Optimizing Main Memory Utilization of Columnar In-Memory Databases Using Data Eviction

Martin Boissier
Supervised by Hasso Plattner
Hasso Plattner Institute, University of Potsdam
August-Bebel-Str. 88, 14482 Potsdam, Germany
martin.boissier@hpi.uni-potsdam.de

ABSTRACT

Despite falling prices for main memory and increasing sizes, main memory is still a scarce resource in database systems. Optimizing main memory utilization is a major objective for main memory databases as more free memory can be used to improve performance or to store larger systems in the database. Several publications proposed separating frequently and less frequently accessed data (i.e., hot and cold data), handling both with different priorities or evicting cold data to secondary storage. However, most of these approaches are optimized for OLTP workloads. In contrast, this PhD project researches how to improve DRAM utilization for mixed workloads including both OLTP and OLAP queries by evicting cold data. As a first step, real-world database workloads are analyzed in order to determine characteristics of hot and cold data as well as aging effects. Also two possible approaches exploiting the results of the workload analyses are outlined.

1. INTRODUCTION

Main memory resilient databases have been in the focus of database research in recent years [9, 12, 15]. Among others, this is caused by falling prices of main memory and increasing main memory sizes per server. Nonetheless, main memory is still a scarce resource and expensive compared to disk. Consequently, improving main memory utilization is a major goal for any in-memory database as more free memory can improve performance (e.g., storing intermediate results), allow larger systems to be stored in the database, or simply to improve cost efficiency by evicting unused data to less expensive storage layers.

Looking at real database workloads shows, that accesses are often highly skewed and frequently access a small fraction of the data. This observation is in line with the working set model that says that for each process there is a subset of pages that are accessed distinctively more frequently [5]. Besides the working set model, another concept plays a significant role for this PhD project: data aging. Data aging describes the issue that recent data is accessed more frequently than older data. The older the data, the less relevant it is for the database system.

Due to resource limitations in main memory databases, a separation into frequently and less frequently used data has become of increasing interest in recent years [4, 6, 11]. These approaches try to prioritize frequently accessed data (i.e., the working set or 'hot data') in order to exploit different data relevancies and thus improve main memory utilization by compressing less relevant data or evicting it to secondary storage.

The main objective of this project is to research data aging ideas in the context of enterprise applications. In particular we look at mixed enterprise workloads. Mixed workloads (also called *OLXP*) consist of transactional as well as analytical queries thus combining OLTP and OLAP [10].

Most publications in the field of data separation based on access frequencies track tuple accesses to prioritize frequently accessed tuples. While tuple-based approaches work well for most OLTP-workloads, it does not solve the problem of relevance-based data separation for mixed workloads. In contrast to those tuple-based approaches, we are focusing our research on a separation that is aware of OLAP-style queries as aggregations or complex joins.

Furthermore, we think it is important to determine real-world requirements on database systems before talking about possible concepts or solutions. Therefore, different database systems have been traced and analyzed over the past months. The findings of these analyses are presented in this paper and we will explain how they influence certain concepts we are looking at.

The objective is to make the following contributions throughout the PhD project:

- A thorough analysis of real-world database workloads captured from different fields of applications as discussed in Section 2.
- An evaluation of existing hot and cold data approaches (see Section 4) using real-world database workloads.
- A simplified approach that enhances memory utilization by exploiting workload characteristics and time correlation of tuple accesses is discussed in Section 3.1. 'Simplified' because this approach does not depend on additional statistics, data reordering, or indices.
- An approach based on horizontal partitioning that exploits skewness and time correlation of tuples accesses is presented in Section 3.2.

Proceedings of the VLDB 2014 PhD Workshop.

2. WORKLOAD ANALYSIS

While the concepts of working sets and data aging are rather simple and well known, we think it is important to actually quantify and thoroughly analyze both using real-world workloads. This includes determining the working set that is eventually processed as well as the time correlation of it to quantify aging effects. To analyze both it is neither sufficient to solely look at data nor it is sufficient to solely look at the workload (i.e., the query log). To quantify how many tuples are accessed, both have to be taken into account.

