Franz Färber, Jonathan Dees, Martin Weidner, Stefan Bäuerle, Wolfgang Lehner

Towards a web-scale data management ecosystem demonstrated by SAP HANA

Erstveröffentlichung in / First published in:

2015 IEEE 31st International Conference on Data Engineering. Seoul, 13.-17.04.2015. IEEE, S. 1259-1267. ISBN 978-1-4799-7964-6.

DOI: http://dx.doi.org/10.1109/ICDE.2015.7113374

Final edited form was published in "2015 IEEE 31st International Conference on Data Engineering. Seoul 2015", S. 1259-1267, ISBN 978-1-4799-7964-6
http://dx.doi.org/10.1109/ICDE.2015.7113374

Towards a Web-scale Data Management Ecosystem Demonstrated by SAP HANA

Franz Faerber [*1], Jonathan Dees [*2], Martin Weidner [*3], Stefan Baeuerle [*4], Wolfgang Lehner [#5]

[*] *SAP SE, Walldorf, Germany*
{[1]franz.faerber, [2]jonathan.dees, [3]martin.weidner, [4]stefan.baeuerle}@sap.com

[#] *TU Dresden, Dresden, Germany*
[5] wolfgang.lehner@tu-dresden.de

***Abstract*—Over the years, data management has diversified and moved into multiple directions, mainly caused by a significant growth in the application space with different usage patterns, a massive change in the underlying hardware characteristics, and–last but not least–growing data volumes to be processed. A solution matching these constraints has to cope with a multidimensional problem space including techniques dealing with a large number of domain-specific data types, data and consistency models, deployment scenarios, and processing, storage, and communication infrastructures on a hardware level. Specialized database engines are available and are positioned in the market optimizing a particular dimension on the one hand while relaxing other aspects (e.g. web-scale deployment with relaxed consistency).**

Today it is common sense, that there is no single engine which can handle all the different dimensions equally well and therefore we have very good reasons to tackle this problem and optimize the dimensions with specialized approaches in a first step. However, we argue for a second step (reflecting in our opinion on the even harder problem) of a deep integration of individual engines into a single coherent and consistent data management ecosystem providing not only shared components but also a common understanding of the overall business semantics. More specifically, a data management ecosystem provides common "infrastructure" for software and data life cycle management, backup/recovery, replication and high availability, accounting and monitoring, and many other operational topics, where administrators and users expect a harmonized experience. More importantly from an application perspective however, customer experience teaches us to provide a consistent business view across all different components and the ability to seamlessly combine different capabilities. For example, within recent customer-based Internet of Things scenarios, a huge potential exists in combining graph-processing functionality with temporal and geospatial information and keywords extracted from high-throughput twitter streams. Using SAP HANA as the running example, we want to demonstrate what moving a set of individual engines and infra-structural components towards a holistic but also flexible data management ecosystem could look like. Although there are some solutions for some problems already visible on the horizon, we encourage the database research community in general to focus more on the Big Picture providing a holistic/integrated approach to efficiently deal with different types of data, with different access methods, and different consistency requirements–research in this field would push the envelope far beyond the traditional notion of data management.

I. Introduction

The last years of database research and development have been dominated by some major trends, that shifted the focus of the academic community as well as the attention of economic actors and stakeholders of the monetary market onto data management. Topics like Main Memory Processing, Big Data analytics, No-SQL, and Internet of Things (IoT)–just to name the most prominent representatives–sparked a huge number of activities in an area which was dominated by the traditional database paradigms (relational data model with ACID consistency) for decades. A huge variety of applications triggered re-thinking all aspects of the data management problem. For example, modern applications require specialized support for:

- different data types: This subsumes all variations from structured via semi- to un-structured data sets with application specific support for image and video processing as well as comprehensive methods for efficient handling of temporal and geospatial data.
- different consumption patterns: This requirement ranges from traditional OLTP to massive OLAP queries over large data sets, more and more as part of iterations with complex data analysis tasks.
- different data models: A multitude of data models as alternatives to the classical relational model has been established by the applications; examples comprise wide-table, graph-centric, or document-centric models but go as far as tensor models for scientific applications.
- different and application-controlled notions of consistency: Again, the full spectrum from ACID to almost zero DB-related consistency support is required from a modern application portfolio.
- different application and query language: NoSQL-query languages allow to express complex analytical tasks more naturally; Also, more and more domain-specific languages (DSLs) are required to be directly supported by the data management layer.
- different levels of scaling: The choice of deployment options ranges from large scale-up scenarios via modest standard IT center compatible scaling to massive scale-out scenarios in hosting environments.
- different hardware capabilities: HW trends have a tremendous impact on storage options (main memory-, flash-, disk-, or even non-volatile RAM-based) as well as compute infrastructure (CPU, GPU, FPGAs, ..)

