B. Mitschang et al. (Hrsg.): Datenbanksysteme für Business, Technologie und Web (BTW 2017),
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2017 545

SAP HANA – The Evolution of an In-Memory DBMS from Pure OLAP Processing Towards Mixed Workloads

Norman May[1], Alexander Böhm[2], Wolfgang Lehner[3]

Abstract: The journey of SAP HANA started as an in-memory appliance for complex, analytical applications. The success of the system quickly motivated SAP to broaden the scope from the OLAP workloads the system was initially architected for to also handle transactional workloads, in particular to support its Business Suite flagship product. In this paper, we highlight some of the core design changes to evolve an in-memory column store system towards handling OLTP workloads. We also discuss the challenges of running mixed workloads with low-latency OLTP queries and complex analytical queries in the context of the same database management system and give an outlook on the future database interaction patterns of modern business applications we see emerging currently.

1 Introduction

The increase of main memory capacity in combination with the availability of multi-core systems was the technical foundation for the In-Memory SAP HANA database system. Based on the SAP Business Warehouse Accelerator (SAP BWA) product, SAP HANA combined a column-based storage engine with a row-store based query engine. As a result, it delivered superb runtime numbers for analytical workloads by fully leveraging the performance of the columnar data organization as well as algorithms highly tuned for in-memory processing (cache awareness, full SIMD support etc.).

The success of the disruptive approach taken by SAP HANA, to move towards a highly parallel and in-memory computing model for data management also triggered to re-think and finally shake the foundations of traditional enterprise data management architectures: While the traditional separation of transactional and analytical processing has clear advantages for managing and specializing the individual systems, it also comes with constraints which cannot be tolerated to serve modern business applications: For example, data on the analytical side is only updated periodically, often only on a nightly basis resulting in stale data for the analytical side. Complex ETL processes have to be defined, tested, and maintained. Data preparation tasks from executing ETL jobs, populating data warehouse databases as well as deriving Data Marts to adhere to specialized physical storage representations of BI tools etc. reflect a highly error-prone as well as time-consuming task within the enterprise data management stack. Last but not least, two separate database management systems need to be licensed and operated.

[1] SAP SE, 69190 Walldorf, norman.may@sap.com
[2] SAP SE, 69190 Walldorf, alexander.boehm@sap.com
[3] TU Dresden, Fakultät Informatik, 01062 Dresden, wolfgang.lehner@tu-dresden.de

Inspired by the potential of SAP HANA, Hasso Plattner started to formulate the vision of an "Enterprise Operational Analytics Data Management System" combining OLTP and OLAP workloads as well as the underlying data set based on a columnar representation ([Pla09]). While this approach was lively discussed within academia as well as openly questioned by senior database researchers ([SC05]) declaring mantra-like that "One Size does not Fit All!", SAP took on the challenge to investigate how far the envelope could be pushed. The goal was to operate one single system providing the basis for realtime analytics as part of operational business processes while also significantly reducing TCO. Specifically, key questions from a technical perspective have been:

- What are the main capabilities of the SAP HANA columnar storage engine, and how could they be exploited for transactional workloads?
- What characteristics of transactional workloads have to be treated with specific optimizations within the SAP HANA engine, and what would be technical solutions?
- How could transactional as well as analytical workload co-exist from a scheduling as well as data layout perspective?

Within this paper, we highlight some technical challenges and give key solutions with respect to these questions. We will show, how the SAP HANA system evolved and continues to evolve from a purely analytical workhorse to an integrated data management platform that is able to serve the largest ERP installations with 5 million queries/h and more than 50K users concurrently. Figure 1 outlines the major evolutionary steps we present in this paper in a chronological order.

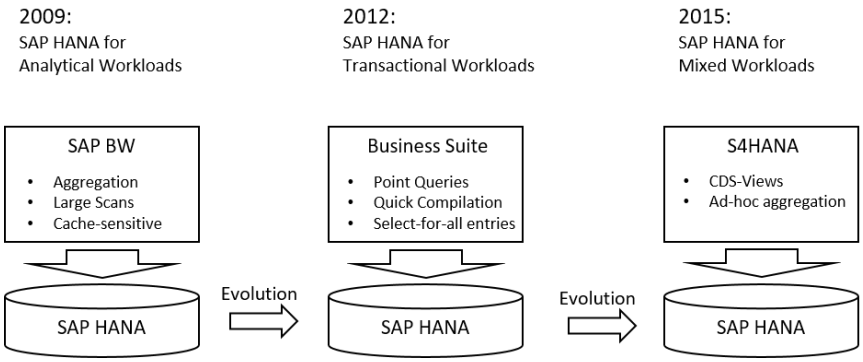

Fig. 1: Evolution of SAP HANA.

Structure of the Paper

The next section summarizes the core ingredients of SAP HANA to provide high performance for analytical workloads (Section 2). The next step within the SAP HANA evolution is then addressed within Section 3 outlining some challenges of supporting transactional query patterns. Section 4 finally shares some requirements and solutions of supporting

S/4HANA, the new SAP flagship product, implementing the vision of a single point of truth for all business related data and any type of interaction with this data set. Section 5 finally concludes with hinting at some challenges as well as opportunities for academia to be tackled within the near future.

2 SAP HANA for Analytical Scenarios

As mentioned above, the original use-case of SAP HANA was to support analytical workloads, motivated by the requirements of the SAP Business Warehouse (BW). The original design therefore heavily focused on read-mostly queries with updates performed in bulks (as the result of ETL processes) and took several architectural choices that strictly favor fast query processing over efficient update handling. Aggregation queries typically addressed star- or snowflake schemas with wide dimension tables and very few large fact tables. Concurrency was limited to a few (in the range of hundreds) users with heavy read-intensive workloads.

2.1 Parallelization at all Levels

One of the core design principles of SAP HANA to cope with these requirements was to perform parallelization at all levels starting from hardware (e.g. heavily using vectorized processing) to query execution. This choice was motivated by the recent trends in hardware development, where CPU clock speed does no longer increase significantly with newer hardware generations, but an ever growing number of CPUs becomes available instead. Consequentially, to make use of an ever growing number of CPUs, SAP HANA parallelizes the execution of queries at different levels:

- Inter-Query Parallelism by maintaining multiple user sessions.
- Intra-Query Parallelism/Inter-Operator Parallelism by concurrently executing operations within a single query.
- Intra-Operator Parallelism using multiple threads for individual operators.