Examining enterprise systems our hypothesis was that real-world workloads are highly skewed – especially more skewed than often expected – both horizontally as well as vertically. Hence, only few attributes are used for query evaluation and only very few tuples are accessed frequently.

Analyzed Workloads

We decided against analyzing benchmarks like TPC-C or YCSB as to our best knowledge no benchmark incorporates realistic age-based access patterns. Furthermore, Krueger et al. have shown that even the enterprise-oriented TPC-C benchmark vastly diverges from characteristics of a real enterprise system [10].

To gain insights into realistic systems we are currently analyzing two systems:

- openHPI [13]
  - German MOOC (Massive Open Online Course) platform
  - web platform build using the Ruby on Rails-based Canvas[1] framework
  - complete workload over eight weeks with ∼200M queries including a database snapshot
- Productive Enterprise Resource Planning System
  - traced financial and controlling module of a productive SAP ERP system
  - sampled workload over three days ∼50M queries)

2.1 Workload Analysis Procedure

The workload analysis is split into two separate tasks, starting with a thorough analysis of the traced queries followed by a query log replay to gain tuple access statistics.

2.1.1 Query Log Analysis

The first part is the query log analysis in which each query is parsed to extract the following information (amongst others):

- Query template: for each query, we create the corresponding query template (similar to the form of prepared statements) by removing the filter values for each selection
- Logical information: for each query its characteristics as "is the query a join, aggregation, or key-select?" et cetera are stored
- Projections: for each query, the list of projected attributes is stored which is of particular interest for vertical partitioning and tuple reconstruction in column stores

[1]https://github.com/instructure/canvas-lms

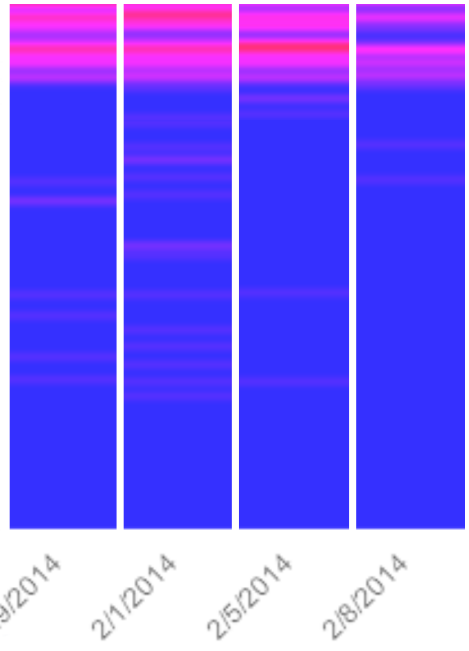

Figure 1: Heat Map Visualizing Table Accesses for Table 'quiz_submissions'.

- Selections: for each query all its selections are stored

By extracting and normalizing query characteristics and storing them queryable in a database it is possible to ask questions as: 'How many queries join table X and Y with range predicates on columns Y and Z?'

2.1.2 Access Analysis

The access analysis is executed on a snapshot of the database. This analysis is basically a query log replay in which each query is modified in order to return the tuple positions instead of the actual tuples. This way we can quantify which tuples are required to answer a query. Obviously this is a kind of black box analysis that does not answer questions concerning the query execution (e.g., which tuples have been accessed to be filtered before returning the result set) but rather an analysis of what is of interest for the requesting application.

Aggregating queries are modified in a way that all tuples that are part of the aggregation are considered as accessed (e.g., by removing the SUM operator from `SELECT SUM(*) FROM A WHERE customerId=42`).

2.2 Preliminary Results

As of now, we have analyzed the openHPI system and a snapshot of the productive SAP ERP system from May 2014. The findings of our analyses are so far:

*SELECT * Projections*

Most queries use `SELECT *` projections or project the majority of attributes (in fact, as openHPI is build using Ruby on Rails' active records all queries are `SELECT *` projections). In the current version of the ERP system (which is running on a columnar database) 20% of all queries still use `SELECT *` projections.

