Adaptive HTAP through Elastic Resource Scheduling

Aunn Raza
aunn.raza@epfl.ch
Ecole Polytechnique Fédérale de Lausanne

Periklis Chrysogelos
periklis.chrysogelos@epfl.ch
Ecole Polytechnique Fédérale de Lausanne

Angelos Christos Anadiotis
angelos.anadiotis@polytechnique.edu
Ecole Polytechnique

Anastasia Ailamaki
anastasia.ailamaki@epfl.ch
Ecole Polytechnique Fédérale de Lausanne
RAW Labs SA

ABSTRACT

Modern Hybrid Transactional/Analytical Processing (HTAP) systems use an integrated data processing engine that performs analytics on fresh data, which are ingested from a transactional engine. HTAP systems typically consider data freshness at design time, and are optimized for a fixed range of freshness requirements, addressed at a performance cost for either OLTP or OLAP. The data freshness and the performance requirements of both engines, however, may vary with the workload.

We approach HTAP as a scheduling problem, addressed at runtime through elastic resource management. We model an HTAP system as a set of three individual engines: an OLTP, an OLAP and a Resource and Data Exchange (RDE) engine. We devise a scheduling algorithm which traverses the HTAP design spectrum through elastic resource management, to meet the workload data freshness requirements. We propose an in-memory system design which is non-intrusive to the current state-of-art OLTP and OLAP engines, and we use it to evaluate the performance of our approach. Our evaluation shows that the performance benefit of our system for OLAP queries increases over time, reaching up to 50% compared to static schedules for 100 query sequences, while maintaining a small, and controlled, drop in the OLTP throughput.

CCS CONCEPTS

• **Information systems → DBMS engine architectures**; **Main memory engines**; *Database transaction processing*; *Online analytical processing engines.*

ACM Reference Format:
Aunn Raza, Periklis Chrysogelos, Angelos Christos Anadiotis, and Anastasia Ailamaki. 2020. Adaptive HTAP through Elastic Resource Scheduling. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD'20), June 14–19, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3318464.3389783

1 INTRODUCTION

Modern business analytics use Hybrid Transactional Analytical Processing (HTAP) systems, where an Online Transaction Processing (OLTP) engine continuously updates the state of the database which serves Online Analytical Processing (OLAP) queries. HTAP introduces *data freshness* as an additional dimension to analytical data processing. As data freshness depends on transactional throughput, an ideal HTAP system provides analytical processing over fresh data without affecting the performance of the transactional engine. Similarly, analytical processing requires that query response times remain unaffected by data freshness traffic.

There is no free lunch: no matter which mechanism is used to guarantee query execution over fresh data, the performance of either the OLTP or the OLAP part of the HTAP system deteriorates. We isolate the engines by using two distinct CPU sockets for each engine, in a four socket server and show in Figure 1 the two extremes of the HTAP design spectrum which incur different overheads for each of the two engines. OLTP runs TPC-C NewOrder transaction with one warehouse per worker thread. To show the performance tradeoff, we run the same query 16 times and vary the snapshotting frequency by taking a new snapshot (after OLTP has made updates) at every query, then every two queries, up to sixteen queries, as depicted on the x-axis. We measure average query execution time and transactional throughput as per values on the left- and right-hand y-axis, respectively.

The experiment denoted "ETL" transfers data from the OLTP to the OLAP engine and then executes the analytical query. Since query execution starts after all updates have been applied to the OLAP engine, the end-to-end query response time is higher. Nevertheless, the cost is amortized

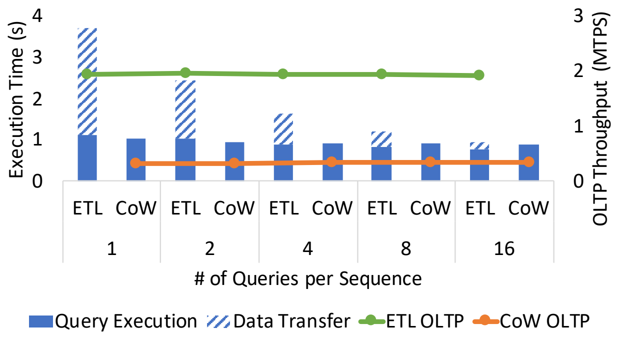


Figure 1: Trade-off between ETL- & CoW-based HTAP.

across all queries executed on top of the same snapshot. The experiment denoted "CoW" relies on hardware-supported Copy-on-Write to allow analytical query execution while running transactions. When the OLTP engine writes a record while a query is running, operating system first duplicates the affected page, and the write is applied on the copy, thereby ensuring consistency while query execution time remains intact. Copying pages, however, makes OLTP performance deteriorate. Without concurrent OLAP queries, OLTP throughput is the same across both experiments (around 2 MTPS).

Current state-of-the-art HTAP systems consider data freshness requirements in their design and they can be classified under two categories, based on their storage design. The first category includes a unified storage, where transactional and analytical workloads are executed concurrently using snapshotting mechanisms. This category is optimal for analytical workloads where every query accesses further fresh data than the previous one. The second category includes a decoupled storage, where the OLTP and the OLAP part of the system are executed in isolation and fresh data are transferred from the OLTP to the OLAP part upon request. This category is optimal for workloads with batches of queries that require the same level of data freshness, or when the data that they need to access is not frequently updated.

However, the amount of fresh data that a query needs to access depends on the workload [6, 8], whereas it can vary at runtime. For instance, reporting workloads that typically run daily and in batches can be efficiently supported through an ETL process. On the other hand, real-time statistics workloads require immediate access to fresh data. Finally, monitoring workloads have queries partly accessing fresh data. Therefore, a generic HTAP system needs to support workloads accessing different parts of the database which are updated with different frequency, while providing freshness and performance guarantees for both OLTP and OLAP. Accordingly, the HTAP system has to adapt its design to the workload requirements, at run-time. In this paper, we refer to the amount of fresh data that a query needs to access, as its *data freshness requirements*. Accordingly, we describe the design and implementation of an HTAP system which supports query workloads with variable data freshness requirements. Our system adapts to the freshness requirements of the workload, while controlling interference between the transactional and the analytical part, through a scheduling algorithm which distributes compute and memory resources across the OLTP and the OLAP engines. Our scheduling algorithm is enforced through a system which is non-intrusive to the design of standard in-memory OLTP and OLAP engines and can traverse the HTAP system design spectrum by changing the distribution of resources between the two engines. Accordingly, we treat HTAP primarily as a resource scheduling problem, where the OLTP and the OLAP engine compete for achieving maximum fresh data locality. In summary, we make the following contributions:

- We approach HTAP as a resource scheduling problem, and we specify a set of states that the system adapts based on the workload requirements for fresh data and the availability of the compute and memory resources that can be exchanged between the engines.
- We demonstrate that, compared to static designs, adaptive resource scheduling increases the performance of OLAP query execution up to 50% for 100 OLAP query sequences while maintaining a small, and controlled, drop in the OLTP throughput.

2 BACKGROUND & RELATED WORK

In this section, we first provide the definition and the metric for data freshness that are relevant to the HTAP problem that we study. Then, we classify existing HTAP approaches based on their storage organization. Finally we present relevant approaches on hybrid workload scheduling.

