Scaling up Mixed Workloads: a Battle of Data Freshness, Flexibility, and Scheduling

Iraklis Psaroudakis[1,3], Florian Wolf[1,4], Norman May[1], Thomas Neumann[2], Alexander Böhm[1], Anastasia Ailamaki[3], and Kai-Uwe Sattler[4]

[1] SAP SE, 69190 Walldorf, Germany
iraklis.psaroudakis@sap.com, florian.wolf01@sap.com, norman.may@sap.com, alexander.boehm@sap.com
[2] TU Munich, Germany
thomas.neumann@in.tum.de
[3] EPFL, Switzerland
iraklis.psaroudakis@epfl.ch, anastasia.ailamaki@epfl.ch
[4] TU Ilmenau, Germany
florian.wolf@tu-ilmenau.de, kus@tu-ilmenau.de

Abstract. The common "one size does not fit all" paradigm isolates transactional and analytical workloads into separate, specialized database systems. Operational data is periodically replicated to a data warehouse for analytics. Competitiveness of enterprises today, however, depends on real-time reporting on operational data, necessitating an integration of transactional and analytical processing in a single database system. The mixed workload should be able to query and modify common data in a shared schema. The database needs to provide performance guarantees for transactional workloads, and, at the same time, efficiently evaluate complex analytical queries. In this paper, we share our analysis of the performance of two main-memory databases that support mixed workloads, SAP HANA and HyPer, while evaluating the mixed workload CH-benCHmark. By examining their similarities and differences, we identify the factors that affect performance while scaling the number of concurrent transactional and analytical clients. The three main factors are (a) data freshness, i.e., how recent is the data processed by analytical queries, (b) flexibility, i.e., restricting transactional features in order to increase optimization choices and enhance performance, and (c) scheduling, i.e., how the mixed workload utilizes resources. Specifically for scheduling, we show that the absence of workload management under cases of high concurrency leads to analytical workloads overwhelming the system and severely hurting the performance of transactional workloads.

Keywords: OLAP, OLTP, CH-benCHmark, SAP HANA, HyPer, data freshness, flexibility, scheduling, workload management

1 Introduction

Traditionally, online transaction processing (OLTP) workloads have been the motivation force behind relational database management systems (DBMS). OLTP

workloads are composed of short-lived transactions that read or modify operational data, and are typically standardized, submitted through application layers such as an Enterprise Resource Planning (ERP) software or an online web shop. The increasing importance of business intelligence led to another class of long-running, scan-heavy, ad-hoc queries, namely online analytical processing (OLAP) workloads. Due to their substantial differences from OLTP workloads, OLAP workloads are supported by specialized database systems, which are typically used in data warehouses. Operational data is periodically replicated from OLTP systems to data warehouses for analytics.

Nowadays, the design gap between OLTP-oriented and OLAP-oriented DBMS or data warehouses is even more prominent, since big data applications demands and performance requirements increase [20]. OLTP-oriented DBMS, such as VoltDB or DB2, are typical row-stores that deliver high throughput for updates and index-based queries. OLAP-oriented DBMS, such as Vectorwise or Sybase IQ or DB2 BLU [19], are typical column-stores that deliver high performance for complex analytical queries, and do not support transactional workloads or offer only a chunk-wise mechanism for loading data.

1.1 Real-time Reporting

In exchange for high performance, OLAP-oriented DBMS typically do not support full ACID transactions. As a result, data analytics queries run on an outdated version of operational data. This is unacceptable for real-time reporting, where organizations and enterprises are increasingly requiring analytics on fresh operational data to gain a competitive advantage or obtain insight about fast-breaking situations [1, 16]. Examples include online games that make special offers based on non-trivial analysis [4], liquidity and risk analysis, which benefits from fresh data while also requiring complex analytical queries [17], and fraud detection analyzing continuously arriving transactional data [15].

The need for real-time reporting necessitates the development of a new class of DBMS that can efficiently support mixed (OLTP and OLAP) workloads processing common data of a common schema [17]. Efficient processing means scaling OLTP clients to as many users as possible, with reasonably short response times [8], while, at the same time, servicing OLAP clients whose longer-running queries should be able to efficiently analyze the live operational data. In this paper, we evaluate the performance of two state-of-the-art mixed workload DBMS: SAP HANA [7], and HyPer [10]. By examining their similarities and differences, we aim to identify the factors that affect the performance of mixed workloads while we scale the number of concurrent clients.

To evaluate mixed workloads, we cannot readily use benchmarks aimed for either OLTP or OLAP, such as TPC-C, TPC-W, TPC-H, TPC-DS [2] or OLTPbench [6]. As a new direction to benchmarking mixed workloads, we adopt the CH-benCHmark [5], which considers concurrent OLAP and OLTP clients in a mixed workload inspired by TPC-C and TPC-H. We find the CH-benCHmark an adequate solution since it allows to scale the number of concurrent transactional and analytical clients independently.

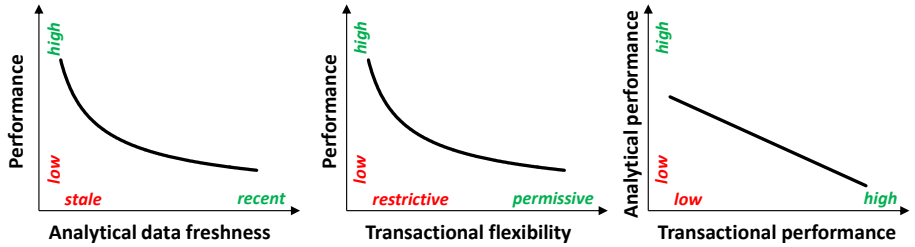


Fig. 1. Conceptual figures of how we expect the performance of mixed workloads to be affected by (a) data freshness, (b) flexibility, and (c) scheduling.

