BatchDB: Efficient Isolated Execution of Hybrid OLTP+OLAP Workloads for Interactive Applications

Darko Makreshanski, Jana Giceva, Claude Barthels, Gustavo Alonso
Systems Group, Department of Computer Science, ETH Zurich
{name.surname}@inf.ethz.ch

ABSTRACT

In this paper we present BatchDB, an in-memory database engine designed for hybrid OLTP and OLAP workloads. BatchDB achieves good performance, provides a high level of data freshness, and minimizes load interaction between the transactional and analytical engines, thus enabling real time analysis over fresh data under tight SLAs for both OLTP and OLAP workloads.

BatchDB relies on primary-secondary replication with dedicated replicas, each optimized for a particular workload type (OLTP, OLAP), and a light-weight propagation of transactional updates. The evaluation shows that for standard TPC-C and TPC-H benchmarks, BatchDB can achieve competitive performance to specialized engines for the corresponding transactional and analytical workloads, while providing a level of performance isolation and predictable runtime for hybrid workload mixes (OLTP+OLAP) otherwise unmet by existing solutions.

1. INTRODUCTION

As workload characteristics and requirements evolve, database engines need to efficiently handle both transactional (OLTP) and analytical (OLAP) workloads with strong guarantees for throughput, latency and data freshness. Running analytics on the latest data, however, must not degrade OLTP performance which is typically bound by strict SLAs for response time and throughput [43]. Furthermore, OLAP is no longer confined to a small set of in-house users without requirement for guaranteed performance. Businesses provide OLAP (in addition to OLTP) as a service to large number of users with SLAs on data-freshness and performance. An example is the airline industry [59], where analytics of flight bookings is offered as a service to travel agents and airline companies.

Efficiently handling both OLTP and OLAP workloads is difficult because they require different algorithms and data structures. A common approach for handling such hybrid workloads is to keep a separate data warehouse for OLAP isolated from the OLTP system. Data warehouse systems are optimized for read-only analytical workloads and are periodically refreshed through a batch job containing the latest data updates. This provides both good performance isolation between the two workloads, and the ability to tune each system independently. The downsides, however, are that the data analyzed is possibly stale and that there is additional overhead of interfacing with multiple systems [45].

To overcome this problem, several alternatives have been recently introduced which target such hybrid workloads (e.g., SAP HANA [17], HyPer [28, 40], SQL Server [30], MemSQL [52], Oracle [29], etc.). However, a big challenge for these systems is the performance impact that workloads have on each other. A recent study by Psaroudakis et al. [47] analyzed cross-workload interference in HANA and HyPer and found that the maximum attainable OLTP throughput, when co-scheduled with a large analytical workload, is reduced by at least three and five times respectively.

This paper presents BatchDB, an alternative design of a database engine architecture, which handles hybrid workloads with guarantees for performance, data freshness, consistency, and elasticity.

To accommodate both OLAP and OLTP, BatchDB primarily relies on replication, trading off space for performance isolation, with a primary replica dedicated for OLTP workloads and a secondary replica dedicated for OLAP workloads. This allows for workload-specific optimizations for every replica and physical isolation of resources dedicated for each workload.

To efficiently maintain the replica up-to-date without affecting OLTP and OLAP performance, BatchDB relies on lightweight update extraction and isolated execution of queries and updates at the OLAP replica. The latter is achieved by having incoming OLAP queries first queued and then scheduled in batches, one batch-at-a-time. Execution of each batch of queries is shared and done as part of a single read-only transaction on the latest version of the data. Propagated OLTP updates are also first queued and then efficiently executed in-between two batches of queries. This enables version-agnostic scan processing at the OLAP replica and logical isolation between the query processor and update propagation. For added elasticity of the system, BatchDB uses an efficient way of extracting, propagating and applying updates both within one and across multiple machines using RDMA over InfiniBand.

All these features help BatchDB achieve performance isolation for hybrid workloads as required by applications that need guaranteed throughput and response times. The experimental results, based on a hybrid TPC-C and TPC-H workload [11], show that BatchDB achieves good performance isolation between the hybrid workloads, having a negligible ten percent overhead on each other's performance. Moreover, we show that the replication mechanism is capable of propagating and applying the updates at the analytical replica at a rate higher than any TPC-C throughput achieved to date and can, thus, be integrated with other transactional engines. Finally, the system has competitive performance for individual OLTP and OLAP workloads and is superior to existing systems designed for hybrid workloads in terms of overall throughput.

SIGMOD'17, May 14-19, 2017, Raleigh, NC, USA

DOI: http://dx.doi.org/10.1145/3035918.3035959

In summary the paper makes the following contributions:

- We identify and experimentally show multiple effects of resource sharing that cause unbalanced performance degradation of OLTP and OLAP workloads:
  - Explicit resource sharing depends on scheduling policies of the database engine and can be avoided by scheduling OLTP and OLAP requests on dedicated resources.
  - Implicit resource sharing (e.g., of memory bandwidth and CPU caches) that entails the need for having separate replicas of OLTP and OLAP workloads
- We propose a method for logical separation of analytical queries and transactional updates using a single snapshot replica, batch scheduling of queries and updates and efficient algorithms for executing the updates.
- We show experimentally that our method leads to very high isolation of OLTP and OLAP performance both when replicas are collocated on a single machine and when they are placed on separate machines.

2. MOTIVATION AND STATE-OF-THE-ART

In this section, we identify the key requirements for engines that aim to handle hybrid transactional and analytical processing workloads (HTAP), and summarize existing systems.

2.1 Design Goals

Performance isolation: Database engines typically have to provide guarantees for latency and throughput as required by SLAs. Unpredictable OLTP performance caused by concurrent OLAP queries in hybrid workloads, and vice-versa, can lead to a significant revenue losses.

Workload-specific optimizations: Database engines should leverage workload-specific optimizations wherever applicable. They should use data structures and operate on data formats suitable for the given workload. Workload-optimized implementations are important for delivering good and stable performance.

Update propagation & data freshness: Ideally, OLAP queries should run on the most recent version of data as many critical business decisions rely on real-time analytics on latest data [45]. Fast update propagation requires using low-latency communication primitives between the individual system elements and having efficient update mechanisms within the components.

Consistency guarantees: Analytical queries should be executed with high consistency guarantees, (i.e., snapshot isolation) to ensure that queries have a consistent view on the data.

Single system interface: Instead of managing a separate system for each workload type, having a single interface which supports both analytical queries and transactions significantly increases the ease of use of the system.

Elasticity: The system should take advantage of all the resources provided by modern distributed environments. To achieve this, the database engine should be able to scale dynamically with an increasing number of machines.

2.2 Overview

The traditional way of analyzing operational data meant running a separate data warehouse system updated periodically with updates from the main OLTP system through an ETL process. This approach provided good performance isolation between the OLTP and OLAP workloads, and allowed for workload-specific optimizations to be applied in each system. However, the main drawback is that most of the analysis is then done on stale data, as the update period can be in the order of hours or days. In today's connected fast-paced world, data freshness is critical and there is a high demand for real-time operational analytics [30, 43, 45].

For this reason, many recent research and commercial systems aim to provide real-time analytics. For instance, SAP HANA [17] relies on multi-version concurrency control with a main and delta data-structures where the delta is periodically merged into the main. Recent work by Goel et al. [21] proposed an architectural design for a scale-out extension of HANA (SAP HANA SOE) to support large scale analytics over real-time data. The first version of HyPer [28] relied on single-threaded in-core partitioned database and used the operating system's fork system call to provide snapshot isolation for analytical workloads. The new version uses an MVCC implementation that offers serializability, fast transaction processing and fast scans [40]. Microsoft's SQL Server combines two specialized engines for OLTP (Hekaton in memory tables) and OLAP (Apollo updateable column-store indices) [30]. Oracle provides dual-format in-memory option [29] to store a table in columnar format for analytic operations in addition to the standard row format for OLTP workloads. It also provides a publish-subscribe mechanism with its Change-Data-Capture [41] and GoldenGate [43] technologies to stream changes from a primary OLTP database to generic secondary databases that can be anything from a relational database to a big-data system. ScyPer [39] extends HyPer to provide scaled-out analytics on remote replicas by propagating updates either using a logical or physical redo log. MemSQL [52] also supports hybrid OLTP and OLAP workloads providing dynamic code generation, distributed query processing, and high performance in-memory data-structures, such as lock-free skip-lists.

There are many other systems which have addressed similar problems or shared some of our design goals. For instance, Ganymed [44] and Multimed [51] showed the benefits of using a primary-secondary replication mechanism for scalability, performance isolation and providing special functionality both across and within multicore machines. KuaFu [24] analyze concurrent replay of transactions in backup database replicas, and Pacitti et al. [42] studied how immediate update propagation (without waiting for the commitment of the transaction) can improve data freshness. HYRISE [22] automatically partitions tables with varying widths depending on whether certain columns are accessed as part of an OLTP or an OLAP workload. Several systems have also explored hybrid storage layouts, and can decide on the fly which layout is best for the respective queries (e.g., [1, 14]).

The technique of batching requests has been previously used in several different contexts. When processing transactional workloads, systems like Calvin [56] and BOHM [16] have grouped transactions in window based batches to determine a global serial order, which then helps in orchestrating the transaction-at-a-time execution to improve performance of distributed transactions and highly scalable concurrency protocol on multicore machines. For analytical workloads, systems like HyPer [28] have proposed grouping of OLAP queries by session so that they read the same snapshot, but are still executed query-at-a-time. Alternatively, analytical requests can be batched and executed concurrently as a group using multi-query optimization. Such an approach was originally proposed by IBM Blink [48], and later adopted by systems such as CJoin [9], Crescando [59], DataPath [2], and SharedDB [19].

By using a shared-scan approach and a delta-main data-structure AIM [7] proposes a system that can run real-time analytics on high data rates coming from streaming systems, but is specifically tailored for a telecommunication industry use-case.

Other relevant systems are federated database engines, targeting big data batch and stream processing systems, such as BigDAWG's Polystore [15] and the Cyclops's DBMS+ [34]. Both follow the

one size does not fit all paradigm, but expose a single user interface to multiple specialized data processing systems, hiding the complexity in the optimizer.