2.1 Data Freshness

In the HTAP design space that we study, we consider two engines, one OLTP and one OLAP, which can be either logically or physically separated. Each engine stores data in its private storage, and we allow them to access each other's storage through predefined access paths.We define as *fresh*, the data resulting from modifications executed by the OLTP engine which are not present in the OLAP private storage when an analytical query arrives. Hence, the fresh data can be accessed by the OLAP engine either by first copying them to its private storage, or by accessing the OLTP storage directly through access paths exposed by the OLTP engine. Following the definitions in [4], we measure data freshness with the *freshness-rate metric*, defined as the rate of tuples that are the same between the private storage of the two engines, over the overall amount of tuples. Accordingly, when

Table 1: HTAP Design Classification

HTAP Storage	System	Snapshot Mechanism	Freshness-Perf. Trade-off
Unified Storage	HyPer-Fork [19], Caldera [2]	CoW	OLTP (CoW)
	HyPer-MVOCC [29], MemSQL, IBM BLU [32]	MVCC	OLAP (version traversal)
	SAP HANA [13]	Delta-Versioning	OLAP (version traversal), OLTP (record chains)
Decoupled Storage	BatchDB [24]	Batch-ETL	OLAP (ETL latency)
	Microsoft SQL Server [21]	MVCC-Delta	OLAP (tail-records scan)
	Oracle Dual-format [20]	Txn Journal & ETL	OLAP (tail-records scan)

the two engines share the same data storage, the freshness-rate metric will always be 1. Instead, when their storage is independent, this metric will generally be less than 1.

In HTAP, the analytical queries are executed on top of a data snapshot with freshness-rate metric equal to 1. This is achieved either by the two engines sharing the same data storage, or by transferring the corresponding delta from the transactional to the analytical storage before the query is executed. In this paper, we study the performance trade-offs of every approach on the transactional and the analytical engine, and we adapt our system design based on the freshness-rate metric, which we measure at query-level.

2.2 HTAP engine design

We classify the existing HTAP systems, based on their storage design, in two high level categories: (i) Unified storage, and, (ii) Decoupled storage. In unified storage, the HTAP system maintains a single consistent snapshot of data for analytical and transactional processing, and isolation between the two engines is achieved through snapshotting. In decoupled storage, the HTAP system maintains a separate storage for analytical and transactional processing, hence replicating and optimizing data formats while extracting data from the transactional engine, transforming them into the appropriate format, and loading them into the analytical engine. Table 1 shows the classification of existing HTAP systems and provides information on the mechanism used for acquiring fresh data snapshots as well as on the trade-offs between performance and data freshness.

Unified Storage. The first version of HyPer [19] relies on partitioned-serial execution for concurrency control among transactions. Analytical queries are executed on isolated snapshots which are taken lazily upon conflicting access between the transactional and the analytical part of the system. The snapshot isolation mechanism is based on CoW and Hyper uses UNIX fork to start a new process when an analytical query arrives thereby providing immediate access on the fresh data to the analytical engine. Caldera [2] is an HTAP prototype system employing GPUs for analytical query execution, which also relies on CoW using page-shadowing.

The most recent version of Hyper [29] relies on optimistic multi-version concurrency control (MVOCC) to mediate access among transactional and analytical queries. Similarly, MemSQL and IBM BLU [32] are commercial HTAP Systems that employ MVCC for snapshot isolation and a tunable option for each table to either store it as a column-major or row-major formats. SAP HANA [13] is another commercial DBMS system that provides HTAP capabilities using a variant of MVCC-based storage. SAP HANA maintains a consistent OLAP-optimized main and OLTP-optimized delta storage which is periodically merged into the main storage.

HTAP systems with unified storage opt for analytical data freshness using either CoW [25] or multi-versioning [34] for snapshot isolation. However, such systems provide data freshness at the cost of performance of either or both analytical and transactional workloads. Specifically, in CoW, when the OLTP engine updates a record, it has to do a full page copy first, thereby trading transactional for analytical performance. On the other hand, by using MVCC alone, the OLAP engine has to traverse the versions kept by the OLTP engine, trading analytical for transactional performance due to random memory accesses. Delta-versioning is a hybrid technique which maintains one OLTP- and one OLAP-optimized snapshot, with second being periodically updated by the first one. This approach is fairer since both engines lose performance for accessing each other's snapshot, the OLTP for reading what was recently migrated to OLAP and the OLAP for reading the recently updated data from OLTP. The trade-offs between data freshness and transactional or analytical performance are further analyzed in [31].

Decoupled Storage. Traditionally, in data warehousing, the data is extracted from transactional stores, and then transformed and loaded into analytical data stores, during an ETL process. Recently, HTAP systems are challenged with more frequent update-propagation mechanisms to provide high data freshness rates. BatchDB [24] schedules OLTP and

OLAP engines across isolation boundaries, for instance different NUMA nodes, and uses a mini batching technique to propagate transactional logs to the analytical data store, either periodically or on-demand. Oracle's dual-format [20] maintains OLTP optimized row-major and OLAP optimized column-major data in-memory. Microsoft SQL Server[21] also maintains two copies of data and propagates data to OLAP storage through an intermediate delta storage to avoid overheads of merging transactionally-hot records.

Decoupled storage has been the conventional way of linking OLTP with OLAP databases. Batch-ETL provides isolation between the engines, at the cost of OLAP performance for transferring the data from the OLTP snapshot. However, this cost is amortized by the execution of query batches. In the MVCC-Delta and the Txn Journal & ETL approach, the corresponding systems keep a separate snapshot for OLTP and OLAP, and they either transfer periodically the recent versions maintained by the MVCC protocol to the OLAP side, or they use the transactional journal log to do the transfer.

Hybrid workload scheduling. HTAP workload scheduling can be considered as a subclass of the hybrid workload scheduling problem, where the goal is to achieve fairness, while avoiding interference across workloads. Works in data-center and cluster scheduling [17] [11] achieve performance isolation between small and large jobs, reducing tail-latencies and head-of-line blocking in a shared cluster. Our approach follows the same principles by elastically trading resources between engines through an elastic HTAP scheduler which guarantees that no engine will starve, while allowing every individual engine to optimize their own schedule internally. However, our focus is on scale-up servers, where the trade-offs are different than scale-out clusters, whereas the metric driving the scheduler decision is the data freshness rate.

While, run-time resource scaling [10] and performance isolation [26] are well-studied topics in DBMS scheduling, existing work focuses on independent workloads. In contrast, HTAP introduces data dependencies due to the data freshness requirements. This causes interference due to implicit (memory bandwidth, CPU caches, hyper-threads) and explicit (locking/latching) resource sharing caused by OLTP and OLAP's concurrent data accesses. Furthermore, works focusing on scheduling in combination with HTAP traditionally consider the location of the data as input to decide task placement and, then, the data access method. For instance, the work of Dziedzic et al. [12] distinguishes and optimizes hybrid workloads by adapting the access methods, that are, B+-tree, column-store index or hybrid, at runtime. In our approach, the location of the data is decided by a scheduler, which indicates whether fresh data should be moved from the OLTP to the OLAP engine, or accessed remotely. Based on resource availability, the scheduler decides how to access the data given the amount of fresh data required by every query. As we consider an independent storage for each engine, we can physically isolate them and then control interference at the CPU and the memory level by changing the distribution of CPUs across the engines and the data access methods.

