Workload-Driven Horizontal Partitioning and Pruning for Large HTAP Systems

Martin Boissier
Hasso Plattner Institute
University of Potsdam, Germany
martin.boissier@hpi.de

Daniel Kurzynski [1]
SAP SE
Potsdam, Germany
daniel.kurzynski@sap.com

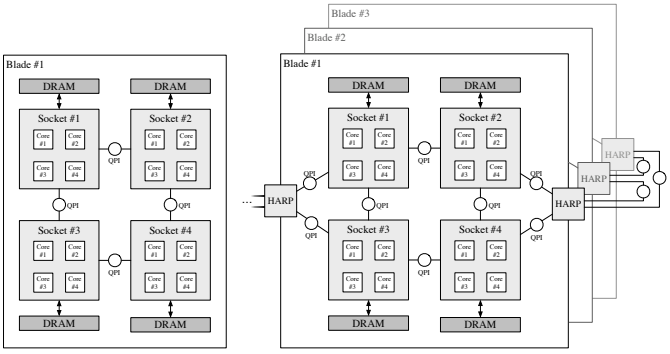
***Abstract*—Modern server systems with large NUMA architectures necessitate (i) data being distributed over the available computing nodes and (ii) NUMA-aware query processing to enable effective parallel processing in database systems. As these architectures incur significant latency and throughout penalties for accessing non-local data, queries should be executed as close as possible to the data. To further increase both performance and efficiency, data that is not relevant for the query result should be skipped as early as possible. One way to achieve this goal is horizontal partitioning to improve static partition pruning.**

As part of our ongoing work on workload-driven partitioning, we have implemented a recent approach called *aggressive data skipping* and extended it to handle both analytical as well as transactional access patterns. In this paper, we evaluate this approach with the workload and data of a production enterprise system of a Global 2000 company. The results show that over 80% of all tuples can be skipped in average while the resulting partitioning schemata are surprisingly stable over time.

I. Partitioning & NUMA

Modern enterprise applications require database systems that handle both transactional day-to-day workloads as well as increasing analytical workloads (also called mixed workloads, OLxP, or HTAP, cf. [1]). One of the main promises of these new systems is to enable real-time analytics on the most recent transactional data (cf. [2], [3]) without expensive data warehouse installations and long-running ETL (extract, transform, and load) processes. Two key factors enabling these systems recently have been (i) vastly increased DRAM capacities and (ii) main memory-resident databases [4].

To meet the increasing demand for large DRAM capacities, hardware vendors developed new server architectures. Recent scale-up multi-socket systems include multiple CPUs with dozens of cores on a single blade. One characteristic of these systems is that the access performance to main memory depends on the location of the particular DRAM module to access. As each DRAM module is directly attached to a CPU, accessing non-local DRAM modules incurs additional costs. Such an architecture is called a *Non-Uniform Memory Access* architecture (NUMA, see Figure 1(a)) where non-local DRAM is accessed via the QPI connected to another CPU (i.e., a hop). Systems such as the SGI UV300H, depicted in Figure 1(b), even expand the system to span multiple blades. Here, the QPIs of different blades are connected via a hardware component called HARP (for more details see [5]). While the UV300H is optimized for low latency connections, even larger systems such as the SGI UV2000 add substantial latencies that are up to an order of magnitude higher when accessing non-local memory, cf. [6]. The higher the number of hops to access data – and with that higher latencies and lower bandwidths – the larger the potential for mechanisms avoiding unnecessary accesses to distant data and reducing inter-node communication.



(a) Depiction of a four node two-hop single-blade NUMA system.

(b) Exemplary depiction of a three-blade SGI UV300H showing the blades connected via the HARP component.

Fig. 1. DRAM access paths in NUMA architectures (FMC-like notation).

Database systems do not need to be modified in order to run on NUMA architectures as these systems provide a fully cache-coherent system that behaves like a single socket system. However, to fully exploit the performance of NUMA architectures, multiple components (i.a., storage manager, query planner, operators) of a database system need to be adapted to counter NUMA effects in order to achieve optimal performance. Kissinger et al. [6] propose treating NUMA systems like distributed systems since communication needs to be minimized and processing should be data-local in order to fully exploit the system. Simple approaches that distribute small horizontal partitions round-robin enable highly parallel processing, but leave much room for improvement, cf. [7].