2.3 Discussion

To discuss the properties of state-of-the-art systems that handle OLTP and OLAP workloads we separate them into two categories: systems that operate on a single instance of the data, and systems that rely on data replication. Examples of the first category of systems are SAP HANA and HyPer. They provide a single system interface, and enable users to analyze the latest (or any) version of the data. This, however, comes at a cost. It limits opportunities for workload-specific optimizations, it results in synchronization penalties and trade-off between data freshness and performance on both sides of the workload spectrum, and often results in interference for the usage of hardware resources. Consequently, there is a significant performance interaction. The extensive analysis by Psaroudakis et al. [47] demonstrated the OLAP workloads can significantly reduce the performance of the OLTP workload. They used the hybrid CH-benCHmark [11] to test such cross-workload performance effects on both HyPer and SAP HANA. The study concluded that the interference is due to resource sharing, as well as synchronization overhead when querying latest data. Finally, we must note that despite operating on a single instance of the data, these systems still rely on distinction between data representations and certain amount of replication. The latter one is implicit through a copy on write mechanism (HyPer), or through keeping the data in a main and a delta containing the most recent updates still not merged with the main (SAP HANA). The latter one can be perceived as a partial replication, where the updates are merged with the main in a batch on certain time intervals.

The second category of systems follows the more classical approach of data warehousing, where data is replicated with separate replicas being used for different workloads. Example systems are ScyPer [39], SQL Server's column-store index, Oracle's GoldenGate [43] and in-memory dual format [29], and the scale-out extension of SAP HANA [21]. The common approach for maintaining an OLAP-specific replica up-to-date with OLTP updates in these systems (in particular employed by SAP HANA, Oracle and SQL Server) is to have updates staged in a separate delta data-structure with the query processor accessing both the delta and main OLAP replica for scan processing. The delta is then periodically merged into the main data-structure. The main challenge in this case, however, is to maintain the OLAP replica up-to-date under high OLTP and OLAP load without sacrificing performance. For instance, the SQL Server recommends storing only cold data in the column-store index to avoid the overheads of applying updates from hot data [53]. The OLAP query execution then automatically scans the column-store index for cold data and the primary OLTP store for hot data. ScyPer's authors note that by executing a physical redo log of update-containing transactions, the updates are applied two times faster than in the primary replica, but their performance is still significantly affected by the OLAP query execution. The enterprise version of MemSQL supports replication, but log recovery at the secondary replica is single-threaded and is not be able to keep up under high OLTP load. Furthermore, MemSQL only provides read-committed isolation level. Finally, Oracle states that their approach leads to single-digit percentage performance overhead to the source (primary OLTP) database [43], but it is unclear whether and by how much the applying of updates affects the execution of analytical workloads.

Hence, we conclude that handling hybrid workloads while satisfying the design goals (described in §2.1) is still an open problem.

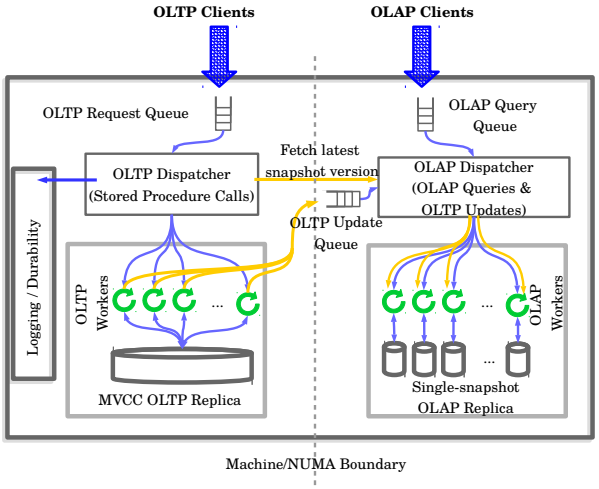


Figure 1: System architecture

3. SYSTEM OVERVIEW

This section provides an overview of BatchDB's key principles, and how they address our design goals and the shortcomings of existing systems. It also includes a discussion on the assumptions and trade-offs we make.

3.1 BatchDB components

Figure 1 shows an overview of BatchDB's architecture. Similar to the second category of systems supporting hybrid workloads described in Section 2.3, BatchDB's architecture is based on *replication*. In particular, it uses a primary-secondary type of replication. The *primary replica* is OLTP-specific and handles all update-containing transactions (Figure 1 left), while the *secondary replica* is OLAP-specific and executes all analytical queries (Figure 1 right). This allows for each replica to be optimized for the particular type of workload it is responsible for (e.g., the OLTP and OLAP components have separate execution engines where the OLTP is designed for pre-compiled stored procedures, while the OLAP targets scan-heavy ad-hoc queries). In BatchDB, the replicas can be placed either on the same shared-memory machine (on different NUMA nodes) or across multiple machines.

Each replica has a dedicated set of resources (worker threads on separate CPU sockets or separate machines) and a dispatcher that schedules the corresponding requests. The OLTP dispatcher is responsible for assigning OLTP requests to worker threads and logging the successful transactions. Worker threads also export a physical log of updates containing information on the snapshot version for each affected tuple to be used for propagating the updates to the analytical replica. We provide more details for the implementation of the OLTP component in Section 4.

The OLAP dispatcher schedules OLAP queries and makes sure that the OLAP replica operates on one-batch-at-a-time processing with data kept in a *single-snapshot-replica*. This is important for efficient application of the propagated updates from the primary replica, without negatively affecting the performance of the OLAP queries. In this case, the physical storage is version-agnostic and maintains only a single version of the data at-a-time, removing the overhead of dealing with multiple snapshots for scan processing. Furthermore, processing a single batch-at-a-time eliminates the overhead of logical contention and synchronization with the

transactional component, and when applying the updates. More details for the analytical component are presented in Section 5.

3.2 Addressing the design goals

Performance isolation: BatchDB partially addresses the first design goal by relying on replication and using separate replicas for the different types of workloads. By design it allows for physical isolation of the allocated hardware resources: either by placing the replicas on different machines, or when executed on the same machine, with replicas isolated on separate NUMA nodes. In the evaluation section we show the impact of implicit sharing of resources on performance isolation. The second key principle addressing the performance isolation objective is removing the logical contention between the two components. This is achieved by the batch-based processing and applying of the updates done by the OLAP dispatcher.

Data freshness: To support a high level of data freshness, OLTP worker threads constantly push updates to the OLAP replica. Updates are pushed at the end of the execution of a batch of OLTP transactions in two cases: 1. if the OLAP dispatcher has asked for the latest snapshot version, 2. if the latest push of updates happened longer than a certain (configurable) period, which in our case is set to 200ms. Therefore the amount of time it takes for a committed transaction's updates to be present at the OLAP replica depends mostly on the duration of a batch of OLTP transactions, which in our case is in the order of 10s of milliseconds. This is generally less than the execution time of analytical queries, so the amount of data freshness, as perceived by users, will be determined only by the response time of the OLAP queries.

Workload-specific optimizations: By design, BatchDB's workload specific replicas can apply any applicable optimization technique for the corresponding workload mix they serve. In our case, the design and implementation of both the OLAP and OLTP components is based on ideas and techniques that have been shown to improve the performance of the particular workload at hand.

Consistency guarantees: BatchDB provides snapshot isolation for both OLTP and OLAP. The OLTP replica uses MVCC to process transactional requests, while the OLAP replica uses a single snapshot version at a time approach to answer analytical queries. We discuss some of the trade-offs BatchDB makes in terms of transactional semantics in the following subsection.

Interfaces: BatchDB exposes a single system image and interface to users of both workloads. There is no explicit requirement for users to distinguish requests for the two replicas. The secondary OLAP replicas can be seen as indexes which operate on a single snapshot version, one batch of read-only queries at a time.

Elasticity: BatchDB can scale by deploying more replicas as the number of machines increases. In addition to different replica types, multiple instances of the same component can be created in order to distribute the load evenly across the whole system. The high bandwidth of modern networks in combination with RDMA primitives makes it possible to distribute updates to a large number of replicas. The design principles we use, can also be applied to provide specialized replicas for other workload types (e.g., offline batch analytics for long running queries, graph processing, etc.).

3.3 Trade-offs

Individual query latency: One trade-off in BatchDB is the individual query latency for OLAP workloads. Namely, we impose the requirement that all concurrent OLAP queries have to be co-scheduled together to isolate them from execution of OLTP updates. The effect of this is that query latency depends on the latency of other concurrent OLAP queries. We argue that this is acceptable in practice as predictable performance for all queries is more important than performance of individual queries. This is particularly true for a system offering online analytics on real-time data as it is typically assumed (and often required due to SLAs) that all queries are executed within few seconds. If a user wishes to run a query that would take longer to execute, for instance minutes or hours, then this query can be handled in a different way as an offline non-real-time query. We discuss this in detail in Section 7.

Transaction semantics: While our system provides snapshot isolation guarantees for OLAP queries it does not provide full transactional flexibility. For instance, the version of the data on which queries are processed is decided by the system when the query (along with other queries in its batch) starts executing. To isolate the execution of OLAP queries and OLTP updates, the OLAP replica is updated at a coarser granularity and users can not choose the exact version of data to run their query on. Hence, interactive sessions with long running read only transactions where users submit queries one after the other are not possible in our current implementation. In Section 7 we discuss possible extensions to handle this and other types of workloads.

4. TRANSACTIONAL COMPONENT (OLTP REPLICA)

The transactional (OLTP) component, as depicted on the left-hand side of Figure 1, contains the primary replica of the database engine whose purpose is to handle both update-containing transactions and short latency-sensitive read-only requests. Apart from having to prepare updates to be propagated to the OLAP replica, the design decisions for implementing the OLTP replica can be oblivious to the requirements of OLAP workloads. Therefore, no compromise needs to be made when handling OLTP workloads and any OLTP-specific optimizations are applicable.

We based our implementation of BatchDB's OLTP component on Hekaton [13], and use multi-version concurrency control [6] and lock-free indexing data-structures, as opposed to partitioning to achieve scalability in multi-core systems. This is unlike the approaches taken by H-Store [26], VoltDB [54], or HyPer [28], which are more suitable for partitionable workloads.

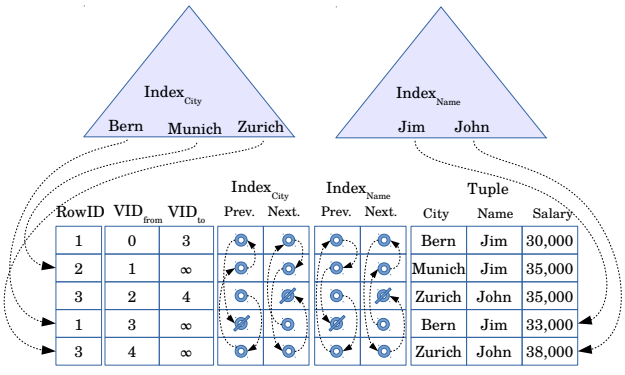


Figure 2: OLTP Record Storage Format and Index Layout Example

Storage layout and indexes: Figure 2 depicts the storage layout and indexing data-structures for a sample relation. Primarily, we use a row-oriented storage as it is most efficient for the point lookups and updates, which are integral to transaction processing. Similar to Hekaton, we use a hash-based and a tree-based index

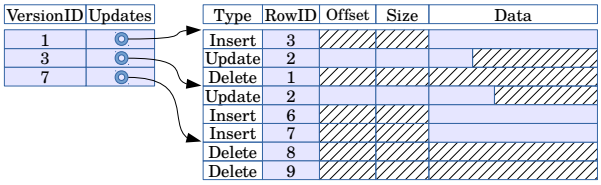


Figure 3: Propagated update format for a specific table from a single OLTP worker thread

based on the lock-free Bw-Tree [32]. We use a simplified version of the Bw-Tree that relies on atomic multi-word compare-and-swap updates [37]. Furthermore, the physical records contain a double-linked list per index to facilitate easier traversal into the indexes.