3 SYSTEM DESIGN & IMPLEMENTATION

This Section describes the design of our HTAP system. As we consider two independent engines that can work together through a thin, scheduling layer, we split our description into four parts. First, we provide an overview of the overall system to give the big picture and link with the design choices of each individual engine. Then, we provide the technical description of the OLTP and the OLAP engines and show that our design is non-intrusive to the standard design of such in-memory engines. Finally, we explain how data and resources are exchanged between the OLTP and the OLAP engine through the Resource and Data Exchange engine.

3.1 Overview

Hybrid Transactional/Analytical Processing addresses workloads where analytical queries process fresh data. In the presence of both the OLTP and the OLAP engine, data freshness is guaranteed either by sharing the storage of the OLTP engine or by copying a part of it to the OLAP engine. There are two important challenges associated with achieving query execution over fresh data: (i) explicit resource sharing causes interference both at the software (extensive copy-on-write or non-contiguous scans) and at the hardware level (sharing CPUs and memory bus), and, (ii) data copying imposes an increased latency which may not be tolerable by the workload. We address these challenges through an elastic system design which adapts to the workload requirements.

Figure 2 depicts the architecture of our HTAP system, which achieves resource isolation and sharing on demand, with respect to the performance requirements of each engine. Each engine executes its workload independently, while they have a one-way dependency: the OLAP engine reads fresh data from the OLTP. As sessions do not span across the engines, each one maintains its own request queue and executes requests independently. The RDE engine assigns resources (CPU and memory) and data to the engines. The resource distribution between the engines is decided by the RDE engine. By introducing RDE as an integration layer, we achieve an adaptive HTAP system design that is minimally intrusive to the design of existing OLTP and OLAP engines.

3.2 OLTP engine

The OLTP engine is depicted on the left-hand side of Figure 2. Following the standard in-memory OLTP system design, our engine includes a Storage Manager (SM), a Transaction Manager (TM) and a Worker pool Manager (WM).

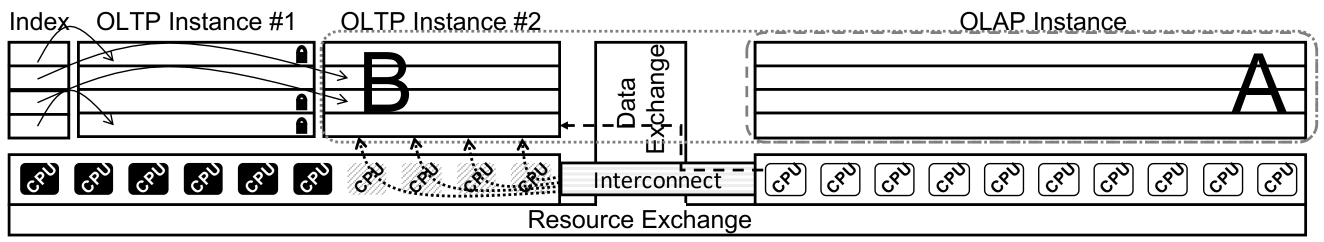


Figure 2: System architecture

Storage Manager. The SM stores data entirely in the main memory in columnar format. Following an approach similar to Twin Blocks [5] and Twin Tuples [25], the SM maintains two *instances* of the data, in addition to a multi-versioned storage. The difference from existing approaches is that each instance keeps data in a columnar layout, to allow the OLAP engine to perform fast scans over the data without having to traverse specific versions. At every time point, only one of the two instances is active. The OLTP engine also maintains an index, implemented using cuckoo hashing [30]. The index always points to the last updated record in either of the two instances. Therefore, even though the inactive instance will remain out-of-date, upon switch, the data will always come from the newest version. The OLTP engine also maintains a delta storage to allow transactions to traverse older versions of the objects in Newest-to-Oldest ordering [34], following the standard multi-versioned concurrency control (MVCC) process. The active instance of the SM is switched upon request. The SM provides an API to switch the instance, which returns the starting address of the inactive instance when no active OLTP worker thread is using it any more. The SM also maintains instance statistics per column, which are the number of records at the time of switch, a flag indicating if the column contains updated-tuples and the epoch number. The SM also maintains an *update indication bit* per record, which is set when the record gets updated. Access to the update indication bits is synchronized using atomic operations, as they can be accessed by different parts of the system.

Transaction Manager. Together with the SM, the TM provides transactional access to the database records by relying on a two-phase locking (MV2PL [34]) concurrency control protocol with deadlock avoidance [1] and transactional isolation level of snapshot isolation [3]. When a transaction tries to access a record, it brings from the index its most recent value. This value may either be on the currently active instance or on the inactive one. When a transaction starts, it requests the SM for the starting address of the column which is on the currently active instance, and performs all the operations there, sets the update indication bit for that record, and then updates the index, if needed. Every update is directly placed on the active instance upon transaction commit, and the older version is pushed to the versioned storage. This provides a significant speedup to the OLAP query execution since it allows sequential scans. Inserts are pushed to both instances, but they are made available through the inactive instance only after a switch.

Worker Manager. The OLTP engine uses one hardware thread per transaction. The WM keeps a worker pool of active threads. We set each thread to first generate a transaction and then execute it, simulating a full transaction queue. The WM exposes an API to set the number of active worker threads and their CPU affinities, thus enabling the OLTP engine can elastically scale up and down upon request.

3.3 OLAP engine

The OLAP engine is based on Proteus [7, 18], a parallel, NUMA-aware, query execution engine with support for CPU-GPU execution [7]. The engine uses code generation to specialize the executed code to each query [28], compute device [7] and access path [18]. Our design considers CPU as the only compute device available, leaving hardware accelerators as future work. Following the description of the OLTP engine, we describe how the Storage Manager (SM) and Query Executor (QE) of the OLAP engine, work.

Storage Manager. The SM considers data stored in the main-memory and is agnostic of their format and layout. The data access paths are decided by input plugins which specify how tuples are accessed and interpreted, following the database schema and the data format. The input plugins further encapsulate different data access methods. In our HTAP setting, we rely on Proteus support for partitioned data across several (contiguous) memory areas, to access only the fresh data from the OLTP storage and the rest from the OLAP storage.

Query Executor. The OLAP engine generates just-in-time specialized code for each query, using the appropriate plugins to access the data. The query plan is translated into pipelines which execute sequences of operations on input tuples without materializing intermediate results. Each pipeline is transformed into code which is compiled and optimized for the current hardware architecture. By default, the pipelines process one block of tuples at a time. During code generation, the plugins specialize the different access methods, based on the location of the data. The OLAP engine parallelizes query execution by routing blocks between different pipelines that execute concurrently. Each pipeline is assigned to a worker which is affinitized to a CPU core. Based on the placement of the data, the OLAP engine balances the load across worker threads using protocols (hash-based, load-aware, locality-aware and combinations). By default, the OLAP engine uses locality-and-load-aware policies, and schedules blocks to pipelines which are executed locally to the data, if possible.

3.4 RDE engine

The Resource and Data Exchange engine is the integration layer between the OLTP and the OLAP engines and supports the operations required for HTAP. Fundamentally, there are two ways to achieve HTAP and they require two different design choices. The first way is to have the two engines running in isolation, which we assume to be at the socket boundary in a single, scale-up server. HTAP is thus supported by transferring fresh data from the OLTP to the OLAP before executing a query. The second way is to have the two engines sharing their resources, effectively having a single HTAP system with transactional and analytical processing capabilities. Our HTAP system design relies on elastic resource management to traverse the design space between the above two approaches, through the RDE engine.

