HyPer: Adapting Columnar Main-Memory Data Management for Transactional AND Query Processing

Alfons Kemper Thomas Neumann Florian Funke Viktor Leis Henrik Mühe
TU München, Faculty of Informatics, firstname.lastname@in.tum.de

Abstract

Traditionally, business applications have separated their data into an OLTP data store for high throughput transaction processing and a data warehouse for complex query processing. This separation bears severe maintenance and data consistency disadvantages. Two emerging hardware trends allow the consolidation of the two disparate workloads onto the same database state on one system: the increasing main memory capacities of several terabytes per server and multi-threaded processing based on multi-core parallelism. The prevalent data representation of hybrid OLTP&OLAP main memory database systems is columnar in order to achieve best possible query execution performance for OLAP applications. In order to shield the OLTP transaction processing from long-running queries without costly locking/latching, all queries are executed on an arbitrarily recent snapshot of the data. The paper contrasts several snapshotting techniques for columnar data (twin block, versioning) with the hardware-supported shadow paging we employ in the HyPer system. While OLAP-query processing can rely mostly on columnar scans, high OLTP throughput requirements necessitate index techniques for exact match and small range queries. Despite the ever growing capacity, main memory is still a scare resource. Therefore, compressing main memory resident databases is beneficial. The paper will devise techniques that achieve good compression ratios without hurting the mission-critical OLTP throughput by adaptively separating cold (i.e. immutable) data for aggressive compression from the hot (i.e. mutable) working set data that remains uncompressed.

1 Introduction

In this paper we want to highlight the architecture of our hybrid OLTP&OLAP main memory database system HyPer which – against common belief – achieves world-record transaction processing throughput and best-of-breed OLAP query response times in **one** system **in parallel** on the **same** database state. The two workloads of online transaction processing (OLTP) and online analytical processing (OLAP) present different challenges for database architectures. Currently, users with high rates of mission-critical transactions have split their data into two separate systems, one database for OLTP and one so-called *data warehouse* for OLAP. While allowing for decent transaction rates, this separation has many disadvantages including data freshness issues due to the delay caused by only periodically initiating the *Extract Transform Load*-data staging and excessive resource consumption due to maintaining two separate information systems. We present an efficient hybrid system, called *HyPer*, that can handle both OLTP and OLAP simultaneously by using hardware-assisted replication mechanisms to

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

maintain consistent snapshots of the transactional data. HyPer is a main-memory database system that guarantees the full ACID properties for OLTP transactions and executes OLAP query sessions (multiple queries) on arbitrarily current and consistent snapshots. The utilization of the processor-inherent support for virtual memory management (address translation, caching, copy-on-write) yields both at the same time: unprecedentedly high transaction rates as high as 100,000 per second and very fast OLAP response times on a single system executing both workloads in parallel. The performance analysis is based on a combined TPC-C and TPC-H benchmark.

In the novel hybrid OLTP&OLAP database system HyPer we separate the OLTP from the OLAP processing by way of snapshotting transactional data via the virtual memory management of the operating system [14]. In this architecture the OLTP process "owns" the database and periodically (e.g., in the order of seconds or minutes) forks an OLAP process. This OLAP process constitutes a fresh transaction consistent snapshot of the database. Thereby, we exploit operating systems functionality to create virtual memory snapshots for new, cloned processes. In Unix, for example, this is done by creating a child process of the OLTP process via the `fork` system call.

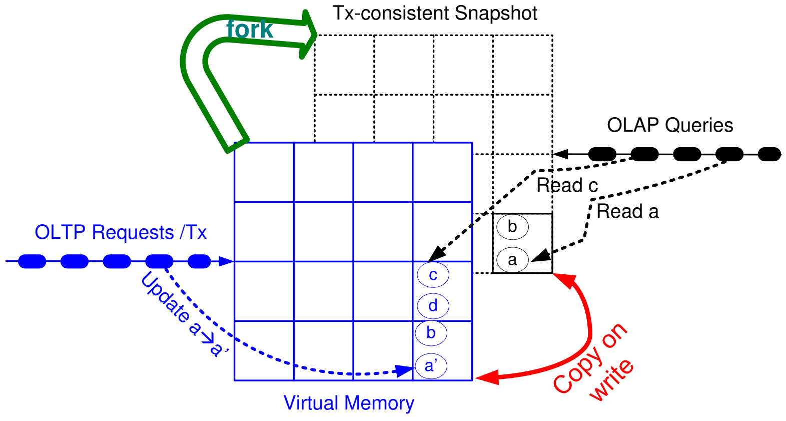

Figure 1: Virtual Memory Snapshots to Separate OLTP & OLAP.