Figure 2 shows the principle of one of the most frequently used operators for analytical workloads–aggregation [TMBS12, MSL+15]: A central (fact) table is logically subdivided into different fragments which may fit into the processor cache. The aggregation algorithm first aggregates chunks of the tuples into thread-local hash tables that fit into the L2 cache. When a certain number of pre-aggregated data is available those tables are merged into the final aggregated results. In this second reduction phase we avoid locking the buckets of hash tables by partitioning the groups to separate threads. Overall, this leads to almost linear scalability of the aggregation operator with the number of available threads. Other relational operators are tuned in a similar fashion for a high-degree of parallelism to exploit modern hardware capabilities.

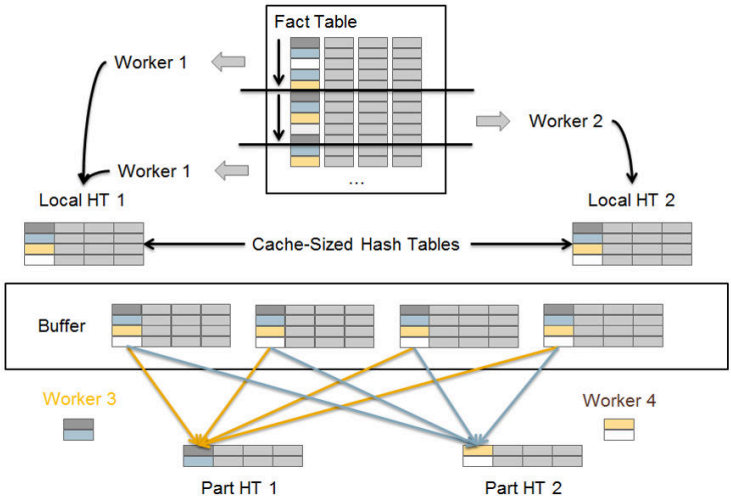

Fig. 2: Principle of parallel aggregation within SAP HANA.

2.2 Scan-friendly Physical Data Layout

SAP HANA relies heavily on a column-oriented data layout, which allows to scan only those columns which are touched by a query. Moreover, since dictionary compression is used for all data types within SAP HANA, the stored column values can be represented by the minimal number of bits required to encode the existing number of distinct values [FML+12]. This storage layout both enables a high compression potential (e.g. run-length encoding, sparse encodings etc.) and allows to fully exploit the capabilities of the CPU's hardware prefetcher to hide the latency induced by the memory access. The dictionary itself is sorted with respect to the actual column values supporting an efficient value lookup for example to resolve search predicates. The columnar storage layout and dictionary compression are also used in other database systems, e.g. [IGN+12, KN11, RAB+13, LCF+13, LCC+15], but using ordered dictionaries for all data within an in-memory database system is a unique feature of SAP HANA.

To avoid re-encoding large amounts of data upon dictionary changes, update operations are buffered within a special data structure (delta index) to keep the main part of the table as static (and highly compacted) as long as possible. An explicit merge operation fuses the buffered entries with the static part of the table and creates a new version of the main part to keep the data in a scan-friendly format.

2.3 Advanced Engine Capabilities

In addition to plain SQL support, SAP HANA supports analytical applications by providing a rich bouquet of specialized "engines". For example, a planning engine provides primitives

to support financial planning tasks like disaggregation of global budgets to different cost centers according to different distribution characteristics. A text engine delivers full text retrieval capabilities from entity resolution via general text search techniques to classification algorithms. Special engines like geospatial, timeseries, graph, etc. in combination with a native connectivity with R as well as machine learning algorithms using the AFL extension feature of SAP HANA complement the built-in services for analytical applications.

In addition to engine support, the SAP HANA database engine is part of the SAP HANA data management platform offering integration points with other systems for example to implement federation via the "Smart Data Access" (SDA) infrastructure or specialized connectivity to Hadoop and Spark systems. "Dynamic Tiering" (DT) is offered to perform data aging transparently for the application. DT allows to keep hot data (actuals) within SAP HANA's main memory and cold data (historical data) within SAP IQ by providing a single system experience (e.g. single transactional context, single user base, single admin console) [MLP+15].

2.4 Summary

SAP HANA's DNA consists in mechanisms to efficiently run analytical workloads for massive data sets stored within SAP HANA's columnar data representation. In order to support analytical workloads beyond traditional single engine capabilities, SAP HANA offers a rich set of add-ons to act as a solid and high performance foundation for data analytics applications.

3 OLTP Optimizations in SAP HANA

The second step within the evolutionary process of SAP HANA was positioned to provide a solid foundation for SAP's OLTP workload. One particular goal was to support SAP's ERP product based on the enterprise-scale ABAP stack which includes the ABAP programming language as well as the associated application server, and application lifecycle management. While academia heavily questioned the possibility to optimize a column-store for OLTP workloads, the goal of SAP was to provide a robust solution to sustain a typical enterprise workload in the range of 50.000 OLTP statements/second, comprising queries as well as concurrent updates. Within this section, we highlight some of the optimizations we could integrate into the productive version of SAP HANA without compromising the analytical performance, and also highlight performance improvements that were made over time.

3.1 Initial Analysis and Resulting Challenges

Since SAP HANA comes with a column as well as a row store, the obvious solution would be to internally replicate transactional data and use the row store for OLTP workloads. While this approach has been followed by other systems (e.g. Oracle 12c [LCC+15], IBM

DB2 [RAB+13] and MS SQL Server [LCF+13]), it would be impractical for ERP scenarios with more than 100.000 tables from a management as well as an overall cost perspective (as this replication-based approach significantly increases the overall memory requirements of the system by storing data twice). Consequently, the column store-based implementation has to be advanced to deal with update intensive scenarios, where insert/update/delete operations are performed on a fine granularity (i.e. primary key access as the most relevant access path compared to scan-based data access), data retrieval typically performed for all attributes in a row (`SELECT *` requires row reconstruction out of individual column values), and typically a very high number of concurrently active users requiring a robust resource management.

