The Impact of Columnar In-Memory Databases on Enterprise Systems

Implications of Eliminating Transaction-Maintained Aggregates

Hasso Plattner
Hasso Plattner Institute for IT Systems Engineering
University of Potsdam
Prof.-Dr.-Helmert-Str. 2-3
14482 Potsdam, Germany
hasso.plattner@hpi.de

ABSTRACT

Five years ago I proposed a common database approach for transaction processing and analytical systems using a columnar in-memory database, disputing the common belief that column stores are not suitable for transactional workloads. Today, the concept has been widely adopted in academia and industry and it is proven that it is feasible to run analytical queries on large data sets directly on a redundancy-free schema, eliminating the need to maintain pre-built aggregate tables during data entry transactions. The resulting reduction in transaction complexity leads to a dramatic simplification of data models and applications, redefining the way we build enterprise systems. First analyses of productive applications adopting this concept confirm that system architectures enabled by in-memory column stores are conceptually superior for business transaction processing compared to row-based approaches. Additionally, our analyses show a shift of enterprise workloads to even more read-oriented processing due to the elimination of updates of transaction-maintained aggregates.

1. INTRODUCTION

Over the last decades, enterprise systems have been built with the help of transaction-maintained aggregates as on-the-fly aggregations were simply not feasible. However, the assumption that we can anticipate the right pre-aggregations for the majority of applications without creating a transactional bottleneck was completely wrong. The superior approach is to calculate the requested information on the fly based on the transactional data. I predict that all enterprise applications will be built in an aggregation and redundancy free manner in the future.

Consider a classic textbook example from the database literature for illustration purposes, the debit/credit example as illustrated in Figure 1. Traditionally, funds are transferred between accounts by adding debit and credit entries to the accounts and updating the account balances within transactions. Maintaining dedicated account balances and paying the price of keeping the aggregated sums up to date on every transfer of funds was the only way we could achieve reasonable performance as calculating the balance on demand would require the expensive summation of all transfers. However, this concept of maintaining pre-built aggregates has three major drawbacks: (i) a lack of flexibility as they do not respond to organizational changes, (ii) added complexity in enterprise applications and (iii) increased cost for data insertion as materialized aggregates have to be updated transactionally safe.

In contrast, the most simplistic approach of only recording the raw transaction data allows for simple inserts of new account movements without the need of complex aggregate maintenance with in-place updates and concurrency problems. Balances are always calculated by aggregating all account movements on the fly. This concept overcomes the drawbacks mentioned above but is not feasible using row-based databases on large enterprise data.

Let us consider the thought experiment of a database with almost zero response time. In such a system, we could execute the queries of all business applications, including reporting, analytics, planning, etc. directly on the transactional line items. With this motivation in mind, we started out to design a common database architecture for transactional and analytical business applications [16]. The two fundamental design decisions are to store data in a columnar layout [3] and keep it permanently resident in main memory. I predicted that this database design will replace traditional row-based databases and today it has been widely adopted in academia and industry [5, 9, 11, 14, 16, 19].

The dramatic response time reduction of complex queries processed by in-memory column stores allowed us to drop all pre-aggregations and to overcome the three drawbacks mentioned above. Although a row-based data layout is better suited for fast inserts, the elimination of maintaining aggregates on data entry results in performance advantages of columnar-based system architectures for transactional business processing [19]. An important building block to compensate for the response time of large aggregations has been the introduction of a cache for intermediate result sets that can be dynamically combined with recently entered data [15]. This caching mechanism is maintained by the database and

Proceedings of the VLDB Endowment, Vol. 7, No. 13

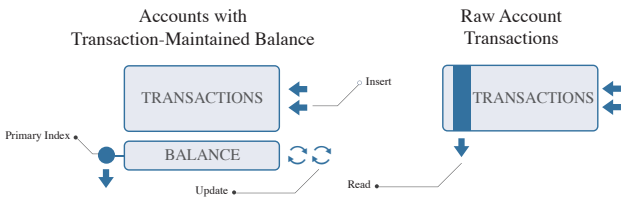

Figure 1: Debit/Credit Example with transaction-maintained balances compared to on-the-fly calculations of balances based on the raw account transactions.

completely transparent to applications. The main results of an application design without pre-aggregated information are a simplified set of programs, a reduction of the data footprint and a simplified data model. For the remainder of the paper, we refer to the classic system architecture that keeps transaction-maintained aggregates on application level and stores data in row-based disk databases as row-based architectures and use column-based architectures in the sense of a simplified architecture on application level removing transaction-maintained aggregates by leveraging in-memory columnar databases.

In the following, this paper summarizes the column-based architecture in Section 2 and provides arguments why it is the superior solution for transactional business processing in Section 3. Additionally, a workload analysis of a new simplified financial application without transactional-maintained aggregates is presented in Section 4. Implications of the adoption of a column-based architecture as well as optimizations for scalability are described in Section 5. The paper closes with thoughts on future research and concluding remarks in Sections 6 and 7.