Access Skewness

That access to data is often skewed is well known. For example, the YCSB benchmark includes a skewness property to skew tuple accesses. We calculated the Zipfian distributions for tables that account for over 80% of the openHPI database size. The average scale factor for those tables was above 3.9. This is particularly interesting as several publications expect significantly lower skewness. E.g., DeBrabant et al. evaluated Zipfian scale factors between 0.5 and 1.5 [4].

Selection Columns

ERP systems often consist of very wide tables (e.g., the main table to store accounting document segments has over 300 attributes). But query evaluation is mostly done on very few attributes.

Number of Accessed Tuples

The number of distinct tuples that were returned in the workload is very low for most transactional tables. We took the three largest openHPI tables that have been accessed and that account for over 90% of the data volume (in fact, the single largest table – a pure logging table – has not been queried once over the eight weeks). On average, less than 3% of the tuples in these tables have been accessed.

For the ERP system, the five largest (by size in main memory) traced tables have an all together over one billion rows. Of these less than 13 million have been accessed during our trace. In contrast, for the two main financial tables *BSEG* and *BKPF* the share of accessed tuples is 16% resp. 37%.

Age-Access Relation

Analyzing the relation between tuple accesses and their age shows that there is a distinctive correlation between both for large transactional tables. For the two largest tables in the openHPI system, a horizontal cut of the most recent 20% is still sufficient to answer 85% of all queries on both tables. This effect is shown in Figure 1. In this heat map, accesses to the 'quiz_submissions' table are shown (the oldest tuple on the bottom, newest on top, divided into time slots of ∼80h). Accessed data regions are drawn in red, regions have not been accessed are drawn in blue.

For smaller tables, e.g., master data as the user table, this correlation is obviously different.

3. ENVISIONED APPROACHES

Starting from the preliminary results, we want to investigate two approaches to exploit aging effects in order to optimize main memory utilization.

3.1 Simplified Data Aging

The simplified approach tries to exploit the access skewness and the low number of columns required to evaluate queries.

We call this approach simplified as it does not require any tuple movements, tuples access statistics, or additional indices. The basic idea is an improved version of a vertical partitioning, which is the most obvious approach to evict cold data in a columnar database. Hereby, columns are considered cold when they are not used in the query execution and are thus evicted to secondary storage. The problem with a pure vertical approach is tuple reconstruction. As seen in our analyses applications often used *SELECT ** projections. Even though queries can be evaluated on the hot columns, the database still has to access cold columns for materialization.

Similar to the vertical approach the simplified approach stores all columns that are required for query evaluation in main memory. This includes all columns that are part of aggregations, selections, group by statements, order by statements et cetera. All other columns that are not required for query evaluation are considered cold. An exemplary visualization of a table using the simplified approach is depicted in Figure 2. But instead of simply moving them to secondary storage two approaches are possible:

Page Buffering Here mmap is used to memory-map cold columns and a page buffer to pin the most recent pages of each column in main memory. As already explained, tuples accesses on large tables have shown to correlate with age. Hence, keeping the most recent 20% of each column in memory would enable the majority of tuple reconstructions to be executed without accessing secondary storage.

Tuple Cache As the number of distinct tuples accessed has shown to be very small, a tuple cache can be even more efficient in reducing the memory footprint. Such a tuple cache – e.g. using LRU cache to persist the latest materialized tuples – has the additional advantage that it would also cover frequently accessed tuples that are not part of the most recent pages.

The simplified approach has several advantages. One is that only little modification is required for column stores. Besides, query performance for the majority of queries is only slightly impacted. The query can be evaluated completely in memory until the point of tuple reconstruction for returning the result set. For the tuple reconstruction, the majority of queries can still be answered from main memory with little penalty caused by the buffer management. In our opinion, the very small penalties on query performance is one the major advantages of the simplified approach. Using a tuple cache, the simplified approach might even outperform current column store implementations in which tuple reconstructions are still a bottleneck.

Preliminary results for the two largest tables in the openHPI system are shown in Table 3.1. The tables have been stored in a dictionary-encoded and bit-compressed column store. The reason that the simplified approach can potentially reduce the memory footprint by factors of up to 700 is that the two largest tables in the openHPI system are both logging tables that store the history of texts. These versioning tables are always accessed via key attributes (integer values). Furthermore, all queries in our workload accessed the most recent version. Please note that the presented sizes do not include the space required to allow fast tuple reconstruction. But since the number of distinct tuples accessed is that low, a tuple cache should not increase the required space significantly.