Based on those observations, a magnitude of optimizations has been realized with SAP HANA; some of the key optimizations applied to productive SAP HANA are discussed in more detail in the subsequent sections.

3.2 Short-Running Queries

While the original query execution model was designed for long-running, scan-based queries, OLTP is characterized by short-running queries. As query compilation time does not much affect the overall runtime of OLAP-style queries, query preparation may be the dominant factor for short-running queries. In many cases, compiling an OLTP query can simply not be tolerated. Therefore, SAP HANA stores optimized SQL statements in a plan cache which associates the SQL string with a compiled query execution plan. Consequently SQL statements are not even parsed when a corresponding entry exists in the plan cache. For parametrized queries, the SQL statement is optimized again and cached when the first parameter value is passed for execution. Plan cache entries are either invalidated when DDL operations are performed that may invalidate the precompiled plan or after a certain number of executions to accommodate for updates to the data.

To emphasize the importance of plan caching we refer to Figure 3. In this figure, we have sorted the top 100 statements of two large productive instances of SAP HANA running an ERP workload based on accumulated query execution time and plot them by their execution count. In both systems, the ten most frequently executed statements comprise one third of all statement executions in the system. The plan cache in these systems consumes a few GB out of a few TB of main memory to guarantee cache hit ratios above 99%, i.e. less than 1% of all incoming queries require a compilation phase. In the two ERP systems shown in Figure 3, cached query execution plans are executed a few thousand times. However, if compilation is necessary, a query of "simple nature", e.g. single table access based on the primary key, does not even enter the full-blown query optimization path but is parametrized and directly executed using specialized access code.

While maximal parallelization within the database engine is the key for OLAP-style queries, plan-level parallelism is disallowed for OLTP queries in order to avoid context switches, scheduling overhead, and the resulting cache misses. The optimizer only allows parallelism,

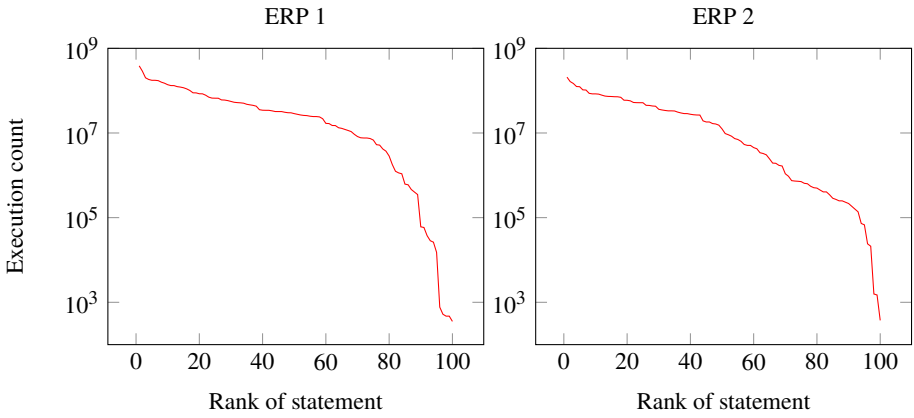

Fig. 3: Execution count of TOP 100 queries in two large productive ERP systems.

if (almost) linear speedup can be expected based on the current load in the system [PSM+15]. A further investigation reveals that parallelization in OLTP scenarios is already implicitly done at the application server level with more than 10.000 concurrent users.

3.3 Optimized Physical Data Representation

OLTP scenarios–especially in the context of SAP ERP–very often exhibit "empty attributes" typically showing only one single default value. The major reason for this is that not all business functionality (i.e. extensions for specific industries) that is backed by corresponding fields in a table is used by all customers, but anyhow fields are retrieved by the default access paths encoded in the application server stack. Consequentially, in productive ERP systems, we often find more than one thousand columns that store only a single (default) value – including several large and hot tables in functional components of the ERP like financials, sales and distribution, or material management. Access on such columns is optimized by completely bypassing the columnar representation and storing the single default values within the "header" of the columnar table, i.e. within the in-memory storage. This reduces the latching overhead when accessing these columns. In internal OLTP benchmarks on ERP data sets we measured about 30% higher throughput and 40% reduced CPU consumption when using this optimization.

As mentioned in the previous section, the columns of a table are stored in delta and main structures to buffer changes to the data and periodically apply these changes by generating a new version of the main part. Those maintenance procedures can be easily weaved into the loading procedures of data warehouse systems. Necessary database reorganizations can thus be directly performed as part of the ETL processes. For OLTP, exclusively locking a table generates massive lock contention because the system continuously has to accept update operations. As a consequence, reorganizations must not interfere with or even completely block concurrently running OLTP queries. Within SAP HANA, the MVCC scheme is extended to explicitly support delta merge operations requiring a table lock only

for switching over to the new version resulting in a very limited impact on concurrent workload.

Finally, queries with a predictable result set cardinality have an optimized physical result vector representation in row-format. This format reduces one allocation per column vector to a single allocation of a compact array. Costly cache misses are thus avoided when materializing the query result because of the dense memory representation.

3.4 Transaction Management

SAP HANA relies on snapshot isolation in two variants: In statement-level snapshot isolation every statement receives a new snapshot and a statement reads all versions that were committed when the statement started. This is the default mode of snapshot isolation used in SAP HANA. With transaction-level snapshot isolation all statements of a transaction read the latest versions committed before the transaction started or that were modified by the transaction itself.

The initial transaction manager of SAP HANA was tracking visibility information via an explicit bit-vector per transaction on table level. Read transactions with the same snapshot shared this bit-vector, but write transactions generated a new bit-vector which was consistent with their snapshot. Bit-vectors representing snapshots of transactions that were older than the snapshot of the oldest open transaction could be removed by a garbage collector. While this design is feasible for OLAP workload with rare updates spanning only a few tables within a single (load) transaction, it turned out to be too expensive and memory consuming for transactions with only a few update operations per table, touching multiple tables sprinkled across the whole database.

This deficit for OLTP workload resulted in a significant extension of the transaction manager and a row-based visibility tracking mechanism. For every row the timestamp when it was created or deleted is stored for the row (note that updates create a new version of the row). As soon as all active transactions see a created row or likewise no active transaction may see a deleted row, these timestamps can be replaced by a bit indicating its visibility. Furthermore, rows that are not visible to any transaction can be removed during the next delta merge operation. This change in the design allows to handle fine-granular updates and is independent of the number of concurrently running transactions. At the same time, OLAP workloads still benefit from the efficient bit-vector-based visibility test for most rows.