2. ARCHITECTURE OVERVIEW

Although the concept behind column stores is not new [3, 21], their field of application in the last 20 years was limited to read-mostly scenarios and analytical processing [21, 12, 6, 20]. In 2009, I proposed to use a columnar database as the backbone for enterprise systems handling analytical as well as transactional processing in one system [16]. In the following section, we describe the basic architecture of this approach and the trends that enable this system design.

The traditional market division into online transaction processing (OLTP) and online analytical processing (OLAP) has been justified by different workloads of both systems. While OLTP workloads are characterized by a mix of reads and writes of a few rows at a time, OLAP applications are characterized by complex read queries with joins and large sequential scans spanning few columns but many rows of the database. Those two workloads are typically addressed by separate systems: transaction processing systems and business intelligence or data warehousing systems.

I strongly believe in the re-unification of enterprise architectures, uniting transactional and analytical systems to significantly reduce application complexity and data redundancy, to simplify IT landscapes and to enable real-time reporting on the transactional data [16]. Additionally, enterprise applications such as Dunning or Available-To-Promise exist, which cannot be exclusively assigned to one or the other workload category, but issue both analytical and transactional queries and benefit from unifying both systems [17, 22]. The workloads issued by these applications are referred to as mixed workloads, or OLXP.

The concept is enabled by the hardware developments over the recent years making main memory available in large capacities at low prices. Together with the unprecedented growth of parallelism through blade computing and multi-core CPUs, we can sequentially scan data in main memory with unbelievable performance [16, 24]. In combination with columnar table layouts and light-weight compression techniques, the technological hardware developments allow to improve column scan speeds further by reducing the data size, splitting work across multiple cores and leveraging the optimization of modern processors to process data in sequential patterns. Instead of optimizing for single point queries and data entry by moving reporting into separate systems, the resulting scan speed allows to build different types of systems optimized for the set processing nature of business processes.

Figure 2 outlines the proposed system architecture. Data modifications follow the insert-only approach and updates are modeled as inserts and invalidate the updated row without physically removing it. Deletes also only invalidate the deleted rows. We keep the insertion order of tuples and only the lastly inserted version is valid. The insert-only approach in combination with multi-versioning [2] allows to keep the history of tables and provides the ability of time-travel queries [8] or to keep the full history due to legal requirements. Furthermore, tables in the hot store are always stored physically as collections of attributes and metadata and each attribute consists of two partitions: main and delta partition. The main partition is dictionary-compressed using an ordered dictionary, replacing values in the tuples with encoded values from the dictionary. In order to minimize the overhead of maintaining the sort order, incoming updates are accumulated in the write-optimized delta partition [10, 21]. In contrast to the main partition, data in the delta partition is stored using an unsorted dictionary. In addition, a tree-based data structure with all the unique uncompressed values of the delta partition is maintained per column [7]. The attribute vectors of both partitions are further compressed using bit-packing mechanisms [24]. Optionally, columns can be extended with an inverted index to allow fast single tuple retrieval [4].

To ensure a constantly small size of the delta partition, we execute a periodic merge process. A merge process combines all data from the main partition with the delta partition to create a new main partition that then serves as the primary data store [10]. We use a multi-version concurrency control mechanism to provide snapshot isolation of concurrently running transactions [2]. This optimistic approach fits well with the targeted mixed workload enterprise environment, as the number of expected conflicts is low and long running analytical queries can be processed on a consistent snapshot of the database [16, 17].

Although in-memory databases keep their primary copy of the data in main memory, they still require logging mechanisms to achieve durability. In contrast to ARIES style logging [13], we leverage the applied dictionary compression [25] and only write redo information to the log on the fastest durable medium available. This reduces the overall log size by writing dictionary-compressed values and allows