Psaroudakis et al. [8] compared data placement strategies for SAP HANA, an in-memory column store. The authors found that *physical partitioning* (PP), where each partition is self-

[1]Daniel Kurzynski worked on partitioning of large HTAP systems as part of his master's thesis at the Hasso Plattner Institute. This work largely summarizes the results of his thesis.

contained with local dictionaries and indices, to perform best for a wide range of scenarios. Further, they mention the potential pruning possibilities with physical partitioning. Nonetheless, the authors recommend another approach as they found PP to (i) be slow during repartitioning and (ii) increase memory consumption due to local dictionaries. We disagree with both points as we found (i) the proposed partitioning scheme for real-world systems presented in this paper to be surprisingly stable (see Section IV) and (ii) the memory overhead of local dictionaries to be neglectable (between -2% and 5% increase for the evaluated enterprise system). Consequently, we see physical partitioning to distribute table data to the available NUMA nodes as the most promising data distribution strategy which further allows efficient NUMA-optimized task scheduling [8].

Horizontal Partitioning for HTAP Systems

We study HTAP-optimized and DRAM-resident databases on large scale-up servers. As previously mentioned, besides query execution strategies and task scheduling for NUMA systems (cf. [9]), data placement is one of the most important aspects. We investigate various horizontal partitioning approaches and how they perform compared to interleaved data allocations, cf. [8]. The goal is to physically partition data horizontally in order to distribute the data over the NUMA nodes and improve data locality. Tuples can only be stored in one partition at a time. Partitioning or declustering [10] enables static partition pruning (or partition elimination), where partitions are skipped when the database can logically rule out that a partition contains tuples that qualify for the given query predicates [11]. As such, partition pruning does not only improve runtime performance but also improves additional aspects such as the energy efficiency by reducing the data that needs to be processed to answer a query while ensuring correct results.

Horizontal partitioning is well studied, especially in the context of transactional OLTP systems. Such systems often hash-partition on a primary key attribute to distribute the load and enable partition elimination for point queries [12]. However, for HTAP workloads with analytical loads, hash partitioning is often not the best choice: (i) it does not support range queries and (ii) HTAP systems are usually not dominated by point accesses. Hence, our goal is to find *a single partitioning scheme* that suits both OLTP-typical point-access queries as well as sequential OLAP queries. Finding a suitable range partitioning scheme, however, is challenging as both the workload as well as data characteristics have to be considered to avoid unbalanced data or access distributions. Moreover, range-partitioned schemata often deteriorate over time, e.g., on date(-correlated) attributes. Nonetheless, due to the requirements of analytical queries in HTAP systems, we consider round-robin and hash-based partitioning as inept.

Our objective is to determine a partitioning scheme that (i) partitions the data set into a given number of partitions (e.g., the number of NUMA nodes in the system) and (ii) is optimized for efficient partition pruning given the database's workload. Horizontal partitioning has been traditionally applied to warehouse systems because the primary access path (i.e., linear scans) largely profits from partition pruning. This is not directly the case for OLTP systems, where the primary access path is usually an index scan. Avoiding unnecessary accesses to partitions with partition-local indices might not improve the average query runtime, but it can significantly lower the overall load of the system or increase throughput. Throughout this paper, our focus lies mainly on the *aggressive data skipping* approach by Sun et al. (Section II) and its *pruning rates* and less on runtime performance.

To evaluate our implementation of aggressive data skipping, we use the data and workload traces of a production enterprise system. This system is an installation of a recent SAP ERP version of a Global 2000 company (cf. [13] for more details about the traced system). A particularly interesting aspect is that the traced system is one of the first systems to exhibit both transactional workloads as well as analytical workloads [1]. Partitioning such a system is particularly challenging as OLTP and OLAP queries have vastly different characteristics.

II. Aggressive Data Skipping

In this paper, we present our experiments with an implementation of the *aggressive data skipping approach* (or *SOP* for *skipping-oriented partitioning*) by Sun et al. [14]. We consider this approach to be one of the most promising approaches for horizontal – pruning-optimized – partitioning which finds efficient partitioning schemata by analyzing the workload while being reasonably fast to create.