In the following, we first provide fundamentals for the RDE engine design, then we describe the discrete states the system marking them as S_1 (Co-located OLTP and OLAP), S_2 (Isolated OLTP and OLAP), and S_3 (Hybrid), and finally we explain how the RDE engine elastically migrates between states. The decision on which state to move at each point in time is taken by the system scheduler and enforced by the RDE engine. Our system design is independent of the scheduling algorithm, and therefore, we leave the description of the latter to the next Section and here we focus only on the way our system can adapt to different configurations that are required by different HTAP workloads.

The RDE engine is the owner of memory and CPU resources and distributes them to the OLTP and the OLAP engines. Still, each engine has its own internal scheduler that decides which resources to use based on the workload. For instance, the scheduler of our OLAP engine makes NUMA-aware decisions on the use of multiple nodes for each query operator. Resources that are not accepted by an engine are returned to the RDE and offered to the other engine.

OLTP active instance switching. The OLTP engine provides consistent and fresh data with with snapshot isolation guarantees. To avoid interference with transaction execution every time the OLAP engine needs access to fresh data, the RDE engine instructs the OLTP engine to switch its active instance. However, this creates a freshness-level inconsistency between the two instances, which can increase with time. For instance, if some records get updated every two OLTP instance switches, then they will be fresh in one instance, leaving the other one behind. Upon the instance switch, the RDE engine checks for updates through a hierarchical update-presence flag in the order of schema, relations, and columns. In the presence of updates, the RDE engine traverses the update indication bit and for the records that are updated, it copies them to the other instance, in case they have not been updated there as well by that time. As the number of inserts and updates corresponds to the data freshness rate, the RDE also maintains these statistics which are provided to the scheduler. With careful engineering, this process has a negligible effect in performance, and it is followed every time we refer in the following to switching the active instance of the OLTP engine. It takes around 10ms to sync around 1 million modified tuples in a database of over 1.8 billion records, while executing TPC-C NewOrder transaction. The size of database corresponds to TPC-H scale-factor 300.

Co-located OLTP and OLAP [S_1]. In this state, the two engines share the memory and the CPUs of all the sockets. The CPUs of each socket are distributed to the engines following the decision made by the scheduler. When an OLAP query arrives, the RDE engine instructs the OLTP engine to switch its active instance. The OLTP engine returns the pointer of its inactive instance, which is then used by the OLAP engine to execute the query. This way, the two engines interfere at the hardware, but not at the software level, since the OLAP query is executed on a part of the memory which is not used by the OLTP engine. The OLTP engine continues transaction execution on its own instance of the data.

An example of this is depicted in Figure 2, on the left-hand socket, where black colored CPUs are used for OLTP and the stripped ones are used for OLAP. The black colored CPUs access only OLTP Instance #1, whereas the stripped ones only OLTP Instance #2,a and the RDE engine changes the ownership of each part of the memory between the engines. In this setting, the OLTP engine has access to both the instances, since the index may point to data stored in the instance used by the OLAP engine. As both engines only read data from the second instance, there is no conflicting operation involved and therefore no need for synchronization.

Related systems to [S_1]. This state represents the class of systems which employ co-location of compute and storage resources, like SAP HANA [13] and HyPer-MVOCC [29]. Co-location of hybrid workloads also represents CoW based systems, since the OLAP engine gets fresh snapshots instantly while OLTP still proceeds on secondary data instance. Our design avoids the CoW overheads for OLTP while providing OLAP access to data stored in columnar layout.

Isolated OLTP and OLAP [S_2]. In this case, the two engines run in the highest isolation level with minimal interference.

Each engine receives resources at the granularity of a CPU socket, following the decision made by the scheduler. When an OLAP query arrives, the RDE engine instructs the OLTP engine to switch its active instance. After the OLTP engine returns the pointer to the inactive instance, the RDE engine transfers the data that have been inserted and the data that have been updated since the last time that the instance of the OLAP engine was updated. Updated data are recognized by the update indication bit, which is set by the OLTP engine. For each record transferred to the OLAP engine, the RDE engine clears the corresponding bit. Data transfers are overlapped with OLTP instance synchronization to avoid rereading the same records. Even though CPU-level isolation is achieved throughout the whole query execution time, there is interference at the memory level when reading the data from the socket where the OLTP engine is executed. This interference is limited by the interconnect bandwidth, which is typically several times smaller than the memory bandwidth, whereas the OLTP engine in any case does not fully utilize memory bandwidth due to random memory accesses.

An example of the distribution of resources to the engines is shown in Figure 2, considering that the OLTP engine occupies the full left-hand socket and the OLAP engine the full right-hand side socket with the long dashed line indicating the memory isolation boundary. Supposing that OLTP Instance#2 was active when the query arrived, after the switch, its data are transferred by the RDE engine to OLAP instance through the socket interconnect. Given that analytical query execution cannot start before all data have been transferred, the RDE engine uses resources of OLAP to transfer the data. Therefore, data transfer time is accounted to the overall query execution time, as there is no benefit in hurting the performance of the OLTP engine at this point.

Related systems to [S_2]. This state represents the class of systems with decoupled storage and full compute isolation, like BatchDB [24] and traditional data-warehouse solutions, where a periodic ETL is performed from OLTP to OLAP engines. These systems isolate OLTP and OLAP workloads across hardware boundaries, NUMA or machines, and provide software and hardware level isolation.

Hybrid OLTP and OLAP [S_3]. In this case, the two engines share memory and, if requested, CPU resources. The key aspects of the hybrid approach are: (i) the OLAP engine accesses only the fresh data that it needs for every query, and, (ii) the OLAP engine accesses fresh data either through the interconnect or directly from the socket where the OLTP engine is executed, by taking CPUs from the OLTP engine. Similarly to the previous states, when an OLAP query arrives, the RDE engine instructs the OLTP to switch its active instance and passes the pointer to the inactive instance to the OLAP engine. Then, the OLAP engine either accesses the fresh data through the interconnect, like in the isolated state, or it gets CPUs on the OLTP engine socket and accesses fresh in full memory bandwidth from these CPUs. The scheduler decides which of the two options to be used. The interference in the hybrid approach at the memory level is bounded at the lower side by the interconnect bandwidth, and at the CPU level is bounded at the upper side by the number of CPUs allowed to be passed from the OLTP to the OLAP engine.

An example of the hybrid approach can be reconstructed from Figure 2, by considering cases A and B referring to the first and the second option, respectively. In case A, the OLAP engine uses its own socket and accesses fresh data from the interconnect, as the long-dashed-line arrow indicates. In case B, the OLAP engine uses the stripped CPUs from the socket of the OLTP engine, and after performing some operations on that socket, they send the data back to the main OLAP socket, as the short-dashed-line arrow indicates. Case B is particularly useful for query operators with a big reduction factor (e.g., an aggregation like a count), which would stress the interconnect if case A would be followed.