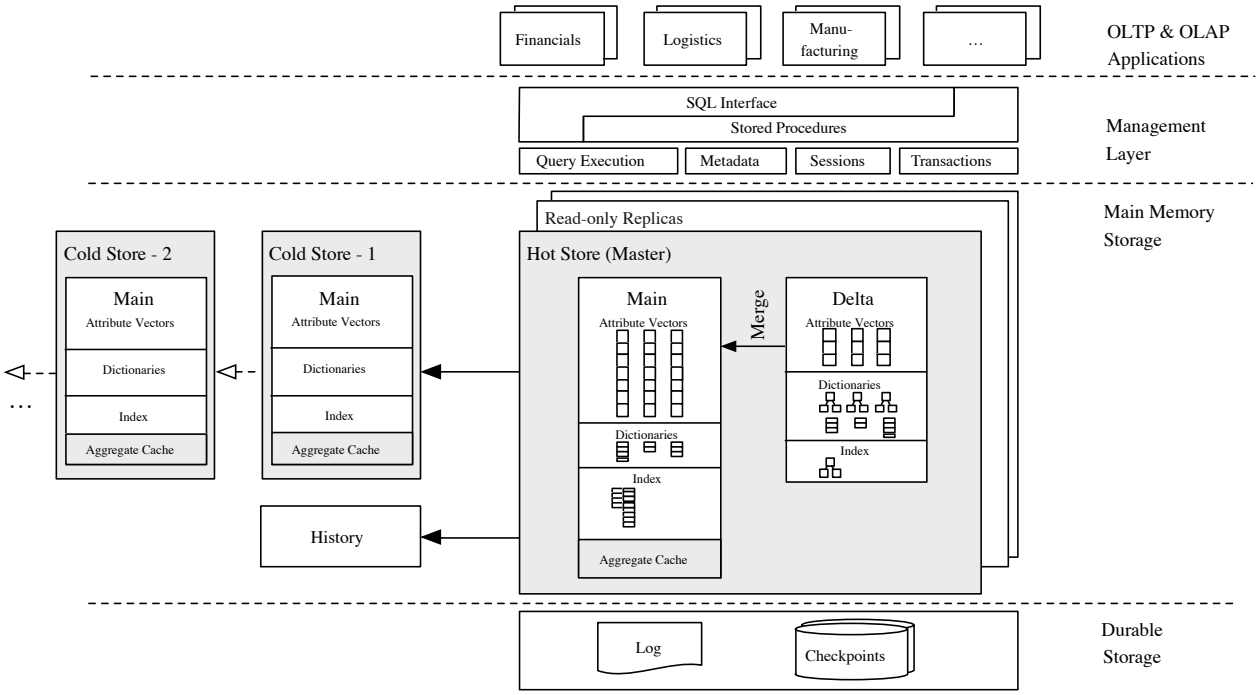

Figure 2: Architecture Blueprint of Columnar In-Memory Database.

for parallel recovery as log entries can be replayed in any order [25]. Periodically created checkpoints provide consistent snapshots of the database on disk in order to speed up recovery. Additional concepts leveraging the read-mostly workload like hot and cold data partitioning, transparent aggregate caches and read-only replication for scalability are discussed in Section 5.

In summary, the fast sequential memory scan speed of today's systems allows for a new database architecture for enterprise systems that combines various database techniques like columnar table layouts, dictionary compression, multi-version concurrency control and the insert only approach. In addition, the proposed database architecture enables the redesign of applications, as fast on-the-fly aggregations are possible and eliminate the need to maintain complex hierarchies of aggregates on application level.

3. TRANSACTION PROCESSING

It is a common belief that a columnar data layout is not well-suited for transactional processing and should mainly be used for analytical processing [1]. I postulate that this is not the case as column-based system architectures can even be superior for transactional business processing if a data layout without transaction maintained aggregates is chosen.

Transactional business processing consists of data entry operations, single record retrieval and set processing. Most single record retrievals are accesses to materialized aggregates and therefore logically retrieve results of aggregated sets of records. Therefore, for our discussion why column-based architectures are faster for transactional business processing than row-based architectures, we consider data entry performance and set processing capabilities.

3.1 Data Entry

The cost of data entry consists of record insertion and potentially updating related materialized aggregates. Single inserts of full records are slower in column-based than in row-based system architectures as the operation requires access to all attributes of a table which are distributed across various locations in main memory instead of one sequential access.

In case of transaction-maintained aggregates, each data entry operation requires an update of all corresponding aggregates, increasing the cost and complexity of the data entry. By dropping all transaction-maintained aggregates, indices and other redundant data structures, we can significantly simplify data entry transactions.

Analyzing typical data entry transactions in detail reveals that the overhead of maintaining aggregates on data entry by far outweighs the added insert costs of columnar tables. As an example, consider the data entry process of the SAP Financials application with the underlying data model as outlined in Figure 3. The master data tables contain customer, vendor, general ledger and cost center information and additional tables that keep track of the total per account. The actual accounting documents are recorded in two tables as an accounting document header and its line items. The remaining tables replicate the accounting line items as materialized views with various filters and different sort orders to improve the performance of frequent queries. Separate tables for open and closed line items exist for vendors (accounts payable), customers (accounts receivable) and general ledger accounts. Additionally, controlling objects are maintained containing all expense line items.