The *aggressive data skipping* approach has been developed to add a fine-granular partitioning level (creating so-called blocks) on top of horizontal partitioning. Using the created metadata, certain blocks can be dynamically pruned.

From our point of view, the addition of such meta data structures is not necessary. Hence, we decided to use the generated features to physically partition the data. The main reason was that the eventual merging phase allows adjusting the number of partitions relatively freely, enabling us to physically partition and distribute the data on our large NUMA system. As the approach potentially yields partitioning schemata with an arbitrary number of criteria, typical composite range-based schemata as used by most databases are not sufficient.

The Process. As a first step, the database workload is parsed in order to obtain representative filter predicates, so-called features, which are conjunctive filter predicates that describe tuples. Similar predicates can be merged when one predicate subsumes another. Using these features, SOP scans the relation and creates feature vectors that characterize each tuple. Eventually, the feature vectors are clustered and merged using a cost model-based merge function to form the partitions.

III. Improvements for HTAP Systems

During the evaluation of the traced SAP ERP system, we made three major observations. First, for complex real-world systems with data being accessed through different applications with different intentions, the limitation to partition only by one dimension turned out to be too often insufficient, because filter attributes often differ for OLTP and OLAP queries. Second,

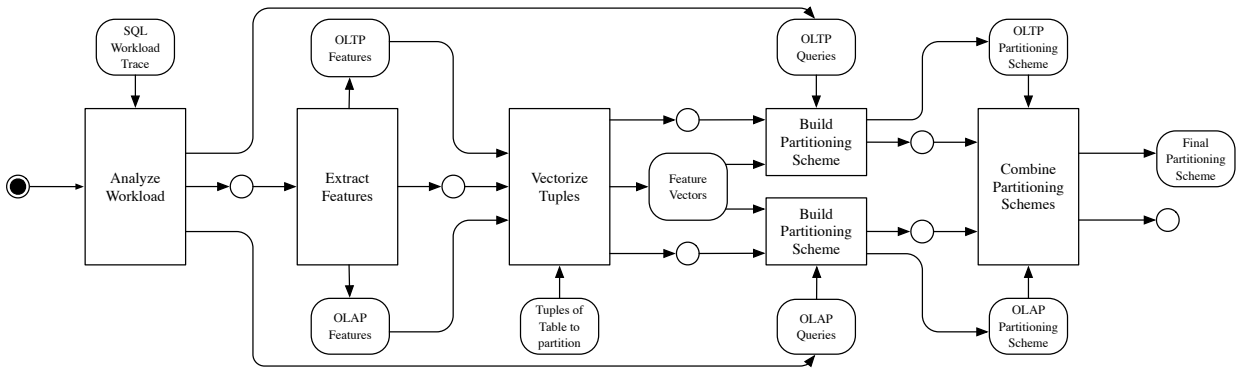


Fig. 2. Petri net of adopted *aggressive data skipping* process (FMC notation). Instead of handling all query types equally, transactional and analytical workloads are separately processed and optimized. Eventually, both schemata are merged.

while SOP expects queries in conjunctive normal form (CNF), the queries of enterprise systems are often formulated in a disjunctive normal form (DNF). Third, the inherit misbalance in the number of queries of different classes (in this context, OLTP and OLAP queries) leads to suboptimal results. In this section, we will discuss how we adapted Sun et al.'s approach towards the characteristics of HTAP enterprise systems.

A. Query Reformulation

Most enterprise systems do not expose the actual database engine beneath by providing an abstraction layer (e.g., SAP's Open SQL, cf. [15]). Comparable to Hibernate[2], such ORM layers automatically generate SQL queries. Those queries often select a disjunctive list of items, where each item is described via conjunctive predicates. In the traced enterprise systems, we found queries with up to 350 disjunctions of conjunctions of predicates. In contrast, SOP has been implemented assuming queries in the conjunctive normal form as they better matched their examined OLAP queries.

To handle the DNF-dominated queries of the enterprise systems, we decided to simply split each disjunctive query into multiple queries. This is a trade-off because, from a query optimizer's perspective, it does not reflect the actually executed workload as a reformulation to a conjunctive form is usually preferable. However, as a reformulation to DNF is known to be computationally expensive, we decided to split queries to keep the analysis runtime manageable. This way, we do not need to change the feature extraction process and traded this gain for a more sophisticated subsuming process. We tracked split queries and rejoin them afterward in order to decide whether the original query is prunable for a given partitioning scheme.

