one of the categories. The queries in this blurred transition between OLTP and OLAP bare a great potential for optimization. It is also the group of queries with a high demand for optimization, because they can have a huge impact on the overall performance of an application.

We therefore propose a third class of workload named *semi-analytical*. Thus, we distinguish the following three workload classes:

Transactional: These queries only process few records to read, update, or delete their content or they insert new

```
SELECT * FROM user u WHERE u.id=5348765;

UPDATE user SET password = 'PASSWORD'
WHERE id = 5348765;

INSERT INTO bid(amount, timestamp, user,
    auction)
VALUES (12.43, NOW(), 5348765, 87523);
```

Code 1: Transactional Examples: Query a user, setting a password and placing a bid.

```
SELECT a.id, a.title, a.end_date
FROM auction a
WHERE a.category = '34'
  AND a.end_date > NOW()
ORDER BY end_date desc LIMIT 100;
```

Code 2: Semi-Analytical Example: The 100 next ending auctions of a category.

```
SELECT u.last_name, u.first_name, (sum(a.
    price) * 0.01) as revenue
FROM user u,
  (SELECT a.user as user , MAX(amount) as
      price
    FROM auction a, bid b
    WHERE b.auction = a.id
    GROUP BY a.id, a.user
  ) as a
WHERE a.user = u.id
GROUP BY u.id, u.last_name, u.first_name
ORDER BY revenue desc LIMIT 100;
```
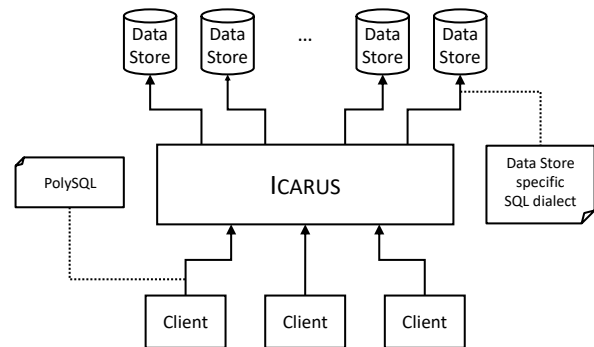
Code 3: Analytical Example: Top 100 seller by revenue.

Figure 3: Logical view on the deployment of ICARUS.

records. Transactional queries only have simple conditions and only very few joins. Code 1 shows a few examples.

Semi-Analytical: Simple analytical queries like aggregations or counts on the whole table or on groups defined by simple conditions, like in Code 2. Also transactional queries having complex conditions belong to this category. Furthermore, update and delete operations that process more than a few records belong to this group, too.

Analytical: Complex and potentially long-running queries that process a huge amount of records. In contrast to the *semi-analytical* queries, the queries in this class operate on many or all records in the database based on complex conditions. An example can be found in Code 3.

Workloads are characterized by the mix of these three transaction types. Moreover, the work reported in this paper is based on the following assumption: *In a common enterprise application, there is only a relatively small amount of queries that differ in their structure. Queries with the same structure are similar in their complexity and therefore also have similar execution times.* Hence, we assume that there is only a finite number of queries that differ in their structure. For SQL, this means that we can group all queries which share the same operation (SELECT, UPDATE, etc.), that operate on the same columns and tables, and that use the same operators and functions.

We define a *Query Class* as a set of queries and statements which are, according to the assumption, similar in their structure and execution time.

IV. SYSTEM MODEL AND ARCHITECTURE

Figure 3 shows a logical view of a deployment of ICARUS: Multiple clients are connected to ICARUS using PolySQL (ICARUS' SQL dialect) as query language. ICARUS itself is connected to multiple *data stores* (various DBMSs with usually different storage engines) and communicates with them using their specific query language or access method. When a query is submitted to ICARUS, it determines "the best" data store where the query should be executed. This decision is based on a *routing table* ICARUS maintains. Finally, ICARUS compiles PolySQL into the specific query language or access method of the underlying data store.

ICARUS itself consists of three main building blocks: The *Core*, the *Executor* and the *Analyzer*:

Core: contains the *Client Interface*, the *Router* and the *SQL Parser*. The *Client Interface* accepts the queries and provides a web interface for executing jobs and for monitoring the status of the system. The *Router* is responsible for sending the jobs to the *Executor(s)* and for routing the results back to the *Client Interface*.

Executor: handles the communication with a data store. For each data store there is at least one *Executor*.

Analyzer: is responsible for calculating the routing table based on the information on previous executions.

ICARUS is designed to allow the *Core* and the *Executor* to be executed on different nodes. Furthermore, each of these two building blocks can be executed multiple times. This allows load-balancing and increases the fault tolerance.