Transaction execution and concurrency control: For efficient handling of OLTP requests the system supports natively compiled stored procedures with clients sending their requests in the form of stored procedure calls. To execute the stored procedures, the OLTP component owns a dedicated set of worker threads. The worker threads are collocated on a single NUMA node to avoid high synchronization overheads over the interconnect for sensitive parts of the architecture, such as epoch management, garbage collection, memory allocation, etc. The current implementation provides snapshot isolation guarantees for transactions using a multi-version concurrency control. Scaling this component over multiple NUMA nodes and across multiple machines is part of future work.

Scheduling: The responsibility for allocation of OLTP requests (stored procedure calls) to worker threads is assigned to the OLTP dispatcher that, similar to the OLAP dispatcher, schedules operations in batches working on one batch at-a-time. As depicted in Figure 1, incoming OLTP requests are first queued up in the OLTP queue while the system is busy executing the current batch of OLTP requests. When the current batch is done, the OLTP dispatcher dequeues all requests from the OLTP queue and enqueues them to the OLTP worker threads in a round robin fashion. This trades off individual query latency to obtain benefits that arise in evaluating many requests as a batch. For instance in our case, the logging of updates to durable storage, the epoch management for the lock-free data-structure, the garbage collection and the propagation of the updates to the OLAP replica all benefit from such batch-based execution. In such mode, threads can combine multiple operations in the same epoch and amortize the cost of modifying contested atomic counters. We have not yet explored using the OLTP request batching for optimizing the performance of the concurrency protocol, as proposed by BOHM [16].

Logging: For durability the OLTP dispatcher logs the successful update transactions to a durable storage medium before the responses are sent to the clients. To minimize the effect of logging on performance, we do command logging (similarly to VoltDB [38]). Note, that as we are using snapshot isolation the information on the read and committed snapshot versions needs to be also logged for correct recovery. Furthermore, logging is performed on a batch basis (as group-commit [12]) to hide the I/O latency for multiple OLTP requests. Note that as the OLAP replica is not backed up by a durable medium, in case of failures it needs to be recovered by reading a snapshot and catching up with new updates from the primary replica. This has been studied in several settings [25, 27].

Update propagation: To keep the secondary replica consistent and up-to-date, the OLTP component also exports a log of updates separate from the durable log. Unlike the durable log that contains logical updates, the propagated updates contain the physical updates to individual records. This enables efficient application of the updates on the secondary replica. To avoid expensive synchronization among OLTP worker threads, each thread prepares its own set of updates to be propagated. An example set of updates from a single thread is depicted in Figure 3. This example contains eight propagated updates from three committed transactions. Updates from a single thread may be interleaved with updates from other threads during the propagation process. For instance, in this example the updates from a transaction with version ID 2 are part of the update set of a different thread. Each update contains:

1. The **Type** of the update can be either a newly inserted tuple, an update to an existing tuple or a delete of an existing tuple;
2. The **RowID** integer which uniquely specifies the tuple that corresponds to this update. The RowID is equivalent to the primary key of the relation and is used to efficiently locate the corresponding tuple at the secondary replica;
3. The **Offset** and **Size** in bytes which are used to update existing tuples on finer sub-tuple granularity;
4. The **Data** which contains either the data of the newly inserted tuple or the payload of the update to an existing record.

As our results also show, propagating the updates as described above adds small performance overhead on this part of the system.

5. ANALYTICAL COMPONENT (OLAP REPLICA)

The analytical component is depicted on the right-hand side of Figure 1, and contains the secondary replica of the database engine.

Query scheduling: In order to avoid synchronization and performance interaction between running the OLAP queries and applying the OLTP updates, the OLAP dispatcher executes queries in batches. A batch is executed as a read-only transaction on the latest snapshot version. Before executing the next batch of queries, the dispatcher retrieves from the OLTP component the latest committed snapshot version, and applies the propagated updates on its replica up-to that version. Furthermore, as only one batch of queries is executed at-a-time, the OLAP engine does not need to store more than a single version of the data, i.e., can be version oblivious.

The batch scheduling of BatchDB is similar to the one used by Crescando [59] and SharedDB [19]. The technique of BatchDB differs from these systems as it batches all concurrent OLAP queries in the system to isolate the OLAP query processing from the updates propagated by the primary OLTP replica. The first version of HyPer also grouped the OLAP queries by *session*, and made sure that all queries in a session operated on the same snapshot version.

Query execution: Since the query scheduler executes queries in batches, even though it is not necessary, we use a query processing engine that also takes an advantage of shared execution. Our OLAP component uses ideas presented by prior work on shared scans [49, 59, 61] (to share memory bandwidth across scans and query predicate evaluation); more complex query processing [2, 8, 19, 36] (to share execution of join operations for more efficient utilization of CPU and DRAM bandwidth); and scheduling optimizations [20]. Prior experiments on this component [36] show that, for large workloads, it can provide higher throughput than state-of-the-art commercial engines in analytical query processing.

Update propagation: Finally, to enable fast application of updates, all tuples of replicated tables are annotated with a *RowID* integer attribute in both replicas. The RowID is essentially a primary key attribute hidden from the user. As described earlier, all

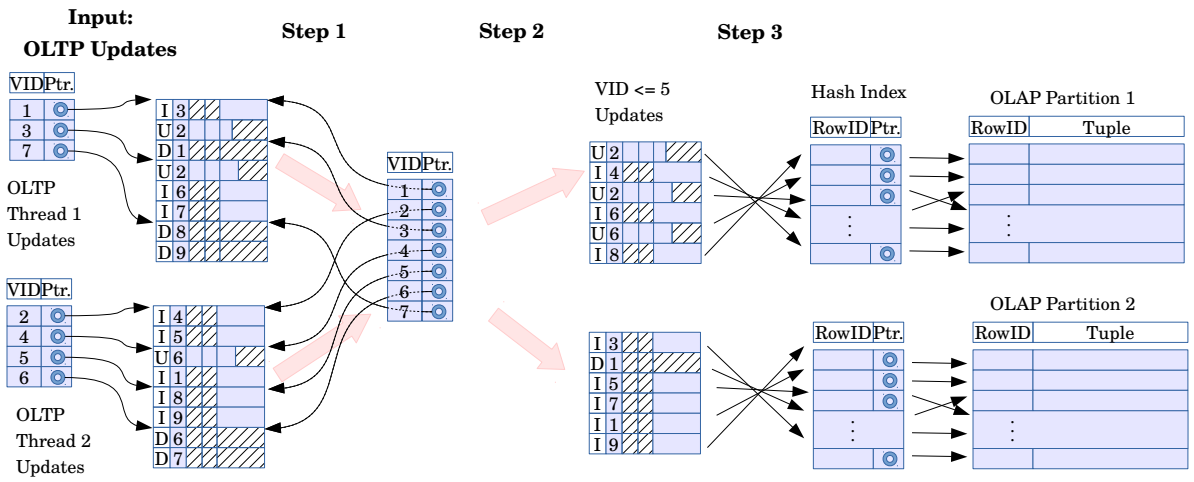


Figure 4: Process of applying propagated updates from OLTP replica into OLAP replica

propagated updates from the OLTP replica contain the RowID attribute which is used at the OLAP replica to uniquely identify the tuples referred by the updates. Furthermore, as show in Figure 1, the data in the OLAP replica is horizontally (soft) partitioned based on a hash value of the RowID attribute. This enables both efficient (NUMA-local) scan processing and fast application of OLTP updates on modern multi-core machines. Similar soft based partitioning has also been used by numerous other systems (e.g., [31, 46]).

To facilitate efficient matching of OLTP updates and tuple locations, the OLAP component maintains a hash index of the data on the RowID attribute. The process of applying the propagated updates using the RowID and the hash indexes consists of three steps which are illustrated in Figure 4[1]:

- In **step 1**, update sets from multiple OLTP threads are ordered by the snapshot version ID (VID). This step is the fastest as it only orders the update pointers using a scan with complexity linear in the number of new snapshot versions.
- **Step 2** is executed when the OLAP dispatcher obtains the latest committed snapshot version ID (in this example is 5) to be used for the current batch of queries. Thereafter, the updates corresponding to snapshot versions up to and including the VID are propagated to the corresponding partitions based on a hash value of the RowID attribute.
- In **step 3**, updates for each partition are applied using the hash index on RowID to find the location of tuples for operations that either update or delete a tuple. When a tuple is deleted, the slot for that tuple is marked as empty, as a signal for the scan processor to ignore the tuple at that location. In case of inserts, the tuple is inserted into the next free slot of the partition (possibly at a location where a tuple was recently deleted) and the hash index is populated accordingly. In case of updates, tuples are updated in place at the granularity of single attributes to avoid rewriting the entire tuple. The attribute to be modified is identified using the *offset* and *size* fields depicted in Figure 3.

All three steps are easily parallelizable. The most time consuming step is step 3, as it contains multiple random accesses. Technically, this step resembles a hash join with the hash index used as a hash table to join the updates with the existing data. In general, there has been a significant work on speeding up joins with state-of-the-art algorithms reaching hundreds of million of tuples per second on modern multi-cores [3]. To put these numbers into perspective the highest reported performance for the industry standard OLTP benchmark (TPC-C) would correspond to about tens of millions updated tuples per second. Therefore, we expect that this approach of propagating updates from an OLTP to an OLAP replica can satisfy the needs of most systems. Our algorithm optimizes for both main-memory bandwidth and latency. Hash buckets from the hash index are stored in a single cacheline and accessed using a grouped software prefetching technique [10] to minimize high random main-memory access latencies.

We also experimented with propagating updates with an array approach, where the RowID is directly used as an offset in the storage of the OLAP replica. However, we found that it did not bring significant performance gains compared to an optimized hash index approach, while on the other side requires coordination among OLTP worker threads in the use and reuse of RowID integers.