B. Combining HTAP Query Classes

Analyzing the SAP ERP system, we found that the original data skipping approach falls short in handling different query classes. In HTAP systems, the majority of queries are transactional queries which process relatively few tuples and mostly filter on attributes of the primary key. In contrast, analytical queries process much larger numbers of tuples and often filter on attributes that are not part of the primary key. As Sun et al.'s cost model is optimized for the majority of queries (which might already be sufficiently covered by index accesses anyways), analytical queries are mostly neglected.

To separate the classification of queries (in our case OLTP and OLAP, but it could be any classification) from the actual optimization of the partitioning scheme, we decided against weighting queries or artificially changing the execution times. We added a preprocessing step that first separates the workload into different classes of query types. We differentiated analytical queries by their average running time, but it is also possible to classify them based on the filters' selectivity or the join/aggregation costs. After we classified each query, we run the whole process independently for both classes. Eventually, the locally optimized partitioning schemata are merged into one aggregated scheme. We found the results of this process to be promising (see Sec. IV). As the feature processing for all classes can be done with one pass-through, runtime is barely affected. The whole adapted process is depicted in Figure 2.

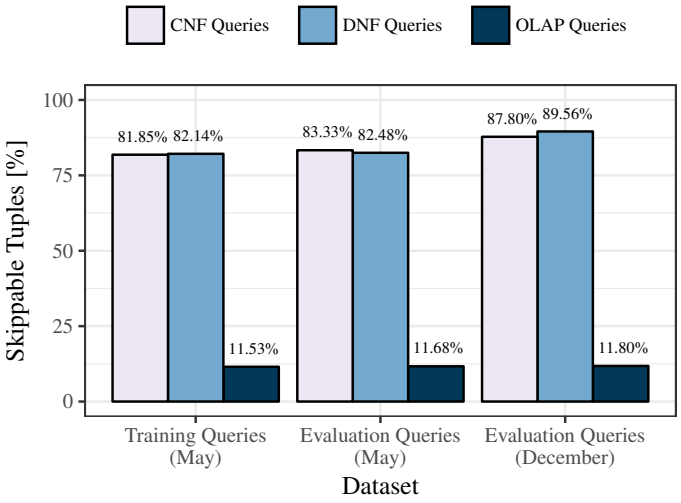
IV. Evaluation

In this section, we present our evaluation using data and workload of the traced enterprise system. We traced the system three times, each time for several days, resulting in several million traced queries (for more details see [13]). We evaluated the workload for the `ACDOCA` table, which is the central table for the financial and controlling modules of an SAP ERP system and exhibits the highest analytical load.

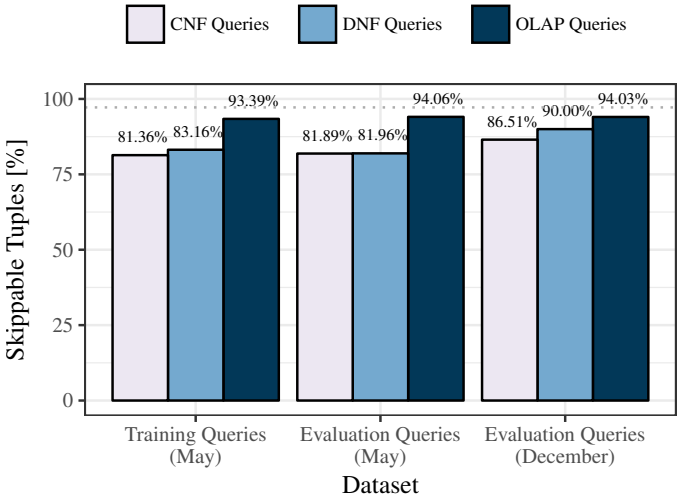
A. Synthetic Database Benchmarks

We briefly evaluated TPC-C and TPC-CH (and with that also the CH-benCHmark [16]). We found both benchmarks to be too simple to demonstrate the effects of a more adept partitioning approach. TPC-C, e.g., can be trivially partitioned by warehouse identifier. The problem is that most benchmarks do neither (i) include a broad range of queries with varying characteristics nor (ii) exhibit any skew in the data or selection criteria. However, we plan to evaluate TPC-DS as part of future work. Due to the length limitations of this paper, we will concentrate on the production SAP ERP system for the remainder of this section.