Further improvements of the garbage collection are possible as detailed in [LSP+16]. Especially, in mixed workloads, OLTP applications trigger many new versions of data objects while long-running OLAP queries may block garbage collection based on pure timestamp comparison. The SAP HANA garbage collector improves the classical garbage collection based on timestamp-based visibility by 1) identifying versions that belong to the same transaction, 2) identifying tables that are not changed by old snapshots, and 3) identifying intervals of snapshots without active transactions. Combining these garbage collection strategies resulted in reduced memory consumption for keeping versions of

records for multiple snapshots. For the TPC-C benchmark the latency of statements could be reduced by 50%. When having mixed workloads with concurrent long-running statements the OLTP throughput of TPC-C was about three times higher with the improved garbage collection compared to the commonly used global garbage collection scheme.

3.5 Latching and Synchronization

Especially on large systems with many sockets, synchronization primitives that protect internal data structures can become a major scalability bottleneck. These latches require cross-socket memory traffic which is factors slower than the cache-conscious code that is protected by these latches. In many cases, the access to the protected data structure is dominated by read operations, e.g. column access or B-tree access. Consequently, HANA carefully optimizes those accesses balancing the complexity of the implementation with the gain in performance. This confirms the work by [LSKN16].

Latching on the column level is needed to protect against unloading of columns in the presence of memory pressure [SFA+16] or installing a new version of a column after a delta merge operation. This scenario is compatible with RCU-based latching, i.e. update operations install a new version of the object while existing readers either keep working on old references and new readers immediately work on the new version of the column [McK04]. In the case of unloading columns, the column appears as unloaded for new readers while the unloading becomes effective only after the last reader dereferenced the column. In the case of delta merge operations a new version of the merged column is installed in parallel to the old version. The merge operation switches the versions of the main fragment and also of the delta fragments in an atomic operation, and thus with minimal impact on both read and update operations.

As a last optimization HANA uses Intel TSX operations to increase scalability. These operations are especially important in the B-tree used for the lookup of value IDs in the delta fragment because the delta fragment is accessed both for read operations and also updated concurrently by any update operation. In [KDR+14] it is shown that using TSX results in far better scalability with the number of concurrent threads. This is especially important on systems with a growing number of available cores as we see them already today in deployments of SAP HANA.

3.6 Application-Server/DBMS Co-Design

As already presented in [Böh15], SAP HANA exploits the potential to optimize across boundaries within the software stack. Obviously, SAP HANA natively optimizes for the ABAP language and runtime.

On the one hand, the "Fast Data Access" (FDA) feature allows to directly read from and write into a special internal data representation which can be shared between the database system and the application server. Bypassing the SQL connectivity stack for data transfer

results in a 20% speedup on average (Figure 4a) depending obviously on the cardinality of the data set.

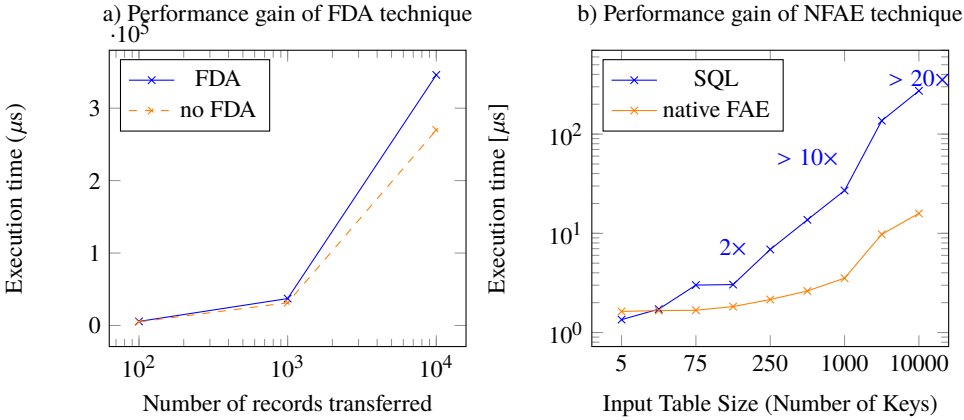

Fig. 4: Impact of SAP HANA optimizations with respect to FDA and NFAE.

On the other hand, the "Native For All Entries" (NFAE)-technique modifies the ABAP runtime to convert a sequence of conjunctions, disjunctions or large IN-lists into a semi-join, which can be more efficiently executed by the underlying database system. For example, the ABAP statement below extracts train departure and arrival information for all trains with an ID and a specific date given within the runtime table `key_tab` residing in the application server.

```
SELECT t.departure, t.arrival FROM trains t
INTO CORRESPONDING FIELDS OF TABLE result_tab
FOR ALL ENTRIES IN key_tab k
WHERE t.ID = k.c1 AND t.DATE = k.c2
```

NFAE exploits FDA to pass the runtime table of the application server efficiently to the SAP HANA process and issues a semi-join between the database table `trains` and the (former) runtime object `key_tab`. Depending on the cardinality of the runtime table, a speedup in the order of a magnitude can be achieved without changing a single line of application code (Figure 4b).

3.7 Results

As outlined, evolving the columnar storage engine and runtime towards an OLTP capable system required substantial changes in all components from transaction management to special support of the SAP application server. To underpin the achievements, Figure 5 quantifies the performance improvements of some of the most critical DB operations (log scale!). The measurements comprise a 2-year period and are derived from the productive SAP HANA code base.

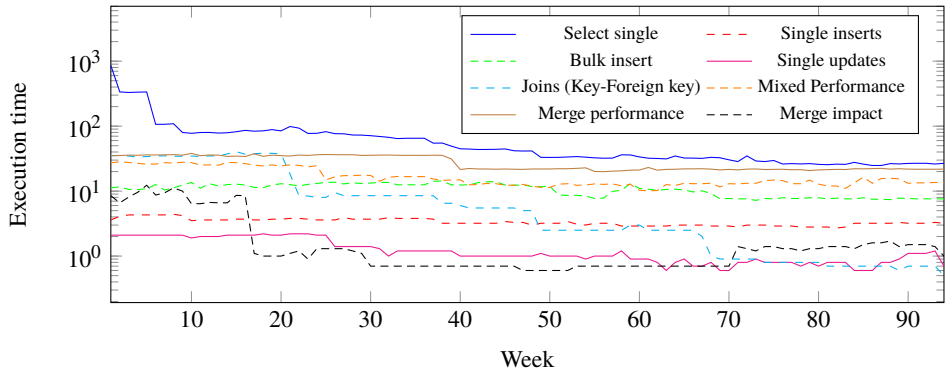

Fig. 5: Impact of SAP HANA optimizations to improve OLTP workload.