Storage layout and data transformation: The current implementation of the OLAP component uses an uncompressed row-oriented storage (as depicted in Figure 4), similar as to the OLTP replica. Note, however, that this is not a design requirement (i.e., we do not rely on physical replication) and as part of future work we plan to optimize it using a column-store format, which has been shown to be more efficient for analytical workloads, and can further benefit from compression, reduced I/O and memory costs. We do, however, evaluate the performance of our update propagation method using a micro-benchmark on a column-oriented format in Section 8.3.

6. NETWORK COMMUNICATION

BatchDB allows the OLAP replica to be either co-located on the same physical machine as the primary replica, or to reside on a different node. In the latter setup, the system makes use of modern low-latency networks to efficiently propagate the updates.

Remote Direct Memory Access (RDMA) [23] is used to improve the latency of small messages and reduce the CPU overhead for large data transfers by avoiding intermediate copy operations inside the network stack. RDMA has been used for several database

[1]Note that the *OLTP Thread 1 Updates* in Figure 4 are identical to the ones presented in Figure 3

systems [33, 50] and the ideas we use are based on the platforms developed for running query-at-a-time joins at large scales [4, 5]

RDMA offers one-sided and two-sided operations. When using one-sided read or write operations, the initiator of the request directly manipulates a section of memory which has previously been exposed by the remote machine. The CPU on the target machine is not notified nor involved in the data transfer. Two-sided operations on the other hand represent traditional message passing semantics. The receiver has to allocate several receive buffers and is notified when a new message has been placed in any of these locations. A downside of RDMA is that the application is burdened with extra complexity of network buffer management [18].

In BatchDB, each machine registers several RDMA receive buffers with the network card. Small messages of less than 1024 KB are directly written to these buffers using two-sided RDMA operations. Larger messages cannot be transmitted directly, and require a handshake operation. During the handshake, the sender transmits the required buffer size to the receiver, which in turn allocates a new buffer and registers it with the network card. The receiver responds with the buffer address and the necessary access credentials. After this exchange, the sender can initiate the data transfer using a one-sided RDMA write operation. Once the transfer is complete, the receiver is notified by the sender. To reduce the overhead of memory allocation and registration, large RDMA-buffers are pre-allocated and cached in a buffer pool.

In addition to a low latency, modern networks also provide high bandwidth, which is important for data-intensive applications. The communication mechanisms mentioned above allow for optimal use of the network bandwidth for both small and large messages. Propagating updates from the primary replica to one secondary copy does not fully saturate the throughput of a 4xFDR InfiniBand network. Not being limited by the network bandwidth enables the primary node to simultaneously transmit updates to multiple secondary copies, thus making the system elastic and scalable.

7. BEYOND OLTP AND OLAP

The system we have described so far is designed to run concurrent OLTP and OLAP workloads providing high performance isolation, and data freshness as well as support for many workload-specific optimizations. To provide these features, however, we make several trade-offs. One of these trade-offs is that due to the batched way of scheduling queries and updates in the OLAP replica, the latencies of all queries is determined by the slowest query. For this reason, the system is designed to process OLAP queries expected to be answered within a predictable time-frame of several seconds.

If a user would like to run a long-running (offline) analytical query that takes minutes or hours this would have a devastating effect on the latency of other concurrent queries. Here we outline a few ways of how we could address such and other types of queries.

Multi-version storage in secondary replica: With this approach the OLAP replica essentially replicates the primary OLTP replica containing fine-grained versions, but without storing the indexes that are necessary for the point-based OLTP workloads. Compared to a single replica, this still provides physical isolation of resources, and possibility to have workload-specific optimizations. Furthermore, concurrent queries can operate on any version of the data, hence long running and short running queries can be mixed without effect on latency. However, the scan processing must deal with the snapshot version information for each tuple and query, plus stale versions have to be garbage collected, leading to increased overall overhead to query processing. Thus, we do not prefer this approach and would recommend one of the options explained below.

Separate replica for different types of workloads: In this approach a separate replica will be used for different types of workloads and the same principles (single-snapshot replication and batch scheduling of queries and updates) will be used for each replica. Each replica will also be specialized for its specific type of workloads, including offline analytics, graph processing, machine learning, etc. This goes in a similar direction to the approach of federated databases that is currently gaining traction with systems like BigDAWG's Polystore [15] or the Cyclops's DBMS+ [34] that provide a single interface to multiple systems specialized for different types of data processing. The disadvantage is that it requires more hardware resources (e.g., more main-memory and CPUs) to keep the higher number of replicas.

Store fixed amount of versions in secondary replica: Another approach is to make a compromise between the above two options and maintain multiple versions at the secondary replica but with a much coarser granularity than at the primary OLTP replica. For instance, analytical queries could be divided into two categories: latency-sensitive online analytical queries and latency-insensitive offline analytical queries. The analytical component would then allow two concurrent batches of queries, one containing the latency-sensitive ones and the other containing the latency-insensitive queries. Compared to the first approach, this would induce less overhead on applying OLTP updates as only two versions will be maintained and processed by the scan. Compared to the second approach, this will not use as many hardware resources.

8. EVALUATION

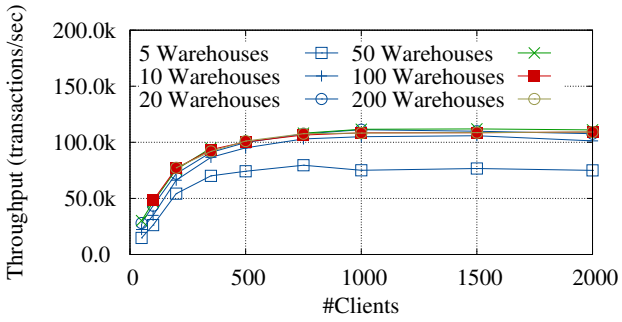
In this section, we provide an experimental analysis of the performance and behaviour of BatchDB. We first start with an experiment using a stand alone OLTP benchmark to evaluate our OLTP component and compare it to the state-of-the-art. We then measure the cost of applying the OLTP updates on the secondary OLAP replica. The main experiment uses a hybrid OLTP and OLAP workload to evaluate BatchDB's performance and predictability, and quantify the amount of cross-workload performance interaction. Using the same benchmark we also evaluate performance isolation for two related systems that support hybrid workloads. Finally, we investigate the performance impact of implicit sharing of resources.

8.1 Experimental setup

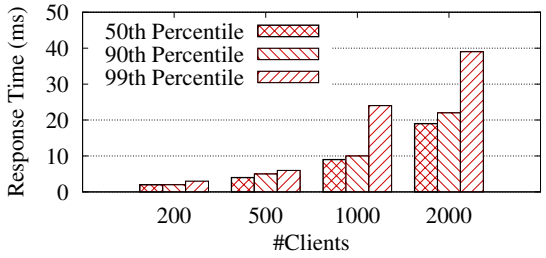
Infrastructure: Our testbed is a cluster of machines each with 4 Intel Xeon E5-4650 v2 (IvyBridge-EP) processors. Each processor has 10 physical cores (20 logical cores) running at 2.4 GHz with 25 MB shared L3 cache and 128 GB of DDR3 RAM running at 1833 MHz. The system is running on a Linux kernel with version 3.16.0-4. The machines are interconnected with a 4xFDR InfiniBand interconnect. In the experiments, BatchDB runs on one or two machines, and the client workload is generated on separate machines that communicate via a 1 Gbit Ethernet network.

Workload: We evaluate our system under a hybrid OLTP and OLAP workload using the CH-benCHmark [11]. The benchmark is a mix of standard OLTP and OLAP benchmarks, namely TPC-C and TPC-H. In particular, the OLTP part is an unmodified version of the TPC-C benchmark, while the OLAP part contains a set of analytical queries inspired by TPC-H. The latter are modified to match the TPC-C schema, plus a few missing TPC-H relations.

We modified the analytical queries of the CH-benCHmark for several reasons: First, as our OLAP query processor benefits from sharing opportunities, it is important that we do not have a workload with a large set of equivalent queries or query predicates. However, we observed that the original set of query predicates does not provide sufficient diversity even after randomizing the parame-



(a) Throughput



(b) Transaction Latencies (200 Warehouses)

Figure 5: TPC-C Performance

ters according to the TPC-H benchmark. One reason for this is that unlike in the TPC-H dataset where most attributes have a broad range of values, many attributes in the TPC-C dataset, and in particular the date fields, cover a narrow scope. Thus, we modified the queries by randomizing the predicate values and adding query predicates where applicable so as to not unduly benefit the shared execution approach. Second, our current OLAP query processor does not provide full SQL, and only supports scan, join and aggregate queries. Therefore, we modified some of the queries to be able to accommodate them. The final set of queries we used, including the randomized query predicates, is shown in Appendix 1. This set of queries is a good approximation for OLAP queries, as they consist of heavy full table scans and expensive join operations. Since the batched query scheduling logically isolates the OLAP query processor from the OLTP component and the update propagation mechanism, having support for full SQL and the original TPC-H queries would not have an impact on our results and conclusions.

8.2 OLTP Performance

In the first experiment we test standalone OLTP performance without propagating the updates to a secondary replica. We allocate 20 OLTP worker threads running on 10 physical cores (20 logical cores). We measure throughput in terms of transactions executed per second and transaction latency in terms of milliseconds, while we vary the number of warehouses (range of 5 to 200) and number of clients (up to 2000). In TPC-C the number of warehouses affects the database size (one warehouse contains about 70MB of data and 500 thousand tuples). The number of warehouses also indirectly impacts the amount of achievable concurrency in the system, as small number of warehouses will lead to higher contention within transactions of concurrent clients.

The results of this experiment are shown in Figure 5. The throughput results (Figure 5a) show that our system is able to achieve a maximum of 110 thousand TPC-C transactions (or around 3 million new order transactions per minute) when running with 200 warehouses and 2000 clients. While this is not the fastest performance reported for TPC-C, it is on par with related systems on comparable hardware. For instance, Silo [58] achieves around 22 thousand transactions per second per core, while BatchDB achieves 11 thousand. We would like to note, however, that Silo's implementation assigns a worker thread per warehouse, and hence limits the number of warehouses to the machine core count. As a result, it only processes data of up to 32 warehouses. Other recent systems, like HyPer [40] or TicToc [60] are able to achieve significantly higher throughput. HyPer achieves a throughput of about 100 thousand transactions per second in a single thread (700 thousand on 20 threads), however it relies on easily partitionable workloads. TicToc achieves a performance of about 4 million transactions per second on 40 threads, however it uses a subset of the TPC-C benchmark that relies on hash indexes exclusively.