[2]Hibernate - Java persistence framework: http://hibernate.org



(a) Skipping performance without HTAP adaption. Due to the few analytical queries, partitions are optimized for OLTP queries (i.e., partitioning along primary key attributes).



(b) Skipping performance with HTAP adaption. The dotted line denotes the maximum skipping performance achieved when optimizing solely for OLAP, showing that the skipping performance for OLAP queries is only slightly impacted when optimizing for the complete workload including OLTP and OLAP queries.

Fig. 3. Skipping results with and without HTAP adaption. Both normal forms show results for the full workload traces including OLTP & OLAP queries.

B. Pruning Performance

Modern HTAP enterprise systems are clearly dominated by reading queries, cf. [13]. As a consequence, our main objective is to optimize for read performance by skipping as many tuples as possible before the first plan operator is executed (via static partition pruning, cf. [11]).

Amongst the most interesting questions for us was how the adaption of the data skipping approach to handle mixed workloads performs. Figure 3(a) shows the results of data skipping when all queries are handled homogeneously. The three clusters show the tracing periods of the production ERP system. We see several important insights: (i) almost completely "out-of-the-box" the data skipping approach reached average pruning rates (i.e., accessed tuples vs. skipped tuples) of over 80%, (ii) the partitioning scheme is surprisingly stable and even slightly improves after several months[3], and (iii) OLAP queries show a comparatively poor pruning rate.

Figure 3(b) shows the results of the adaption for HTAP workloads. OLAP skipping performance improves significantly while deteriorating marginally compared to OLAP-only optimization. The dotted line denotes the maximum pruning ration for OLAP queries when solely optimizing for OLAP (measured for the last trace). Hence, we accept a minor reduction of the average skipping performance (cf. Figures 3(a) and 3(b)) in exchange for significantly improved OLAP performance.

Figure 4 shows the pruning rate and the size (denoted by bar width) for each created partition. We can make two major observations: (i) only a few small partitions have a pruning rate below 85% and (ii) partition sizes are skewed. The reason for the few small partitions that can often not be pruned is the creation of generalized partitions that take remaining tuples. These partitions cannot be pruned for the majority of queries as their criteria are very broad. However, our adapted SOP automatically limits the size of these partitions.

The size of the created partitions varies to a large extent. The partition scheme shown has been created as we assumed a 32-socket server machine (e.g., SGI UV300H) and thus tried to create ~32 partitions. However, we see the partition size skew as neglectable because the resulting partitioning scheme has excellent pruning results and enterprise systems often contain thousands of tables. Instead of enforcing an even data distribution, we would rather balance the data distribution when placing partitions from other tables on the same node. As the majority of tables do not have an inter-table dependency, we consider a balanced distribution as doable.

C. Runtime Performance

To evaluate SOP, we prototypically implemented it in a columnar in-memory database. As the tested database system has been built with different assumptions about partitioning and is not fully capable of physically partitioning by an arbitrary number of attributes, we cannot make credible statements about end-to-end performance. Hence, we will only briefly discuss the runtime performance. For the executed tests, the runtime impact of the partitioning is two-sided. On the one hand, the results for scan-dominated queries show a speed up almost linear with the share of skipped tuples. Further, the distribution of physical partitions to the CPU sockets helped to improve the scan performance as the dictionary was node-local (cf. [8]). On the other hand, joins have been mainly impacted negatively as the tested database system creates mapping structures between each partition pair to join. These mappings can quickly become prohibitively expensive for a larger number of partitions to

[3]We attribute the pruning improvement over time to the large impact of the fiscal year in financial systems. In May, several processes still access items from the previous fiscal year, this rate is slowing decreasing over time, improving the partition scheme over time. Unfortunately, we have no query trace from the beginning of the following fiscal year.

join. In general, the performance depends on the number of partitions a query has to access, whether it combines data from distant nodes (e.g., joining) where the QPI can quickly become a bottleneck, and whether distant data can be directly compared or not (cf. global vs. local dictionaries).