1.2 Scaling up Mixed Workloads

We identify three main factors that affect the performance of mixed workloads while we scale the number of concurrent clients: (a) *data freshness*, (b) *flexibility*, and (c) *scheduling*. In Figure 1, we sketch how we expect the performance of the DBMS to be affected by these factors.

Data freshness refers to how recent the data, that is processed by analytical queries, is. On the one hand, data can be stale, as is the case for typical data warehouses where operational data is periodically replicated. This separation, with a low level of data freshness, allows for various optimizations such as decoupling transactions and analytics, minimizing the interference between them, and having additional materialized views or indexes for analytics (which may be otherwise expensive to maintain with a high level of data freshness). On the other hand, as the refresh rate is increased, we compromise performance, because we need to sustain the overhead of frequent snapshots and respect transactional semantics of the concurrent OLTP workload.

Flexibility refers to the restrictions that a DBMS may impose on the transactional features or expressiveness in order to increase optimization choices to enhance performance. For example, a system can restrict flexibility by requiring that transactions are instantiated from templates that are known in advance, allowing for pre-compilation of transactions [21]. Another example is restricting interactivity, i.e., transactions cannot have multiple rounds of communication with a remote client, which allows optimizing execution [21]. Moreover, techniques like just-in-time (JIT) compilation, introduce a compilation overhead (of e.g., several milliseconds), but can improve performance of ad-hoc queries [14].

Scheduling determines how, and the order in which transactions and analytical queries use the system's resources, including potential workload management techniques. For the typical case of high concurrency with numerous OLTP and OLAP clients and a fully saturated system, the DBMS may opt to either prioritize transactions in the expense of analytical queries, or the reverse.

1.3 Contributions

In this paper, we survey, evaluate, and compare two state-of-the-art main-memory DBMS for mixed workloads: SAP HANA and HyPer. We evaluate the CH-

benCHmark, which we implement using C++ and ODBC, and provide it as open source[5]. Through our analysis, we detail how (a) data freshness, (b) flexibility, and (c) scheduling affect the performance of mixed workloads while we scale the number of concurrent clients. The most significant findings of our experimental evaluation (see Section 5) are:

- DBMS, that maintain separate versions of the operational data for analytics, can suffer a decrease in performance of up to 40% for high refresh rates.
- DBMS, which are optimized for the execution of less flexible or less expressive transactions, can achieve up to one order of magnitude better transactional throughput than DBMS optimized for flexible and interactive transactions.
- The absence of workload management in cases of high concurrency, that fully saturate the system, results in long-running and complex analytical queries overwhelming the system, and significantly hurting the performance of short-lived transactional workloads.

Paper outline. In sections 2 and 3, we survey how SAP HANA and HyPer handle mixed workloads. In Section 4, we describe how we implement the CH-benCHmark. Our experimental evaluation is presented in Section 5. Finally, we conclude our paper in Section 6.

2 Mixed Workloads in SAP HANA

SAP HANA is a commercial main-memory relational DBMS that supports mixed OLTP and OLAP workloads. It incorporates four storage engines to support various workloads: (a) a column-store, that efficiently supports OLAP-dominated and mixed workloads, (b) a row-store, that is suited for OLTP workloads, (c) a text engine, and (d) a graph engine [7]. For the evaluation of our paper, we use the column-store as it is better suited for mixed workloads. We note that a hybrid data layout [3, 9] can improve the performance of mixed workloads, but, in this paper, we focus on assessing the scalability of concurrency than different data layout approaches.

Each column in the column-store is composed of two parts: the *main*, and the *delta*, as shown in Figure 2a. Data in the main is dictionary encoded using a sorted dictionary. The dictionary-encoded data is static, bit-compressed, and further compressed for fast scanning. The delta supports transactional operations, and includes recently added, updated, and deleted data. The delta's dictionary is unsorted, and a cache-sensitive B+-tree is employed for fast lookups. To respect transactional semantics, read operations query both the main and the delta. The transaction manager uses multi-version concurrency control (MVCC).

Allowing the delta part to grow incessantly compromises performance of both analytical and transactional operations due to the increasing bookkeeping overhead of the delta's dictionary and index. Thus, the delta is periodically merged

[5] Available online at: `http://www3.in.tum.de/research/projects/CHbenCHmark/`

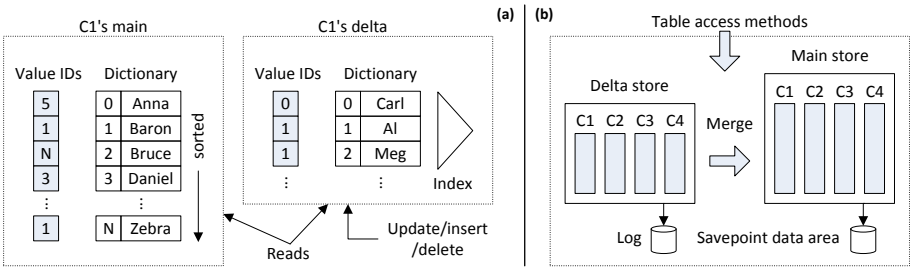


Fig. 2. (a) The core data structures of the main and the delta parts of a column. (b) The delta part of a column is periodically merged into the main part.

into the main part, as shown in Figure 2b, reconstructing the static data structures of the main part, and preparing an empty delta for new data. For recovery, the merge operation may store a savepoint in persistent memory, and further transactional operations (in the delta) are typically logged.

Next, we discuss the issue of data freshness for analytical queries, how flexible are transactions, and how scheduling works in SAP HANA.