Figure 5 shows the fifteen consecutive steps for posting a vendor invoice in the classic SAP Financials application,

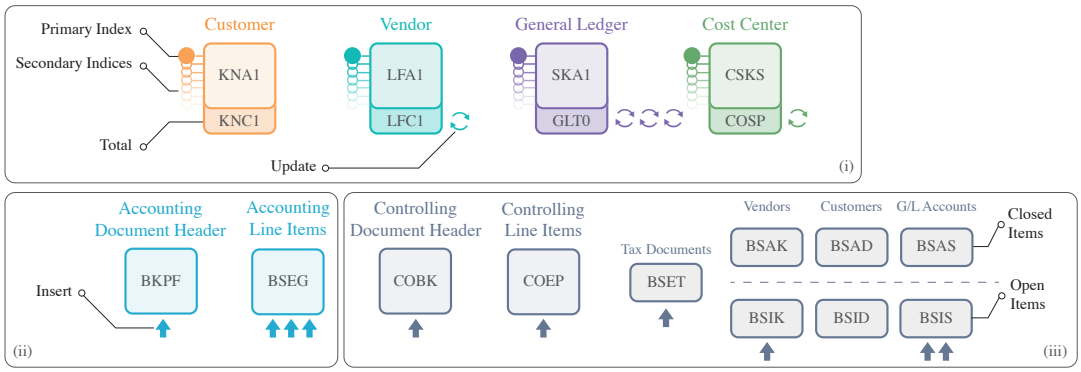

Figure 3: Selected tables of the SAP Financials data model, illustrating inserts and updates for a vendor invoice posting: (i) Master data for customers, vendors, general ledger and cost centers with transaction-maintained totals. (ii) Accounting documents. (iii) Replicated accounting line items as materialized views.

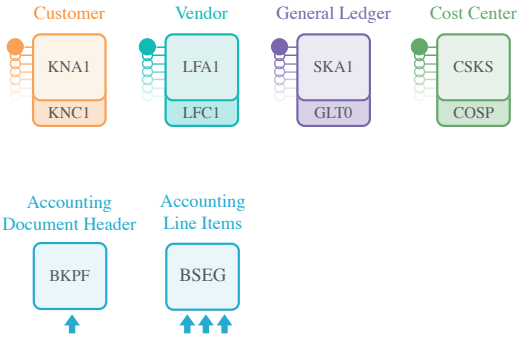

Figure 4: Simplified SAP Financials data model on the example of a vendor invoice posting illustrating the remaining inserts for the accounting document.

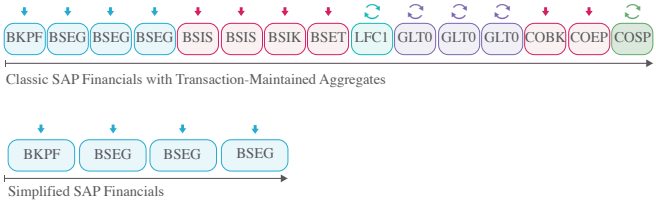

Figure 5: Executed steps for vendor invoice posting on classic SAP Financials with transaction-maintained aggregates compared to simplified application.

consisting of ten inserts and five updates. First, the accounting document is created by inserting the header and a vendor line item, an expense line item and a tax line item. Additionally, the expense and tax line items are inserted into the list of open items organized by general ledger accounts and the vendor line item is inserted into the list of open items organized by vendors. The tax line item is also inserted into the list of all tax line items. Then, the maintained total for the respective vendor account balance is updated. Afterwards, the general ledger totals are updated by writing the general ledger account balance, expense account balance and vendors reconciliation account balance. Finally, a controlling object is created by inserting a document header and one expense line item plus updating the respective cost center account balance.

In contrast, the simplified approach removes all redundant data and only records the accounting documents containing all relevant information as depicted in Figure 4. There is no need to update additional summarization tables or secondary indices. Consequently, the only necessary steps for the booking of a vendor invoice are the inserts of the accounting document header and its three line items as depicted in Figure 5. Although the single inserts into the column-store take longer, the reduction of complexity eliminates most of the work during data entry and results in significant performance advantages on the simplified data schema. In turn, data entry becomes actually faster on an in-memory column store.

We measured the runtime of both transactions in an productive setting, finding the simplified data entry transaction on an in-memory column store to be 2.5 times faster than the classic data entry transaction on a disk-based row-store.

3.2 Transactional Set Processing

For row-stores maintaining pre-built aggregates, it is essential to distinguish between anticipated queries and ad-hoc queries. Anticipated queries can leverage the predefined, transaction-maintained aggregates, whereas analytical ad-hoc queries require full table scans and are therefore in practice not executed on the transactional systems.

In contrast, by dropping the transaction-maintained aggregates in our column-based system architecture, there is no need to distinguish between ad-hoc queries and anticipated queries. All set processing queries can aggregate the required business data on the fly, taking advantage of fast sequential data access and are therefore not limited by the fixed predefined set of aggregates.