V. Related Work

There is a vast array of published work about database partitioning and declustering. In this section, we briefly discuss the most relevant works in the field of physical partitioning. Due to the length limitations of this paper, we skip the discussion of dynamic pruning approaches using additional statistical data structures (cf. [11], [17], [18]).

Agrawal et al. present AutoAdmin in [12], the physical design advisor for SQL Server. The authors present a holistic approach that considered both vertical and horizontal partitioning, as well as indexing and materialized views. As the state space is accordingly large, the presented approach strongly prunes possible partitioning criteria, does not consider data characteristics, and omits partitioning criteria with more than one dimension. While this approach is interesting in terms of making complex design decisions in a reasonable time frame, its focus is clearly on disk-based and row-oriented databases for OLTP workloads, which is orthogonal to our focus.

The constraints for physical partitioning shifted with fully DRAM-resident databases as they allow much faster data transfers, enabling iterative partitioning with frequent adaptions.

Curino et al. [19] presented a workload-aware partitioning approach that falls into the category of OLTP-optimized approaches (cf. [20], [21]) which aim to minimize partition-spanning transactions in order to improve throughput.

Sun et al. presented the successor of the data-skipping approach in [22]. In their derivative called GSOP (generalized skipping-oriented partitioning), tables are no longer partitioned solely in a horizontal fashion. To better handle diverse potential partitioning criteria, the authors implemented a column-major format that can partition each column on its own. For HTAP settings, we consider the negative impact of not having direct access to all attributes of a tuple as prohibitively expensive.

Jindal et al. proposed an online partitioning approach [23] that is both applicable for vertical as well as horizontal partitioning. Similar to Navathe's well-known vertical partitioning approach [24], the authors use an affinity matrix-based approach. While the presented approach is highly interesting – especially its focus on online repartitioning – it is not directly applicable for our domain: (i) it only considers one-dimensional partitioning schemata which we have found to be not sufficient for real-world systems and (ii) skewed data and access patterns are not considered. Ghandeharizadeh et al. [10] and Boral et al. [25] presented early approaches for multi-dimensional declustering (i.e., range-based horizontal partitioning) for large parallel analytical systems spanning multiple server nodes.

Höppner et al. presented a partitioning approach for SAP HANA to tier infrequently accessed columns to secondary storage [26]. While this approach was optimized for HTAP workloads, it did not prune data on a horizontal level.

Psaroudakis et al. compared data placement and task scheduling strategies for SAP HANA and found that both are highly relevant for an efficient usage of NUMA architectures [8]. They compared various strategies for distributing partitioned columns (cf. Section I).

A rather orthogonal path is taken by database cracking [27], where data is not directly partitioned but replicated to incrementally create indices. We consider this approach as too expensive since we focus on in-memory databases and thus want to avoid any replication of data and keep a small memory footprint.

VI. Future Work & Challenges

As part of our research on large NUMA systems, we are currently implementing the foundation to support a wide range of partitioning alternatives in Hyrise$_2$, a column-oriented in-memory research database[4]. We focus two aspects: flexibility and repartitioning. As Hyrise$_2$ is a research database, the implemented partitioning mechanism should allow flexible partitioning schemata (feature-based approaches as SOP being one of them). Another important aspect – that has not been covered sufficiently in research from our point of view (apart from OLTP-optimized partitioning approaches focusing node-spanning transactions) – are efficient online repartitioning techniques for HTAP systems. We think that this aspect is crucial for any self-adapting and highly available database system. We are working on a cost model that determines small iterative changes to the partitioning scheme with minimal impact on the systems runtime performance while still ensuring sufficiently fast repartitioning. Depending on the available resources and workload, the database can reinforce repartitioning performance or lower the overhead of repartitioning.

With respect to our work on partitioning algorithms, we want to improve the data locality of regular join partners. Looking at large transactional tables (e.g., accounting documents and deliveries) revealed that there is a mismatch between search and join predicates. The typical header-item relationship in normalized systems is interesting as join predicates are often a subset of the search predicates. Usually, partitioning the smaller header tables is deemed as neglectable as it introduces a certain partitioning overhead without improving performance. However, it might allow joining tables entirely node-locally by applying the same partition scheme and node assignment to both tables. This could significantly improve access locality (and with that runtime performance) for joins and lower the pressure on QPI links.