Data freshness. The fact that both analytical and transactional operations target the same data means that SAP HANA allows analytics to query the most recent version of operational data. As soon as an OLTP operation, e.g., updates data in the delta of a column, the new version is immediately available by the MVCC to upcoming analytical queries. Allowing OLTP and OLAP to target common data, however, comes with the cost of synchronization for the common data structures, such as the index of the delta's dictionary.

Flexibility. SAP HANA supports fully interactive ACID transactions [11], which can contain multiple round-trips to the client. Efficient and flexible support for distributed transactions is available. Upon first execution, queries are compiled and the cached plan is available in subsequent invocations of the same query. It supports multiple interfaces, including SQL and specialized languages [7].

Scheduling. SAP HANA employs a pool of threads for servicing network clients and a task scheduler for servicing heavy-weight requests [18]. Analytical queries can be expressed as single tasks or as multiple tasks (intra-query parallelism) which are dispatched to task queues. One worker thread is employed per hardware context that continuously gets tasks from the queues and executes them. The scheduler takes care to maintain the number of active worker threads as close as possible to the number of hardware contexts, avoid unnecessary involuntary context switches, and allow stealing tasks to balance task queues.

When the machine is fully saturated, scheduling decides how transactions and analytical queries utilize resources. We show that the default configuration of SAP HANA favors analytical throughput over transactional throughput. By decreasing parallelism of analytical queries, however, we can increase transactional throughput to the detriment of analytical throughput (see Section 5).

3 Mixed Workloads in HyPer

HyPer is a research prototype main-memory relational DBMS that supports mixed OLTP and OLAP workloads [10]. The aim is to support high OLTP throughput, as well as efficient concurrent execution of OLAP workloads. The storage engine can be configured to be a row-store or a column-store. In this paper, we use the column-store configuration.

OLTP clients are serviced serially with a single thread [10]. This avoids the usage of locks or latches for data structures, and, due to the absence of I/O, allows transactions to be executed in one-shot, uninterrupted and efficiently. Multiple threads for OLTP are supported if the schema is manually partitioned or the machine supports hardware transactional memory [13]. In this paper, we use the default single-threaded behavior.

For serving OLAP clients, HyPer uses an innovative way to provide snapshots of operational data. As shown in Figure 3a, OS- and hardware-supported virtual memory facilities are leveraged to efficiently create snapshots. Currently, each arriving OLAP client forks the main OLTP process into another process, getting a virtual memory snapshot to work on. The lazy copy-on-update strategy ensures that a virtual page is not physically replicated, and OLTP and OLAP are reading the same physical page. The OS creates a new physical copy only in the case a transaction modifies a page. In this case, the parent OLTP process has the latest version, and the OLAP process refers to the older version of the page. The capability to update OLAP snapshots on demand in a single system is far more efficient than the usual two system setup (one for OLTP and one for OLAP), since data does not need to be replicated from system to the other.

Data freshness. Conceptually, HyPer's main OLTP process is similar to SAP HANA's delta and the OLAP processes are similar to versions of the main. Forking is similar to the merge operation. In contrast to SAP HANA, analytics read their snapshot and not the freshest data from the OLTP process. This allows decoupling of OLTP and OLAP, and synchronization overhead is avoided. Also, since the OLAP client can update its snapshot on demand, data freshness is customizable: on the one hand, the client can opt to take a snapshot and never update it, or, on the other hand, update its snapshot after every couple of queries. The downside of this tactic, however, compared to SAP HANA, is that, in the case that OLAP clients wish to keep their snapshots as fresh as possible, the virtual memory snapshot overhead is increased (see Section 5).

Flexibility. HyPer is optimized for the execution of prepared statements or precompiled transactions [10]. Ad-hoc queries and ACID transactions are both supported and compiled by a just-in-time (JIT) compiler. The overhead of the elaborate compilation may be amortized for multiple invocations of the same query or transaction, but can limit the scalability of short-lived ad-hoc OLTP.

HyPer restricts flexibility for clients on purpose to allow for further optimizations. For example, clients need to define if they are OLTP or OLAP clients. Also, for an OLTP client, the whole client transaction is performed in a single batch, i.e. there cannot be multiple round-trips to a client in a transaction. The re-

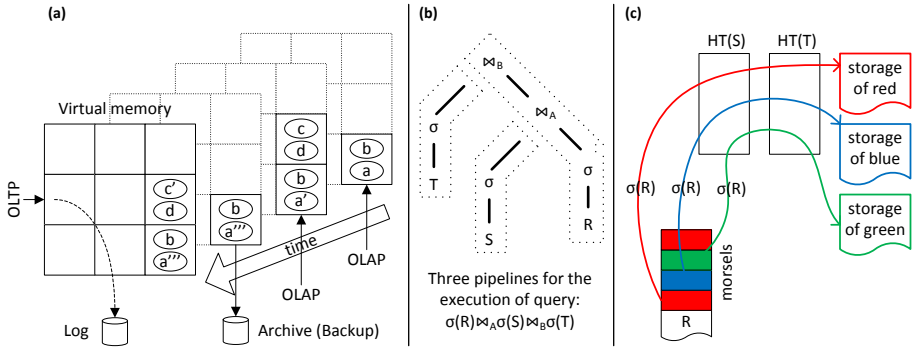


Fig. 3. (a) HyPer design using virtual memory facilities to create snapshots for analytics. (b) Restricting flexibility to support further optimizations of query plans. (c) Conceptual figure of scheduling for pipeline R of the query plan of (b).

strictions for OLTP clients allow for, e.g., analysis of the transaction, regarding the accessed and updated tables, or the control-flow [14]. These optimizations, along with the serialization of transactions, can achieve significantly high OLTP throughput (see Section 5). Read-only OLAP clients access a read-only snapshot; ad-hoc queries are fully supported because they are compiled as they arrive on the database server.