The routing table lists all known *Query Classes* together

Query Class	Data Store A	Data Store B	Data Store C
1	60 %	40 %	-
2	-	50 %	50 %
3	-	-	100 %
4	33 %	33 %	34 %

Table I: Exemplary routing table. Here, 60 % of all *Query Class 1* queries are executed by *Data Store A* and the remaining 40 % by *Data Store B*, whereas *Query Class 3* is only executed by *Data Store C*.

with the percentage of `SELECT` queries that should be routed to this store. `INSERT` and `UPDATE` statements have to be executed on all data stores and are therefore not listed in the routing table. Table I shows an example for such a routing table. The probability for having a query forwarded to a certain data store solves two problems: First, a certain parallelism can be exploited. Even if a data store does not execute the query as fast as the fastest, its capacity can be used to maximize throughput. Second, by executing the query also on other data stores, ICARUS is able to adapt to changes in the execution time. For example, the fastest data store can get a lesser probability if its execution time happens to be greater than the second fastest data store.

The *Analyzer* uses three thresholds to build the routing table. First, the *Short running to long running threshold* ($SRLRT$) (in milliseconds) specifies for one *Query Class*, with respect to the respective fastest data store for that *Query Class*, if it is either short or long running (cf. OLTP vs. OLAP). For example, if *Query Class* QC1 needs on average 63 ms to be executed on the – for this *Query Class* – fastest data store and $SRLRT$ is specified to be 1000 ms it is considered as "short running" whereas *Query Class* QC2 having an average execution time of 1652 ms on its fastest store is considered as "long running". The averages are calculated over a certain period of time using the simple moving average (SMA) algorithm.

Second and third, the two *similar thresholds*, namely *Short running similar threshold* ($SRST$) and *Long running similar threshold* ($LRST$) are required to allow the execution of *Query Classes* not only on their respective fastest data store. These thresholds specify up to which execution times data stores are considered for the calculation of the percentage values. $SRST$ is applied to the previously as "short running" classified *Query Classes* and $LRST$ is applied to the "long running" *Query Classes*, respectively. The values of $SRST$ and $LRST$ are specified as percentage values, more precisely as "additional percentage values". The resulting maximum execution time can be calculated as follows ($SRST$, $LRST$ respectively):

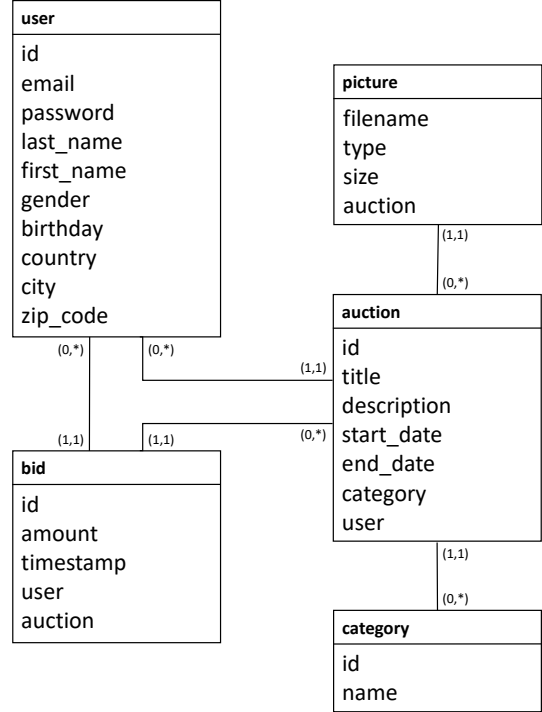$$\text{MaxExecTime} = \text{FastestExecTime} \times (1 + SRST/100)$$

Figure 4: Database Schema for the Gavel Benchmark.

Extending the previous example: Let *Query Class* QC1 require on average 63 ms on data store A, 88 ms on data store B and 547 ms on data store C. Also let $SRLRT = 1000$ ms and $SRST = 100$ % (allow up to twice the fastest execution time). Then data stores which lead to an execution time less than 126 ms are considered for the subsequent calculation of the percentage values. Data stores exceeding the 126 ms are ignored (here, data store C). If $SRST = 0$, then only the fastest data store is considered.

V. GAVEL BENCHMARK

For evaluating the ICARUS system, we have devised a benchmark from the Gavel scenario outlined in Section II.

The database schema can be found in Figure 4. For each table – except for the table `picture` – there is a primary key index created on the `id` attribute. The query templates for the Gavel benchmark can be found on Github [1]. The missing information (e. g., the IDs) are randomly generated. The number of queries derived from each of the templates are taken as parameters. The order of the queries is randomly determined.

