Workload Optimization using SharedDB

Georgios Giannikis Darko Makreshanski
Gustavo Alonso Donald Kossmann

Systems Group, Department of Computer Science, ETH Zurich, Switzerland
{giannikg,darkoma,alonso,kossmann}@inf.ethz.ch

ABSTRACT

This demonstration presents SharedDB, an implementation of a relational database system capable of executing all SQL operators by sharing computation and resources across all running queries. SharedDB sidesteps the traditional query-at-a-time approach and executes queries in batches. Unlike proposed multi-query optimization ideas, in SharedDB queries do not have to contain common subexpressions in order to be part of the same batch, which allows for a higher degree of sharing. By sharing as much as possible, SharedDB avoids repeating parts of computation that is common across all running queries. The goal of this demonstration is to show the ability of shared query execution to a) answer complex and diverse workloads, and b) reduce the interaction among concurrently executed queries that is observed in traditional systems and leads to performance deterioration and instabilities.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Query processing, Relational databases*

General Terms

Experimentation, Measurement, Performance

Keywords

Main Memory, Shared Query Processing

1. INTRODUCTION

Traditionally designed to execute individual queries as fast as possible, database systems face considerable challenges when dealing with complex workloads composed of large amounts of transactional and analytical queries. With the advent of multicores, this query-at-a-time approach will typically handle such workloads by allocating and time-sharing one or several cores to individual queries, and executing them at the same time. This creates unpredictable resource interference and contention [3] that is especially problematic for systems that require to meet service-level agreements (SLAs) that involve for instance a maximum response time.

Current approaches to meet SLAs in these situations include replication, materialized views, indexing, caching and reusing (recycling) of query results. While highly effective in certain situations, they are all sensitive to high update rates, and further require the work of skilled database administrators to be properly configured given a certain workload. They also sometimes involve making certain trade-offs. For instance, replication of transactional data is used to eliminate the interference of analytic workload at the expense of losing data freshness guarantees, increased operational and maintenance costs and replication overhead. So-called NoSQL (Dynamo, Cassandra) systems are able to handle large update and query rates, however they lack support for SQL operators such as joins, sort and top-n.

What is sought for is a simple solution to the problem imposed by complex, dynamic workloads and strict SLA requirements. The goal is a single system that is able to minimize resource contention and interference and provide stable performance while handling such workloads.

For this purpose, we have developed a novel system, called SharedDB [1], that optimizes the execution of the whole workload instead of optimizing each query individually. In SharedDB query processing is distributed over multiple cores, with each core dedicated to an always-active operator (scan, sort, join and so on), processing batches of possibly thousands of requests concurrently. Compared to the traditional query-at-a-time approach, this processing model gives us response times that depend more on the complexity of the queries and less on the state and load of the system.

In this demonstration we showcase SharedDB's performance using the TPC-W benchmark, simultaneously comparing it to the performance of a traditional query-at-a-time system, namely MySQL. The demonstration will allow the conference participants to understand the trade-offs of using a single plan for all the queries and the advantages of shared query execution.

2. SHAREDDB

In this section we outline the main design ideas and features of SharedDB. Additionally, we present the key differences of a shared query processing system to traditional query-at-a-time approaches, as well as, details of how sharing can increase the performance of a database system. Further details about SharedDB can be found in [1].