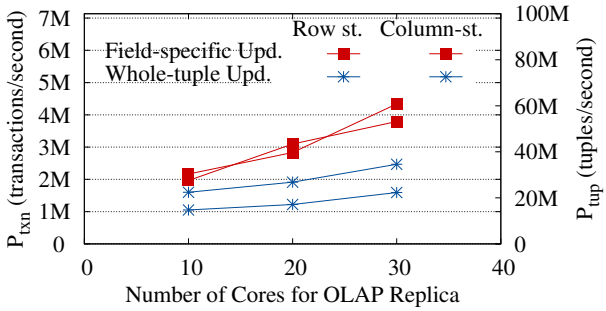


Figure 6: Update Propagation Power at OLAP Replica

The results depicted in Figure 5b show the impact of batching queries on transaction latencies. We note that at the point when maximum throughput is reached, the 99th percentile is less than 30ms, which is well below the 5 second limit for 90th percentile as per the TPC-C specification.

8.3 Update Propagation

The next experiment evaluates the efficiency of BatchDB's update propagation mechanism. In this setup, we enable replication and have the OLTP component propagate the updates to the OLAP component. In general, updates are pushed every 200ms, or when the OLAP component queries the OLTP component for the latest snapshot version. Updates are propagated only for relations that are used in the analytical workload, i.e., Stock, Customer, Order and Orderline. Together they account for a majority (around 85%) of all updated tuples. For this experiment we measure CPU time spent applying the OLTP updates at the OLAP replica (t_{upd}), the total number of update tuples ($\#Tup$), and the total number of transactions committed ($\#Txn$) during the experiment. $\#Tup$ is the total number of tuples that are either inserted, modified or deleted during the experiment. Using this we derive the *update propagation power* in terms of updated tuples per second (eq. 1) and transactions per second (eq. 2). The update propagation power denotes the average update rate achieved throughout the experiment, providing an estimate on the overhead of keeping the OLAP replica up-to-date.

$$P_{tup} = \frac{\#Tup}{t_{upd}} \quad (1)$$

$$P_{txn} = \frac{\#Txn}{t_{upd}} \quad (2)$$

The results are shown in Figure 6, where we increase the number of cores allocated to the OLAP replica and measure its update propagation power when varying two factors:

Table 1: CPU Time Spent per Step and Relation for Update Propagation

		Relations (**S**tock, **C**ust., **O**rder, **O**rder**L**ine)				
		S	C	O	OL	Total
% of updated tuples		31	6	3	27	67
% of inserted tuples		0	0	3	30	33
Total		31	6	6	57	100
% of CPU time on	S1	2	2	2	2	8
applying whole	S2	7	2	2	10	21
records updates	S3	40	9	2	20	71
Total		49	13	6	32	100
% of CPU time on	S1	3	3	3	3	12
applying field	S2	11	5	3	16	35
specific updates	S3	9	11	2	31	53
Total		23	19	8	50	100

1. Propagating field-specific or whole record updates,
2. Using a row- or column-oriented storage format.

First, we compare the update propagation power to the performance at which the OLTP component operates. Looking at the case when the OLAP replica uses the same amount of resources as the OLTP replica (10 cores), we can see that the update propagation power is around 2 million transactions per second, while the OLTP component processes about 110 thousand transactions per second. Thus, updates at the secondary replica are applied 20 times faster than they are generated at the primary replica hinting at the negligible overhead it has on the OLAP performance.

Second, we note that since the OLAP data is partitioned by the row ID attribute, the update propagation speed improves as we increase the amount of allocated resources to the secondary replica.

Third, we can observe that using field-specific updates does not result in any significant performance degradation when using a column oriented storage. However, when updating whole tuples its performance decreases by more than two fold. This is to be expected, as updating whole records in a column-store results in more random DRAM accesses.

Finally, we note that the current implementation with 30 cores is able to sustain an update rate of about 4 million transactions per second (or 250 million transactions per minute). For reference this is more than the best reported TPC-C results, confirming that this approach is applicable to other systems.

We also investigate the CPU time spent applying the updates for each step and per relation for a row-store format. The results of this experiment are shown in Table 1. The update tuple distribution per relation is shown in the first two rows (% of updated and % of inserted tuples). The table only shows the relations which are used in the analytical workload and are modified by the TPC-C benchmark. These numbers highlight the difference on how the two approaches apply the updates to existing tuples. In the first approach where updates are applied to *full records*, most of the CPU time is spent updating the Stock relation (row 7), even though most of the updated tuples belong to the Orderline relation (row 3). The reason for this is that Stock records are very wide (319 bytes), making an update to an existing record expensive. In the second approach, when using *field-specific* updates to update only the changed attributes, the updates to Stock tuples take significantly less CPU Time (last row), which in turn improves performance (recall Figure 6). Table 1 also shows that step 1 from the update propagation algorithm described in Section 4 is only a small fraction of the total CPU time spent, and that the applying of updates in step 3 is the most expensive part.

8.4 Hybrid Workload Performance

In the following experiment we test our system with a hybrid workload based on the CH-benCHmark and measure the impact that the OLTP and OLAP workloads have on each others' performance. We use an initial dataset of 100 warehouses for all measurements. Each run is 2 minutes long, where the first 30 and last 10 seconds are warmup and cooldown periods. We vary both the transactional and analytical clients from 0 to 2000 and measure: throughput; 50^{th}, 90^{th} and 99^{th} response time percentiles; and CPU utilization.

The results are shown in Figure 7, where the top two rows show the impact that the OLTP workload has on the OLAP throughput (Figure 7a) and latencies (Figure 7b), and the bottom two rows show the CPU utilization (Figure 7c) and how much the OLAP workload affects the performance of the OLTP throughput (Figure 7d), and response times (Figure 7e).

On a high level, the results demonstrate that BatchDB achieves good performance isolation for both workloads, which leads to predictable performance. The throughput degradation reaches up to 10% when the two replicas are distributed across two machines, and up to 20% when they are co-located on the same machine. The distributed case also has a smaller impact on the 99th response time percentiles for both workloads.

We now analyze the graphs in more detail. The OLAP throughput is shown in Figure 7a. The left-most plot, titled Local (NUMA) Replicas, shows a significant drop in OLAP throughput as the OLTP load increases. This is a result of the database size increasing during the execution of the benchmark. Running the TPC-C workload on the OLTP replica at a rate of 110 thousand transactions per second (about 3 million new orders per minute) for 2 minutes results in an increase of the dataset size for almost 200 warehouses of new orders and orderlines (or triple the initial size). This coincides with the registered 50% drop on average throughput.

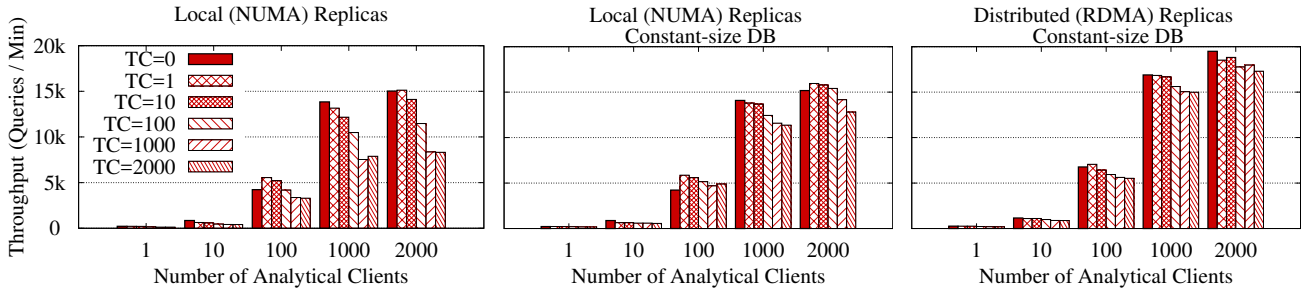
To isolate the effects of increased dataset size, we modified the TPC-C benchmark so that the New-Order transaction also deletes old orders and orderlines to keep the database size constant. The results of this experiment are presented in the plots on the right side, which show a much lower overhead of up to 10% and 20% depending on whether the OLAP replica runs on the same machine (separate NUMA nodes) or on a different machine.

The increase in analytical throughput at 100 analytical clients under OLTP load happens due to an artifact that comes from sharing of query execution in the OLAP replica. In particular, when the OLAP replica is underloaded (i.e., when the processing time of a batch is sublinear to the number of queries in the batch), slowing down the OLAP query scheduler (due to the OLTP workload) causes queries to wait less in the OLAP query queue, which improves overall performance. This effect goes away when the OLAP replica operates at maximum throughput.

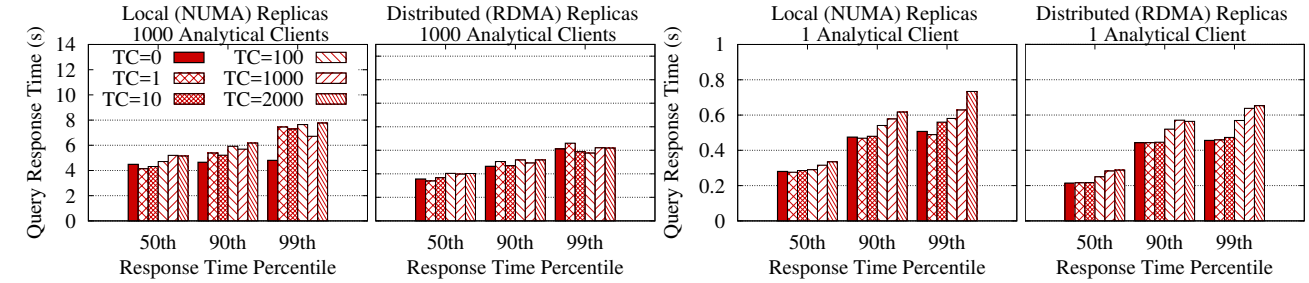
The CPU utilization shown in Figure 7c demonstrates two important aspects of BatchDB. First, it shows the clear separation of resources dedicated to each component (1 socket to the OLTP and 3 sockets to the OLAP component). Second, it demonstrates the effect of shared query execution, which was used in our OLAP component as even though the CPU of the OLAP component is already saturated for 1 client, throughput keeps increasing as the number of clients goes up to 1000 clients. Note that the main design principles of BatchDB, do not depend on shared query execution within the OLAP component to achieve high performance isolation.

Figure 7b shows the impact of the OLTP workload on OLAP response times. The results indicate that response times remain predictable, as the 99th percentile is increased by at most 50%.