As can be seen, point queries (select single) have improved by over an order of magnitude, which also applies for the impact of delta merge operations on concurrently running database workloads. Dramatic improvements have been achieved also for key-foreign key joins, which are–in addition to point queries–the most performance critical query patterns for OLTP workloads.

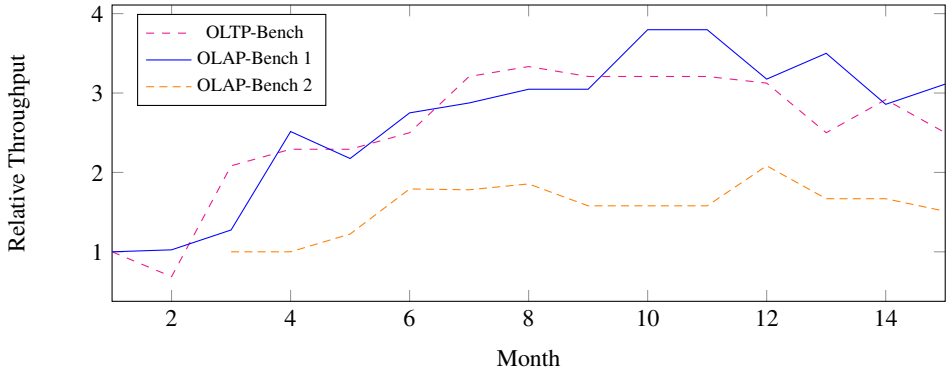

Fig. 6: Performance evolution of application scenarios.

Figure 6 presents how the throughput for two analytical and one transactional application scenarios improved over a 15 month period. Due to legal reasons we present only relative throughput numbers here: For the OLAP benchmarks the throughput is measured in queries per hour, and for the transactional throughput it is measured in the number of concurrent clients which execute transactional queries and updates with a bounded response time. The experiments were executed on a 4TB system with 16 sockets and logical 480 cores. For the analytical benchmarks the number of queries per hour processed by the system increased by up to 4 times. At the same time the number of concurrent users that could be handled by SAP HANA while guaranteeing a maximal response time per query also improved by a factor of three.

In total, the improvements presented so far resulted in

- a factor 25+ efficiency gain in highly concurrent OLTP select operations.
- a factor 15+ improved runtime performance for short-running join operations.
- a factor 8+ improved performance in order-to-cash benchmark.
- no congestion/queuing situations due to delta merge operations.

Overall, SAP HANA provides more than "good-enough" throughput for OLTP operations for typical ERP scenarios, although the system was a priori not designed as a highly specialized OLTP engine but took most design choices in favor of superior analytics performance. Most notably, the above-mentioned optimizations could be achieved without compromising the OLAP performance of the system.

4 SAP HANA Optimizations for Mixed Workloads

As already motivated in the introduction, the success of SAP HANA triggered a re-thinking of business processes and business application support. In order to efficiently operate a modern enterprise, the technically required decoupling of transactional and analytical scenarios can no longer be tolerated. In order to close the gap between making and executing operational decisions based on analytical evaluations and (more and more) simulations, the tight coupling of these two worlds based on a common data management foundation is becoming a must-have. In addition to application requirements, the accompanying operational costs of running two separate systems is no longer accepted.

As a consequence, SAP HANA was extended to deal with mixed workloads on the same database instance. This implies that a specific schema for analytical scenarios (e.g. star-/snowflake schemas) are no longer the base of executing OLAP workloads. Instead, the system must be able to (a) schedule different types of queries (OLTP as well as OLAP) with significantly different resource requirements as well as (b) efficiently run OLAP-style queries on-top of an OLTP-style database schema.

The following sections share some insights into different optimization steps applied to the productive version of SAP HANA.

4.1 Resource Management for Mixed Workloads

A major challenge of mixed workloads we see in SAP HANA are thousands of concurrent transactional statements in addition to hundreds of concurrent analytical queries. The former class of statements requires predicable and low response times, but this is hard to achieve as complex concurrent analytical queries acquire significant amounts of CPU and memory for a real-time reporting experience. Additionally, complex ad-hoc queries may turn out to be

far more resource-intensive than expected, and such statements must not block business critical transactional operations.

One solution to achieve robustness for transactional workload while serving concurrent analytical workload is to assign resource pools to these workloads. However, it is known that this leads to sub-optimal resource utilization [Liu11], and thus is adverse to reducing operational costs. Consequently, instead of over-provisioning ressources, SAP HANA detects cases of massive memory consumption and aborts the associated analytical statements. This is considered acceptable because given a proper sizing of the system such statements should be exceptional.

Likewise, SAP HANA adapts the concurrency of a statement based on the recent CPU utilization [PSM+15]. This means that on a lightly loaded server an analytical query can use all available CPU resources. Under heavy CPU load, i.e. when many statements are processed concurrently, the benefit of parallelization vanishes, and consequently most statements use very few threads avoiding the overhead of context switches. Considering NUMA architectures with 16 sockets or more and about 20 physical cores per socket it also turns out that it is most efficient to limit the concurrency of a statement to the number of cores per socket. This nicely complements the processing of statements on the sockets where the accessed data is allocated [PSM+15].

As a final principle for scheduling mixed workloads with SAP HANA, a statement is classified as OLTP style within the compilation phase (potentially supported by hints coming directly from the application server). For query processing, such transactional statements are prioritized in order to achieve a guaranteed response time, which is crucial for interactive ERP applications. Otherwise, OLAP queries would dominate the system and queuing effects would result in significantly delaying (short running) OLTP queries [WPM+15].