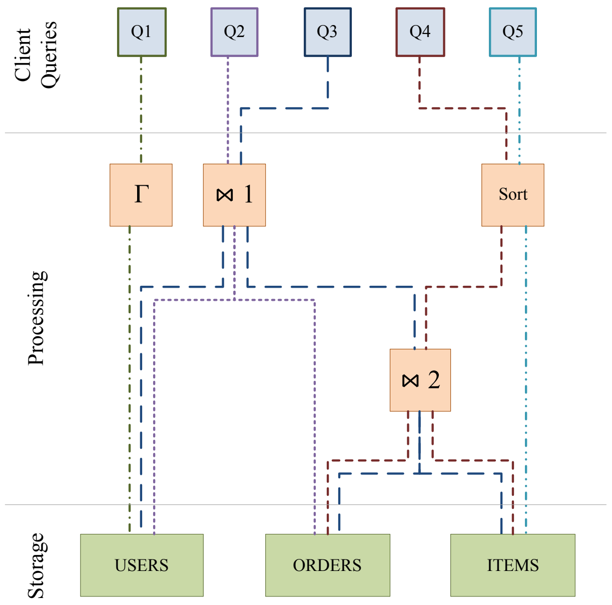
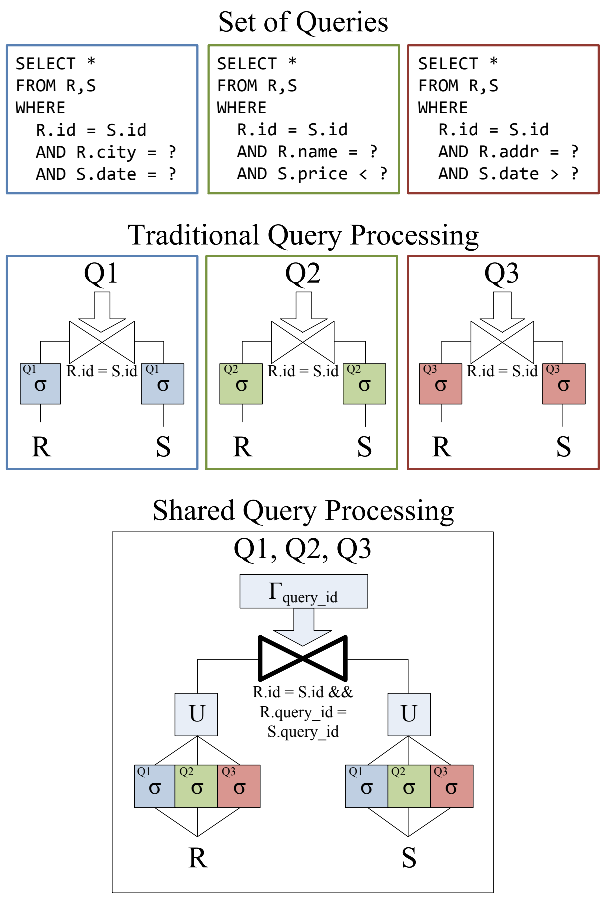


Figure 1: Example of a Global Query Plan



```
SELECT *
FROM R,S
WHERE
  R.id = S.id
  AND R.city = ?
  AND S.date = ?
```

```
SELECT *
FROM R,S
WHERE
  R.id = S.id
  AND R.name = ?
  AND S.price < ?
```

```
SELECT *
FROM R,S
WHERE
  R.id = S.id
  AND R.addr = ?
  AND S.date > ?
```

Figure 2: Shared Join

2.1 Overview

SharedDB is a relational database system, able to execute all SQL operators. In SharedDB queries share computation and resources which avoids repetition of computations across running queries.

Instead of compiling every query into a separate query plan, SharedDB compiles the whole workload of the system into a single global query plan. The global query plan may serve hundreds or thousands of concurrent SQL queries and updates and may be reused over a long period of time, possibly for the entire lifetime of the system. An example of a global query plan for a simple workload can be seen in Figure 1. In this example, three query processing operators and three storage engines are used to answer five different query types. During the demonstration, we will use the complete set of queries of TPC-W.

SharedDB operators are able to process multiple queries concurrently. For instance, in the global query plan of Figure 1, the Sort operator process tuples that belong to Q_4 and Q_5 at the same time. This is achieved by *batching* queries and executing each batch serially. Queries that arrive while a batch is being executed are queued and will be part of the following batch. In SharedDB all result tuples, including intermediate results, are "tagged" with the `query_id` of the query that generated them. Furthermore, result tuples can belong to multiple queries by means of a list of `query_id`s. An important advantage of this approach is that it minimizes the amount of result tuples that are processed. In SharedDB hotspots inflict only a constant load, no matter how popular a tuple is across running queries.

The introduction of the `query_id` tagging enables the concurrent processing of tuples that belong to different queries. Figure 2 shows how SharedDB processes joins for three different queries (Q_1, Q_2, Q_3). All three queries involve a join between tables R and S, but each query has different predicates on the S and R tables. Instead of compiling each query individually as shown in the middle of Figure 2, SharedDB will merge the individual query plans into a single shared plan that meets the requirements of all three queries.