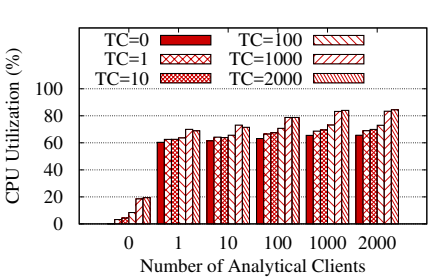
The impact of OLAP on OLTP performance is shown in Figure 7d (throughput) and Figure 7e (response time). For reference,



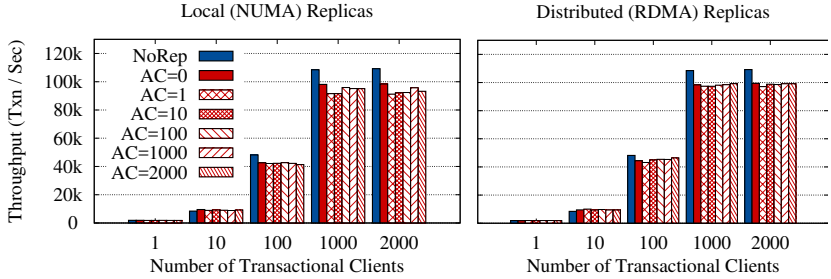
(a) Impact of OLTP Workload on OLAP Throughput



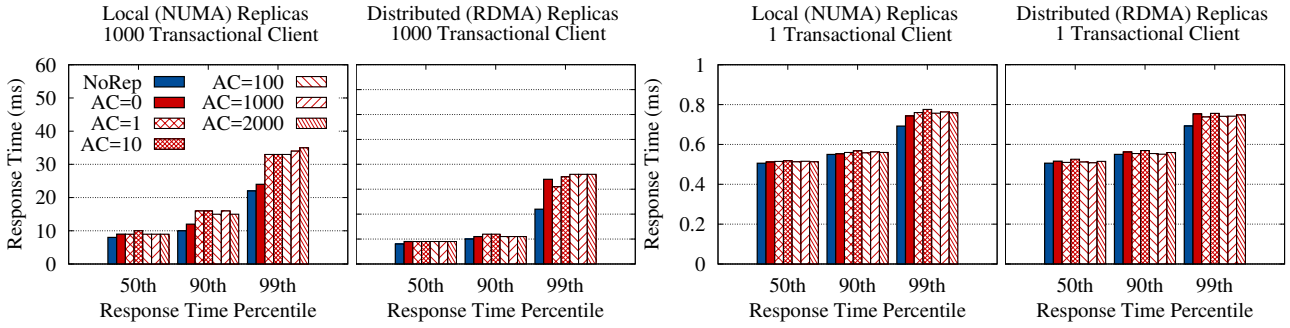
(b) Impact of OLTP Workload on OLAP Response Times



(c) CPU Utilization (Local Replicas)



(d) Impact of OLAP Workload on OLTP Throughput



(e) Impact of OLAP Workload on OLTP Response Times

Figure 7: BatchDB Hybrid CH (OLTP + OLAP) Benchmark Performance for 100 Warehouses

we also include the numbers for a system setup where updates are not propagated to an OLAP replica (NoRep). We observe that the maximum throughput is degraded by at most 10% when propagating the updates. Running the OLAP workload on the same machine as the OLTP workload incurrs an insignificant additional overhead of maximum 7%, which is not even present in the case where OLAP runs on a separate machine. This shows that our approach of replicating both within a machine and across machines allows us to achieve high isolation. Looking at the response time graphs, at maximum performance the increase in the 99th percentile is due to the periodic pushing of propagated updates. In the current implementation of BatchDB the updates are pushed either every 200ms or when the OLAP dispatcher schedules a new batch of queries, whichever comes first. Nevertheless, the maximum 99th percentile is still under 40ms and is well bellow the 5s limit of TPC-C.

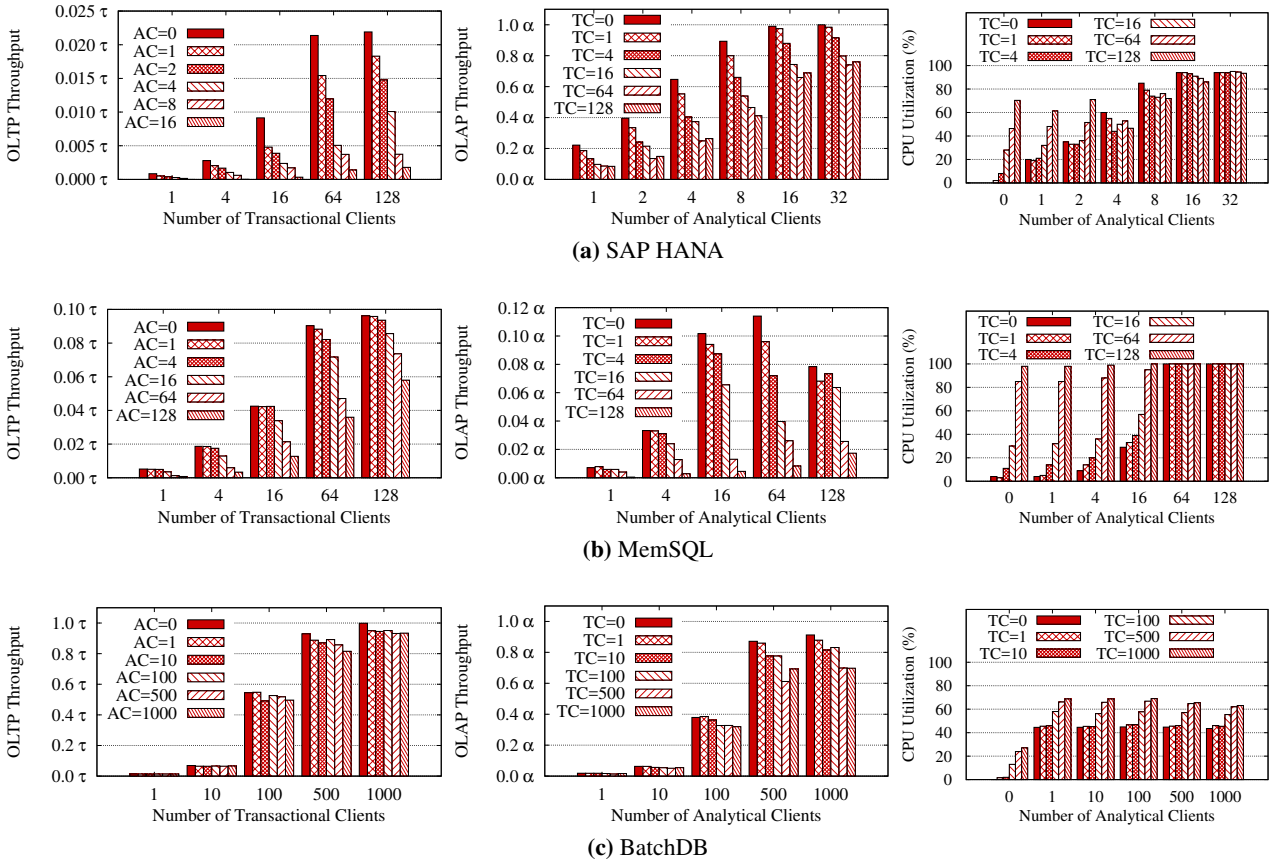


Figure 8: Hybrid Workload Interaction for SAP HANA, MemSQL and BatchDB on a Cloud Machine.

Another important take away from these results is the smoothing effect that the batch scheduling has on query latency, which is also part of the trade-off we make to achieve the performance isolation. In our case, all queries take the same amount to execute, leading to the 50th, 90th and 99th query latency percentiles be almost identical. As discussed before, we believe this is prefered by users expecting predictable response times based on certain SLAs.

8.5 Comparison to Related Systems

Next we provide experimental comparison with related systems that handle hybrid workloads. We ran the CH-benCHmark on two state-of-the-art commercial engines that support hybrid transactional and analytical workloads – SAP HANA and the community edition of MemSQL. We ran the benchmark for all three systems on a virtual machine on the Amazon cloud (as it is not possible to run SAP HANA on our local machine).

As a benchmark driver we took the code from Psaroudakis et al. [47] and modified the OLAP part to use the same set of queries we used for evaluating BatchDB. Note that in the case of HANA, we essentially corroborated the results published by Psaroudakis et al. [47], albeit for a modified workload and on different hardware. The Amazon cloud machine we used is an 'r3.8xlarge' containing two Xeon E5-2670 v2 processores with 32 vCPUs in total, and 244 GBs of main memory. The clients were run on a separate virtual machine in the same availability zone.

For MemSQL we used a separate benchmark driver for the OLTP part based on tpcc-mysql [57] (the OLTP part of the CH-benCHmark is identical to TPC-C), which was modified not to use prepared statements (as they are not supported by MemSQL). Even though MemSQL is designed for distributed query processing, we primarily focused on how well it can isolate the performance of the OLTP and OLAP workloads on a single node. More concretely, we ran MemSQL on a single machine, by directly querying a leaf node – we found that this provides highest throughput for both OLTP and OLAP. Furthermore, we colocated the TPC-C clients on the same machine as the database, as we found that when they were running on a separate machine, the database engine was not saturated. To avoid performance interference due to the co-location, we dedicate 4 physical cores of the machine to the TPC-C clients and the remaining 12 to the MemSQL engine.

The results are shown in Figures 8a, 8b, and 8c for HANA, MemSQL and BatchDB respectively. We use finer granularity for the workload size for HANA and MemSQL as both systems reach maximum throughput and exhibit higher performance interactions already for smaller number of clients compared to BatchDB. For all systems we show (from left to right): the impact of OLAP workload on OLTP throughput; the impact of OLTP workload on OLAP throughput; and the CPU utilization during each experiment. As we can not show absolute performance numbers, similar to the survey by Psaroudakis et al. [47], we only report relative system throughput. I.e., τ and α correspond to the maximum transactional and analytical throughput observed on the cloud machine.

With the results shown in Figure 8a we corroborate the findings of Psaroudakis et al. [47] that, for HANA, resource interfer-

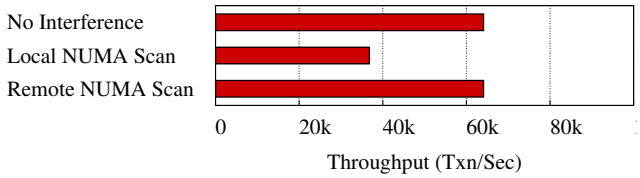


Figure 9: Performance effect on OLTP workload when colocated with a multi-threaded scan operation

ence has a significant effect on the system performance. In particular the transactional throughput is heavily affected when collocated with large OLAP workloads, dropping by more than 5 times. Psaroudakis et al. [47] reported similar degradation for OLTP throughput for the HyPer version which relied on fork system call to handle hybrid workloads. Note, that the new version of HyPer has a different method [40], but there are no published performance numbers when running hybrid workloads for comparison. Additionally, we could not get a hold of a new binary due to proprietary constraints. For MemSQL (Fig. 8b) the situation is reversed – the OLAP throughput is the one that suffers from high OLTP workload, also dropping by more than five times. Unlike for SAP HANA and MemSQL, BatchDB (Fig. 8c) is able to provide high performance isolation for both transactional and analytical workloads similarly as to the results for our on-premises machine (Fig. 7).