Related systems to [S_3]. This state represents the class of systems employing hybrid data access techniques, like tail OLTP record scan for OLAP, which is equivalent to accessing fresh data from the OLTP inactive instance in our case, in addition to scanning OLAP-local storage. Representative systems includes Microsoft SQL Server [21], and Oracle Dual-format [20]. Further, state S_3 follows the design of elastic resource allocation in cloud systems to distribute resources between the two engines at runtime.

4 ADAPTIVE HTAP SCHEDULING

The elasticity at the system design level is driven by a scheduler which distributes resources to the OLTP and the OLAP engine, mainly considering the data freshness rate for each query. Accordingly, the scheduler selects a system state and the OLAP engine adapts its resource allocation and data access methods to provide maximum data freshness for each analytical query. As execution ranges across three states: co-location, isolation, and hybrid, the scheduler requests the system to migrate to a state by changing the size of the worker pool of each engine and the affinity of the worker threads. Accordingly, workload execution adapts to the resources that are made available to the engines every time.

4.1 System model

We model the HTAP system as a set of memory and computing resources and we assign them to the engines, which use them internally to optimize their execution. We rely on the scan operators of the analytical query plan to find the data that the query will access and we extract the fraction of fresh data. Further, we assume that data are stored in a columnar layout and that there no additional overheads to scanning other than any potential NUMA effects that are caused by using CPUs that are remote to the data. Observe that NUMA overheads are fully controllable by the scheduler, since it distributes the resources to the engines. Finally, we assume that the database administrator can set restrictions on the amount of resources that can be revoked from the OLTP and the OLAP engine, to abide with performance guarantees and control interference between the engines.

We do not consider any optimizations through indices or related data structures and we assume that the system will perform a full scan of each column. In the presence of such optimizations, the HTAP system will still need to schedule the maintenance of the associated data structures considering the data updates. Even though we expect that our scheduler will still be applicable, we leave this as future work which can rely on our current findings.

The main overheads that we consider in the execution are the remote memory accesses, since they can affect significantly execution times, especially in OLAP engines that have to process big amounts of data. Given that state-of-the-art analytical query engines can saturate the memory bandwidth while scanning the data, we can quantify the overhead for remote vs local memory access to be equal to the difference in bandwidth between the main memory bus and the CPU interconnect. For OLTP engines, which are characterized by random data accesses and therefore use only a part of the memory bandwidth, we assume that the overhead for remote data access is less than the analytical ones. Given that the profile of the transactions typically does not change over time, we assume that the scheduler can easily learn and experimentally quantify this overhead and adjust it at run-time, if the profile of the transactional workload changes. Finally, both engines can scale as they use more CPU cores even from a socket which is remote to the data, despite the interconnect bottleneck, which leads to lower performance.

The scheduler makes decisions in two levels. The first level is the state selection. The scheduler selects the state that optimizes access locality to the fresh data required by the query. The second level is the resource distribution. Memory-wise, in the co-located state, the OLTP and the OLAP engine take one of the OLTP instances each. In the isolated case, the OLTP and the OLAP keep their own instances. In the hybrid case, the OLTP keeps one of its active instances and shares the second with OLAP. Computing-wise, in the co-located state, the OLTP and the OLAP share at maximum all CPU sockets. In the isolated case, there is no change in the CPU distribution. In the hybrid case, the OLAP engine may use a number of CPU cores that primarily belonged to the OLTP engine and interfere with its execution. The thresholds on the number of CPU cores to exchange between the two engines are set by the database administrator, as they affect workload execution in both engines, and therefore resource allocation has to remain compliant to any performance guarantees.

4.2 Elastic resource scheduling

Our systems achieves adaptivity by migrating across different states through fine-grained, elastic resource scheduling. Elasticity allows the RDE engine to provide different data-access paths to the OLAP engine to achieve maximum data freshness, while having controlled interference in OLTP performance. In [33], we give a detailed description of the state migration algorithms which rely on the system elasticity.

Algorithm 1: Freshness-driven resource scheduling

```
   Data: $F_{el}$ = Elasticity availability flag
   Data: $M_{el}$ = Elasticity mode: {Hybrid, Co-location}
   Data: $N_{fq}$ = Amount of fresh data in query
   Data: $N_{ft}$ = Amount of fresh data in database
   Data: $\alpha$ = ETL sensitivity
1  ResourceSchedule()
2      if $N_{fq} < \alpha N_{ft}$ AND !QueryBatch then
3          if !$F_{el}$ then
4              MigrateState$S_3$(ISOLATED)
5          else if $M_{el}$ == HYBRID then
6              MigrateState$S_3$(NON-ISOLATED)
7          else
8              MigrateState$S_1$()
9      else
10         MigrateState$S_2$()
```

Algorithm 1 describes the scheduling strategy for migrating the system across different states. The decisions of the algorithm are based on the freshness-rate metric for every query. Recall that the freshness-rate metric in our HTAP system is the rate of tuples that are the same in the OLAP and the currently active OLTP instance when the query arrives. Algorithm 1 calculates the freshness-rate metric only for the columns which will be accessed by every query. The scheduler retrieves from the RDE engine the amount of fresh data that the OLAP engine needs to fetch from the OLTP instance to satisfy the current query N_{fq} with freshness-rate 1 and the amount of fresh data to update the whole OLAP instance N_{ft}. The parameter α is defined within $[0, 1]$ and as it decreases, the scheduler prefers to do ETL by migrating to S_2. If ETL is not preferred due to the amount of fresh data, the scheduler will check whether elasticity is allowed, denoted with the flag F_{el}. If it is not allowed, then it will instruct the OLAP engine to read the data needed for the query remotely from the OLTP instance, after migrating to state $S_3 - ISOLATED$ ($S_3 - IS$). If elasticity is allowed, then the system will either migrate to the state $S_3 - NON - ISOLATED$ ($S_3 - NI$) or to S_1, depending on the performance requirements of the OLTP engine. Therefore, the decision on the elasticity mode M_{el} is based on the service level agreement for the OLTP engine. The number of CPU cores to be passed from the OLTP to the OLAP engine is subject to the workload and the OLTP performance requirements.

Given that the performance of an HTAP system is determined by the performance of both the OLTP and the OLAP engine, Algorithm 1 is a heuristic which tries to optimize the performance of OLAP given the restrictions of the OLTP engine. For this reason, it first favors for OLAP to take compute resources from OLTP ($S_3 - NI$), then to trade them with the OLTP (S_1) and finally to just do remote access ($S_3 - IS$). In all cases, when there is enough fresh data, as defined by α, the algorithm migrates to S_2 to keep the OLAP instance fresh and provide data locality for future queries.

Query Batch. We consider as batch, a set of queries that are executed over the same data snapshot with the same freshness rate. Thus, the execution of the batch depends only on the OLAP engine and it is orthogonal to the scheduler. As the number of queries is increased, so does the probability of them accessing all the fresh data of the OLTP instance, making N_{fq} to approach N_{ft} and leading the scheduler to migrate to S_2. This also applies for individual queries, where the scheduler is expected to initially trigger states S_1 and S_3 which do not update the OLAP instance, but at some point the rate of fresh data per query to the overall amount of fresh data will approach 1, eventually migrating to S_2.