Obviously such factors are not always possible. Depending on the data that a table stores and the access patterns (e.g., key-selects vs. aggregation queries), the theoretical results of simplified approach differ vastly.

Storing Cold Columns

Another interesting question is how to store cold columns on secondary storage. Because cold columns should only be

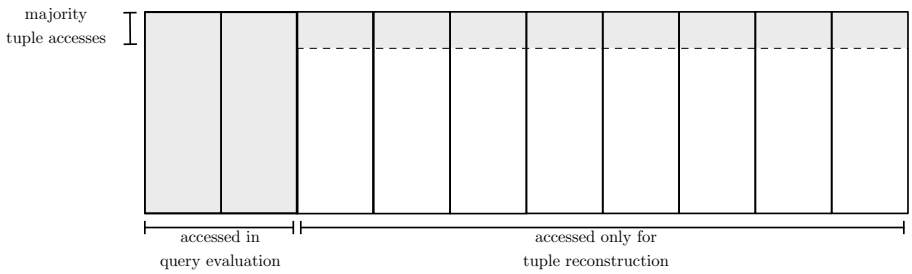

Figure 2: Simplified Data Aging: Exemplary Visualization of a Table Partitioned in Hot and Cold Columns Using Page Buffering.

Table	`versions`	`quiz_submissions`
Tuple count	1,423,502	815,518
Distinct tuples accessed	20,034	37,344
Column count	6	23
Hot columns	3	4
Size of table	7.7 GB	4.6 GB
Size of hot columns	11 MB	12 MB

Table 1: Statistics for the two Largest Tables in openHPI System.

accessed for tuple reconstruction, storing them dictionary-encoded (what is the default compression for columns in our scenario) is expected to yield poor performance, as the materialization of n attributes requires at least $2*n$ accesses to secondary storage. One possible solution is to store the columns uncompressed.

In case of dynamically changing workloads it cannot be avoided that scans on cold columns incur. In this case, neither the uncompressed nor the dictionary-encoded storage format is perfect. But as disk storage is comparably inexpensive today, we want to evaluate storing cold columns mirrored in a row- as well as column-oriented manner.

Besides the storage format, the storage layer on which the cold columns are stored is part of our planned research. PCIe-connected solid state disks are a middle layer between disk and main memory. Columns that are not used for query evaluation but for the majority of tuple reconstructions can be stored on that middle layer, while columns almost never used for tuple reconstruction can remain on disk.

3.2 Partitioning-Based Data Aging

We expect several cases where the simplified approach will not evict a substantial part of the table. Examples are tables in which columns that are required for query evaluation account for the majority of the table size.

Therefore, we want to investigate in another approach we call partitioning-based data aging. The idea of this approach is to partition tables horizontally into one master partition and n read-only partitions.

The master partition stores the latest fraction of a table, e.g., the most recently added tuples required to answer 90% of all single tuple selects (i.e., key-selects). Our analyses showed that this is already the case for transactional tables when storing the most recent 25% of the table. The n read-only partitions are distributed to o server nodes with $o \ll n$. Hereby, partitions are created depending on the given workload. We envision a dynamic partitioning system comparable to Curino et al.'s work [3]. The main contrast here is that the focus does not lie on distributing the workload amongst partitions as far as possible. Instead we are trying to skew the distribution as much as possible towards the active partition(s). Apart from keeping workload statistics, no additional overhead as tuple access counting or tuple reordering is being done.

The goal is to answer the vast majority of OLTP-queries solely from the master partition, e.g., using a tuple cache for key-selects to tuples that are not part of the most recent tuples stored. Threfore, it is important to avoid accesses to read-only partitions as far as possible for transactional workloads whenever possible. For OLAP queries, we think it is eventually impossible to avoid partition-spanning joins. But dynamic partitioning based on the workload and stored partition profiles on the master node can minimize partition-spanning operations. Such partition profiles are stored in memory on the master partition for each read-only partition and include data statistics for the columns that are most often queried. How these partition profiles look is part of our research we plan to investigate in the future.