For the benchmark, the database is prepared with test data using the parameters shown in Table II. The five tables of the Gavel benchmark schema contain around 90 million records and have – exported as CSV files – a size of 4.5 GB. With

[1] https://gist.github.com/dbisUnibas/370dfa0a88865e9606fdbb026282e05e

Gavel Dataset	
Number of categories	35
Number of users	1 500 000
Number of auctions	750 000
Length of an auction title	Between 2 and 8 words
Length of auction description	Between 5 and 40 sentences
Number of bids per auction	Between 30 and 200
Pictures per auction	Between 1 and 6
Duration of an auction	10 days
Maximum age of an auction	4 years

Table II: Properties of the Gavel test data. On average a "word" contains 6.1 characters and a "sentence" 8.3 words.
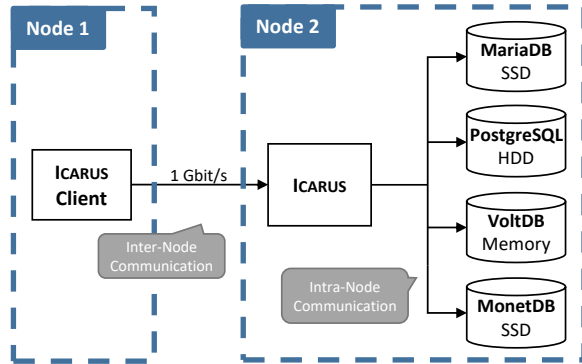

Figure 5: Overview of the evaluation setup with two nodes.

4.3 GB, the majority of the data is held by the `bid` table. The size of the data in the other tables are: 695 MB for `auction`, 157 MB for `user`, 124 MB for `picture`, and finally 4 KB for `category`.

VI. EVALUATION

The evaluation of ICARUS has two goals: The first goal is to find the best configuration of ICARUS and thereby to show the influence of the various parameters of the ICARUS system. The results of this parameter evaluation can be found in Section VI-C. The second goal is to show that the standard benchmarks TPC-C and TPC-H are not well suited for multistore DBMSs like ICARUS and that ICARUS increases the overall performance if compared against its underlying data stores for the better suited Gavel benchmark which combines elements from OLTP and OLAP. These results are discussed in Section VI-D.

A. Evaluation Environment

The evaluation environment depicted in Figure 5 consists out of two nodes: the node running ICARUS CLIENT performing the benchmarks (*Node 1*) and the node on which

Gavel Benchmark	
Total number of queries	10000
Percentage of Query Types	
Read-only queries	75 %
Update queries	10 %
Insert queries	5 %
Semi-Analytical queries	9.8 %
Analytical queries	0.2 %
Read-only Queries	
Get a random auction	20 %
Get a random bid	30 %
Get a random user	30 %
Get all bids on a random auction	7 %
Get the currently highest bid on a random auction	7 %
Search for an auction with a random search string	6 %
Update Queries	
Change password of a user account	50 %
Change an auction	50 %
Insert Queries	
Adding a new auction	50 %
Bid on an auction	50 %
Semi-Analytical Queries	
Number of running auctions per category	10 %
The 100 next ending auctions of a category	10 %
Total number of auctions	40 %
Total number of bids	40 %
Analytical Queries	
Top 10 auctions of a category by number of bids	25 %
Top 100 seller by number of auctions	25 %
Top 100 seller by revenue	25 %
Top 10 cities by number of customers	25 %

Table III: Properties of the Gavel benchmark.

ICARUS and the data stores are deployed (*Node 2*). Table IV summarizes the hardware specifications of the two nodes.

As a general note: If not mentioned otherwise, there is a warm-up phase executed before the main evaluation run.

B. Data Stores

The four data stores used by ICARUS are: i.) MariaDB [2] version 10.0.29 as a relational DBMS with the *MyISAM* storage engine with a dedicated SSD as storage device, ii.) PostgreSQL [3] version 9.5.5 as a relational DBMS with a dedicated HDD as storage device, iii.) VoltDB [4] Community

[2] https://mariadb.org/
[3] https://www.postgresql.org/
[4] https://www.voltdb.com/

Hardware Specifications	
CPU	Intel Xeon E5-2630 v4
RAM	Kingston Value RAM (4 x 32GB, DDR4-2133)
OS SSD	ADATA SP600 (256GB, M.2 2242)
SSD	Samsung 850 EVO Basic (4 x 500GB, 2.5 Inch)
HDD	WD Red (2 x 4000GB, 3.5 Inch)

Table IV: Hardware specification of the evaluation setup.

Figure 6: Execution time of the Gavel benchmark for different numbers of *Client Threads*.

Edition version 6.6.4 as a relational in-memory DBMS with the *Temp Table Size* set to 20 GB, the *Query Timeout* set to 300 seconds and where every table is of type "replicated", and finally iv.) MonetDB [5] version 1.7 as a column-oriented DBMS working on a dedicated SSD. Please note that MonetDB has an *Optimistic Concurrency Control* [2]. To keep MonetDB consistent with the other stores, we allow only one writing transaction at a time being executed on the MonetDB store, independent of the number of *Executor Threads* configured.