Final edited form was published in "2015 IEEE 31st International Conference on Data Engineering. Seoul 2015", S. 1259-1267, ISBN 978-1-4799-7964-6
http://dx.doi.org/10.1109/ICDE.2015.7113374

and high-speed interconnects with opportunities like remote DMA.

A. Reactions from the DB Community

With this new interest and the possibility to provide business value for customers (lower TCO as well as increase of application opportunities), a plethora of new database engines has evolved striving to tackle some of the arising challenges. Especially "NoSQL" approaches try to question the traditional design decisions in multiple directions, for example by rethinking traditional consistency models, by favoring extreme scale out mechanisms, and providing non-relational data models like JSON or pure key-value stores. Representatives like MongoDB, CloudDB, or Cassandra are already quite popular and are attracting more and more attention of application developers looking for flexibility beyond the traditional relational model. Coming from the other end of a priori schemaless dataset, graph engines distinguish themselves mainly in storing individual data items (entities with potentially individual entity types) as nodes in graph structures with edges to represent individual relationships between the entities. With respect to consistency and scale out, current graph engines like Neo4j, Allegrograph, etc. behave like standard relational systems.

Recently, many projects in the Hadoop ecosystem like Spark/Splash, Stinger, HBase, Hadapt, Hive and many others received a lot of attention. These engines either share the persistency model (HDFS), the runtime environment (MapReduce/TEZ), the resource management services (Yarn) or combinations of it. As these systems can partially cooperate and exchange data on a technical level, common query execution and semantic operations have to be provided by the application for example by providing scenario-specific MapReduce jobs. Although a common technical infrastructure exists, a common query processing infrastructure (with local as well as global components) allowing a seamless integration of languages beyond SQL, a common repository for higher-level business concepts (again with local derivatives), or even a common lifecycle management infrastructure allowing to consistently roll individual software components (application as well as system software) forward to new releases is missing.

B. SAP HANA Ecosystem

With SAP HANA we take the first steps towards our goal of an integrated as well as dynamically deployable data management ecosystem. We strive to combine different engines, data models, and data processing paradigms and deliver one solution for the application which logically consists of one execution runtime, one persistency, one infrastructure and one administration experience. We carefully separate between a physical storage model level and a logical data model level. For the storage model we rely on a flexible interpretation of column groups as a foundation for different logical data models like the standard relational model, graph-like or tensor-based models potentially coming along with individual languages which are mapped to a common SQL-like internal query language for the storage layer. While most of the engines deployed within the SAP HANA ecosystem are based on the main-memory centric column store of SAP HANA and implicitly take advantage of its superior performance characteristics, the ecosystem also comprises external systems like "R" or a native HDFS/YARN-integration using SAP HANA.

Figure 1 illustrates the impact of novel application domains like IoT in addition to traditional OLTP/OLAP-style data processing. While SAP HANA is optimized to run OLTP and OLAP workloads in extreme speed, data sets coming in from sensors and other devices may initially be loaded into Hadoop-based environments and further propagated within a data refinement process into the In-Memory structures. In the opposite, transactional data may age and–guided by business rules–moved to extended storage and potentially into HDFS-based systems.

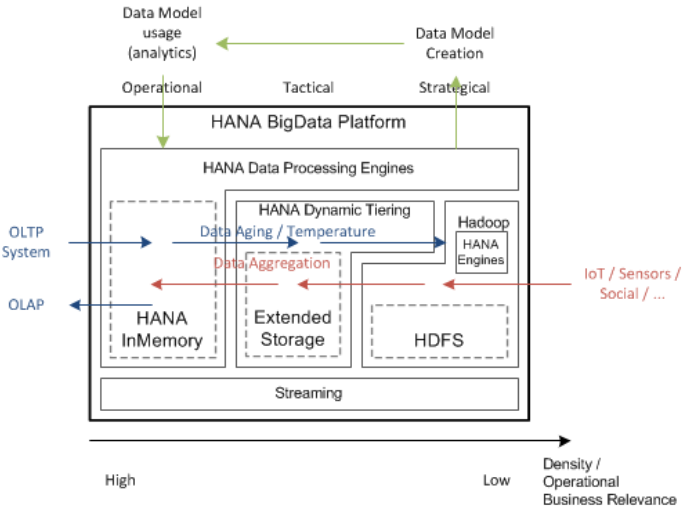

Figure 1. Positioning of the SAP HANA Data Management Ecosystem

C. Outline and Goal

In the following we will first give a summary of the state of SAP HANA with respect to functional extensions and integration of application context. We will then provide a detailed outlook on the upcoming massive scale-out extension, SAP HANA SOE (**S**cale-**O**ut **E**xtension, codename "Velocity Engine") as another specialized engine within the SAP HANA family. We will also outline the integration of Hadoop into the HANA ecosystem. The paper will close with an in-depth look into different real world scenarios, where an efficient combination of different engines is crucial for the customer use case.