Figure 7 quantifies the results of our optimizations for NUMA-aware scheduling of mixed workloads executed on the same system as the experiment of figure 6. In this figure we plot how the throughput of a typical OLTP scenario on the x-axis versus the throughput of a complex analytic workload on the y-axis improved over time relative to the base line in SAP HANA Rev. 70.

4.2 Heavy on-the-fly aggregation

Due to limited database performance, the ERP application code maintained a huge number of redundant aggregate tables reflecting business-logic specific intermediate results like "sales by region" or even pre-aggregates ("total sales per year") [Pla14]. This application architecture allowed on the one hand to lookup aggregate values within OLTP operations but showed some significant drawbacks on the other hand:

- The application gets more complex because it needs to maintain aggregates besides implementing actual business logic.

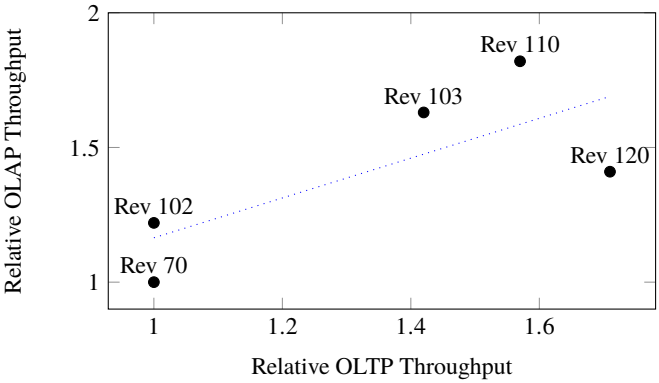

Fig. 7: Evolution of Mixed-workload performance of SAP HANA.

- Pre-aggregation is inflexible because the decision on what needs to be precomputed has to be taken upfront.
- The approach to be taken comes along with high maintenance overhead, scalability issues due to in-place updates, as well as non-negligible storage costs (the tables storing the pre-aggregates can get quite big).

The ability to run mixed workloads on top of SAP HANA allowed to significantly "slim down" application code. By replacing these redundant aggregate tables by compatible view definitions on base tables the application does not need to maintain these views during write transactions. This implies that aggregate values are now computed on-the-fly within transactional-style ERP applications [Pla14]. As a consequence, the workload changed towards a mixture of short-running lookup/update operations and long-running aggregate queries even for running only the transactional operations of an ERP system.

As an optimization, SAP HANA exploits an aggressive caching strategy to improve on the overall scalability of query processing. Using state-of-the-art view matching techniques during query compilation, the cache mechanism may provide full transparency for the application.

SAP HANA supports a static caching strategy which refreshes the cache based on a configurable retention time. This alternative is useful when accessing rather static data or for applications that can be satisfied with data that is a few minutes old. Using the static cache we observe a significantly reduced CPU consumption which ultimately leads to a higher throughput for analytical queries. Even refresh intervals of a few minutes add low overhead for refreshing the static cache. In this scenario, it is important that the application makes the age of the returned data transparent to the user.

As a second alternative SAP HANA supports a dynamic caching strategy which keeps a static base version of a view, calculates the delta with respect to the base version and merges it into the result of a transaction-consistent snapshot [BLT86]. Clearly, this alternative trades fresh data for higher CPU consumption for the query execution.

On the application level an optional declarative specification of cached views allows the application code designer to convey hints of caching opportunities to the database.

4.3 Complex Analytics on Normalized, Technical Database Schemas

The fact that OLTP transactions are executed on the same physical database schema as complex analytic queries also implies that there is no ETL step involved which brings the data into a form that is suitable for reporting purposes. Together with the removal of aggregates maintained by the application this leads to the decision to create a layered architecture of database views [MBBL15]. At the lowest level, these views map tables in the physical schema to the most basic business objects which are meaningful to applications like S/4HANA, e.g. sales orders. Higher-level views build on top of these views and represent higher-level business entities, e.g. a cost center or a quarterly sales report. Analytics performed by business applications like S/4HANA execute seemingly simple SQL queries on this stack of views. Clearly, these reporting queries do not run on star- or snowflake schemas as they are commonly used in data warehouses. Instead, complex reports work on the technical, normalized database schema with, e.g. header-lineitem structures and (optionally) references to multiple auxiliary tables, e.g. for application-specific configuration or texts. Figure 8 visualizes this layered architecture of views - also called Core Data Services (CDS) views - as nodes, and the nesting relationship and associations as edges. In a typical S/4HANA development system in 2015 there are 6,590 CDS views with 9,328 associations which sum up to 355,905 lines of code for the view definitions.

The complexity of these business-oriented database views is notable: After unfolding all referenced views, the CDS view with the highest number of referenced tables referenced 4,598 base tables. Specifically, there are many views with a high complexity, i.e. there are 161 CDS views that reference more than 100 tables. Considering these numbers it is clear that analytical queries on these complex nested views are very challenging to optimize and evaluate in an efficient way. So far, only few research papers consider queries of this complexity, e.g. [DDF+09]. One way to handle such complex queries is to apply caching strategies as described already in Section 4.2. Another challenge is the issue of supportability for such complex scenarios, e.g. how can one characterize complex views, or even analyze performance issues considering that a plan visualization of queries on such a complex view does not even fit on a very large poster? Finally, it is difficult to offer metrics to developers who define these complex nested CDS views so that they can assess the complexity of the views they model. For example, only counting the number of referenced tables might be misleading because this measure does not consider the size of the accessed tables. Other metrics like the usage of complex expressions may also be relevant for this assessment.