VII. Conclusion

For large server systems with NUMA architectures, the combination of workload-driven partitioning, partition pruning, and locality-aware query execution can vastly improve efficiency and runtime performance. The topic of declustering is well tackled for transactional workloads and key-value stores, but less so for modern HTAP systems. We have presented the results of our implementation of the aggressive data skipping

[4]Hyrise$_2$ on GitHub: https://github.com/hyrise/hyrise

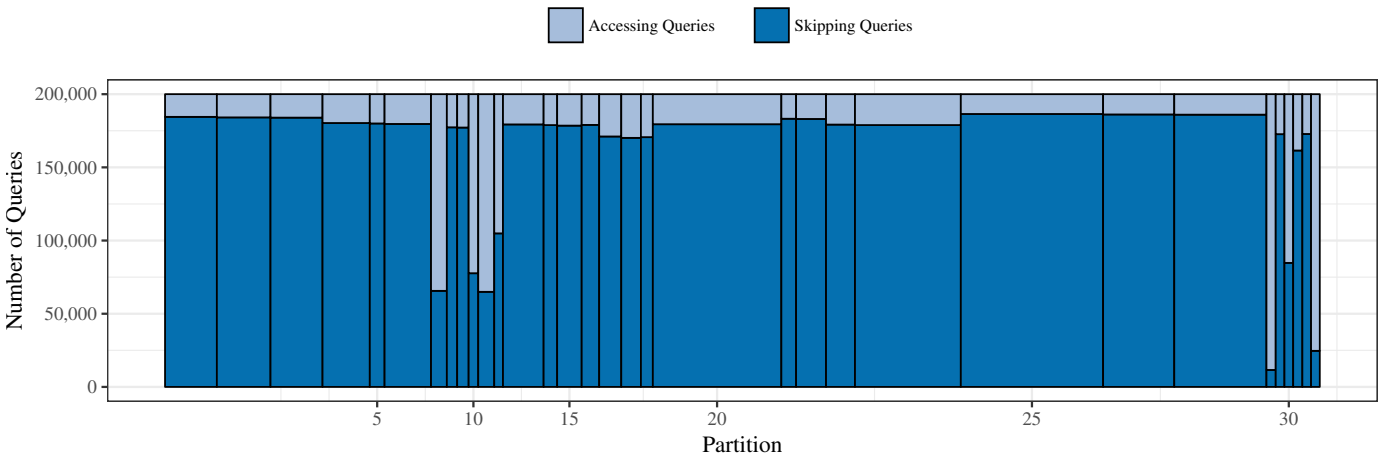


Fig. 4. Skipping rates and partition sizes (depicted by bar width) for the final HTAP-aware partition scheme.

approach by Sun et al. and evaluated the approach for a production enterprise system. We extended the aggressive data skipping approach by adapting it for HTAP workloads. The results are promising. Especially for analytical workloads, where the primary access path is linear scanning, efficient partition pruning has the potential to improve performance substantially. On average, more than 90% of tuples of a table can be pruned for OLAP queries of the enterprise system.

With the ability to repartition and restructure data on-the-fly in DRAM-resident databases, we see HTAP-optimized and workload-driven partitioning becoming of increasing importance. Especially with the trend towards autonomous and self-pimping databases. With that, the question how to maintain, iteratively improve with a low overhead, and develop (range-based) partitioning schemata over time needs to be further studied. Moreover, the interplay of adaptable partition schemata and learned access patterns (e.g., from time series analyses) is highly valuable and part of our future research.

REFERENCES

[1] H. Plattner, "The impact of columnar in-memory databases on enterprise systems," *PVLDB*, vol. 7, no. 13, pp. 1722–1729, 2014.

[2] S. Klauck *et al.*, "Interactive, flexible, and generic what-if analyses using in-memory column stores," in *Proc. DASFAA, Part II*, 2015, pp. 488–497.

[3] C. Tinnefeld *et al.*, "Available-to-promise on an in-memory column store," in *Proc. BTW*, 2011, pp. 667–686.

[4] F. Färber *et al.*, "The SAP HANA database – an architecture overview," *IEEE Data Engineering Bulletin*, vol. 35, no. 1, pp. 28–33, 2012.

[5] M. Dreseler *et al.*, "Hardware-accelerated memory operations on large-scale NUMA systems," in *Proc. ADMS at VLDB*, 2017.