As shown in Figure 3b, the query plans created by the optimizer are composed of operator pipelines through which tuples are pushed. Pipelines are broken by operators that cannot be pipelined (e.g., a sort). By pushing tuples through a whole pipeline of several operators, performance can be significantly improved with JIT compilation, better data locality, and predictable branch layout [14].

Scheduling. HyPer includes a NUMA-aware (non-uniform memory access) task scheduler for queries. Each phase of a query is parallelized, and the scheduler takes care to distribute work evenly across sockets, using task stealing and elastic parallelism, and optimize for data locality [12]. In Figure 3c, we show an example of how the scheduler executes the probe phases of the hash-joins of pipeline R (of the query of Figure 3b), using three of the sockets of a machine (depicted in different colors). Relation R is partitioned to small fragments, called *morsels*. A thread continuously takes a morsel from relation R, local to its socket, and passes it through the pipeline, probing the hash tables for relations S and T, finally storing locally the result. In comparison to SAP HANA, we show in our experiments (see Section 5), that HyPer's scheduling also favors analytical throughput over transactional throughput in cases of high concurrency and saturation.

4 Setting up the CH-benCHmark

The CH-benCHmark builds upon the widely used TPC-C and TPC-H benchmarks [2]. TPC-C is used to analyze the performance of transactional workloads

in a scenario of order processing, while TPC-H analyzes the performance of analytical workloads in the context of a wholesale supplier. The goal of the CH-benCHmark [5] is to combine TPC-C and TPC-H in a unified schema, in order to analyze the performance of the mixed OLTP and OLAP workload. Next, we give an overview of the CH-benCHmark, and how we adapt and implement it.

4.1 Overview of the CH-benCHmark

The database schema of the CH-benCHmark is shown in Figure 4. The schema uses the nine tables of TPC-C and adds the tables `NATION`, `REGION`, and `SUPPLIER` from TPC-H. As in TPC-C, the size of the database scales with the number of warehouses. The integrated schema required the following changes:

- `NATION` contains 62 rows instead of 25, and `SUPPLIER` is fixed to 10,000 rows.
- `CUSTOMER` and `NATION` can be joined on columns `N_NATIONKEY` and `C_STATE`. Column `C_STATE`, however, is defined as a two-character code while `N_NATIONKEY` is defined as integer. To solve this mismatch, the following join condition was proposed in the original definition of the CH-benCHmark: `NATION.N_NATIONKEY = ASCII(SUBSTR(CUSTOMER.C_STATE,1,1))`. This is also the reason for increasing the number of entries in `NATION` from 25 to 62. Notice that this join condition cannot easily be detected as foreign-key relationship.
- Similarly, `SUPPLIER` and `STOCK` can be joined using the following condition: `SUPPLIER.SU_SUPPKEY = MOD(STOCK.S_W_ID * STOCK.S_I_ID, 10000)`. Again, this is not easily detected as a foreign-key relationship.

Regarding the workload, the CH-benCHmark uses the five transactions defined in TPC-C for the OLTP workload. In contrast to TPC-C, an OLTP client randomly chooses a warehouse, and there is no correlation between the number of warehouses and the number of clients.

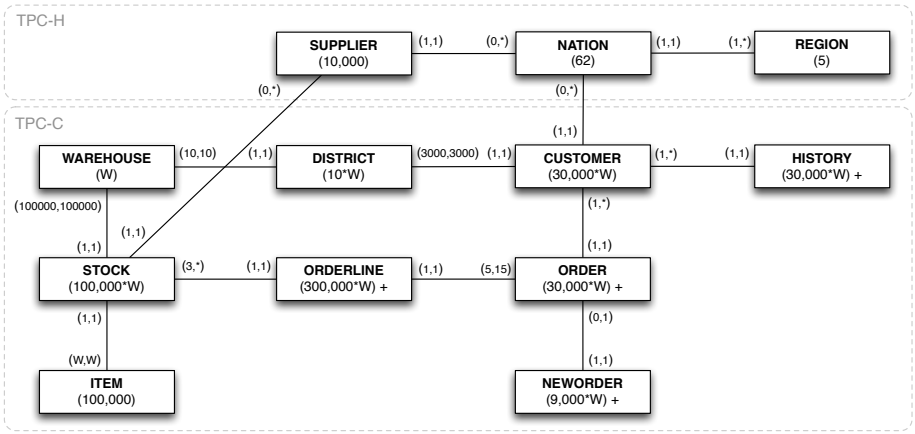


Fig. 4. Schema of the CH-benCHmark.

The OLAP workload is based on the 22 queries defined in TPC-H, but adapted to the modified schema (see Figure 4). A client either executes the OLTP workload or the OLAP workload. Hence, the number of clients for each type of workload can be scaled independently.

4.2 Adapting the CH-benCHmark

As discussed above, the schemas of TPC-C and TPC-H were originally integrated in an ad-hoc fashion using expressions in the join conditions of the queries. For foreign-key relationships, as defined between tables `CUSTOMER` and `NATION` as well as tables `SUPPLIER` and `STOCK`, real-world schemas would avoid such expressions. This leads us to the decision to materialize the join expressions explicitly in the database because it allows us to use standard equi-joins for queries joining these tables. Thus, we introduce the following:

- A column `C_N_NATIONKEY` in table `CUSTOMER` computed as `ASCII(SUBSTR( CUSTOMER.C STATE,1,1))`.
- A column `S_SU_SUPPKEY` in table `STOCK` computed as `MOD(STOCK.S_W_ID * STOCK.S_I_ID, 10000)`.