Data stored in read-only partitions is supposed to be more cost efficient than being stored in main memory, e.g., by persisting the data on disk and only keeping indices, materialized aggregates, and statistics in memory. Similar to the simplified approach we want to evaluate different mirroring approaches to store data in different formats on disk.

Transaction Handling

As already said, we try to avoid any OLTP-workloads on the read-only partitions. Consequently, the read-only partitions do not incur any changes to their data. If a tuple of a read-only partition is about to be modified the row is marked as invalid and a new version is stored on the master partition (similar to insert-only approaches). For such scenarios and transactions that span multiple partitions, we want to test transaction abortion and restarting comparable to the idea proposed by DeBrabant [4]. Here, whenever a transaction requires a modifying access to tuples on read-only partitions, the transaction is aborted. Then, the required tuples are moved to the master partition and the transaction is restarted afterwards.

With partition profiles and transaction handling that is optimized towards our partitioning scheme we hope to work towards a systems that is capable of mixed workloads in a

multi-node setup. Such a system could provide stable OLTP performance and sufficient OLAP performance in scale out scenarios.

4. RELATED WORK

The research field of data aging has been studied in many publications over the past decades. Garcia-Molina et al. already discussed the workload-based distinction of hot and cold data in the context of in-memory databases in 1992 [7].

More recently, Funke et al. presented a compaction-based approach to handle hot and cold data [6]. Using capabilities of modern server systems to track data accesses, the database compacts partitions based on their access frequencies. While the data is still kept in memory, cold and frozen data is compressed to better utilize main memory. Particularly interesting for us is the idea of using the MMU to track page-table accesses as this concept works for row- or column-oriented storage formats.

Levandoski et al. discussed the separation into hot and cold data based on asynchronous sampled query log analyses [11]. This work is of particular interest for us since our work focuses on a thorough analysis of database query logs as well. The project is called Siberia, an extension to the Hekathon OTLP-optimized main memory engine to manage hot and cold data. Another very interesting work – also part of Siberia – are "Adaptive Range Filters" by Alexiou et al. [2]. With the help of such filters accesses to cold data can be pruned for range queries. We want to evaluate such filters on their own using realistic workloads and combine them with partition profiles of read-only partitions to better prune analytical queries of mixed workloads.

In many ways similar to Siberia, DeBrabant et al. implemented hot and cold separation in the row-based in-memory database H-Store [4, 9]. The authors call this approach *Anti-Caching* to underline that hot data is no longer cached in main-memory but cold data is evicted to secondary storage. To trace accesses to tuples, tuples are stored in an LRU chain per table.

Besides research on data aging and its variants, there has been plenty of work about storage formats for disk-resident data. As mentioned in Section 3, we would like to investigate redundant storage of data on disk as disk is comparably cheap and solely row- or column-oriented storage formats incur many shortcomings depending on the access patterns.

Ailamaki et al. use a columnar-like format that groups attributes per page, called PAX (Partition Attributes Across) [1]. PAX still allows fast tuples reconstructions while improving cache utilization.

Ramamurthy et al. went one step further and redundantly stored data (i.e., mirroring) in both horizontally and vertically partitioned storage formats and adapted the query execution engine to automatically chose the appropriate storage format [14].

5. RESEARCH AGENDA

In the course of this PhD project we'd like to focus on both of the discussed directions in the upcoming year.

Workload Analysis

As of now, the workload analysis has already provided valuable insights into aspects of data aging and workloads in general.

We think it will be very interesting to evaluate published data aging approaches (see Section 4) using real-world workloads. We expect to see that most approaches work very well for OLTP workloads in which data is mostly queried using key-selects. But as we are focusing on mixed workloads with transactional as well as analytical queries, we expect tuple-based approaches to perform insufficiently for queries that require scans, aggregations, et cetera. To evaluate possibly upcoming mixed workload system we are currently looking into analytical systems. By combining transactional workloads extracted from actual enterprise systems with analytical workloads extracted from analytical systems a realistic mixed workload can be estimated.