The forked child process obtains an exact copy of the parent processes address space, as exemplified in the Figure by the overlayed page frame panel. This virtual memory snapshot that is created by the `fork`-operation will be used for executing a session of OLAP queries. These queries can be executed in parallel threads or serially, depending on the system resources or client requirements. In essence, the virtual memory snapshot mechanism constitutes a OS/hardware supported shadow paging mechanism as proposed decades ago for disk-based database systems by Lorie [17]. However, the original proposal incurred severe costs as it had to be software-controlled and it destroyed the clustering on disk. Neither of these drawbacks occurs in the virtual memory snapshotting as clustering across RAM pages is not an issue. Furthermore, the sharing of pages and the necessary copy-on-update/write is managed by the operating system with effective hardware support of the MMU (memory management unit) via the page table that translates VM addresses to physical pages and traps necessary replication (copy-on-write) actions. Therefore, the page replication is extremely efficiently done in 2 μs as we measured in a micro-benchmark.

HyPer's OLTP throughput is better than that of dedicated OLTP engines (like VoltDB) and HyPer's OLAP query response times match those of the best pure OLAP engines (e.g., MonetDB). It should be emphasized that HyPer can match (or beat) these two best-of-breed transaction (VoltDB) and query (MonetDB) processing engines **at the same time** by performing both workloads in parallel on the same database state. This performance evaluation was carried out on the basis of a new business intelligence benchmark, called the CH-benCHmark [6], that combines the transactional workload of TPC-C with the OLAP queries of TPC-H – executed against the **same** database state. Hyper's performance is due to the following design choices:

- HyPer relies on in-memory data management without the ballast of traditional database systems caused by DBMS-controlled page structures and buffer management. The SQL table definitions are transformed into simple vector-based virtual memory representations – which constitutes a column-oriented physical storage scheme.
- The OLAP processing is separated from the mission-critical OLTP transaction processing by **fork**-ing virtual memory snapshots. Thus, no concurrency control mechanisms other than the hardware-assisted

VM management are needed to separate the two workload classes.

- Transactions and queries are specified in SQL and are efficiently compiled into LLVM assembly code [20]. The transactions are specified in an SQL scripting language and registered as precanned, stored procedures. For the JIT-compiled interactive queries the very fast compilation into assembly language (LLVM) as opposed to a slow cross-compilation (into C/C++) is essential. The query evaluation follows a data-centric paradigm by applying as many operations on a data object as possible in between pipeline breakers. This evaluation scheme goes one step beyond cache-locality towards register-locality.
- As in VoltDB, the parallel transactions are separated via lock-free admission control that allows only non-conflicting transactions at the same time. Parallelism in this serial execution model is achieved by logically partitioning the database and admitting multiple partition-constrained transactions in parallel. However, for executing partition-crossing transactions the scheduler resorts to strict serial execution, rather than costly locking-based synchronization.
- HyPer relies on logical logging where, in essence, the invocation parameters of the stored procedures / transactions are logged via a high-speed network. The serial execution model in combination with partitioning and group committing achieves extreme scalability in terms of transaction throughput – without compromising the "holy grail" of ACID (that was sacrificed by the recent NoSQL/key value stores).
- While in-core OLAP query processing can be based on sequential scans, this is not possible for transaction processing as we require execution times of a few microseconds only. Therefore, we have developed sophisticated main-memory indexing structures based on hashing, balanced search trees (e.g., red black trees) and radix trees [15]. Hash indexes are indispensable for exact match (e.g., primary key) accesses that are most common in transactional processing while the tree structured indexes are essential for small-range queries, that are commonly encountered in transactional scripts as well.

2 Snapshotting

Since HyPer relies on the ability to efficiently create a transaction-consistent snapshot of the database, we compare different mechanisms for snapshot creation which do not diminish OLTP performance. These techniques are our hardware page shadowing approach as illustrated in Section 1 which we refer to as `vm-fork`, tuple shadowing which was used – for instance – by SolidDB [16] and a variant of the so called ZigZag approach as evaluated by Cao [5]. For brevity, we refer to implementation details in [19].

We extended all mechanisms to enable high performance query execution on snapshots as their most important use – instead of just recovery as previously suggested, e.g., by Molina et al. [9] or in [22]. Because of this, our approaches yield higher order snapshots for query processing as opposed to those snapshots primarily used for recovery.