4.4 Data Aging

In an in memory database like SAP HANA, the data accessed by SQL statements must reside in memory. By default, full columns are loaded into main memory upon first access and only

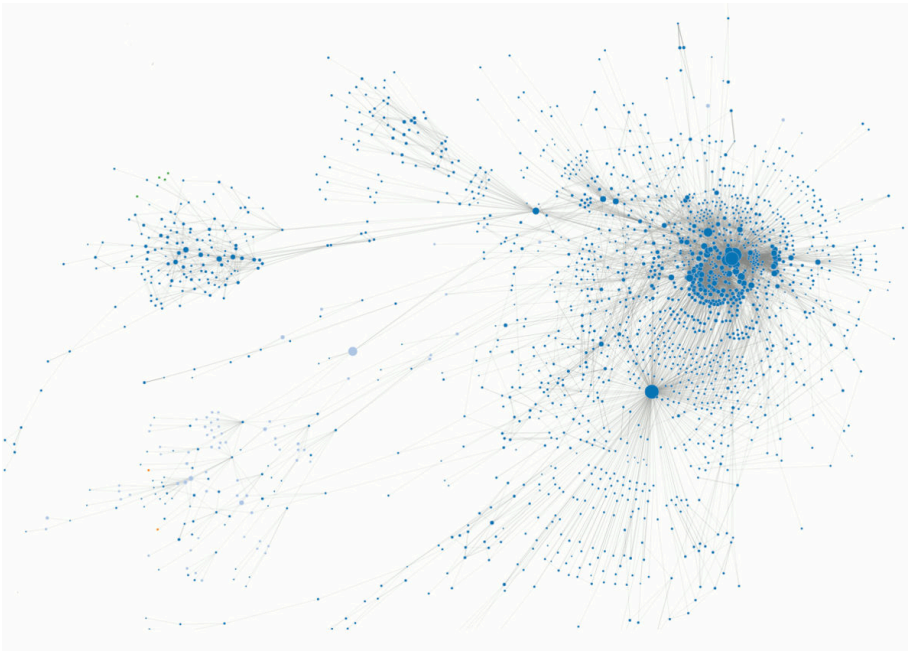

Fig. 8: Relationship of nested CDS views.

memory pressure may force the system to unload columns from memory. Because historical data needs to be kept for a long time, e.g. for legal reasons, keeping all accessed data in memory may often be considered too expensive, especially for rarely accessed, historical data. Other systems, like Hekaton or DB2, offer mechanisms to detect rarely used data and avoid loading it into main memory [CMB+10, AKL13]. However, in our experience, application knowledge that indicates which data is expected to be used rarely is required in order to exploit a maximum of memory footprint reduction while at the same time providing low-latency query response times, and avoiding an accidential placement of data on slow storage media like disk. Consequentially, as part of an *aging run*, the application marks certain data as warm or cold data, and based on these flags, SAP HANA can optimize the storage.

The easiest solution available in SAP HANA are table partitions which are declared to have page-loadable columns [SFA+16]. A more sophisticated approach relies on a completely separate storage engine using dynamic tiering [MLP+15]: Dynamic tiering exposes remote tables stored in the SAP Sybase IQ storage engine as virtual tables inside SAP HANA. This way, SAP HANA can store massive amounts of data and keep it accessible for applications via the efficient disk-based analytic query engine offered by SAP Sybase IQ. This setup has the advantage that applications are completely agnostic to the concrete physical schema design, and all relational data is accessible via one common SQL interface.

Nevertheless, access to the cold data stored on disk should be avoided during query processing. Therefore, SAP HANA relies on 1) partition pruning based on the metadata of

partitioned tables, 2) hints provided by the application that indicate that only hot data is relevant in this particular query, and 3) potentially pruning partitions at runtime based on statistics of accessed table columns and available query parameters.

4.5 Summary

In this section, we presented how SAP HANA can deal with mixed workloads of concurrent OLAP and OLTP statements. The scalability on both dimensions heavily depends on the effective use of memory and CPU resources. Furthermore, interactive analytics on complex hierarchies of views requires techniques like caching but also guidance for developers of analytic applications to analyze the impact of queries on these views. Finally, we discussed how the system supports data aging which keeps cold data stored on disk and loads this data on demand into memory with low overhead.

5 Conclusions and Outlook

In this paper we report on our journey of extending SAP HANA from a system heavily optimized for analytical query processing to a database engine that can also handle transactional workloads such that it can sustain sufficient throughput for large customer systems running SAP ERP workload. As we explain and underline with experiments, this required various specific localized but also architectural changes. As a result, we can report that it is indeed possible to handle OLTP workloads "fast enough" for the majority of workloads we encounter in large customer scenarios, while allowing them to run realtime analytics on the same operational data set.

Nevertheless, we are also convinced that this journey is not over yet. The new opportunities to handle both analytical and transactional workload efficiently without redundancy, complex ETL, or data staleness in a single database results in an increase of truly mixed workload. The demand in this direction goes well beyond what is currently possible - handling mixed workload gracefully with predicable and low response times, high system throughput, and high resource utilization is an ongoing challenge. In addition, while applications become more and more aware of the opportunities of doing analytics, planning, as well as running complex machine learning algorithms on transactional data, we expect that even more data-intensive logic needs to be handled within the database. Finally, as the data sizes grow further, using persistent storage needs to be reconsidered as an integral part for an in-memory database like SAP HANA, e.g. to age warm and cold data to NVM, SSDs or spinning disks.

In summary, the journey of SAP HANA towards a high performance as well as robust foundation of large software applications in combination with demands coming from operating SAP HANA within on-cloud, on-premise as well hybrid deployment settings is not over. Stay tuned!

6 Acknowledgements

We want to thank all members of the global SAP HANA database development team for bringing the system described in this paper from vision to reality. We are also grateful to our colleagues from the ABAP application server development team for the excellent collaboration (e.g. on the FDA and NFAE features), and for providing Figure 8.

References

[AKL13] Karolina Alexiou, Donald Kossmann, and Per-Ake Larson. Adaptive Range Filters for Cold Data: Avoiding Trips to Siberia. *PVLDB*, 6(14):1714–1725, 2013.

[BLT86] Jose A. Blakeley, Per-Ake Larson, and Frank Wm Tompa. Efficiently Updating Materialized Views. *SIGMOD Rec.*, 15(2):61–71, June 1986.

[Böh15] Alexander Böhm. Novel Optimization Techniques for Modern Database Environments. In *Datenbanksysteme für Business, Technologie und Web (BTW)*, pages 23–24, 2015.

[CMB+10] Mustafa Canim, George A Mihaila, Bishwaranjan Bhattacharjee, Kenneth A Ross, and Christian A Lang. SSD Bufferpool Extensions for Database Systems. *PVLDB*, 3(1-2):1435–1446, 2010.

[DDF+09] Nicolas Dieu, Adrian Dragusanu, Françoise Fabret, François Llirbat, and Eric Simon. 1,000 Tables Under the Form. *PVLDB*, 2(2):1450–1461, August 2009.