4.3 Implementing the CH-benCHmark

The core of our implementation is a C++ program that can be started as a:

- *OLAP or OLTP client*, which generates the test workload, and measures benchmark performance values.
- *Server sampler*, which runs on the DBMS host and periodically captures performance metrics, such as system utilization, with a configurable interval.
- *Benchmark coordinator*, which manages all benchmark processes.

To be able to connect to various DBMS, we use unixODBC, and access SAP HANA via the ODBC interface. This allows us to connect to various databases without changing the benchmarking code, and it still allows us to use proprietary SQL extensions. We use SQL prepared statements for both OLTP and OLAP, and compile these during the initialization phase of our benchmark. Transactions are fully interactive and managed by standard ODBC calls.

For HyPer, we use its available client interface, as it does not offer an ODBC interface yet. We use pre-compiled statements for OLTP (non-interactive), and send SQL statements for the OLAP workload. The caching of OLAP query plans in HyPer, however, is similar to using prepared statements.

5 Experimental Evaluation

In this section, we evaluate and compare SAP HANA (see Section 2) and HyPer (see Section 3) using the CH-benCHmark (see Section 4). First, we detail the experimental configuration, how we setup each system, and the performance

metrics we measure (see Section 5.1). Then, we present the results of the experimental evaluation for SAP HANA (see Section 5.2) and HyPer (see Section 5.3), while detailing the implications of the results: how data freshness, flexibility, and scheduling affect the performance of mixed workloads.

5.1 Experimental Configuration

To execute the CH-benCHmark, we use a server that has eight ten-core processors Intel Xeon E7-8870 at 2.40GHz, with hyper-threading enabled (for a total of 160 hardware contexts), and 1TB of RAM. Each core has a 32KB L1 cache and a 256KB L2 cache. Each processor has a 30MB L3 cache, shared by all its cores. The OS is a 64-bit SMP Linux (SuSE), with a 3.0 kernel. We use read committed for the isolation level of SAP HANA. HyPer executes transactions using timestamp ordering with a single thread.

In our experiments we use a one minute warm-up period, followed by a five minutes period to collect throughput information. We use 100 warehouses, which amount to 6.7GB of raw CSV files to be imported. We note that we observe similar trends for a higher number of warehouses. We are, however, more interested in assessing the scalability of concurrency than the increase in data size. We scale the number of OLAP and OLTP clients exponentially (power of 2) between 0 and 2^7 leading to 81 different combinations. Since the result of combination 0/0 is trivial, we are left with 80 combinations for the clients of the mixed workload.

For an OLTP client, the benchmark reports throughput in tpmC, as defined in TPC-C, i.e., the number of successful new order transactions per minute. For an OLAP client, throughput is reported in QphH, i.e., the finished TPC-H queries per hour. Defining an aggregated metric for the whole benchmark is difficult in practice, and thus we follow the original benchmark proposal and analyze both measures independently.

For each system, we present a figure showing the analytical throughput of all combinations of OLTP and OLAP clients, and another figure showing the transactional throughput of all combinations. In this pair of plots, each experiment is displayed twice. As an example we refer the reader to Figure 5, where the black bar (T=32) in section A=8 represents a single experiment with 8 analytical (OLAP) and 32 transactional (OLTP) clients. We also measure the average CPU utilization of the host machine as we increase the load.

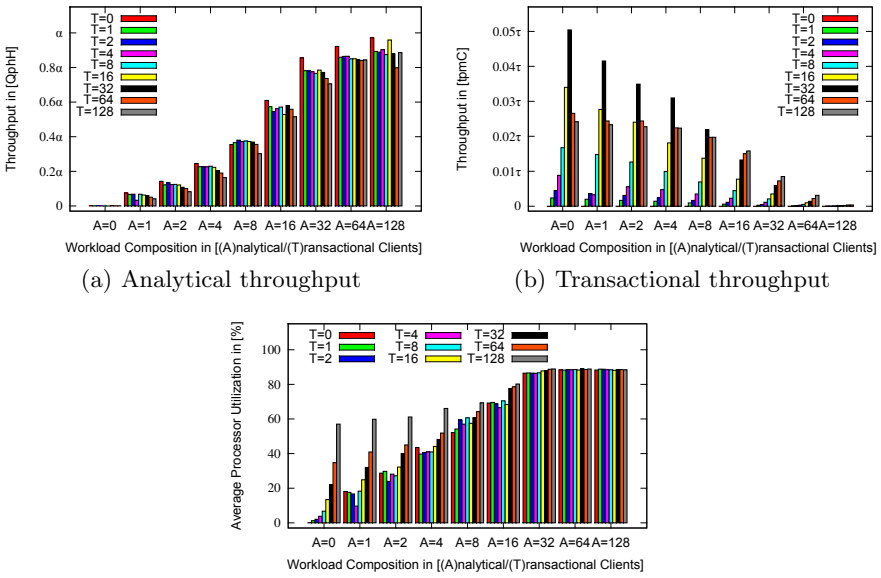
Due to legal reasons, we do not disclose absolute numbers. For this reason, all throughput results are normalized to undisclosed constants α for OLAP and τ for OLTP, where α and τ are the maximum observed throughput values for OLAP and OLTP respectively. This does not hinder us from showing the implications of our experiments, because our focus is on the scalability of the mixed workload as we increase the number of clients, and comparing SAP HANA and HyPer as to how they handle mixed workloads.