The overall goal of the paper is to convey the message that a modern data management system has to tackle at least three aspects to be successful with respect to modern data management challenges:

- providing a variety of specialized data structures and embedded algorithms: Within SAP HANA, we provide a large number of non-traditional data types with highly-tuned algorithms. The portfolio ranges from semantic text analytics to scientific computations on matrices as outlined in Section II.
- listening to the application: The data management layer can extremely benefit from application knowledge to improve performance significantly. We will provide some examples in Section III.

Final edited form was published in "2015 IEEE 31st International Conference on Data Engineering. Seoul 2015", S. 1259-1267, ISBN 978-1-4799-7964-6
http://dx.doi.org/10.1109/ICDE.2015.7113374

- orchestrating highly specialized data engines running in different environments ranging from edge cloud devices (e.g. mobile devices, base stations of cellular network to large Hadoop-based installations), see Section III.

II. SAP HANA: Beyond relational Data Processing

As motivated, a modern web-scale data management ecosystem is based on two pillars. First, the ecosystem has to provide a rich set of functionality and a tight integration of application knowledge–the pure relational model with plain SQL as the query language does no longer satisfy the huge variety of different use cases and resulting requirements. Second, the ecosystem requires a high-performing as well as scalable implementation of this functionality in order to be useful for Big Data scenarios. Before diving into some architectural details of the SAP HANA SOE, we list specific functional extensions already existing in the SAP HANA system as the foundation for an efficient software stack for business applications.

In 2010 SAP announced the in memory column store as the foundation for SAP's data management and future application development. SAP HANA is a fully ACID compliant relational database system with all the state of the art capabilities like backup, recovery, and HA mechanisms [1], [2]. Figure 2 shows some core components of the HANA system. As it can be seen, the system comprises multiple data processing engines to refine data streams coming from different "raw" execution engines. The section will highlight some of the non-standard processing engines like text, graph, or capabilities for processing time series data efficiently. Figure 2 also gives an impression of the current execution engines ranging from the main-memory based engines via the extended storage engine (based on IQ technology) to a comprehensive federation framework (SDA = smart data access) in order to reach out to a huge variety of external data sources.

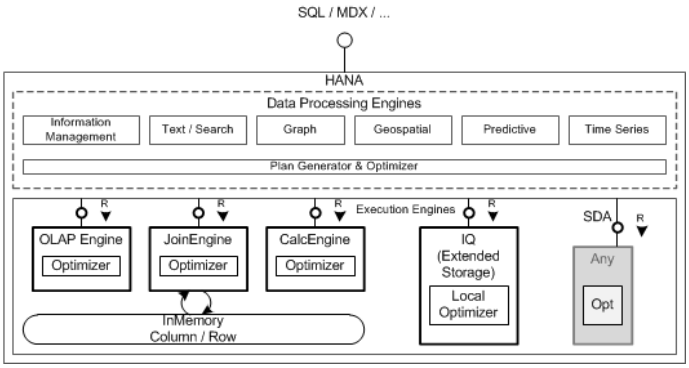

Figure 2. SAP HANA System

A. Main Memory Column Store: Bringing OLAP and OLTP together

By loading data completely into main memory, applying multiple compression techniques and optimizing for multi-core architectures, the HANA system achieves data access and performance characteristics, which allows to recombine OLTP and OLAP workloads into one single system. Contrary to prevailing methods, the main memory column store is also used for heavy transactional load, which has proven to be by far fast enough in thousands of customer installations. Hasso Plattner and others have already outlined in [3] that read operations massively dominates write operations in enterprise business systems which is becoming even more evident if redundant system tables are removed. The combination of both workloads in one system allows to avoid the expensive replication costs between OLTP and OLAP systems and provides access for all analytic questions in real time.

B. Data Mining: Next Level of Analytics

We embedded some critical data mining features directly into the column store engine. Examples are distributed basket analysis and a variety of forecasting algorithms. In addition, access to the system R and other data mining providers like SAS are applicable directly out of the system in order to tap the sheer amount of algorithms and methods already existing in the community. Access to R is implemented as a special operator into the internal data flow graph of the database engine allowing the optimizer to embrace the call to the external system. More functionality is integrated as stored procedure calls into the native SQL model; we therefore can combine it seamlessly with any kind of other SQL access.