Technically, the *union* of all R and S tuples that the three queries are interested in are considered as part of the join. Furthermore, the join predicate is amended, thereby considering the `query_id`. This way, an R tuple that is only relevant for Query Q_1 does not match an S tuple that is only relevant for Query Q_2. Finally, the routing of the join results to the relevant queries is carried out using a *grouping operator* (Γ) by `query_id`.

2.2 Implementation Details

In contrast to existing relational database systems that assign each query to one or more threads, SharedDB statically assigns threads to always-on shared operators. This "staging" [2] of operators creates a data flow network, in which result tuples flow through operators. This idea originates from data stream processing and increases performance due to the better instruction cache locality.

In order to reduce scheduling overhead and working set migration, each operator thread uses hard affinity to bind to a dedicated CPU core. Furthermore, SharedDB benefits from non-uniform memory architectures by statically allocating the working set of each operator in the closest NUMA memory region. This working set includes the stack of the threads, as well as, all intermediate buffers and hash tables that are required to process the queries.

Finally, in SharedDB data is stored and processed in main memory, which further improves the performance of the system. This applies to all operators in SharedDB. Specifically for the storage engine operators, SharedDB uses a simple key-value store as an index and Crescando [4] to evaluate queries that require full table scans.

3. DEMONSTRATION OVERVIEW

In this demonstration, we offer the attendees the opportunity to understand the trade-offs of using a single plan for all the queries and the advantages of shared execution by means of an interactive demo. The demo highlights the system's performance on the TPC-W benchmark, a simulation of an online bookstore. The workload will be answered by an instance of SharedDB, as well as an instance of MySQL, a widely used query-at-a-time system. The participants will be able to modify certain parameters of the workload and explore how this affects the performance on both systems.

3.1 System Setup

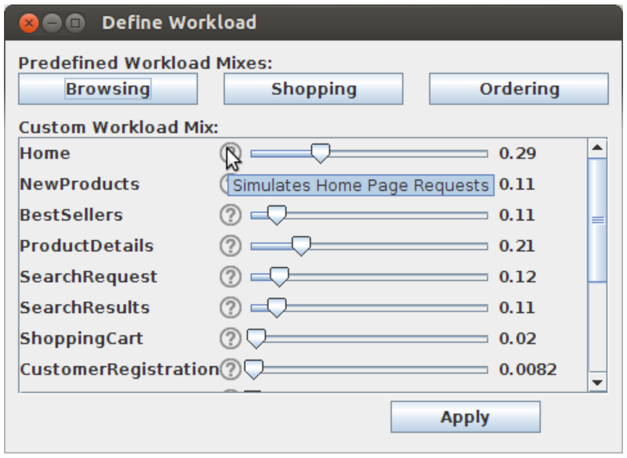
The demonstration uses two identical machines, each built from four 12-core AMD Opteron 6174 "Magny-Cours", running at 2.2 GHz, and 128 GBytes of DDR3 RAM. One of the machines is going to host SharedDB while the second will be used to run MySQL 5.5. MySQL has been carefully configured for the given workload. The data files are located on an in-memory filesystem, while all MySQL buffers are big enough to accommodate the intermediate results. Since MySQL does not scale linearly to the number of CPU cores [3], we allow the conference participants to define the number of CPU cores available to both systems. Additionally, a set of eight 16-core machines will be used to generate the workload for the two systems.

During the demonstration, a small pool of laptops, one of them connected to a projector, will allow the conference participants to interact with the two systems by modifying the workload in terms of query complexity as well as query load (cf. Figure 4). This forms the "performance" part of our demo. At the same time, the conference participants will be able to specify a synthetic load on the system consisting of very diverse queries, and observe how the execution of one query affects the performance of others on the two systems (cf. Figure 5). This forms the "query interaction" part of our demonstration.

The laptops will be connected to a server cluster running at ETH Zurich. This setup naturally requires an Internet connection. In the event of bad internet connectivity, we will run an appropriately scaled-down instance of the two systems on the laptops themselves.