C. Parameter Evaluation Results

Before the actual performance evaluation, we validate important parameters of ICARUS using the Gavel benchmark. These parameters are: 1) the number of clients querying ICARUS determining the number of clients needed to max out ICARUS' capacity, 2) the used *Analyzer* to compare the learned with a supervised approach, and 3) the used *Router* to verify the query class routing approach.

1) Number of Clients: The ICARUS CLIENT supports running multiple *Client Threads* in parallel. This allows to evaluate how ICARUS performs under concurrent workloads. These *Client Threads* take their queries from a query list and execute them. This allows to compare the runtime of the benchmark as measurement on how well ICARUS performs with this number of continuous requests.
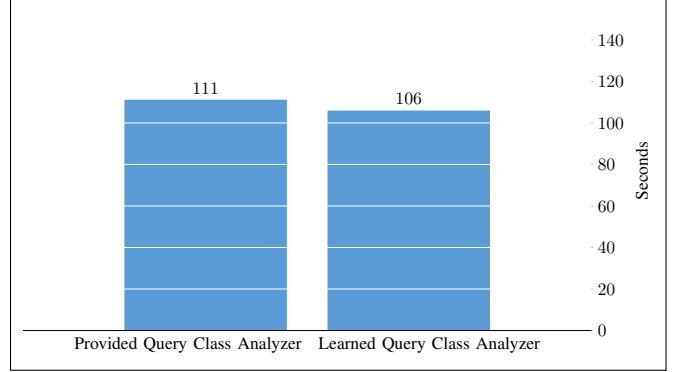
[5] https://www.monetdb.org/

Figure 7: Comparison of the *Provided Query Class Analyzer* and the *Learned Query Class Analyzer*.

The result in Figure 6 depicts the execution times of the Gavel benchmark for different numbers of *Client Threads*. The evaluation shows that the system on our hardware has a massive performance increase for the first three threads. This has two reasons: First, if there is more than one client thread, ICARUS does not always idle while the ICARUS CLIENT analyzes the result of an executed query. And second, multiple read-only queries can be executed on different stores simultaneously, leading to a better resource utilization of the whole system.

2) Analyzer: The *Analyzer* is responsible for building the routing table based on information about previous executions. It also provides the logic for assigning *Query Classes* to incoming jobs. There are currently two implementations of ICARUS' *Analyzer* interface. The *Provided Query Class Analyzer* uses the *Query Class* provided with the request for clustering similar queries. Instead, the *Learned Query Class Analyzer* clusters the queries based on the accessed columns, tables and operators and automatically assigns artificial *Query Classes* to this groups. Because the provided *Query Classes* are correctly assigned for each query in the benchmark, the *Provided Query Class Analyzer* can be seen as a reference.

In Figure 7 the results of the comparison of the two *Analyzer* implementations are depicted. They show that for the Gavel benchmark the *Learned Query Class Analyzer* is not inferior in quality compared to the *Provided Query Class Analyzer* and therefore confirm our approach.

3) Router: This evaluation compares the two implementations of ICARUS' *Router* interface. While the *All Stores Router* simply sends all jobs to all stores, the *Query Class Router* forwards the jobs based on the routing table generated by the *Analyzer*. For this benchmark we use the *Learned Query Class Analyzer*.

Figure 8 depicts the results of the evaluation. As expected, sending the queries to all stores results in a worse performance compared to an intelligent routing.
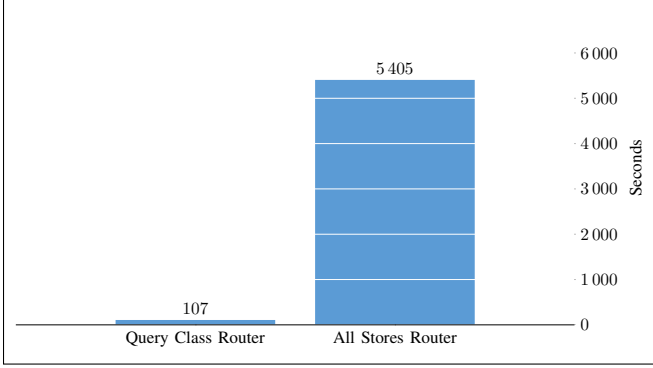
Figure 8: Comparison of the *Query Class Router* and the *All Stores Router*.

D. Performance Evaluation Results

In this section we first show on the basis of the routing tables that the TPC-C and TPC-H benchmarks are not well suited for multistore DBMSs. Conversely, this also means that multistore DBMSs do not provide (much) gain for pure TPC-H or pure TPC-C workloads since then, just one of the optimized stores underneath can be used, relinquishing the additional flexibility of the multistore. Following this, we compare ICARUS against its four underlying data stores using the Gavel benchmark and show if ICARUS is worth its costs. Additionally, we show the speedup gained for deploying ICARUS for different read / write ratios.