Comparing the two system architectures, we conclude that a row store is only faster in the artificial case if all aggregates are anticipated and do not change. Anytime an ad-hoc query requires an aggregate that has not been pre-build, the row-based architecture is by orders of magnitude slower. And changing the aggregate structure would mean that we have

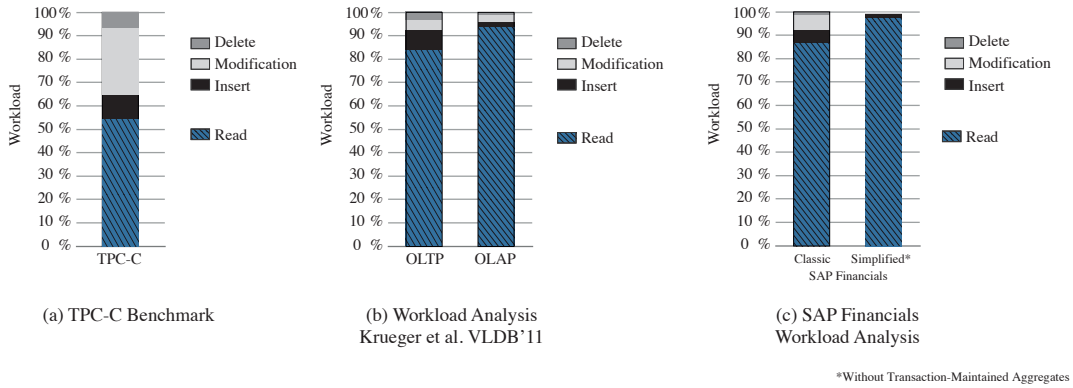

Figure 6: Workload Analysis outlining ratio between read and write queries. (a) Classic TPC-C benchmark. (b) Analysis of traditional enterprise system workload [10]. (c) Workload analysis of simplified SAP Financials.

to reconstruct the aggregation tables which instantly leads to down time with synchronized changes in the application code. Therefore, we consider a column-based architecture as faster for set processing when taking anticipated and ad-hoc queries into account.

4. WORKLOAD ANALYSIS OF SIMPLIFIED APPLICATIONS W/O AGGREGATES

Applications optimized for a column-based system architecture without transaction-maintained aggregates lead to a change in the database workload pattern. The share of read queries in the workload increases for three reasons: (i) the reduction of inserts and updates during data entry as shown in Section 3, (ii) the redesign of current applications to directly access the transactional data instead of materialized aggregates and (iii) the introduction of new, interactive applications with analytical capabilities.

Traditional applications, such as customer segmentation, dunning, or material resource planning, have typically been built around materialized aggregates. Redesigning such applications to calculate all information on the fly from the transactional schema leads to more complex queries. This increases the read share of the total workload, as the runtime of a query increases in the column-based system architecture compared to merely reading a single materialized aggregate in the row-based system architecture.

Enabled by the column-based system architecture, interactive analytical applications on the transactional schema present opportunities for new user groups to derive business value from the data. The possibility to ask follow-up questions in interactive applications and the availability on mobile devices lead to an increased usage and more queries. Following the law of supply and demand, the usage of such applications dramatically increases, since results are available whenever they are needed.

A workload analysis of the simplified SAP Financials application verifies the trend towards a read-dominated workload. The workloads were analyzed before and after the introduction of the simplified application without transaction-maintained aggregates, replacing a classic financial application with transaction-maintained aggregates. The workloads consists of the queries issued by thousands of users over one week each.

Figure 6 summarizes the findings and compares the workload patterns with TPC-C [23] and previous workload analyses [10]. Compared to the classic application, the share of read queries of the total execution time increased from 86 percent to 98 percent. With the absence of transaction maintained aggregates, the update share decreased significantly, representing only 0.5 percent of the total workload. In the classic application, each insert of a line item led to an average of 3.4 inserts or modifications to other tables within the same application module. The simplified application issues no additional inserts and calculates all information by filtering and aggregating the line items on the fly.

In contrast, the workload of the TPC-C benchmark does not reflect the share of reads and writes as observed in the financial systems. Instead, database systems for business applications have to be optimized for a read-dominated workload, with an increasing amount of analytical-style queries that aggregate data on the finest level of granularity, the actual business transactions.

5. IMPLICATIONS AND OPTIMIZATIONS

Transforming an IT landscape from a row-based system architecture to a column-based system architecture goes along with a simplified application development and a reduced data footprint, as described in this section. Furthermore, we discuss optimizations to the column-based system architecture to keep the system scalable.

5.1 Simplified Application Development

Filtering, grouping and aggregation of large datasets can be done interactively on in-memory column-stores and without the need to prepare indices upfront. This fundamental performance characteristic implies a complete shift in application development: Instead of anticipating a few analytical use cases that we optimize our programs for, a column-based system architecture allows us to query all transactional data with response times in the order of seconds.