5.2 Experimental Evaluation of SAP HANA

Figure 5 shows the performance of the default configuration of SAP HANA as we scale the mixed workload. Figure 5a shows how analytical throughput scales as we increase the number of analytical clients. For each case of analytical clients, we also show how analytical throughput scales as we increase the number of transactional clients. As shown in the figures, analytical throughput increases almost linearly up to 32 analytical clients. After that, as the system gets saturated (see Figure 5c), the increase of throughput levels out.

Figure 5b demonstrates the scaling behavior of the transactional throughput as we increase the number of analytical clients. For a small number of concurrent OLAP clients (up to 8), transactional throughput generally increases as we increase the number of OLTP clients up to 32, after which, OLTP throughput drops. This is due primarily to the fact that more and more transactions contend for modifying common data, resulting in higher abort rates, and, secondarily, in increased synchronization overhead (in the latches of the deltas' indexes). As we add more OLAP clients, overall transactional throughput is generally hurt, as it almost reaches zero throughput for the case of 128 concurrent analytical clients.

We call this scaling behavior the *house pattern*, due to the increasing overall OLAP throughput and the decreasing overall OLTP throughput as we increase the number of OLAP clients. This effect is intrinsic to the behavior of not distinguishing between short-lived transactions and complex analytical queries. The scheduler of SAP HANA employs the machine's resources for analytical queries

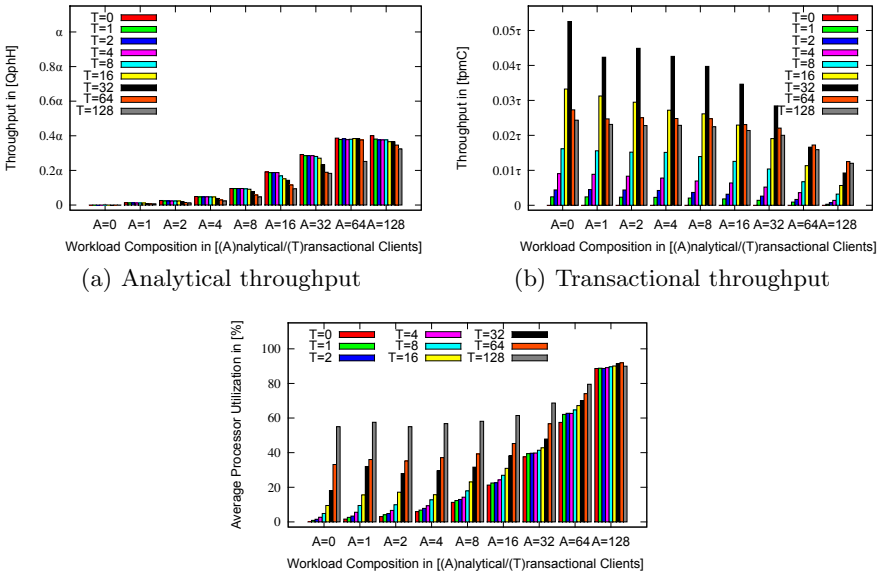


(a) Analytical throughput

(b) Transactional throughput

(c) Average CPU utilization

Fig. 5. Performance of the default configuration of SAP HANA.



(a) Analytical throughput

(b) Transactional throughput

(c) Average CPU utilization

Fig. 6. Performance of SAP HANA when intra-query parallelism is disabled.

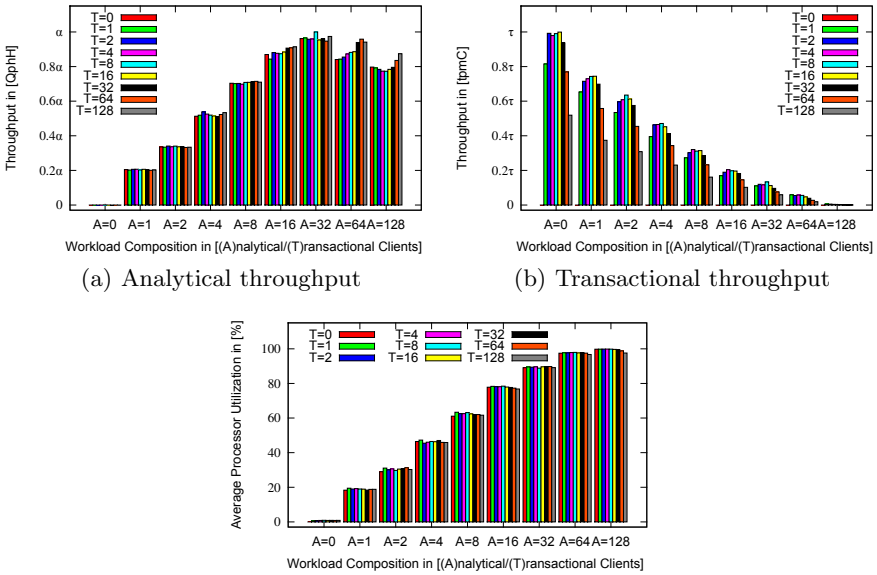
for long durations, and does not leave enough space for the continuously arriving short-lived OLTP transactions. That is why, as we add more OLAP clients, overall OLTP throughput decreases.

To reinforce our argument, we evaluate SAP HANA under a configuration which disables intra-query parallelism, and decreases the effect of analytical queries overwhelming execution. We show the results in Figure 6. In this configuration, OLTP transactions and OLAP queries are mostly executed with a single thread (or task) each. OLAP throughput is overall lower than the default configuration, since queries do not benefit from parallel execution any more. Still, OLAP throughput increases as we increase the number of OLAP clients. The positive effect is that OLTP throughput is overall improved in comparison to the default configuration. System utilization is lower than the default configuration, and is only saturated for 128 analytical clients.