For the following performance evaluations ICARUS uses the *Learned Query Class Analyzer* together with the *Query Class Router* and is queried by seven *Client Threads*.

1) TPC-C and TPC-H: The routing table of TPC-C (see Table V) shows that no query is executed using the column-oriented DBMS MonetDB, whereas for the TPC-H benchmark all queries are executed using MonetDB (Table VI). The reason is that, per design, both benchmarks do not generate diverse, i. e., mixed workloads. However, multistore DBMSs like ICARUS require benchmarks generating mixed workloads to fully utilize different data stores so that they can benefit from that diversity. For homogeneous workloads, the overhead added by multistore DBMSs leads to a worse performance compared to those data stores which are optimized for that specific workload.

As a side note: All values marked in boldface font in the TPC-C routing table (Table V) would be 100 % if the threshold $SRST = 0$.

2) Gavel Benchmark: In contrast to the routing tables for TPC-C and TPC-H, the table for the Gavel benchmark depicted in Table VII shows a more diverse workload. Therefore, the Gavel benchmark is better suited for multistore DBMSs allowing them to utilize the various data stores. Since, as mentioned in Section IV, the routing table only contains read-only *Query Classes* and two of

Query Class	MonetDB	PostgreSQL	VoltDB	MariaDB
1	-	**51 %**	-	49 %
2	-	31 %	**38 %**	33 %
3	-	33 %	**35 %**	32 %
4	-	**36 %**	32 %	32 %
5	-	44 %	**56 %**	-
6	-	49 %	-	**51 %**
7	-	32 %	33 %	**35 %**
8	-	-	-	**100 %**
9	-	31 %	30 %	**39 %**
10	-	36 %	**64 %**	-
11	-	**51 %**	-	49 %
12	-	**36 %**	31 %	33 %
13	-	32 %	32 %	**36 %**
14	-	**35 %**	32 %	33 %
15	-	**51 %**	-	49 %
16	-	**38 %**	29 %	33 %
17	-	32 %	30 %	**38 %**
18	-	-	-	**100 %**
19	-	30 %	29 %	**41 %**
20	-	32 %	30 %	**38 %**
21	-	**39 %**	30 %	31 %
22	-	30 %	30 %	**40 %**
23	-	**38 %**	31 %	31 %
24	-	**38 %**	31 %	31 %
25	-	31 %	30 %	**39 %**
26	-	32 %	30 %	**38 %**
27	-	31 %	30 %	**39 %**
28	-	32 %	30 %	**38 %**

Table V: Routing table for TPC-C and $SRST = 100$. The percentage values marked in boldface font would be 100 % if $SRST = 0$.

Query Class	MonetDB	PostgreSQL	VoltDB	MariaDB
1 to 14	100 %	-	-	-

Table VI: Routing table for TPC-H and $LRST = 0$. All 14 queries are executed by Monet DB.

Gavel's SELECT queries belong to the same *Query Class*, Table VII has only 13 entries instead of all 18 queries of the benchmark.

The result of the comparison of ICARUS against its underlying data stores depicted in Figure 9 shows that ICARUS is more than two times faster than the fastest of the underlying data stores, MonetDB. Because the benchmarks for the individual stores are executed by the ICARUS CLIENT directly against the respective store, the result shows the real

Query Class	MonetDB	PostgreSQL	VoltDB	MariaDB
1	100 %	-	-	-
2	100 %	-	-	-
3	-	-	100 %	-
4	100 %	-	-	-
5	-	-	-	100 %
6	100 %	-	-	-
7	100 %	-	-	-
8	100 %	-	-	-
9	100 %	-	-	-
10	-	-	-	100 %
11	-	-	-	100 %
12	-	45 %	-	55 %
13	100 %	-	-	-

Table VII: Routing table for the Gavel benchmark with $SRST = 100$ and $LRST = 0$.
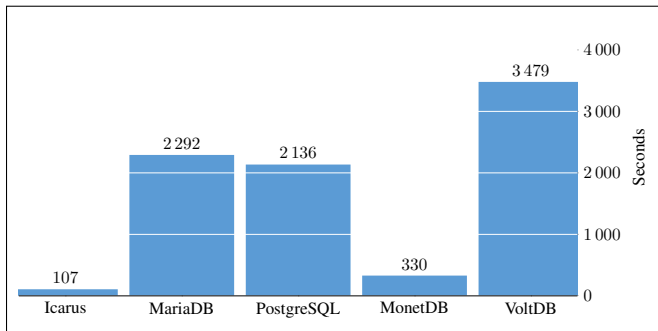
Figure 9: ICARUS compared to its underlying data stores using the Gavel Benchmark.

improvements gained by deploying ICARUS.