ETL Sensitivity. ETL sensitivity is given by the parameter α in Algorithm 1 and it represents the threshold for copying the fresh data from the OLTP to the OLAP instance. Small values of α increase the sensitivity of the scheduler into performing an ETL by migrating to state S_2. This is beneficial for workloads where every query is expected to touch the same attributes as the previous ones, or workloads where only a small fraction of the data gets updated. Instead, big values of α are beneficial for workloads where every query is expected to access a small subset of the updated data.

Elasticity and Interference. Elasticity introduces interference between the OLTP and the OLAP engine. Bandwidth-intensive OLAP can starve OLTP with a few hardware threads by consuming memory bandwidth apart from the compute resources. Limits in the use of CPUs and memory bandwidth can be set by using hardware tools of server-grade CPUs [16] or software-based solutions [22, 27]. In [33], we analyze the effect of elasticity on interference between the workloads. To better utilize the hardware resources, real-time performance monitoring can be employed [14, 15, 23], as they will allow the scheduler to distribute resources between the engines until a certain performance degradation threshold.

5 EXPERIMENTAL EVALUATION

This section presents the results of our experimental evaluation. First, we describe the hardware and software setup that we used for our experiments, and the benchmark that we used to derive our workload. Then, we derive a query mix and we evaluate the adaptivity properties of our scheduler throughout the workload execution. We tune our scheduler, based on the results of a sensitivity analysis presented in [33].

5.1 Hardware & Software setup

Hardware. All the experiments were conducted on a server equipped with 2x14-core Intel Xeon Gold 6132 processor (32-KB L1I + 32-KB L1D cache, 1024-KB L2 cache, and 19.25-MB LLC) clocked at 2.60 GHz, with Hyper Threads, summing to a total of 56 hardware threads, and 1.5-TB of DRAM.

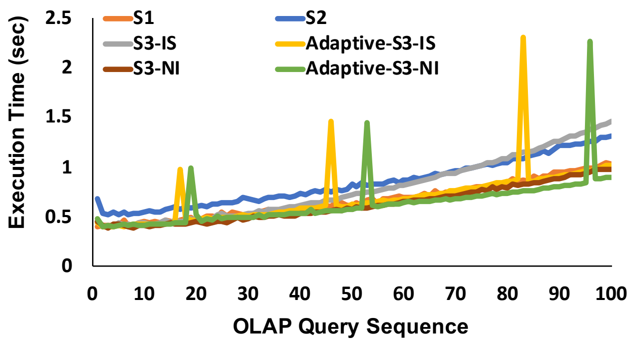
Software. At system bootstrap, the OLTP and the OLAP engines get one CPU socket of the server, which corresponds to the full isolation state (S_2). Before every experiment, we execute a warm-up phase, and then we report steady-state analytical query response time and transactional throughput.

Benchmark. We performed our experiments with the CH-benchmark [9] which combines two industry standard benchmarks, TPC-C and TPC-H, for transactional and analytical processing systems, respectively. To better analyze the effects of different HTAP schedules, we scale the database size following the TPC-H approach by a scale factor SF and the size of the `LineItem` table becomes $SF * 6,001,215$. We fix 15 `OrderLines` per `Order` when initializing the database and we scale the number of records in `OrderLine` to $SF * 6,001,215$. In contrast to TPC-H, and as per TPC-C specification, every `NewOrder` transaction generates five to fifteen order lines per order. For the transactional workload, we assign one warehouse to every worker thread, which generates and executes transactions simulating complete transactional queue.

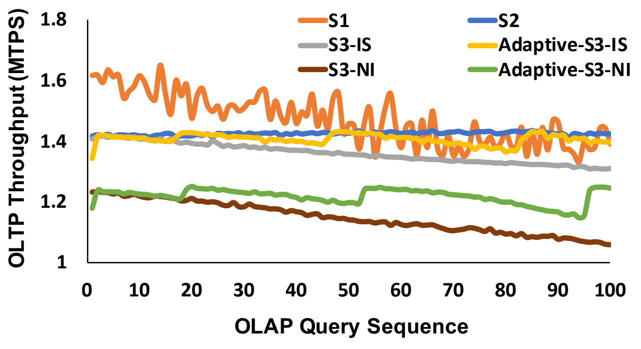
5.2 Adaptive HTAP scheduling

Figure 3 shows the performance of adaptive scheduling and of every state. We initialize the database at SF 30 before we synchronize the storage of the engines setting the freshness-rate to 1. The OLAP engine executes a set of 3 queries for 100 times in a sequence, one after the other and we report the total execution time of a sequence, including snapshotting or ETL. We use two scan-heavy queries, Q1 & Q6, and one join-heavy, Q19. The OLTP engine executes TPC-C `NewOrder`, concurrently to the OLAP queries. We remove the `LIKE` condition from Q19, as it is not supported by the OLAP engine.

OLAP performance. Figure 3(a) shows the OLAP sequence execution times for different states. As the OLTP engine inserts data, execution time increases. The selection of state for $S_3 - IS$, S_1, and $S_3 - NI$ depends on the performance requirements for the OLTP engine, from stricter to looser



(a) OLAP



(b) OLTP

Figure 3: HTAP performance under different scheduling states

ones, respectively. With the availability of elastic resources, the scheduler either opts for $S_3 - NI$ or S_1, depending on the OLTP engine requirements. Observe that the design of our OLTP engine with two instances, synchronized upon query arrival, inherently corresponds to the decision of our scheduler by setting $\alpha = 0$, in the S_1 case. As this approach for the OLTP engine provides flexibility to our design, we only report the performance for the adaptive S_1.

State S_2 is the slowest one, as it has to do an ETL for every query. Given that we are not executing batches, the ETL cost only gets amortized with respect to $S_3 - IS$ after 75 query sets because at that point, in S_2 the OLAP engine has more socket-local data than in $S_3 - IS$ where data are read from the remote socket. State $S_3 - NI$ provides further performance improvement due to the use of data-local compute resources allowing OLAP to transfer less data over the interconnect. In all cases, we observe that adaptivity improves performance, at the cost of a single ETL where one query pays with additional latency. The timing of this ETL depends on the value of α, which we currently set to 0.5. Smaller values of α reduce the tail latency at the cost of smaller benefit for the rest of the queries. Finally, in all cases, we observe that the gap between the adaptive and the non-adaptive case widens with time. Across states, it goes up to 50% ($Adaptive - S_3 - NI$ and $S_3 - IS$) showing the benefits to migrate from one state to an adaptive one. We have executed further experiments with up to 300 queries of the same mix for the state $S_3 - NI$ which has the slowest convergence, and we observed that this gap starts from 11% for 100 queries, goes to 22% for 200 queries, 25% for 250 and 30% for 300 queries.

OLTP Performance. Figure 3(b) shows the OLTP throughput for the given query sequence, under different system scheduling states. The throughput degrades slightly due to increased memory pressure to OLTP-local DRAM as OLAP has hybrid scans accessing data over the interconnect in additional to OLAP-local scans. In all the adaptive schedules, the throughput increases after every ETL as it reduces the OLAP pressure on the OLTP-local memory-bus, which corroborates the above claim. $S_3 - NI$ has lower throughput compared to isolated states due to the reallocation of CPU cores to the OLAP engine. Finally, S_1 has variance due to the co-location of an OLTP and and an OLAP in the same socket.