5.3 Experimental Evaluation of HyPer

In Figure 7 we show the experimental results for the most performant case of HyPer. In this case, we keep the initial snapshot for OLAP clients throughout the whole experiment duration, i.e., OLAP clients do not see any updates from the OLTP clients. This configuration minimizes the overhead of creating snapshots, and minimizes any interference between the OLTP and OLAP workloads.

As we see in Figure 7a, the analytical throughput increases as we add more analytical clients, reaching the maximum at around 32 analytical clients. Addi-



(a) Analytical throughput

(b) Transactional throughput

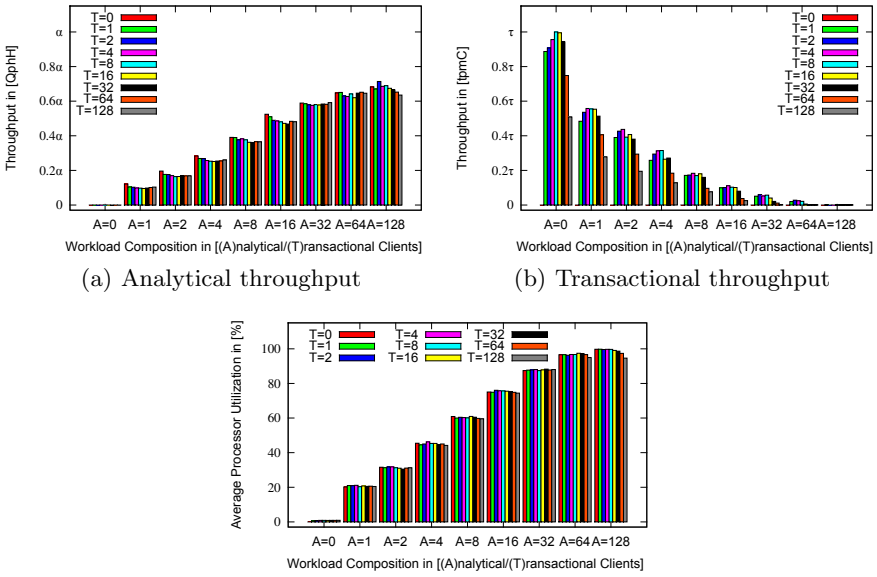
(c) Average CPU utilization

Fig. 7. Performance of HyPer with the lowest level of analytical data freshness.

tional analytical clients drop analytical throughput slightly, due to overwhelming the system with threads. Limiting the overall number of used threads, similar to SAP HANA's task scheduler, can avoid this effect. In comparison to SAP HANA, analytical throughput reaches almost the same maximum, indicating that both systems are similar in parallelizing and executing analytical queries. Also, analytical throughput is not affected by scaling the transactional clients. This is expected, since transactions are executed separately with a single thread.

Transactional throughput, as shown in Figure 7b, is significantly higher (up to an order of magnitude) than that of SAP HANA. This is attributed to several reasons including: (a) transactions are non-interactive whereas transactions in SAP HANA are interactive (with multiple round-trips to the client as defined in TPC-C), (b) transactions are pre-compiled for fast execution, and (c) a single thread executes transactions serially, avoiding any synchronization overhead. Conceptually, we can place HyPer to the left-most part of Figure 1b, and place SAP HANA to the right-most part of the figure.

The trend of the OLTP throughput, however, is similar to SAP HANA. Firstly, we notice a similar drop in throughput for more than 32 OLTP clients, for most experiments. As with SAP HANA, numerous OLTP clients target common data, and result in high abort rates. Secondly, we also identify the same *house pattern* as in SAP HANA: while we increase the number of analytical clients, overall OLTP throughput drops and reaches almost zero for the case of 128 concurrent OLAP clients. Both SAP HANA and HyPer fall in the left-most part of Figure 1c: under cases of high concurrency and saturated resources, the



(a) Analytical throughput

(b) Transactional throughput

(c) Average CPU utilization

Fig. 8. Performance of HyPer with an intermediate level of analytical data freshness.

scheduler favors analytics over transactions. This shows a need for advanced workload management for mixed workloads, that can enable the DBMS administrator to dynamically tip the scales of performance to either analytics or transactions, choosing a spot across the whole span of the line of Figure 1c.

Next, we show how the performance is affected by a different level of data freshness. Figure 8 shows the performance of an intermediate level of data freshness, where every OLAP client takes a new snapshot from the OLTP process after executing all queries of TPC-H (after every 22 queries). Performance is overall decreased in comparison to the best performant case of the lowest level of data freshness, supporting our expectations (see Figure 1a). Analytical throughput is decreased by around 40%. Transactional throughput is decreased as soon as the first OLAP client is added, by around 30%. This is mainly due to the overhead of forking the OLTP process to create snapshots for OLAP clients. It actually interrupts the single-threaded OLTP process and presents an overhead.

We note that increasing the level of data freshness further is not desirable in HyPer because the extremely frequent forks at a fine granularity can significantly deteriorate performance. In such cases, a sort of *snapshot bundling* can be implemented to decrease the snapshot overhead at a small expense of data freshness: instead of every OLAP client forking the OLTP process, several OLAP clients can be batched and serviced on a single snapshot.

While SAP HANA aims for the highest level of data freshness, HyPer provides the opportunity to the DBMS administrator to choose the level of data freshness for analytics. This is a desirable property when it is acceptable not to consider

the latest updates in reports. For cases where extreme real-time reporting is required, SAP HANA's approach to executing both OLTP and OLAP workloads on common data structures can be better for analytical throughput.