Figure 10 shows the speedup gained by deploying ICARUS instead of using the respective fastest of the underlying data stores. Each benchmark in this evaluation has a constant share of 9.8 % semi-analytical and 0.2 % analytical queries in the workload (same percentages for the individual query types than for the other Gavel benchmarks). The percentage of update queries in the workload is increased starting from 0 % to 90 %. The remaining queries in the workload are read-only queries. The evaluation shows that ICARUS improves performance also for heavily writing workloads (up to 80 %). The possible speedup ranges from 1.1 (80 % write queries) to 3 (0 % write queries). Even for read-heavy workloads (up to 50 % writing queries), it is more than two times as fast as the fastest data store.

E. Evaluation Summary

The results show that ICARUS is worth its costs as long as the workload is diverse enough and is not too write-heavy.
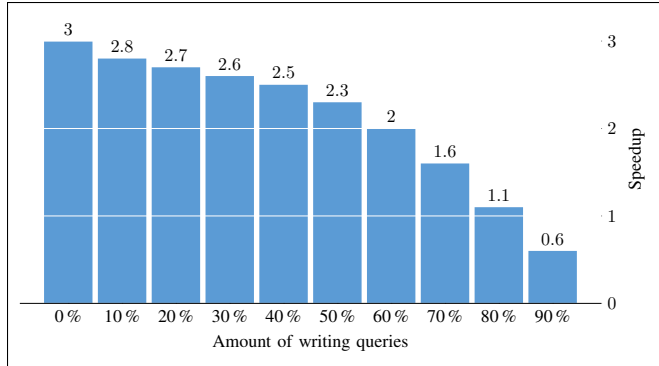
Figure 10: Speedup with ICARUS compared to the fastest of its underlying stores for different percentages of write workload. Semi-Analytical workload is always 9.8 % and analytical workload is 0.2 %. The remaining parts of the workload are transactional read queries.

Furthermore, the queries in the workload have to be derived from a limited number of *Query Classes*.

The results show that pure OLTP or OLAP benchmarks like TPC-C or TPC-H providing homogeneous workloads are not well suited for multistore DBMSs. In contrast to them, the Gavel benchmark provides and represents a good use case and produces a diverse workload. Multistore DBMSs are much better suited for such diverse workloads since they can fully utilize the strengths of their differently optimized underlying data stores. However, the drawback of the Gavel benchmark is that there are not yet results from other systems to compare with.

Note that the small fluctuations in the runtime in the above queries are due to the random ordering of the queries and the network communication. Therefore, we consider differences in runtime of less than two seconds as negligible.

VII. RELATED WORK

Recently, two systems have been published that follow a similar approach as ICARUS. *ESTOCADA*, introduced in [3], [4], is a hybrid store which combines multiple data stores with different data models and allows to query these stores using a unified API. [5] introduces *MuSQLE* a multistore DBMS, like ICARUS, build using Apache Spark [6]. It allows queries to be executed on and optimized for multiple DBMS engines (i.e., data store). MuSQLE's optimizer supports cost estimation for each individual data store allowing MuSQLE to speedup up to an order of magnitude by selecting the best data store for a large set of queries. However, this speedup is only gained if the data is not located on all data stores, i.e., it is gained if the data is split by storing the tables in a specific engine. In contrast to that, ICARUS provides a speedup up to three times with respect to the fastest store in the case where the data is stored on all underlying stores.

PelotonDB [7] stores the data tuples which are frequently accessed within an OLTP workload in a row-layout and the other tuples in a column-layout. The system therefore also takes the predicted workload into account.

In [8] three kinds of polyglot database architectures are described, namely *Polyglot Persistence*, *Lambda Architecture*, and *Multi-Model Databases*. *Polyglot Persistence* databases spread their data over multiple DBMS requiring an integration layer in the middleware which is responsible to split the users' queries into sub-queries for execution and afterwards to merge the results from the different underlying DBMSs. The *Lambda Architecture* combines batch and real-time stream processing to process data streams. *Multi-Model Databases* provide different APIs for different data methods (e.g., a graph and a document API) to work with the data while storing the data into a single database.

The lack of a universal query language for relational and NoSQL data stores is addressed in [9] and [10]. Both have designed a middleware which provides a REST API and allows to execute Create-Read-Update-Delete (CRUD) operations on different NoSQL and relational databases.

Hybrid Data Stores are databases which are using different storage media simultaneously. *HeteroDrive* [11] is an example for such a system. *HybridB tree*, introduced in [12], is a B^+ tree based index for SSD/HDD-based hybrid storage systems which reduces the random writes to the SSD without sacrificing performance. In [13], caching policies for hybrid database storage systems consisting of SSDs and HDDs are introduced.