The experiments on related systems show that having dedicated resources for each workload type is necessary for isolating performance as, depending on the database engine's scheduling policies, performance of one or another workload may be significantly affected. An immediate solution for HANA and MemSQL would be to have two dedicated sets of CPU cores for OLTP and OLAP workloads respectively. However, as these systems handle both OLTP and OLAP requests on the same replica there will be implicit sharing of resources such as CPU caches and memory bandwidth, which can also cause performance degradation for the OLTP workload as we show in Section 8.6.

8.6 Implicit Resource Sharing

In this experiment we evaluate the impact that a standard OLAP operation, (i.e., a multi-threaded scan operator), can have on the performance of an OLTP workload when colocated on the same NUMA node. Unlike in the previous experiments, in this case we disable the update propagation from the OLTP replica, and both the OLTP component and the analytical scan run on separate datasets and in a separate process.

We allocate half a CPU, i.e., 5 physical cores (10 logical cores) to the OLTP process and measure its performance in three different scenarios: (1) Where it runs on its own, without anything else running on the machine; (2) Where it is co-located with the bandwidth-intensive scan running on the other 5 cores, belonging to the same CPU with its data also placed on the same NUMA node; and (3) Where the scan runs on 5 cores on a separate CPU, with data also placed on a separate NUMA node (local to the scan). The second case corresponds to a setup where both OLAP and OLTP are handled using the same replica of the data sharing hardware resources such as CPU caches and main memory bandwidth. The last case resembles a setup where OLTP and OLAP data are replicated across NUMA nodes, which is what we are using in our experiments.

The results in Figure 9 show that a CPU and NUMA co-located bandwidth-intensive scan results in almost 50% performance degradation of the OLTP throughput. Similarly, we observed no performance degradation for the performance of the scan compared to its run in isolation (result not shown in the graph). This experiment also shows that colocating OLTP and OLAP workload on the same NUMA node can cause unbalanced sharing of hardware resources and unfair disadvantage to OLTP performance. Similar observations for cross-workload interference were also reported by Tang et al. [55] and Lo et al. [35] in their evaluation of effects of sharing the memory sub-system among Google's datacenter applications.

This shows that in order to achieve performance isolation, it is very important to isolate the data of OLTP and OLAP workloads on different NUMA nodes, which entails the need for replication.

8.7 Major Takeaways

The above experiments demonstrate the need for dedicated CPU resources (§ 8.5) and separate replicas (§ 8.6) for isolating OLTP and OLAP performance in a single machine. Not having dedicated resources leads to unbalanced performance degradation of OLTP or OLAP depending on the scheduling policy, while dedicating CPU cores but handling both OLTP and OLAP on the same (or partially replicated) data can degradate OLTP performance due to implicit sharing of hardware resources such as CPU caches and memory bandwidth. Therefore, to achieve performance isolation, we need to trade-off space and handle OLTP and OLAP workloads on separate replicas, isolated on at least separate NUMA nodes.

The challenge with replication is to sustain high performance for OLAP workloads while keeping the OLAP replica up-to-date under heavy OLTP load. The experiments from § 8.3 show that, by using a physical log at fine granularity and using techniques from fast in-memory hash join algorithms, updates can be applied faster than any published TPC-C performance. The experiments from § 8.4 show that, by isolated execution of OLTP updates and OLAP queries (through scheduling of one batch of queries and updates at-a-time) and maintaining a single snapshot version of the data at-a-time, the OLAP workload exhibits negligible overhead to performance while operating on the latest versions of the data and under heavy OLTP load. Furthermore, replicas can also be distributed over RDMA and InfiniBand for added elasticity of the system.

Note that these techniques are generic and can be used in conjunction with existing methods that optimize OLAP processing such as dual column storage formats and compression.

9. CONCLUSION

In this paper we presented BatchDB, a system that efficiently handles hybrid OLTP and OLAP workloads with strong performance isolation. BatchDB relies on primary-secondary form of replication with the primary replica handling OLTP workloads and updates propagated to a secondary replica that handles OLAP workloads. To enable query processing on latest data with snapshot isolation guarantees and minimum impact on query performance, the queries and updates at the OLAP replica are queued and scheduled in batches with the system working on one batch at-a-time. Furthermore, updates are extracted and applied from the OLTP replica at the OLAP replica efficiently, incurring a small overhead to overall execution time. Updates can also be propagated to a remote replica via RDMA over InfiniBand for added scalability and isolation of performance. The results confirm that, unlike existing systems, BatchDB is able provide high performance isolation between the workloads leading to predictable performance.

Acknowledgements

Jana Giceva's work has been funded in part by a Google Fellowship. The work of Darko Makreshanski and Claude Barthels have been funded in part by a grant from Oracle Labs. Some of the work on which BatchDB is based has been funded in part by Amadeus.

10. REFERENCES

[1] J. Arulraj, A. Pavlo, and P. Menon. Bridging the Archipelago Between Row-Stores and Column-Stores for Hybrid Workloads. SIGMOD, pages 583–598, 2016.

[2] S. Arumugam, A. Dobra, C. M. Jermaine, N. Pansare, and L. Perez. The DataPath System: A Data-centric Analytic Processing Engine for Large Data Warehouses. In *SIGMOD*, pages 519–530, 2010.

[3] C. Balkesen, J. Teubner, G. Alonso, and M. T. Özsu. Main-memory hash joins on multi-core CPUs: Tuning to the underlying hardware. In *ICDE*, pages 362–373, 2013.

[4] C. Barthels, S. Loesing, G. Alonso, and D. Kossmann. Rack-Scale In-Memory Join Processing using RDMA. In *SIGMOD*, pages 1463–1475, 2015.

[5] C. Barthels, I. Müller, T. Schneider, G. Alonso, and T. Hoefler. Distributed join algorithms on thousands of cores. *PVLDB*, 10(5):517–528, 2017.

[6] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1987.

[7] L. Braun, T. Etter, G. Gasparis, M. Kaufmann, D. Kossmann, D. Widmer, A. Avitzur, A. Iliopoulos, E. Levy, and N. Liang. Analytics in Motion: High Performance Event-Processing AND Real-Time Analytics in the Same Database. In *SIGMOD*, pages 251–264, 2015.

[8] G. Candea, N. Polyzotis, and R. Vingralek. A Scalable, Predictable Join Operator for Highly Concurrent Data Warehouses. *VLDB*, pages 277–288, 2009.

[9] G. Candea, N. Polyzotis, and R. Vingralek. Predictable Performance and High Query Concurrency for Data Analytics. *VLDBJ*, pages 227–248, 2011.

[10] S. Chen, A. Ailamaki, P. B. Gibbons, and T. C. Mowry. Improving Hash Join Performance Through Prefetching. In *Proc. ICDE 2004*, pages 116–, 2004.

[11] R. Cole, F. Funke, L. Giakoumakis, W. Guy, A. Kemper, S. Krompass, H. Kuno, R. Nambiar, T. Neumann, M. Poess, K.-U. Sattler, M. Seibold, E. Simon, and F. Waas. The Mixed Workload CH-benCHmark. In *DBTest*, pages 8:1–8:6, 2011.

[12] D. J. DeWitt, R. H. Katz, F. Olken, L. D. Shapiro, M. R. Stonebraker, and D. A. Wood. Implementation techniques for main memory database systems. In *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, SIGMOD '84, pages 1–8, New York, NY, USA, 1984. ACM.

[13] C. Diaconu, C. Freedman, E. Ismert, P.-A. Larson, P. Mittal, R. Stonecipher, N. Verma, and M. Zwilling. Hekaton: SQL Server's Memory-optimized OLTP Engine. In *SIGMOD*, pages 1243–1254, 2013.

[14] J. Dittrich and A. Jindal. Towards a One Size Fits All Database Architecture. In *CIDR*, 2011.

[15] A. Elmore, J. Duggan, M. Stonebraker, M. Balazinska, U. Cetintemel, V. Gadepally, J. Heer, B. Howe, J. Kepner, T. Kraska, S. Madden, D. Maier, T. Mattson, S. Papadopoulos, J. Parkhurst, N. Tatbul, M. Vartak, and S. Zdonik. A Demonstration of the BigDAWG Polystore System. *PVLDB*, 8(12):1908–1911, Aug. 2015.

[16] J. M. Faleiro and D. J. Abadi. Rethinking Serializable Multiversion Concurrency Control. *PVLDB*, 8(11):1190–1201, July 2015.

[17] F. Färber, N. May, W. Lehner, P. Große, I. Müller, H. Rauhe, and J. Dees. The SAP HANA Database – An Architecture Overview. *IEEE Data Eng. Bull.*, 35(1):28–33, 2012.

[18] P. W. Frey and G. Alonso. Minimizing the hidden cost of RDMA. In *ICDCS*, pages 553–560, 2009.

[19] G. Giannikis, G. Alonso, and D. Kossmann. SharedDB: Killing One Thousand Queries with One Stone. *VLDB*, pages 526–537, 2012.

[20] J. Giceva, G. Alonso, T. Roscoe, and T. Harris. Deployment of Query Plans on Multicores. *PVLDB*, 8(3):233–244, 2014.

[21] A. K. Goel, J. Pound, N. Auch, P. Bumbulis, S. MacLean, F. Färber, F. Gropengiesser, C. Mathis, T. Bodner, and W. Lehner. Towards Scalable Real-time Analytics: An Architecture for Scale-out of OLxP Workloads. *PVLDB*, 8(12):1716–1727, Aug. 2015.

[22] M. Grund, J. Krüger, H. Plattner, A. Zeier, P. Cudre-Mauroux, and S. Madden. HYRISE: A Main Memory Hybrid Storage Engine. *PVLDB*, 4(2):105–116, 2010.

[23] J. Hilland, P. Cully, J. Pinkerton, and R. Recio. RDMA Protocol Verbs Specification (Version 1.0). Technical report, RDMA Consortium, 04 2003.

[24] C. Hong, D. Zhou, M. Yang, C. Kuo, L. Zhang, and L. Zhou. KuaFu: Closing the parallelism gap in database replication. In *Proc. ICDE 2013*, pages 1186–1195, 2013.

[25] R. Jiménez-Peris, M. Patiño Martínez, and G. Alonso. Non-Intrusive, Parallel Recovery of Replicated Data. In *SRDS '02*, pages 150–, 2002.