Additionally, we adapted the mechanisms for use in main memory database systems: In case of Lorie's [17] page shadowing approach, we can show that the limitations when used in on-disk database systems can completely be alleviated in main memory. With Cao et al.'s twin objects approach (called ZigZag approach in [5]), we extended the implementation for use in a general purpose database system instead of a specialized application. We evaluate the different approaches taking both OLTP as well as OLAP throughput into account, since we want to evaluate their suitability for a hybrid OLTP & OLAP database system like HyPer.

The three techniques discussed here can be subdivided by the method they use to achieve a consistent snapshot while still allowing high throughput OLAP transactions on the data. The hardware page shadowing approach uses a hardware supported copy on write mechanism to create a snapshot. In contrast to that, tuple shadowing as well as the twin object approach use software mechanisms to keep a consistent snapshot of the data intact while modifications are stored separately.

Figure 2 shows the throughput for OLTP transactions without parallel OLAP execution ("raw" column) as well as OLTP and OLAP throughput while run at the same time. The OLTP transactions correspond to those of the TPC-C. The OLAP queries consist of queries semantically equivalent to queries 1 and 5 of the TPC-H. The two representative OLAP queries are repeatedly executed in an alternating pattern.

Looking at OLTP throughput, it can be observed that techniques based on Hardware Page Shadowing yield higher throughput. This has two major reasons: First, Hardware Page Shadowing allows for faster reorganization than software controlled mechanisms. Second, there is no indirection as opposed to Tuple Shadowing where a shadow tuple has to be checked, or Twin Tuples, where the tuple to be read has to be found using a bit flag.

OLAP query performance is influenced less by the choice of snapshotting mechanism. Compared to a 50% slowdown as seen in OLTP throughput, OLAP queries run about 25% slower when a software controlled snapshotting mechanism is employed.

Here, the slowdown is caused by two main factors: Reorganization time and the delay caused by quiescing OLAP queries. All backends have been architected so that OLAP query performance is as high as possible. This is achieved by maintaining tuples included in the snapshot in their original form and position and adding redirection only for new, updated or deleted tuples which can only be seen by transactions, not OLAP queries.

Backend	Raw OLTP	Comb. OLTP	Comb. OLAP
VM-Fork	85k	60k	10.3
Tuple	25k	22k	6.8
Twin	33k	29k	7.0

Figure 2: OLTP and OLAP throughput per second combining TPC-C and TPC-H.

3 Database Compaction: Hot/Cold Clustering and Compression

Compression techniques for columnar database systems is a topic extensively studied, primarily in the context of analytical systems [1, 12, 26, 21]. While some of the existing work addresses the problem of updates in compressed databases [11, 3], none of the techniques developed for OLAP systems can be easily adapted for OLTP-style workloads. Modern in-memory database systems have fast and lean transaction models that penalize additional processing severely which often prevents them from compressing data in favor of transaction throughput. A good example is the lock-free transaction processing model pioneered by H-Store/VoltDB [13, 24] that executes transactions serially on private partitions without any overhead from buffer management or locking. This model allows for record-breaking transaction throughput, but necessitates that all transactions execute quickly to prevent the serial execution pipeline from becoming congested.

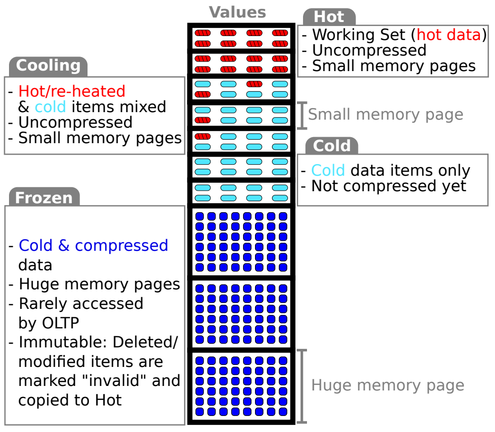

Figure 3: Hot/cold clustering for compression. 🟥 = hot (volatile) data item, ▭ = cold data item, ● = cold & compressed data item.

To avoid hurting transactional throughput, OLTP engines often refrain from compressing their data and thus waste memory space. The lack of compression becomes even more impeding, when the database system is capable of running OLAP-style queries on the transactional data, like the HyPer system [14] or SAP HANA [8]. In this scenario, compression can not only reduce memory consumption significantly due to columnar storage, but also promises faster query execution [25, 1, 4, 10].