3.2 Workload Definition

As mentioned earlier, the demonstration is based on the TPC-W benchmark. TPC-W was chosen because of the great diversity of the queries involved and the transactional constraints. In TPC-W, users issue requests (called web interactions) that are translated into database transactions with a couple of SQL queries. Most of the queries involved are small, point queries that can be answered just by joining a few tuples. For instance, the "Home" web interaction simulates the user visiting the home page of the online bookstore and requires the execution of a small join that fetches the promotions. On the other hand, certain web interactions include heavy analytical queries that involve processing of much larger sets of tuples. An example is the "Best Sellers" web interaction which simulates the user querying for the most popular books of the last orders, grouped by category. In TPC-W, the best seller books are analytically calculated on every such web interaction and not cached. The query that computes the best sellers involves joins over four tables plus a group by and a top-n operator.



Home	0.29
NewProducts	0.11
BestSellers	0.11
ProductDetails	0.21
SearchRequest	0.12
SearchResults	0.11
ShoppingCart	0.02
CustomerRegistration	0.0082

Figure 3: Synthesis of User-Defined Workloads

The selected size for the TPC-W benchmark is 10,000 items and a user-defined number of TPC-W clients (up to 3,000). In order to simplify the demonstration, we do not use the full TPC-W benchmark stack which typically includes web servers and http terminals, rather than just the database component and the clients.

In TPC-W, web interactions have to be answered in a given time. Web interactions that timeout are not counted as successful. The timeout value for every web interaction is different, based on how complex it is. In this demo we respect the timeout values and consider as throughput only the web interactions that where answered in time.

3.3 Performance Demo

The performance demo allows the participants to compare the relative performance of SharedDB with query-at-a-time systems under different workloads. The interface will offer the option to choose one of the three different web interactions mixes of the TPC-W specification. These are:

- the Browsing mix: a read-most workload that includes a considerable amount of analytical queries,
- the Ordering mix: a write intensive workload, and
- the Shopping mix: a workload that includes a good mix of writes and analytical queries.

Furthermore, the interface will allow the participants to synthesize an arbitrary mix of web interactions of TPC-W. This window, shown in Figure 3, can be used to define how frequently the TPC-W web interactions appear in the mix. This enables creation of i.e., extreme mixes where all the queries are analytical. The window includes a set of helpful tooltips and dialog messages that provide a description of each web interaction, as well as information about the SQL queries that need to be executed.

Once the workload has been defined, the user of the demo will be able to define the workload rate. This affects the