6 Conclusions

In this paper, we analyze two state-of-the-art main-memory DBMS for mixed workloads: SAP HANA and HyPer as they promise high performance for mixed OLTP and OLAP workloads. For our experimental evaluation, we evaluate the CH-benCHmark by scaling the number of concurrent transactional and analytical clients. Through our evaluation, we find that the most important factors that affect the performance of mixed workloads are (a) data freshness, i.e., how recent is the data that analytical queries are processing, (b) flexibility, i.e., optimizing the performance of transactions and queries by restricting interactivity and/or expressiveness, and (c) scheduling, i.e., how the DBMS utilizes resources for OLTP and OLAP clients.

Concerning data freshness, SAP HANA's design, where OLTP and OLAP clients target common data, is suited for cases where the highest level of data freshness is required, whereas HyPer's design is suitable for cases where the DBMS administrator wishes to toggle the trade-off between performance and data freshness. Concerning flexibility, we show that HyPer's less interactive statements allow for pre-compilation and achieve a very high transactional throughput. Finally, concerning scheduling, we show that both systems exhibit a house pattern, i.e., increasing OLAP clients can significantly hurt OLTP throughput in cases of high concurrency and saturated resources. This behavior stresses the need for workload management in mixed workloads, where OLTP statements can be distinguished from OLAP statements and can be prioritized differently. Quantifying this effect of priorities in more detail is part of our future work.

This analysis indicates that it is difficult to achieve maximum performance for both OLAP and OLTP while at the same time working on the freshest data. Both systems have made a significant step towards the vision of supporting mixed workloads in a single database system. Requirements regarding the freshness of data certainly depend on the application requirements, and it will be important to analyze this aspect further in real-world applications. Striking a good balance between high and stable OLTP throughput while at the same time offering efficient OLAP performance still seems to be solved partially only. We plan to investigate this topic further as part of our future work.

References

1. SAP HANA Live for SAP Business Suite. `http://help.sap.com/hba` (2014)
2. Transaction processing performance council. `http://www.tpc.org` (2014)
3. Alagiannis, I., Idreos, S., Ailamaki, A.: H2O: A Hands-free Adaptive Store. SIGMOD (2014)

4. Cao, T., Salles, M.A.V., Sowell, B., Yue, Y., Demers, A.J., Gehrke, J., White, W.M.: Fast checkpoint recovery algorithms for frequently consistent applications. In: SIGMOD (2011)
5. Cole, R., Funke, F., Giakoumakis, L., Guy, W., Kemper, A., Krompass, S., Kuno, H.A., Nambiar, R.O., Neumann, T., Poess, M., Sattler, K.U., Seibold, M., Simon, E., Waas, F.: The mixed workload CH-benCHmark. In: DBTest (2011)
6. Difallah, D.E., Pavlo, A., Curino, C., Cudré-Mauroux, P.: OLTP-Bench: An extensible testbed for benchmarking relational databases. PVLDB 7(4) (2014)
7. Färber, F., May, N., Lehner, W., Große, P., Müller, I., Rauhe, H., Dees, J.: The SAP HANA database – an architecture overview. IEEE Data Eng. Bull. 35(1), 28–33 (2012)
8. Florescu, D., Kossmann, D.: Rethinking cost and performance of database systems. SIGMOD Rec. 38(1), 43–48 (Jun 2009)
9. Grund, M., Krüger, J., Plattner, H., Zeier, A., Cudre-Mauroux, P., Madden, S.: HYRISE: A Main Memory Hybrid Storage Engine. PVLDB 4(2) (2010)
10. Kemper, A., Neumann, T.: HyPer: A hybrid OLTP&OLAP main memory database system based on virtual memory snapshots. In: ICDE (2011)
11. Lee, J., Kwon, Y.S., Färber, F., Muehle, M., Lee, C., Bensberg, C., Lee, J.Y., Lee, A.H., Lehner, W.: SAP HANA distributed in-memory database system: Transaction, session, and metadata management. In: ICDE (2013)
12. Leis, V., Boncz, P., Kemper, A., Neumann, T.: Morsel-driven parallelism: A NUMA-aware query evaluation framework for the many-core age. In: SIGMOD (2014), to appear
13. Leis, V., Kemper, A., Neumann, T.: Exploiting hardware transactional memory in main-memory databases. In: ICDE (2014)
14. Neumann, T.: Efficiently compiling efficient query plans for modern hardware. In: VLDB (2011)
15. Nguyen, T.M., Schiefer, J., Tjoa, A.M.: Sense & response service architecture (saresa): An approach towards a real-time business intelligence solution and its use for a fraud detection application. In: Proceedings of the 8th ACM International Workshop on Data Warehousing and OLAP (2005)
16. Olofson, C., Morris, H.: Blending transactions and analytics in a single in-memory platform: Key to the real-time enterprise. Tech. rep., IDC (Feb 2013), `http://www.saphana.com/docs/DOC-4132`
17. Plattner, H.: A common database approach for OLTP and OLAP using an in-memory column database. In: SIGMOD (2009)
18. Psaroudakis, I., Scheuer, T., May, N., Ailamaki, A.: Task scheduling for highly concurrent analytical and transactional main-memory workloads. In: ADMS (2013)
19. Raman, V., Attaluri, G., Barber, R., Chainani, N., Kalmuk, D., Samy, V.K., Leenstra, J., Lightstone, S., Liu, S., Lohman, G.M., Malkemus, T., Mueller, R., Pandis, I., Schiefer, B., Sharpe, D., Sidle, R., Storm, A., Zhang, L.: DB2 with BLU acceleration: So much more than just a column store. In: VLDB (2013)
20. Stonebraker, M., Cetintemel, U.: "One Size Fits All": An idea whose time has come and gone. In: ICDE (2005)
21. Stonebraker, M., Weisberg, A.: The VoltDB main memory DBMS. IEEE Data Eng. Bull. 36(2) (2013)