We plan to publish the workload analysis tool set on a demo track of an international database conference. Furthermore, we are eager to get access to additional systems on which we can run the query log analysis as well as the access analysis.

Envisioned Data Aging Approaches

In respect to the presented data aging approaches, we'd like to further investigate both approaches and implement first prototypes. Therefore, we plan to use the in-memory database HYRISE [8].

To analyze different storage layers and their performance metrics (e.g., non-volatile memory prototypes or PCI-express connected solid state disks) we are in contact with various hardware vendors.

6. CONCLUSION

As outlined in this paper, the objective of the PhD project is to contribute to the field of main memory utilization for columnar in-memory databases. We consider this as an important contribution since main memory is a scarce resource and current approaches do not cover the requirements of modern enterprise systems combining transactional and analytical workloads. Furthermore, we think that a thorough workload analysis of real-world systems can be a significant contribution to the general field of database research.

7. REFERENCES

[1] A. Ailamaki, D. J. DeWitt, M. D. Hill, and M. Skounakis. Weaving relations for cache performance. In P. M. G. Apers, P. Atzeni, S. Ceri, S. Paraboschi, K. Ramamohanarao, and R. T. Snodgrass, editors, *VLDB*, pages 169–180. Morgan Kaufmann, 2001.

[2] K. Alexiou, D. Kossmann, and P.-A. Larson. Adaptive range filters for cold data: Avoiding trips to siberia. *PVLDB*, 6(14):1714–1725, 2013.

[3] C. Curino, Y. Zhang, E. P. C. Jones, and S. Madden. Schism: a workload-driven approach to database replication and partitioning. *PVLDB*, 3(1):48–57, 2010.

[4] J. DeBrabant, A. Pavlo, S. Tu, M. Stonebraker, and S. B. Zdonik. Anti-caching: A new approach to database management system architecture. *PVLDB*, 6(14):1942–1953, 2013.

[5] P. J. Denning. The working set model for program behaviour. *Commun. ACM*, 11(5):323–333, 1968.

[6] F. Funke, A. Kemper, and T. N. 0001. Compacting transactional data in hybrid oltp & olap databases. *PVLDB*, 5(11):1424–1435, 2012.

[7] H. Garcia-Molina and K. Salem. Main memory database systems: An overview. *IEEE Trans. Knowl. Data Eng.*, 4(6):509–516, 1992.

[8] M. Grund, P. Cudr-Mauroux, J. Krueger, S. Madden, and H. Plattner. An overview of hyrise - a main memory hybrid storage engine. *IEEE Data Eng. Bull.*, 35(1):52–57, 2012.

[9] R. Kallman, H. Kimura, J. Natkins, A. Pavlo, A. Rasin, S. B. 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, 2008.

[10] J. Krueger, C. Kim, M. Grund, N. Satish, D. Schwalb, J. Chhugani, H. Plattner, P. Dubey, and A. Zeier. Fast updates on read-optimized databases using multi-core cpus. *PVLDB*, 5(1):61–72, 2011.

[11] J. J. Levandoski, P.-A. Larson, and R. Stoica. Identifying hot and cold data in main-memory databases. In *ICDE*, pages 26–37. IEEE Computer Society, 2013.

[12] J. Lindstroem, V. Raatikka, J. Ruuth, P. Soini, and K. Vakkila. Ibm soliddb: In-memory database optimized for extreme speed and availability. *IEEE Data Eng. Bull.*, 36(2):14–20, 2013.

[13] C. Meinel and C. Willems. *openHPI: the MOOC offer at Hasso Plattner Institute*. Technische Berichte des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam.

[14] R. Ramamurthy, D. J. DeWitt, and Q. Su. A case for fractured mirrors. In *VLDB*, pages 430–441. Morgan Kaufmann, 2002.

[15] V. Sikka, F. Frber, W. Lehner, S. K. Cha, T. Peh, and C. Bornhvd. Efficient transaction processing in sap hana database: the end of a column store myth. In *SIGMOD Conference*, pages 731–742. ACM, 2012.