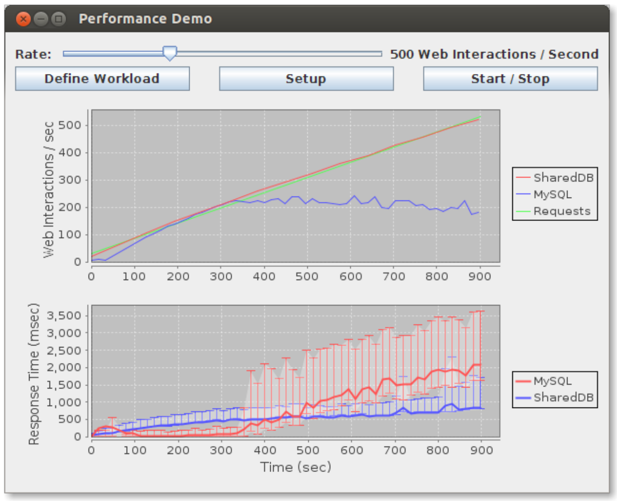


Figure 4: Screenshot of Performance Demonstration

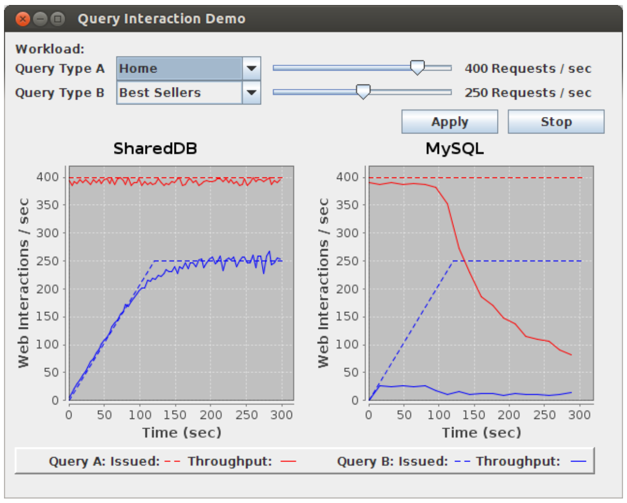


Figure 5: Query Interaction Demo

number of TPC-W clients that are used to issue requests and the think time between two requests. Finally, the interface will display the performance of the two systems in terms of achieved throughput per second. Additionally, the response time of the web interactions will be plotted, including the standard deviation. This will help the participants understand the advantages of sharing operators across queries and executing them as batches rather than one-at-a time.

Figure 4 shows a screenshot of the proposed interface during a sample execution.

3.4 Query Interaction Demo

The second part of the demonstration showcases how sharing reduces query interaction and improves the predictability of a system. The query interaction demo allows the user to select two web interactions and modify the rate at which they are issued, while monitoring their achieved throughput.

This part of the demo makes it easy to investigate how the performance of one query is affected when the system is concurrently executing other queries. Figure 5 is a screenshot of the proposed interface during a sample execution. A suggested execution is the concurrent execution of the lightweight "Home" web interaction with the heavy "Best Sellers" web interaction. In this case, the Best Sellers queries will dominate the load of the systems and as a result, MySQL fails to answer not only the "Best Sellers" queries but also most of the "Home" queries. On the contrary, SharedDB provides better isolation and as a result the only penalized queries are the "Best Sellers".

Last but not least, the users will be able to see which operators are used in each query. The "Global Query Plan Monitor" window displays which SharedDB operators are used to execute the given workload, as well as how many queries of each type are handled by the operators currently. Figure 6 shows a sample screenshot of the monitoring window. This allows the participants to see how many queries share the same operator at runtime.

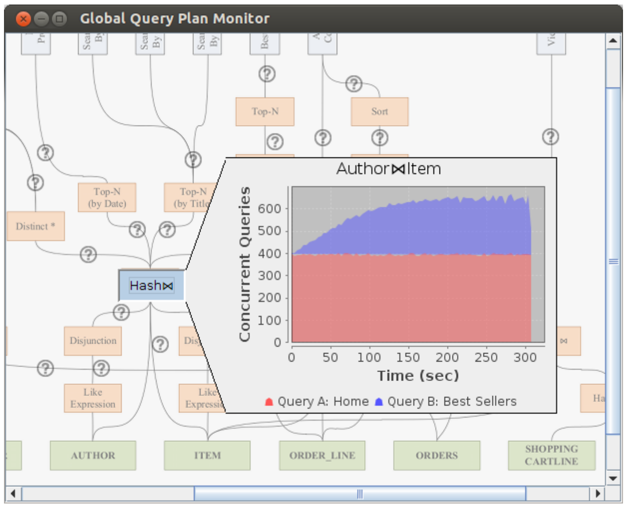


Figure 6: Global Query Plan Monitor Window

4. ACKNOWLEDGMENTS

This work was funded by the Enterprise Computing Center of ETH Zurich (www.ecc.ethz.ch). Furthermore, the work of G. Giannikis was supported by a Swiss National Science Foundation grant as part of its Pro-Doc program.

5. REFERENCES

[1] G. Giannikis, G. Alonso, and D. Kossmann. SharedDB: Killing one Thousand Queries with one Stone. *Proc. VLDB Endow.*, 5(6):526–537, Feb. 2012.

[2] S. Harizopoulos and A. Ailamaki. StagedDB: Designing Database Servers for Modern Hardware. *IEEE Data Eng. Bull.*, 28(2):11–16, 2005.

[3] T.-I. Salomie, I. E. Subasu, J. Giceva, and G. Alonso. Database Engines on Multicores, Why Parallelize when you can Distribute? In *Proc. EuroSys*, pages 17–30, 2011.

[4] P. Unterbrunner, G. Giannikis, G. Alonso, D. Fauser, and D. Kossmann. Predictable Performance for Unpredictable Workloads. In *Proc. VLDB*, pages 706–717, 2009.