Insights. Freshness-driven scheduling in HTAP adapts across feasible states, where feasibility is set as the workloads' isolation level. Isolated adaptive mode achieves 30% speedup over S3-IS with 100 sequences. with 4-elastic cores, adaptive mode achieves 11%, 22% and 26% performance gains at 100th, 200th and 250th sequence execution, and with time, adaptive schedule amortizes cost of data movement. In general, adaptive case builds upon hybrid-states while stressing to trigger ETL based on the freshness ratio in order to balance data-access across OLAP-local and fresh-data.

6 CONCLUSION

We approach HTAP as a scheduling problem, by tuning the OLTP and OLAP engines performance based on the data freshness and performance requirements of the workload. We define the HTAP design space ranging from fully co-located engines to fully isolated, and we devise an elastic system which traverses this space by distributing compute and memory resources to the OLTP and the OLAP engine. We provide a scheduling algorithm which drives resource allocation decisions, and we evaluate its performance using the CH-Benchmark, showing that our system adapts to the workload requirements for data freshness and performance.

7 ACKNOWLEDGMENTS

We would like to thank the reviewers and the shepherd for their valuable feedback. Angelos Anadiotis was at EPFL until after the initial submission of the paper, and contributed to the revision while at the Ecole Polytechnique. This work was partially funded by the FNS project "Efficient Real-time Analytics on General-Purpose GPUs" subside no. 200021_178894/1 and the EU H2020 project SmartDataLake (825041).

REFERENCES

[1] Raja Appuswamy, Angelos-Christos G. Anadiotis, Danica Porobic, Mustafa Iman, and Anastasia Ailamaki. 2017. Analyzing the Impact of System Architecture on the Scalability of OLTP Engines for High-Contention Workloads. *PVLDB* 11, 2 (2017), 121–134. https://doi.org/10.14778/3149193.3149194

[2] Raja Appuswamy, Manos Karpathiotakis, Danica Porobic, and Anastasia Ailamaki. 2017. The Case For Heterogeneous HTAP. In *CIDR 2017, 8th Biennial Conference on Innovative Data Systems Research, Chaminade, CA, USA, January 8-11, 2017, Online Proceedings*. http://cidrdb.org/cidr2017/papers/p21-appuswamy-cidr17.pdf

[3] Hal Berenson, Philip A. Bernstein, Jim Gray, Jim Melton, Elizabeth J. O'Neil, and Patrick E. O'Neil. 1995. A Critique of ANSI SQL Isolation Levels. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, California, USA, May 22-25, 1995*, Michael J. Carey and Donovan A. Schneider (Eds.). ACM Press, 1–10. https://doi.org/10.1145/223784.223785

[4] Mokrane Bouzeghoub and Verónika Peralta. 2004. A Framework for Analysis of Data Freshness. In *IQIS 2004, International Workshop on Information Quality in Information Systems, 18 June 2004, Paris, France (SIGMOD 2004 Workshop)*, Felix Naumann and Monica Scannapieco (Eds.). ACM, 59–67. https://doi.org/10.1145/1012453.1012464

[5] Tuan Cao, Marcos Antonio Vaz Salles, Benjamin Sowell, Yao Yue, Alan J. Demers, Johannes Gehrke, and Walker M. White. 2011. Fast checkpoint recovery algorithms for frequently consistent applications. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, Athens, Greece, June 12-16, 2011.* 265–276. https://doi.org/10.1145/1989323.1989352

[6] Biswapesh Chattopadhyay, Priyam Dutta, Weiran Liu, Ott Tinn, Andrew McCormick, Aniket Mokashi, Paul Harvey, Hector Gonzalez, David Lomax, Sagar Mittal, Roee Ebenstein, Nikita Mikhaylin, Hung-Ching Lee, Xiaoyan Zhao, Tony Xu, Luis Perez, Farhad Shahmohammadi, Tran Bui, Neil Mckay, Selcuk Aya, Vera Lychagina, and Brett Elliott. 2019. Procella: Unifying serving and analytical data at YouTube. *PVLDB* 12, 12 (2019), 2022–2034. http://www.vldb.org/pvldb/vol12/p2022-chattopadhyay.pdf

[7] Periklis Chrysogelos, Manos Karpathiotakis, Raja Appuswamy, and Anastasia Ailamaki. 2019. HetExchange: Encapsulating heterogeneous CPU-GPU parallelism in JIT compiled engines. *PVLDB* 12, 5 (2019), 544–556. http://www.vldb.org/pvldb/vol12/p544-chrysogelos.pdf

[8] James Cipar, Gregory R. Ganger, Kimberly Keeton, Charles B. Morrey III, Craig A. N. Soules, and Alistair C. Veitch. 2012. LazyBase: trading freshness for performance in a scalable database. In *European Conference on Computer Systems, Proceedings of the Seventh EuroSys Conference 2012, EuroSys '12, Bern, Switzerland, April 10-13, 2012*, Pascal Felber, Frank Bellosa, and Herbert Bos (Eds.). ACM, 169–182. https://doi.org/10.1145/2168836.2168854

[9] Richard L. Cole, Florian Funke, Leo Giakoumakis, Wey Guy, Alfons Kemper, Stefan Krompass, Harumi A. Kuno, Raghunath Othayoth Nambiar, Thomas Neumann, Meikel Poess, Kai-Uwe Sattler, Michael Seibold, Eric Simon, and Florian Waas. 2011. The mixed workload CH-benCHmark. In *Proceedings of the Fourth International Workshop on Testing Database Systems, DBTest 2011, Athens, Greece, June 13, 2011*, Goetz Graefe and Kenneth Salem (Eds.). ACM, 8. https://doi.org/10.1145/1988842.1988850

[10] Sudipto Das, Feng Li, Vivek R. Narasayya, and Arnd Christian König. 2016. Automated Demand-driven Resource Scaling in Relational Database-as-a-Service. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, Fatma Özcan, Georgia Koutrika, and Sam Madden (Eds.). ACM, 1923–1934. https://doi.org/10.1145/2882903.2903733

[11] Pamela Delgado, Florin Dinu, Anne-Marie Kermarrec, and Willy Zwaenepoel. 2015. Hawk: Hybrid Datacenter Scheduling. In *2015 USENIX Annual Technical Conference, USENIX ATC '15, July 8-10, Santa Clara, CA, USA*, Shan Lu and Erik Riedel (Eds.). USENIX Association, 499–510. https://www.usenix.org/conference/atc15/technical-session/presentation/delgado

[12] Adam Dziedzic, Jingjing Wang, Sudipto Das, Bolin Ding, Vivek R. Narasayya, and Manoj Syamala. 2018. Columnstore and B+ tree - Are Hybrid Physical Designs Important?. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, Gautam Das, Christopher M. Jermaine, and Philip A. Bernstein (Eds.). ACM, 177–190. https://doi.org/10.1145/3183713.3190660

[13] Franz Färber, Norman May, Wolfgang Lehner, Philipp Große, Ingo Müller, Hannes Rauhe, and Jonathan Dees. 2012. The SAP HANA Database – An Architecture Overview. *IEEE Data Eng. Bull.* 35, 1 (2012), 28–33. http://sites.computer.org/debull/A12mar/hana.pdf

[14] Benjamin Farley, Ari Juels, Venkatanathan Varadarajan, Thomas Ristenpart, Kevin D. Bowers, and Michael M. Swift. 2012. More for your money: exploiting performance heterogeneity in public clouds. In *ACM Symposium on Cloud Computing, SOCC '12, San Jose, CA, USA, October 14-17, 2012.* 20. https://doi.org/10.1145/2391229.2391249

[15] Robert Grandl, Srikanth Kandula, Sriram Rao, Aditya Akella, and Janardhan Kulkarni. 2016. GRAPHENE: Packing and Dependency-Aware Scheduling for Data-Parallel Clusters. In *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016.* 81–97. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/grandl_graphene

[16] Intel. 2020. Intel Resource Director Technology. https://www.intel.com/content/www/us/en/architecture-and-technology/resource-director-technology.html.