The fast filtering and aggregation capabilities support new interactive applications which were not possible with predefined materialized aggregates. Example applications are

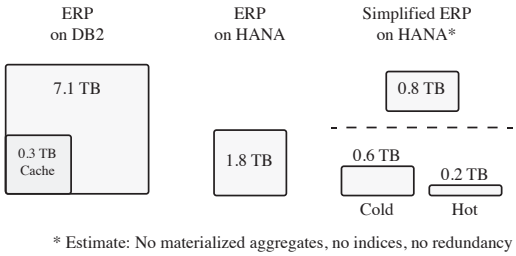

Figure 7: Data footprint reduction of SAP ERP system by moving to simplified ERP on HANA.

an available-to-promise check [22], sub-second dunning with customer segmentation [17] and real-time point-of-sale analytics [18]. Since these applications have short response times in the order of merely seconds, the results can finally be used within the window of opportunity, e.g. while the customer is on the phone or in the store.

5.2 Data Footprint Reduction

The removal of materialized aggregates and redundant data in combination with compression factors enabled by columnar storage reduces the data footprint significantly. Figure 7 outlines the data footprint reduction of the SAP ERP system.

Starting with the ERP system stored on DB2 and a size of 7.1 TB and 0.3 TB cache in main memory, the transition from storing the data in a columnar data layout using efficient dictionary compression on HANA results in a database size of 1.8 TB, yielding a compression factor of approximately four. The elimination of all transaction-maintained aggregates and other redundant data by redesigning the applications is estimated to reduce the data footprint down to 0.8 TB. Techniques such as hot and cold data partitioning can further reduce the data footprint in main memory, resulting in 0.2 TB of hot data storage with an impact on a variety of processes including data recovery and archiving. As a result, the columnar in-memory database needs less main memory for its hot storage than a traditional database uses for its cache in main memory. Note that besides reduced storage requirements, a reduced data footprint also accelerates all downstream processes such as system backup, replication and recovery.

5.3 Aggregate Cache

When resource-intensive aggregate queries are executed repeatedly or by multiple users in parallel, we need efficient means to keep the system scalable. As shown in Section 3, reading tuples of a materialized aggregate is faster than aggregating on the fly.

Recent work has shown that the main-delta architecture of the column-based system architecture is well-suited for an aggregate cache, a strategy of transparently caching intermediate results of queries with aggregates and applying efficient incremental view maintenance techniques [15]. The cached aggregates are defined only on the part of a user query that covers the main partition, as depicted in Figure 8. Since new records are only inserted to the delta partition, the main partition is not affected. This way, the definition of the cached aggregate on the main partition remains consistent with respect to inserts and does only need to be

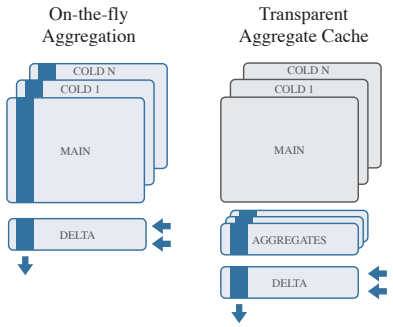

Figure 8: Schematic overview of transparent aggregate cache with efficient incremental view maintenance techniques.

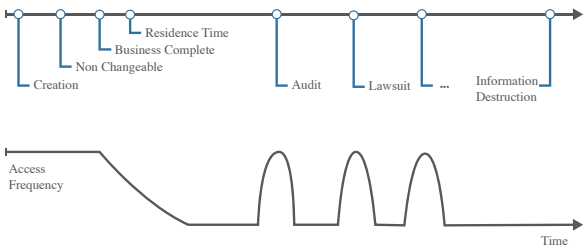

Figure 9: Lifecycle of Business Objects.

maintained during the online delta merge process. When a query result is computed using the aggregate cache, the final, consistent query result is computed by aggregating the newly inserted records of the delta partition and combining them with the previously cached aggregate of the main partition.

The aggregate cache can be used as a mechanism to handle the growing amount of aggregate queries issued by multiple users.

5.4 Hot and Cold Data Partitioning

Transactional enterprise data has to be kept in the system for many years due to internal and external reasons like controlling and legal regulations. However, as illustrated in Figure 9, after a certain period of time, data is only accessed for exceptional reasons and not as a result of ongoing business execution. Therefore, I propose a separation into hot and cold data regions so that cold data can be excluded for the majority of query executions while still guaranteeing correct results and improving query runtimes as well as main memory utilization. The proposed concept is a hot and cold partitioning on application level that leverages specific characteristics of business objects and the way they are typically accessed. Therefore, the concept needs to be finely tuned based on the type of objects and the relevant business semantics. In addition, automatic data aging mechanisms could be applied on database level as an orthogonal concept.