Approaches that maintain two separate data stores, an uncompressed store for freshly inserted data and a compressed store for older data, require costly merge phases that require exclusive locking of tables when moving data to the compressed store. They also tend to complicate and slow down query and transaction processing.

Our approach to compression in hybrid OLTP & OLAP column stores is based on the observation that while OLTP workloads frequently modify the dataset, they often follow the working set assumption [7]: Only a small subset of the data is accessed and an even smaller subset of this working set is being modified (cf. Figure 3). In business applications, this working set is mostly comprised of tuples that were added to the database in the recent past, as it can be observed in the TPC-C workload [23].

Our system uses a lightweight monitoring component to observe accesses to the dataset and identify opportunities to reorganize data such that it is clustered into hot and cold parts. After clustering, the database system compresses cold chunks to reduce memory consumption and streamline queries.

While compression reduces the memory consumption for the data set, we perform further optimizations to narrow the memory consumption in the presence of snapshots. Lorie's shadow paging [17] and its modern, hardware-assisted reincarnation, HyPer's `fork`-based snapshot mechanism, both maintain separate page tables for the current database state as seen by OLTP workers and snapshots. Creating a new snapshot requires the duplication of the current page table. Maintaining the snapshot requires the replication of those pages that are modified by transactions in order to preserve the consistent snapshot state from the time of its creation. The performance of both operations, page table duplication and page replication, highly depends on the size of the pages: While huge pages result in smaller page tables, and thus faster snapshot creation, smaller pages incur less replication overhead. Hot/cold clustering is an elegant solution to this problem, as the cold bulk of the data can be stored on huge memory pages (and thus shrink the page table size) while the hot, frequently modified working set remains on regular memory pages that can be replicated inexpensively. The frozen, huge data pages are never modified; if a frozen data object is changed, after all, it is invalidated (via an invalidation vector) in the frozen partition and re-inserted into the hot working set. The invalidation vector can be maintained as a position tree [11] to retain the fast scanning speed without having to check the validity of every data object upon access.

In addition to the smaller page table, the use of huge pages has further advantages:

1. Scanning huge pages is faster than scanning regular pages. Manegold et al. [18] analyze the impact of translation lookaside buffer (TLB) misses on query performance and conclude with regard to the page size that the use of huge pages reduces the probability of TLB misses.
2. The TLB of the processor's memory management unit has separate sections for huge and normal pages on most platforms. Since the bulk of the database is cold and resides on huge pages, table-scans in OLAP queries mostly access the huge pages TLB. Transactions operate almost entirely on hot data that is stored on normal pages. Thus, separate hardware resources are used and no "TLB-thrashing" occurs.

4 Conclusion and Ongoing Work

In-memory technology has facilitated a reunion of OLTP and OLAP systems by separating the two disparate workloads via snapshotting. So, after all, a "one size fits all" system appears possible. We have presented a comparison of snapshotting techniques demonstrating the benefits of hardware-assisted shadow paging over software approaches. We have made the case for hot/cold clustering to store frequently accessed tuples together on regular memory pages while cold, immutable tuples can reside on huge pages. This leads to the advantageous combination of page table size (and thus snapshot creation costs) and replication overhead. In addition, it allows to compress the majority of the database without causing OLTP throughput declines.

In the future, we will investigate the following issues:

Long-running transactions: Currently, HyPer focuses on the execution of short, pre-canned transactions which are known in advance. Future research will extend this set of workloads to include transactions of arbitrary length. For the execution of these long-running transactions, an optimistic execution strategy which does not interfere with the execution of good-natured short transactions has to be designed.

Scale-out: So far, HyPer was designed as a single server system as we believe it to be more economical to scale up before you scale out. Nevertheless, for very large data volumes scaling out the database across multiple servers is necessary. To enable this, the query engine is being redesigned to distribute query plans and a query coordinator is being designed to manage intermediate query results. A global transaction manager is needed to control the execution of inter-site transactions and for creating globally consistent snapshots. The synchronization of global transactions relies on the execution model for long-running transactions.

Memory efficient index structures: Unlike systems which concentrate purely on OLAP, HyPer does not store data in a sorted fashion to allow for high OLTP throughput. Index structures are used both for efficient access to single tuples using their key as well as ordered access to a relation. Since index structures make up a significant share of the total memory used by the DBMS, we are developing an efficient, compact index structure which works well with our snapshotting approach. As we regularly keep snapshots of index structures as well as the data, modifications to an index need to be as local as possible to cause only few copy-on-write operations.