[17] Calin Iorgulescu, Reza Azimi, Youngjin Kwon, Sameh Elnikety, Manoj Syamala, Vivek R. Narasayya, Herodotos Herodotou, Paulo Tomita, Alex Chen, Jack Zhang, and Junhua Wang. 2018. PerfIso: Performance Isolation for Commercial Latency-Sensitive Services. In *2018 USENIX Annual Technical Conference, USENIX ATC 2018, Boston, MA, USA, July 11-13, 2018*, Haryadi S. Gunawi and Benjamin Reed (Eds.). USENIX Association, 519–532. https://www.usenix.org/conference/atc18/presentation/iorgulescu

[18] Manos Karpathiotakis, Ioannis Alagiannis, and Anastasia Ailamaki. 2016. Fast Queries Over Heterogeneous Data Through Engine Customization. *PVLDB* 9, 12 (2016), 972–983. https://doi.org/10.14778/2994509.2994516

[19] Alfons Kemper and Thomas Neumann. 2011. HyPer: A hybrid OLTP&OLAP main memory database system based on virtual memory snapshots. In *Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11-16, 2011, Hannover, Germany.* 195–206. https://doi.org/10.1109/ICDE.2011.5767867

[20] Tirthankar Lahiri, Shasank Chavan, Maria Colgan, Dinesh Das, Amit Ganesh, Mike Gleeson, Sanket Hase, Allison Holloway, Jesse Kamp, Teck-Hua Lee, Juan Loaiza, Neil MacNaughton, Vineet Marwah, Niloy Mukherjee, Atrayee Mullick, Sujatha Muthulingam, Vivekanandhan Raja, Marty Roth, Ekrem Soylemez, and Mohamed Zaït. 2015. Oracle Database In-Memory: A dual format in-memory database. In *31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April 13-17, 2015.* 1253–1258. https://doi.org/10.1109/ICDE.2015.7113373

[21] Per-Åke Larson, Adrian Birka, Eric N. Hanson, Weiyun Huang, Michal Nowakiewicz, and Vassilis Papadimos. 2015. Real-Time Analytical Processing with SQL Server. *PVLDB* 8, 12 (2015), 1740–1751. https://doi.org/10.14778/2824032.2824071

[22] David Lo, Liqun Cheng, Rama Govindaraju, Parthasarathy Ranganathan, and Christos Kozyrakis. 2015. Heracles: improving resource efficiency at scale. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture, Portland, OR, USA, June 13-17, 2015*, Deborah T. Marr and David H. Albonesi (Eds.). ACM, 450–462. https://doi.org/10.1145/2749469.2749475

[23] Kshiteej Mahajan, Mosharaf Chowdhury, Aditya Akella, and Shuchi Chawla. 2018. Dynamic Query Re-Planning using QOOP. In *13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018, Carlsbad, CA, USA, October 8-10, 2018.* 253–267. https://www.usenix.org/conference/osdi18/presentation/mahajan

[24] Darko Makreshanski, Jana Giceva, Claude Barthels, and Gustavo Alonso. 2017. BatchDB: Efficient Isolated Execution of Hybrid OLTP+OLAP Workloads for Interactive Applications. In *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017.* 37–50. https://doi.org/10.1145/3035918.3035959

[25] Henrik Mühe, Alfons Kemper, and Thomas Neumann. 2011. How to efficiently snapshot transactional data: hardware or software controlled?. In *Proceedings of the Seventh International Workshop on Data Management on New Hardware, DaMoN 2011, Athens, Greece, June 13, 2011.* 17–26. https://doi.org/10.1145/1995441.1995444

[26] Vivek R. Narasayya, Sudipto Das, Manoj Syamala, Badrish Chandramouli, and Surajit Chaudhuri. 2013. SQLVM: Performance Isolation in Multi-Tenant Relational Database-as-a-Service. In *CIDR 2013, Sixth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 6-9, 2013, Online Proceedings.* www.cidrdb.org. http://cidrdb.org/cidr2013/Papers/CIDR13_Paper25.pdf

[27] Vivek R. Narasayya, Sudipto Das, Manoj Syamala, Badrish Chandramouli, and Surajit Chaudhuri. 2013. SQLVM: Performance Isolation in Multi-Tenant Relational Database-as-a-Service. In *CIDR 2013, Sixth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 6-9, 2013, Online Proceedings.*

[28] Thomas Neumann. 2011. Efficiently Compiling Efficient Query Plans for Modern Hardware. *PVLDB* 4, 9 (2011), 539–550. https://doi.org/10.14778/2002938.2002940

[29] Thomas Neumann, Tobias Mühlbauer, and Alfons Kemper. 2015. Fast Serializable Multi-Version Concurrency Control for Main-Memory Database Systems. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015.* 677–689. https://doi.org/10.1145/2723372.2749436

[30] Rasmus Pagh and Flemming Friche Rodler. 2004. Cuckoo hashing. *J. Algorithms* 51, 2 (2004), 122–144. https://doi.org/10.1016/j.jalgor.2003.12.002

[31] Iraklis Psaroudakis, Florian Wolf, Norman May, Thomas Neumann, Alexander Böhm, Anastasia Ailamaki, and Kai-Uwe Sattler. 2014. Scaling Up Mixed Workloads: A Battle of Data Freshness, Flexibility, and Scheduling. In *Performance Characterization and Benchmarking. Traditional to Big Data - 6th TPC Technology Conference, TPCTC 2014, Hangzhou, China, September 1-5, 2014. Revised Selected Papers.* 97–112. https://doi.org/10.1007/978-3-319-15350-6_7

[32] Vijayshankar Raman, Gopi K. Attaluri, Ronald Barber, Naresh Chainani, David Kalmuk, Vincent KulandaiSamy, Jens Leenstra, Sam Lightstone, Shaorong Liu, Guy M. Lohman, Tim Malkemus, René Müller, Ippokratis Pandis, Berni Schiefer, David Sharpe, Richard Sidle, Adam J. Storm, and Liping Zhang. 2013. DB2 with BLU Acceleration: So Much More than Just a Column Store. *PVLDB* 6, 11 (2013), 1080–1091. https://doi.org/10.14778/2536222.2536233

[33] Aunn Raza, Periklis Chrysogelos, Angelos Christos Anadiotis, and Anastasia Ailamaki. 2020. Adaptive HTAP through Elastic Resource Scheduling. (2020). https://infoscience.epfl.ch/record/276997

[34] Yingjun Wu, Joy Arulraj, Jiexi Lin, Ran Xian, and Andrew Pavlo. 2017. An Empirical Evaluation of In-Memory Multi-Version Concurrency Control. *PVLDB* 10, 7 (2017), 781–792. https://doi.org/10.14778/3067421.3067427