C. Text Engine: Bringing structured and unstructured Data Together

Analyzing textual data has many facets in a data management solution. First, we think of simple text search which we all know from web search engines. In order to support customers all over the globe, many languages have to be supported natively with functionality like stemming, part of speech tagging, and others. On top of it, sophisticated text mining and analytical capabilities like text classification, clustering, sentiment analysis, and other text mining operations can be supported. In addition, we are able to extract entities (like names, addresses, companies, ...) and sentiments from documents with a rule based approach on top of the natural language functionality. The extracted entities and sentiments can be stored as structured data. Since text processing is deeply integrated into the HANA engine, results of text analytics can now be combined with structured data already stored in the relational part of the database. This integration bridges the gap between different methods of analyzing structured and unstructured information. The text analysis and feature extraction process is triggered automatically when new or changed documents are brought into the data management system.

D. Planning Extensions: Successful and nevertheless overlooked

Planning functionality and operators are heavily overlooked in the research community. Planning applications for sales planning, financial planning or operations planning are very attractive for customers as–for example–the commercial success of companies like Anaplan[1] or others show. Planning, in this definition, is the process of defining and aligning the sales,

[1]https://www.anaplan.com/

Final edited form was published in "2015 IEEE 31st International Conference on Data Engineering. Seoul 2015", S. 1259-1267, ISBN 978-1-4799-7964-6
http://dx.doi.org/10.1109/ICDE.2015.7113374

financial or other numbers for the foreseeable future (normally for a 1 year period). The planning process requires heavy CPU based database functionality like disaggregation or copy processes, providing logical snapshots or versioning and other operators. We have integrated these operators directly into the relational engine and access them with SQL extensions.

E. Graph/Hierarchy Engine: The embedded Graph Story

Many business applications like transport and logistics, supply chain traceability, business networks, fraud detection, and others work inherently with graph structures and benefit directly from explicit graph representations inside the data management layer. As Paradies et al. have explained in [4], explicit graph structures help applications to express complex business logic more explicitly and execute the operations more effectively. As Finis et al. have discussed in [5] hierarchies as a special kind of a graph are used in almost all kinds of business applications. Special support for time dependent and versioned hierarchies is therefore a crucial functionality in the database layer for business applications. As the graphs and hierarchies are not isolated objects in the application context, graph data has to be combined during modeling but especially during query execution with other types of data, especially the structured relational data, but also geospatial and other data types. Therefore we have decided to embed a graph engine into the HANA data management leveraging the main memory column store. This allows to interpret data in columns (structured relational data) as graph or hierarchy structures by defining hierarchy or graph views on top of the relational data. Within these graph structures, state of the art graph processing functionality (like distance, siblings, shortest path, and others) is provided. We are currently defining extensions to SQL in order to integrate most of the operations into the standard processing model as well as providing a domain-specific language to fully exploit the graph data model without the constraints imposed by the relational representation.

F. GIS, Time Series: Bringing more semantics to the data

Geospatial and time series are supported as native data types inside the relational engine. They are built with mechanisms of the main memory column store and implement powerful compression mechanisms, which is especially useful for sensor data. As operations on both data types provide results in form of tuples, implementing them as data types deep in the engine is a natural fit. We extended the SQL syntax in order to allow the definition of points or polygons, and to support query operators like `WithinDistance`, `Contains` or `Area`. Combination with relational data is obvious in areas like geo-location based analytics, with queries like, e.g., “get the areas in US with the most revenue” or “get all customers within a distance of 10 kilometer having payments due”. But also a combination with graph data for example in routing algorithms is a common use case. Time series data types are mainly interesting for the financial sector or Internet of Things for storing sensor data. Besides providing large compression factors, they provide functionality like resolution adoption, comparison functions, correlation, transformations, and others. Again, integration into the relational world is the key, because the queries always combine the selection of elected sensor data with time series functionality.

G. Scientific Engine: Use the full table scan speed for heavy computation

In [6], Kernert et al. show the significant advantage of bringing linear algebra operations like eigenvalue calculation on large matrices into a main memory column store. Beside the scientific area, many applications in the financial area and business warehouse optimization benefit from having these kinds of functionality inside the data management layer. Since decision-support data is persisted and kept consistently within the data management ecosystem, no redundant copying from other data sources to external libraries is needed, reducing the pain to control consistency and define and monitor data movement processes. Furthermore, the corresponding metadata of data sets can be updated synchronously and consistently with the numerical data. Even more interestingly for database management research, large matrices are no static objects in common analytic workflows. As they are manipulated in an iterative process, the data manipulation capabilities of a database will meet the analytical demands better than the tedious maintaining of multiple data files and therefore significantly improve the efficiency of the domain expert by relieving it from organizing large file repositories.