Partitioning can be achieved through multiple workload-specific metrics. A very simple, yet efficient partitioning scheme is to use all data that is part of ongoing transactional business for the hot partition. This data can be identified by selecting the current fiscal year along with open transac-

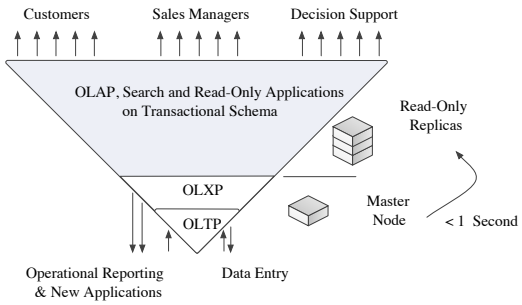

Figure 10: Read-Only Replication.

tions such as open invoices or open customer orders. Since it is often desirable to compare the current year with the last year, we apply the same logic to last years data. All other transactional data can be considered as cold. Master data and configuration data always remains hot as it is frequently requested and only consumes little main memory capacity. If we want to access historic data, we simply access both hot and cold data partitions. On the other hand we can concentrate all business activities, including monthly, quarterly, or yearly reports on the hot data partition only. Our analysis shows that the hot data volume is typically between two and ten percent of the data volume of a traditional database. This is even less than a traditional database typically uses for its in-memory cache.

5.5 Read-only Replication

Despite the aggregate cache, the increasing demand of new applications, more users and more complex queries can eventually saturate a single-node in-memory database system.

To keep up with the growing demand for flexible reports, we propose a read-only replication of the transactional schema. The scale-out is performed by shipping the redo log of the master node to replications that replay transactions in batches to move from one consistent state to the next [14]. Figure 10 provides an overview of this design. The need for real-time information depends on the underlying business functionality. While many applications of the day-to-day business such as stock level or available-to-promise checks need to run on the latest data to produce meaningful results, others can work with relaxed isolation levels. I propose that a powerful master node handles transactions and OLXP workload with strong transactional constraints.

Modern IT solutions create business value through analytical applications on top of the transactional data. Customers can access the system to track the status of their orders, sales managers are enabled to analyze the customers profile and use analytical applications such as recommendations and managers access run decision support queries.

These applications create the majority of the system load, but they have relaxed real time constraints. The queries can work on snapshots and evaluated on read-only replicas of the same transactional schema. Contrary to existing data warehousing solutions with an ETL process, the applications can be created with all flexibility, as all data is accessible up to the finest level of granularity.

These read-only replicas can be scaled out and perform the share of the read-only workload that has relaxed transactional constraints. Since there is no logic to transform data into a different representation, the replication for typical enterprise workloads can be performed with a delay of less than a second. In turn, the transactional workload is not hindered and all applications can use the same transactional schema, without the need for complex and error-prone ETL processes.

6. FUTURE RESEARCH

Our ongoing research efforts are concentrated on workload management features for the proposed database architecture [26], lightweight index structures for column stores [4] and optimized transaction handling for highly contentious workloads. For future work, we foresee several areas of research for further investigation.

Although hierarchies can be modeled in the data schema by using techniques as adjacency lists, path enumeration models and nested set models, querying complex hierarchies expressed in standard SQL can be cumbersome and very expensive. Consequently, we plan to further investigate new ways of calculating and maintaining hierarchies of dynamic aggregates. Additionally, applications need to be redesigned in order to allow users to define new hierarchies. Intuitive mechanism for describing and modeling hierarchies are necessary. This can be beneficial for both, the caching of dynamic aggregates and for new applications including enterprise simulations and 'what-if' scenarios.

The availability of large capacities of main memory has been one of the hardware trends that make the proposed database architecture a viable solution. New changes to the commodity hardware stack, such as non-volatile memory and hardware transactional memory are on the horizon and the database architecture will be adapted to leverage these technologies. In a first step, non-volatile memory can easily be used as a fast storage medium for the database log. In the future, the primary persistence might be stored on non-volatile memory allowing to significantly decrease recovery times, introducing the new challenge of directly updating the durable data using a consistent and safe mechanism.

The new provided flexibility in maintaining multiple reporting hierarchies or analyzing recent business data in near real-time will lead to completely new types of applications. The possibility of predicting future trends or quickly reacting on changing trends by running complex enterprise simulations directly on the actual transactional data will change the way businesses are organized and managed.

7. CONCLUSION

In 2009, I proposed a column-based system architecture for databases that keeps data permanently resident in main memory and predicted that this database design will replace traditional row-based databases [16]. Today, we can see this happening in the market as all major database vendors follow this trend. In the past 5 years, we have proven the feasibility of this approach and many companies have this database architecture already in productive use. The experiences gained by rewriting existing applications and writing new applications, which I have not even dreamed that they are possible ten years ago, have confirmed that column-based systems without any transaction-maintained

aggregates are the superior architecture for enterprise applications. I predict that all enterprise applications will be built in an aggregation and redundancy free manner in the future.

8. ADDITIONAL AUTHORS

Martin Faust, Stephan Müller, David Schwalb, Matthias Uflacker, Johannes Wust.