The term "Hybrid Data Store" is also used for databases which support a row- and column-oriented data management. An example for such a DBMS is *SAP HANA* [14]. In [15] a storage advisor for *SAP HANA* is introduced, which recommends the optimal store based on the data and query characteristics. Besides the per-table decision, the tool also considers horizontal and vertical partitioning of the data and the placement of the partitions on different stores. [16] outlines at the example of medical data the challenges that arise when storing heterogeneous data in the Cloud and proposes a hybrid row-column database architecture.

HYRISE [17], [18] is also a main-memory hybrid storage engine. In [19], the developers of *HYRISE* introduce the vision of an in-memory database storage architecture that combines different storage types like row-wise, column-wise, hybrid partitions and graphs in a single system.

BigDAWG, introduced in [20], is a federated database system for multiple, disparate data models. It executes queries using so-called islands of information. Each island represents a data model, a query language and a set of data management systems.

SnappyData [21] is a DBMS build on top of *Apache Spark* [6] and *Apache Geode* [6]. It provides a unified engine for streaming, transactions, machine learning and analytics in a single cluster.

VIII. Conclusion

In this paper, we have introduced ICARUS, a novel multistore database that jointly supports several heterogeneous data stores. Based on a routing table that contains, per query type, the best suited underlying data store, ICARUS is able to properly forward queries for execution. The evaluation of ICARUS on the basis of the Gavel benchmark that defines a mixed transaction workload, i.e., a workload motivated by an auction house scenario that jointly contains OLTP and OLAP elements, has shown a speedup of up to a factor of three. This speedup is based on the fact that the underlying data stores are either optimized for short OLTP transactions or for long-running OLAP queries – but they are not able to cope with both at the same time. Therefore, the per-query routing of ICARUS is able to select, for each query, the best data store which outperforms the individual stores when used separately.

In our future work, we plan to relax the constraint of storing all data on all stores. Combining partial replication and data partitioning will allow not only to increase the performance of the individual stores, it will also reduce the required amount of storage capacity. Moreover, we also plan to further address the effect of different storage media and database technologies on the performance of the multistore. Currently, we are working on support for data stores with other data models such as, for example, key/value stores. We also plan to evaluate other combinations of DBMSs to find the best trade-off between performance and storage costs for different workloads.

References

[1] M. Stonebraker, "Technical perspective – One size fits all: an idea whose time has come and gone," *Commun. ACM*, vol. 51, no. 12, p. 76, 2008. [Online]. Available: http://doi.acm.org/10.1145/1409360.1409379

[2] H. King and J. Robinson, "On Optimistic Methods For Concurrency Control," in *Fifth International Conference on Very Large Data Bases, 1979.*, vol. 6, no. 2. IEEE, 1981, pp. 351–351. [Online]. Available: http://ieeexplore.ieee.org/document/718150/

[3] F. Bugiotti, D. Bursztyn, A. Deutsch, I. Manolescu, and S. Zampetakis, "Flexible Hybrid Stores: Constraint-Based Rewriting to the Rescue," in *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*. IEEE, may 2016, pp. 1394–1397. [Online]. Available: http://ieeexplore.ieee.org/document/7498353/

[4] F. Bugiotti, D. Bursztyn, A. Deutsch, I. Ileana, and I. Manolescu, "Invisible Glue: Scalable Self-Tuning Multi-Stores," in *Conference on Innovative Data Systems Research (CIDR)*, Asilomar, CA, USA, 2015. [Online]. Available: http://cidrdb.org/cidr2015/Papers/CIDR15_Paper7.pdf

[5] V. Giannakouris, N. Papailiou, D. Tsoumakos, and N. Koziris, "MuSQLE: Distributed SQL Query Execution Over Multiple Engine Environments," in *2016 IEEE International Conference on Big Data (Big Data)*. Washington DC, USA: IEEE, dec 2016, pp. 452–461. [Online]. Available: www.cslab.ntua.gr/~dtsouma/index_files/musqle_bigdata.pdf

[6] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster Computing with Working Sets." in *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*. Boston, MA: USENIX Association, 2010. [Online]. Available: http://www.usenix.org/events/hotcloud10/tech/full_papers/Zaharia.pdf

[7] A. Pavlo, G. Angulo, J. Arulraj, H. Lin, J. Lin, L. Ma, P. Menon, T. C. Mowry, M. Perron, I. Quah, S. Santurkar, A. Tomasic, S. Toor, D. V. Aken, Z. Wang, Y. Wu, R. Xian, and T. Zhang, "Self-Driving Database Management Systems," in *CIDR 2017, Conference on Innovative Data Systems Research*, Chaminade, California, 2017. [Online]. Available: http://db.cs.cmu.edu/papers/2017/p42-pavlo-cidr17.pdf