[6] T. Kissinger *et al.*, "ERIS: A NUMA-aware in-memory storage engine for analytical workload," in *Proc. ADMS at VLDB*, 2014, pp. 74–85.

[7] M. Albutiu, A. Kemper, and T. Neumann, "Massively parallel sort-merge joins in main memory multi-core database systems," *PVLDB*, vol. 5, no. 10, pp. 1064–1075, 2012.

[8] I. Psaroudakis *et al.*, "Scaling up concurrent main-memory column-store scans: Towards adaptive NUMA-aware data and task placement," *PVLDB*, vol. 8, no. 12, pp. 1442–1453, 2015.

[9] V. Leis, P. A. Boncz, A. Kemper, and T. Neumann, "Morsel-driven parallelism: a NUMA-aware query evaluation framework for the many-core age," in *Proc ACM SIGMOD*, 2014, pp. 743–754.

[10] S. Ghandeharizadeh, D. J. DeWitt, and W. Qureshi, "A performance analysis of alternative multi-attribute declustering strategies," in *Proc. ACM SIGMOD*, 1992, pp. 29–38.

[11] A. Nica *et al.*, "Statisticum: Data statistics management in SAP HANA," *PVLDB*, vol. 10, no. 12, pp. 1658–1669, 2017.

[12] S. Agrawal, V. R. Narasayya, and B. Yang, "Integrating vertical and horizontal partitioning into automated physical database design," in *Proc. ACM SIGMOD*, 2004, pp. 359–370.

[13] M. Boissier *et al.*, "Analyzing data relevance and access patterns of live production database systems," in *Proc. CIKM*, 2016, pp. 2473–2475.

[14] L. Sun *et al.*, "Fine-grained partitioning for aggressive data skipping," in *Proc. ACM SIGMOD*, 2014, pp. 1115–1126.

[15] A. Kemper, D. Kossmann, and B. Zeller, "Performance tuning for SAP R/3," *IEEE Data Eng. Bull.*, vol. 22, no. 2, pp. 32–39, 1999.

[16] R. L. Cole *et al.*, "The mixed workload ch-benchmark," in *Proceedings DBTest*, 2011, p. 8.

[17] H. Lang *et al.*, "Data blocks: Hybrid OLTP and OLAP on compressed storage using both vectorization and compilation," in *Proc. SIGMOD*, 2016, pp. 311–326.

[18] G. Moerkotte, "Small materialized aggregates: A light weight index structure for data warehousing," in *Proc. VLDB*, 1998, pp. 476–487.

[19] C. Curino, Y. Zhang, E. P. C. Jones, and S. Madden, "Schism: a workload-driven approach to database replication and partitioning," *PVLDB*, vol. 3, no. 1, pp. 48–57, 2010, VLDB Endowment.

[20] M. Serafini *et al.*, "Clay: Fine-grained adaptive partitioning for general database schemas," *PVLDB*, vol. 10, no. 4, pp. 445–456, 2016.

[21] R. Taft *et al.*, "E-store: Fine-grained elastic partitioning for distributed transaction processing," *PVLDB*, vol. 8, no. 3, pp. 245–256, 2014.

[22] L. Sun *et al.*, "Skipping-oriented partitioning for columnar layouts," *PVLDB*, vol. 10, no. 4, pp. 421–432, 2016.

[23] A. Jindal *et al.*, "Relax and let the database do the partitioning online," in *BIRTE Workshop 2011, Revised Selected Papers*, 2012, pp. 65–80.

[24] S. B. Navathe, S. Ceri, G. Wiederhold, and J. Dou, "Vertical partitioning algorithms for database design," *ACM Transactions on Database Systems*, vol. 9, no. 4, pp. 680–710, 1984.

[25] H. Boral *et al.*, "Prototyping bubba, A highly parallel database system," *IEEE Trans. Knowl. Data Eng.*, vol. 2, no. 1, pp. 4–24, 1990.

[26] B. Höppner *et al.*, "An approach for hybrid-memory scaling columnar in-memory databases," in *Proc. ADMS at VLDB*, 2014, pp. 64–73.

[27] H. Pirk *et al.*, "Database cracking: fancy scan, not poor man's sort!" in *Proc. DaMoN 2014*, 2014, pp. 4:1–4:8.