[FML+12] Franz Färber, Norman May, Wolfgang Lehner, Philipp Große, Ingo Müller, Hannes Rauhe, and Jonathan Dees. The SAP HANA Database – An Architecture Overview. *IEEE Data Eng. Bull.*, 35(1):28–33, 2012.

[IGN+12] Stratos Idreos, Fabian Groffen, Niels Nes, Stefan Manegold, Sjoerd Mullender, and Martin Kersten. MonetDB: Two decades of research in column-oriented database architectures. *IEEE Data Eng. Bull.*, 35(1):40–45, 2012.

[KDR+14] Tomas Karnagel, Roman Dementiev, Ravi Rajwar, Konrad Lai, Thomas Legler, Benjamin Schlegel, and Wolfgang Lehner. Improving in-memory database index performance with Intel® Transactional Synchronization Extensions. In *20th IEEE International Symposium on High Performance Computer Architecture, HPCA 2014*, pages 476–487, 2014.

[KN11] Alfons Kemper and Thomas Neumann. HyPer: A hybrid OLTP&OLAP main memory database system based on virtual memory snapshots. In *Proc. IEEE ICDE*, pages 195–206, 2011.

[LCC+15] Tirthankar Lahiri, Shasank Chavan, Maria Colgan, Dinesh Das, Amit Ganesh, Mike Gleeson, Sanket Hase, Allison Holloway, Jesse Kamp, Teck-Hua Lee, et al. Oracle Database In-Memory: A Dual Format In-Memory Database. In *Proc. IEEE ICDE*, pages 1253–1258, 2015.

[LCF+13] Per-Ake Larson, Cipri Clinciu, Campbell Fraser, Eric N. Hanson, Mostafa Mokhtar, Michal Nowakiewicz, Vassilis Papadimos, Susan L. Price, Srikumar Rangarajan, Remus Rusanu, and Mayukh Saubhasik. Enhancements to SQL server column stores. In *Proc. ACM SIGMOD*, pages 1159–1168, 2013.

[Liu11] Huan Liu. A Measurement Study of Server Utilization in Public Clouds. In *Proc. IEEE Int. Conf. on Dependable, Autonomic and Secure Computing*, pages 435–442, 2011.

[LSKN16] Viktor Leis, Florian Scheibner, Alfons Kemper, and Thomas Neumann. The ART of Practical Synchronization. In *Proceedings of the 12th International Workshop on Data Management on New Hardware*, DaMoN '16, pages 3:1–3:8, 2016.

[LSP+16] Juchang Lee, Hyungyu Shin, Chang Gyoo Park, Seongyun Ko, Jaeyun Noh, Yongjae Chuh, Wolfgang Stephan, and Wook-Shin Han. Hybrid Garbage Collection for Multi-Version Concurrency Control in SAP HANA. In *ACM SIGMOD*, pages 1307–1318, 2016.

[MBBL15] Norman May, Alexander Böhm, Meinolf Block, and Wolfgang Lehner. Managed Query Processing within the SAP HANA Database Platform. *Datenbank-Spektrum*, 15(2):141–152, 2015.

[McK04] Paul E. McKenney. *Exploiting Deferred Destruction: An Analysis of Read-Copy-Update Techniques in Operating System Kernels*. PhD thesis, OGI School of Science and Engineering at Oregon Health and Sciences University, 2004.

[MLP+15] Norman May, Wolfgang Lehner, Shahul Hameed P., Nitesh Maheshwari, Carsten Müller, Sudipto Chowdhuri, and Anil K. Goel. SAP HANA - From Relational OLAP Database to Big Data Infrastructure. In *Proc. EDBT*, pages 581–592, 2015.

[MSL+15] Ingo Müller, Peter Sanders, Arnaud Lacurie, Wolfgang Lehner, and Franz Färber. Cache-Efficient Aggregation: Hashing Is Sorting. In *ACM SIGMOD*, pages 1123–1136, 2015.

[Pla09] Hasso Plattner. A Common Database Approach for OLTP and OLAP Using an In-Memory Column Database. In *Proc. ACM SIGMOD*, pages 1–2, 2009.

[Pla14] Hasso Plattner. The Impact of Columnar In-memory Databases on Enterprise Systems: Implications of Eliminating Transaction-maintained Aggregates. *PVLDB*, 7(13):1722–1729, August 2014.

[PSM+15] Iraklis Psaroudakis, Tobias Scheuer, Norman May, Abdelkader Sellami, and Anastasia Ailamaki. Scaling Up Concurrent Main-Memory Column-Store Scans: Towards Adaptive NUMA-aware Data and Task Placement. *PVLDB*, 8(12):1442–1453, 2015.

[RAB+13] Vijayshankar Raman, Gopi Attaluri, Ronald Barber, Naresh Chainani, David Kalmuk, Vincent KulandaiSamy, Jens Leenstra, Sam Lightstone, Shaorong Liu, Guy M. Lohman, Tim Malkemus, Rene Mueller, Ippokratis Pandis, Berni Schiefer, David Sharpe, Richard Sidle, Adam Storm, and Liping Zhang. DB2 with BLU Acceleration: So much more than just a column store. *PVLDB*, 6(11):1080–1091, 2013.

[SC05] Michael Stonebraker and Ugur Cetintemel. "One size fits all": an idea whose time has come and gone. In *IEEE ICDE*, pages 2–11, 2005.

[SFA+16] Reza Sherkat, Colin Florendo, Mihnea Andrei, Anil K. Goel, Anisoara Nica, Peter Bumbulis, Ivan Schreter, Günter Radestock, Christian Bensberg, Daniel Booss, and Heiko Gerwens. Page As You Go: Piecewise Columnar Access In SAP HANA. In *ACM SIGMOD*, pages 1295–1306, 2016.

[TMBS12] Frederik Transier, Christian Mathis, Nico Bohnsack, and Kai Stammerjohann. Aggregation in parallel computation environments with shared memory, 2012. US Patent App. 12/978,194.

[WPM+15] Florian Wolf, Iraklis Psaroudakis, Norman May, Anastasia Ailamaki, and Kai-Uwe Sattler. Extending database task schedulers for multi-threaded application code. In *Proc. SSDBM*, pages 25:1–25:12, 2015.