9. REFERENCES

[1] D. J. Abadi, S. R. Madden, and N. Hachem. Column-Stores vs. Row-Stores: How Different Are They Really? *ACM*, 2008.

[2] P. A. Bernstein, V. Hadzilacos, and N. Goodman. Concurrency control and recovery in database systems. Boston, MA, USA, 1986.

[3] G. P. Copeland and S. N. Khoshafian. A decomposition storage model. *SIGMOD*, 1985.

[4] M. Faust, D. Schwalb, J. Krüger, and H. Plattner. Fast lookups for in-memory column stores: Group-key indices, lookup and maintenance. In *ADMS@VLDB*, 2012.

[5] M. Grund, J. Krueger, H. Plattner, A. Zeier, P. Cudre-Mauroux, and S. Madden. HYRISE—A Main Memory Hybrid Storage Engine. *VLDB*, 2010.

[6] S. Idreos, F. Groffen, N. Nes, S. Manegold, K. S. Mullender, and M. L. Kersten. Monetdb: Two decades of research in column-oriented database architectures. *IEEE Data Eng. Bull.*, 2012.

[7] T. Karnagel, R. Dementiev, R. Rajwar, K. Lai, T. Legler, B. Schlegel, and W. Lehner. Improving in-memory database index performance with intel transactional synchronization extensions. *HPCA*, 2014.

[8] M. Kaufmann, P. Vagenas, P. M. Fischer, D. Kossmann, and F. Färber. Comprehensive and interactive temporal query processing with SAP HANA. *VLDB*, 2013.

[9] A. Kemper and T. Neumann. HyPer: A hybrid OLTP&OLAP Main Memory Database System based on Virtual Memory Snapshots. *ICDE*, 2011.

[10] J. Krüger, C. Kim, M. Grund, N. Satish, D. Schwalb, J. Chhugani, P. Dubey, H. Plattner, and A. Zeier. Fast updates on read-optimized databases using multi-core cpus. *VLDB*, 2011.

[11] P.-A. Larson, S. Blanas, C. Diaconu, C. Freedman, J. M. Patel, and M. Zwilling. High-performance concurrency control mechanisms for main-memory databases. *VLDB*, 2011.

[12] R. MacNicol and B. French. Sybase IQ Multiplex - Designed for Analytics. *VLDB*, 2004.

[13] C. Mohan, D. Haderle, B. Lindsay, H. Pirahesh, and P. Schwarz. ARIES: a transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging. *TODS*, 1998.

[14] T. Mühlbauer, W. Rödiger, A. Reiser, A. Kemper, and T. Neumann. ScyPer: elastic OLAP throughput on transactional data. *DanaC*, 2013.

[15] S. Müller and H. Plattner. Aggregates caching in columnar in-memory databases. *IMDM@VLDB*, 2013.

[16] H. Plattner. A common database approach for oltp and olap using an in-memory column database. *SIGMOD*, 2009.

[17] H. Plattner. SanssouciDB: An In-Memory Database for Processing Enterprise Workloads. *BTW*, 2011.

[18] D. Schwalb, M. Faust, and J. Krüger. Leveraging in-memory technology for interactive analyses of point-of-sales data. *ICDEW*, 2014.

[19] V. Sikka, F. Färber, W. Lehner, S. K. Cha, T. Peh, and C. Bornhövd. Efficient Transaction Processing in SAP HANA Database: The End of a Column Store Myth. *SIGMOD*, 2012.

[20] D. Ślęzak, J. Wróblewski, and V. Eastwood. Brighthouse: an analytic data warehouse for ad-hoc queries. *VLDB*, 2008.

[21] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. O'Neil, P. O'Neil, A. Rasin, N. Tran, and S. Zdonik. C-store: A column-oriented dbms. *VLDB*, 2005.

[22] C. Tinnefeld, S. Müller, H. Kaltegärtner, and S. Hillig. *Available-To-Promise on an In-Memory Column Store.* BTW, 2011.

[23] Transaction Processing Performance Council (TPC). TPC-C Benchmark. http://www.tpc.org/tpcc/.

[24] T. Willhalm, N. Popovici, Y. Boshmaf, H. Plattner, A. Zeier, and J. Schaffner. SIMD-Scan: Ultra Fast in-Memory Table Scan Using on-Chip Vector Processing Units. *VLDB*, 2009.

[25] J. Wust, J.-H. Boese, F. Renkes, S. Blessing, J. Krueger, and H. Plattner. Efficient logging for enterprise workloads on column-oriented in-memory databases. *CIKM*, 2012.

[26] J. Wust, M. Grund, K. Höwelmeyer, D. Schwalb, and H. Plattner. Concurrent execution of mixed enterprise workloads on in-memory databases. *DASFAA*, 2014.