[8] L. Wiese, "Polyglot database architectures," in *Proceedings of the LWA 2015 Workshops: KDML, FGWM, IR, and FGDB.*, no. October, Trier, Germany, 2015, pp. 422–426. [Online]. Available: http://ceur-ws.org/Vol-1458/H05_CRC85_Wiese.pdf

[9] F. Gessert, S. Friedrich, W. Wingerath, M. Schaarschmidt, and N. Ritter, "Towards a scalable and unified REST API for cloud data stores," in *Lecture Notes in Informatics (LNI)*, vol. P-232, 2014, pp. 723–734. [Online]. Available: https://vsis-www.informatik.uni-hamburg.de/vsis/publications/lookpub/522

[10] R. Sellami, S. Bhiri, and B. Defude, "ODBAPI: A Unified REST API for Relational and NoSQL Data Stores," in *IEEE International Congress on Big Data*. IEEE, Jun. 2014, pp. 653–660. [Online]. Available: http://ieeexplore.ieee.org/document/6906841/

[11] S.-H. Kim, D. Jung, J.-S. Kim, and S. Maeng, "HeteroDrive: Reshaping the Storage Access Pattern of OLTPWorkload Using SSD," in *Proceedings of 4th International Workshop on Software Support for Portable Storage (IWSSPS 2009)*, 2009, pp. 13–17. [Online]. Available: http://camars.kaist.ac.kr/~maeng/pubs/iwssps2009.pdf

[12] P. Jin, P. Yang, and L. Yue, "Optimizing B^+-tree for hybrid storage systems," *Distributed and Parallel Databases*, vol. 33, no. 3, pp. 449–475, Sep. 2015. [Online]. Available: http://link.springer.com/10.1007/s10619-014-7157-7

[13] X. Liu and K. Salem, "Hybrid storage management for database systems," *Proceedings of the VLDB Endowment*, vol. 6, no. 8, pp. 541–552, jun 2013. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2536354.2536355

[14] F. Färber, S. K. Cha, J. Primsch, C. Bornhövd, S. Sigg, and W. Lehner, "SAP HANA Database – Data Management for Modern Business Applications," *ACM SIGMOD Record*, vol. 40, no. 4, p. 45, jan 2012. [Online]. Available: http://dl.acm.org/citation.cfm?id=2094126

[15] P. Rösch, L. Dannecker, F. Färber, and G. Hackenbroich, "A storage advisor for hybrid-store databases," *Proceedings of the VLDB Endowment*, vol. 5, no. 12, pp. 1748–1758, aug 2012. [Online]. Available: http://arxiv.org/abs/1208.4169

[16] B. Mohamad, L. D'Orazio, and L. Gruenwald, "Towards a Hybrid Row-Column Database for a Cloud-based Medical Data Management System," in *Proceedings of the 1st International Workshop on Cloud Intelligence – Cloud-I'12*. New York City, NY, USA: ACM Press, 2012, pp. 1–4. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2347673.2347675

[17] M. Grund, J. Krueger, H. Plattner, A. Zeier, P. Cudre-Mauroux, and S. R. Madden, "HYRISE – A Main Memory Hybrid Storage Engine," *Proceedings of the VLDB Endowment*, vol. 4, no. 2, pp. 105–116, nov 2010. [Online]. Available: http://dl.acm.org/citation.cfm?doid=1921071.1921077

[18] M. Grund, J. Kr, A. Zeier, P. Cudre-Mauroux, and S. Madden, "A Demonstration of HYRISE — A Main Memory Hybrid Storage Engine," *Proceedings of the VLDB Endowment*, vol. 4, no. 12, pp. 1434–1437, 2011.

[19] M. Grund, P. Cudre-Mauroux, J. Krueger, and H. Plattner, "Hybrid Graph and Relational Query Processing in Main Memory," in *2013 IEEE 29th International Conference on Data Engineering Workshops (ICDEW)*. IEEE, apr 2013, pp. 23–24. [Online]. Available: http://ieeexplore.ieee.org/document/6547419/

[20] J. Duggan, S. Zdonik, A. J. Elmore, M. Stonebraker, M. Balazinska, B. Howe, J. Kepner, S. Madden, D. Maier, and T. Mattson, "The BigDAWG Polystore System," *ACM SIGMOD Record*, vol. 44, no. 2, pp. 11–16, aug 2015. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2814710.2814713

[21] B. Mozafari, J. Ramnarayan, S. Menon, Y. Mahajan, S. Chakraborty, H. Bhanawat, and K. Bachhav, "SnappyData: A Unified Cluster for Streaming, Transactions, and Interactive Analytics," in *CIDR 2017, Conference on Innovative Data Systems Research*, Chaminade, California, 2017, pp. 1–8. [Online]. Available: http://cidrdb.org/cidr2017/papers/p28-mozafari-cidr17.pdf