Multi-core parallel query processing: To make operational BI a reality, it is necessary to exploit the vast computing power of modern multi-core servers effectively. For this purpose, the OLAP query engine of HyPer is currently extended by intra-operator parallelism. We are developing a new massively parallel sort merge-join algorithm [2] that scales linearly in the number of cores involved in the join computation.

References

[1] D. Abadi, S. Madden, and M. Ferreira. Integrating compression and execution in column-oriented database systems. In *SIGMOD*, pages 671–682, 2006.

[2] M. Albutiu, A. Kemper, and T. Neumann. Massively Parallel Sort Merge Join in Main Memory Multi-core Database Systems. Technical report, 2012, TU München, 2012.

[3] C. Binnig, S. Hildenbrand, and F. Färber. Dictionary-based order-preserving string compression for main memory column stores. In *SIGMOD*, 2009.

[4] P. Boncz, M. Zukowski, and N. Nes. MonetDB/X100: Hyper-Pipelining Query Execution. In *CIDR*, 2005.

[5] T. Cao, M. Salles, B. Sowell, Y. Yue, J. Gehrke, A. Demers, and W. White. Fast Checkpoint Recovery Algorithms for Frequently Consistent Applications. In *SIGMOD*, 2011.

[6] R. Cole, F. Funke, L. Giakoumakis, W. Guy, A. Kemper, S. Krompass, H. A. Kuno, R. O. Nambiar, T. Neumann, M. Poess, K.-U. Sattler, M. Seibold, E. Simon, and F. Waas. The mixed workload ch-benchmark. In *DBTest 2011*, 2011.

[7] P. J. Denning. The Working Set Model for Program Behaviour. *CACM*, 11(5):323–333, 1968.

[8] F. Färber, S. Cha, J. Primsch, C. Bornhövd, S. Sigg, and W. Lehner. SAP HANA database: data management for modern business applications. *SIGMOD Record*, 40(4):45–51, 2011.

[9] H. Garcia-Molina and K. Salem. Main Memory Database Systems: An Overview. *IEEE TKDE*, 4(6):509–516, 1992.

[10] G. Graefe and L. Shapiro. Data Compression and Database Performance. In *Symp. On Applied Computing*, 1991.

[11] S. Héman, M. Zukowski, N. Nes, L. Sidirourgos, and P. Boncz. Positional update handling in column stores. In *SIGMOD*, 2010.

[12] A. Holloway, V. Raman, G. Swart, and D. DeWitt. How to barter bits for chronons: compression and bandwidth trade offs for database scans. In *SIGMOD*, 2007.

[13] R. Kallman, H. Kimura, J. Natkins, A. Pavlo, A. Rasin, S. Zdonik, E. Jones, S. Madden, M. Stonebraker, Y. Zhang, J. Hugg, and D. Abadi. H-store: a high-performance, distributed main memory transaction processing system. *PVLDB*, 1(2):1496–1499, 2008.

[14] A. Kemper and T. Neumann. HyPer: A Hybrid OLTP&OLAP Main Memory Database System Based on Virtual Memory Snapshots. In *ICDE*, 2011.

[15] V. Leis. Main-memory index structures for modern hardware. Master's thesis, TU Muenchen, 2012.

[16] A.-P. Liedes and A. Wolski. SIREN: A Memory-Conserving, Snapshot-Consistent Checkpoint Algorithm for in-Memory Databases. In *ICDE*, 2006.

[17] R. A. Lorie. Physical Integrity in a Large Segmented Database. *TODS*, 2(1), 1977.

[18] S. Manegold, P. Boncz, and M. L. Kersten. What Happens During a Join? Dissecting CPU and Memory Optimization Effects. In *VLDB*, 2000.

[19] H. Mühe, A. Kemper, and T. Neumann. How to efficiently snapshot transactional data: hardware or software controlled? In *DaMoN*, 2011.

[20] T. Neumann. Efficiently compiling efficient query plans for modern hardware. *PVLDB*, 4(9):539–550, 2011.

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

[22] K. Salem and H. Garcia-Molina. Checkpointing memory-resident databases. In *ICDE*, 1989.

[23] Transaction Processing Performance Council. TPC-C specification, 2010.

[24] VoltDB. Technical Overview. `http://www.voltdb.com`, March 2010.

[25] T. Westmann, D. Kossmann, S. Helmer, and G. Moerkotte. The Implementation and Performance of Compressed Databases. *SIGMOD Record*, 29(3):55–67, 2000.

[26] M. Zukowski, S. Héman, N. Nes, and P. Boncz. Super-Scalar RAM-CPU Cache Compression. In *ICDE*, 2006.