[26] R. Kallman, H. Kimura, J. Natkins, A. Pavlo, A. Rasin, S. Zdonik, E. P. C. Jones, S. Madden, M. Stonebraker, Y. Zhang, J. Hugg, and D. J. Abadi. H-store: A High-performance, Distributed Main Memory Transaction Processing System. *PVLDB*, 1(2):1496–1499, Aug. 2008.

[27] B. Kemme, A. Bartoli, and O. Babaoglu. Online reconfiguration in replicated databases based on group communication. In *DSN '01*, pages 117–126, July 2001.

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

[29] T. Lahiri, S. Chavan, M. Colgan, D. Das, A. Ganesh, M. Gleeson, S. Hase, A. Holloway, J. Kamp, T. H. Lee, J. Loaiza, N. Macnaughton, V. Marwah, N. Mukherjee, A. Mullick, S. Muthulingam, V. Raja, M. Roth, E. Soylemez, and M. Zait. Oracle Database In-Memory: A dual format in-memory database. In *ICDE*, pages 1253–1258, April 2015.

[30] P.-A. Larson, A. Birka, E. N. Hanson, W. Huang, M. Nowakiewicz, and V. Papadimos. Real-time Analytical Processing with SQL Server. *PVLDB*, 8(12):1740–1751, 2015.

[31] V. Leis, P. Boncz, A. Kemper, and T. Neumann. Morsel-driven Parallelism: A NUMA-aware Query Evaluation Framework for the Many-core Age. In *SIGMOD*, pages 743–754, 2014.

[32] J. J. Levandoski, D. B. Lomet, and S. Sengupta. The Bw-Tree: A B-tree for New Hardware Platforms. In *ICDE*, pages 302–313, 2013.

[33] F. Li, S. Das, M. Syamala, and V. R. Narasayya. Accelerating relational databases by leveraging remote memory and RDMA. In *SIGMOD*, pages 355–370, 2016.

[34] H. Lim, Y. Han, and S. Babu. How to Fit when No One Size Fits. In *CIDR*, 2013.

[35] D. Lo, L. Cheng, R. Govindaraju, P. Ranganathan, and C. Kozyrakis. Heracles: Improving Resource Efficiency at Scale. In *Proceedings of the 42Nd Annual International Symposium on Computer Architecture*, ISCA '15, pages 450–462, 2015.

[36] D. Makreshanski, G. Giannikis, G. Alonso, and D. Kossmann. MQJoin: Efficient Shared Execution of Main-memory Joins. *PVLDB*, 9(6):480–491, Jan. 2016.

[37] D. Makreshanski, J. Levandoski, and R. Stutsman. To Lock, Swap, or Elide: On the Interplay of Hardware Transactional Memory and Lock-free Indexing. *PVLDB*, 8(11):1298–1309, July 2015.

[38] N. Malviya, A. Weisberg, S. Madden, and M. Stonebraker. Rethinking main memory OLTP recovery. In *ICDE*, pages 604–615, March 2014.

[39] T. Mühlbauer, W. Rödiger, A. Reiser, A. Kemper, and T. Neumann. ScyPer: A Hybrid OLTP&OLAP Distributed Main Memory Database System for Scalable Real-Time Analytics. In *DanaC*, pages 499–502, 2013.

[40] T. Neumann, T. Mühlbauer, and A. Kemper. Fast Serializable Multi-Version Concurrency Control for Main-Memory Database Systems. In *SIGMOD*, pages 677–689, 2015.

[41] Oracle Change Data Capture. https://docs.oracle.com/cd/B28359_01/server.111/b28313/cdc.htm.

[42] E. Pacitti and E. Simon. Update propagation strategies to improve freshness in lazy master replicated databases. *VLDBJ*, 8(3):305–318, 2000.

[43] O. W. Paper. Oracle GoldenGate 12c: Real-Time Access to Real-Time Information. Technical report, Oracle, 03 2015.

[44] C. Plattner, G. Alonso, and M. Özsu. Extending DBMSs with satellite databases. *VLDBJ*, 17(4):657–682, July 2008.

[45] H. Plattner. A Common Database Approach for OLTP and OLAP Using an In-memory Column Database. In *SIGMOD*, pages 1–2, 2009.

[46] I. Psaroudakis, T. Scheuer, N. May, A. Sellami, and A. Ailamaki. Scaling Up Concurrent Main-memory Column-store Scans: Towards Adaptive NUMA-aware Data and Task Placement. *PVLDB*, 8(12):1442–1453, Aug. 2015.

[47] I. Psaroudakis, F. Wolf, N. May, T. Neumann, A. Böhm, A. Ailamaki, and K.-U. Sattler. Scaling Up Mixed Workloads: A Battle of Data Freshness, Flexibility, and Scheduling. In *Performance Characterization and Benchmarking. Traditional to Big Data, TPCTC 2014*, pages 97–112.

[48] L. Qiao, V. Raman, F. Reiss, P. J. Haas, and G. M. Lohman. Main-memory Scan Sharing for Multi-core CPUs. *VLDBJ*, pages 610–621, 2008.

[49] V. Raman, G. Swart, L. Qiao, F. Reiss, V. Dialani, D. Kossmann, I. Narang, and R. Sidle. Constant-Time Query Processing. In *ICDE*, pages 60–69, 2008.

[50] W. Rödiger, T. Mühlbauer, A. Kemper, and T. Neumann. High-speed query processing over high-speed networks. *PVLDB*, pages 228–239, 2015.

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

[52] N. Shamgunov. The MemSQL In-Memory Database System. In J. J. Levandoski and A. Pavlo, editors, *IMDM@VLDB*, 2014.

[53] Get started with Columnstore for real time operational analytics. https://msdn.microsoft.com/en-us/library/dn817827.aspx.

[54] M. Stonebraker and A. Weisberg. The VoltDB Main Memory DBMS. *IEEE Data Eng. Bull.*, 36(2):21–27, 2013.

[55] L. Tang, J. Mars, N. Vachharajani, R. Hundt, and M. L. Soffa. The Impact of Memory Subsystem Resource Sharing on Datacenter Applications. In *ISCA*, pages 283–294, 2011.

[56] A. Thomson, T. Diamond, S.-C. Weng, K. Ren, P. Shao, and D. J. Abadi. Calvin: Fast Distributed Transactions for Partitioned Database Systems. SIGMOD, pages 1–12, 2012.

[57] https://github.com/Percona-Lab/tpcc-mysql.

[58] S. Tu, W. Zheng, E. Kohler, B. Liskov, and S. Madden. Speedy transactions in multicore in-memory databases. In *SOSP*, pages 18–32, 2013.

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

[60] X. Yu, A. Pavlo, D. Sanchez, and S. Devadas. Tictoc: Time traveling optimistic concurrency control. In *SIGMOD*, pages 1629–1642, 2016.

[61] M. Zukowski, S. Héman, N. Nes, and P. Boncz. Cooperative Scans: Dynamic Bandwidth Sharing in a DBMS. In *VLDB*, pages 723–734, 2007.

APPENDIX

A. ANALYTICAL QUERIES

Listing 1 shows the analytical query templates of the OLAP workload used for the experiments. The join predicates have been ommited for brevity. In the remaining predicates `[NNAME]` is chosen at random from the 62 different nation names, `[RNAME]` is chosen randomly from the 5 different region names, `[CHAR]` is a random small case character, `[DATE]` is a random first day of a month between 1993 and 1997, `[PRICE]` is a random integer between 0 and 100, and `[QUANTITY]` is a random integer between 0 and 10;

Listing 1: Analytical Queries Used

```
-- Query 2
SELECT SUM(s_quantity)
FROM region, nation, supplier, stock, item
WHERE [JOIN_PREDICATES]
  AND s_i_id = i_id AND r_name = [RNAME]
  AND i_data like '[CHAR]%';
-- Query 3
SELECT SUM(ol_amount)
FROM nation, customer, orders, orderline
WHERE [JOIN_PREDICATES] AND n_name = [NNAME];
-- Query 5
SELECT SUM(ol_amount)
FROM region cr, region sr, nation cn, nation sn,
  customer, supplier, orders, orderline
WHERE [JOIN_PREDICATES
  AND cr.r_name = [RNAME] AND sr.r_name = [RNAME];
-- Query 7
SELECT SUM(ol_amount)
FROM nation cn, nation sn, customer, supplier,
  stock, orders, orderline
WHERE [JOIN_PREDICATES]
  AND cn.n_name = [NNAME_1]
  AND sn.n_name = [NNAME_1]
  AND ol_delivery_d BETWEEN DATE '2000-01-01'
                        AND DATE '2020-12-31';
-- Query 8
SELECT SUM(ol_amount)
FROM region cr, nation cn, nation sn, customer,
  orders, orderline, item, supplier
WHERE [JOIN_PREDICATES]
  AND cr.r_name = [RNAME]
  AND sn.n_name = [NNAME]
  AND i_data like '[CHAR]%';
-- Query 9
SELECT SUM(ol_amount) FROM orderline, item
WHERE [JOIN_PREDICATES]
  AND i_data LIKE '[CHAR1][CHAR2]%';
-- Query 10
SELECT SUM(ol_amount)
FROM orderline WHERE ol_delivery_d >= [DATE];
-- Query 11
SELECT SUM(s_order_cnt)
FROM nation, supplier, stock
WHERE [JOIN_PREDICATES] AND n_name = [NNAME];
-- Query 12
SELECT COUNT(*) FROM orders, orderline
WHERE [JOIN_PREDICATES]
  AND ol_delivery_d >= [DATE]
  AND o_carrier BETWEEN 1 AND 2;
-- Query 14
SELECT SUM(ol_amount) FROM item, orderline
WHERE [JOIN_PREDICATES]
  AND i_data LIKE '[CHAR1][CHAR2]%'
  AND ol_delivery_date >= [DATE];
-- Query 16
SELECT COUNT(*)
FROM item, supplier, orderline
WHERE [JOIN_PREDICATES]
  AND i_data NOT LIKE '[CHAR1][CHAR2]%'
  AND s_comment LIKE '%Complaints%';
-- Query 17
SELECT SUM(ol_amount), SUM(ol_quantity)
FROM item, orderline
WHERE [JOIN_PREDICATES]
  AND i_data LIKE '[CHAR]%'
  AND ol_quantity >= [QUANTITY];
-- Query 19
SELECT SUM(ol_amount)
FROM item, orderline
WHERE [JOIN_PREDICATES]
  AND i_data LIKE '[CHAR]%'
  AND i_price BETWEEN [PRICE] AND [PRICE] + 10
  AND ol_quantity BETWEEN 1 AND 10;
-- Query 20
SELECT COUNT(*)
FROM item, nation, supplier, orderline
WHERE [JOIN_PREDICATES]
  AND i_data LIKE '[CHAR]%'
  AND n_name = [NNAME];
```