H. NoSQL Extension: Bringing Flexibility into the Relational World

The “NoSQL world” is often defined by three main criteria: flexible data structures like the document model or key-value stores, massive and easy scale out and the freedom and option to relax consistency. All three topics should also be integrated into a standard data management system. In HANA, so called “flexible tables” can be defined where column definition is not a DDL but implicitly triggered via a DML operation (similar to the document or key-value store paradigm). A flexible table can be queried via SQL as it would be the case if they were standard columns. Technically, metadata about unknown columns are automatically created as soon as records with values for new columns are inserted. The number of columns is not limited (technically there is a 32 bit limit to express the number of columns). Internal compression methods can handle also very sparse columns to achieve compression rates. In addition, we are integrating a document based data model (similar to JSON format) into HANA by introducing a data type “document” for a column of a relational table. The content (the document) is structured in an arbitrary JSON format. The documents themselves are queried by an XQuery like language which is embedded into the SQL statement. The outcome of a “document” query is a set of rows of the table which contains the document as a cell. Obviously there are multiple options, how to use this model. For example, users can just store their JSON documents with the document key as standard columns in the “parent table”. Additional join keys can be stored in the parent table as well. Another option is to model a document structure on top of relational tables for even faster access, for example, in a header–item–subitem structure we can assume a 1:N cardinality between header-item and item-subitem. If we additionally can ensure from an application perspective that corresponding table entries are always inserted/updated/deleted together and retrieval requests are mainly reading the complete ”object” (which reflects the standard procedure for many objects in business software like SAP ERP), such an “object” could be stored in a JSON document as a kind of materialized

Final edited form was published in "2015 IEEE 31st International Conference on Data Engineering. Seoul 2015", S. 1259-1267, ISBN 978-1-4799-7964-6
http://dx.doi.org/10.1109/ICDE.2015.7113374

index on top of the relational data. It will be modeled as special kind of a join index in the database and transparently exploited by the retrieval process. With such non-standard mechanisms, even existing applications and data models can benefit from the document storage.

I. Scale Out: Reconciling the Hadoop and the Relational World

In recent years there was a clear trend for scale out using clusters of commodity hardware especially for Hadoop infrastructures. But also in other analytic workload environments, the trend towards scale out versus scale up by adding more resources to a single server is obvious. Very recent research [7] claims, that the majority of the data volumes we see at customers could be handled with one or just a few bigger servers, which provides benefits in performance TCO and server density. Until now we focused with SAP HANA mainly on use cases with up to a few hundred Terabytes of data which can be handled by a modest number of high end commodity servers. We suggest even to our main customers a model where hot data will be stored on one single database server and colder data is distributed on a set of cheaper nodes. With the Big Data trend especially in connection with the massive amount of data generated from sensors, we see a high demand of storing and analyzing data in the high Petabyte range. In addition to storing large data sets collected from some devices, the need arises to integrate such data sets with existing databases like ERP data, master data, logistic data and to run algorithms on top of such integrated data sets. For example a producer of soap for washrooms wants to start a route planning for their servicer team to fill the dispensers, once sensors have detected, that some are almost empty. They even want to fill them earlier, if they have notice that a major event will be held in locations where they have installed dispensers. In addition, the company wants to direct the service team into the correct direction. For those requirements a combination of logistic, master data and planning data is the key. Therefore, we are currently extending the SAP HANA system with a massive scale out option and a deep integration with the Hadoop system. We do not argue to bring our standard applications to a massive scale out data management system. We still believe in the advantages of a small setup. But with the new requirements in different areas and the need of deep integration, we argue that we need both deeply integrated. In Section IV we describe the low footprint single processing units, the Scale-out infrastructure, and the Hadoop integration of our solution.

III. Bridging the Gap between Applications and Data Management

With more than 100.000 customers for its different business solutions (ERP, Analytics, HCM, networks, and others) SAP has gained a lot of experience in application development. Almost all applications are built on top of a relational database engine. A huge variety of tools, libraries, and services have been created in the application layer in order to support the efficiency of application developers. Many of these tools reuse components logically belonging to the data layer, but are implemented in the application layer in order to achieve independence from the underlying database. The problem of this approach manifests in conducting many data centric calculations within the application layer and not within the database layer as close as possible to the original data sets. This implies that a significant amount of data has to be transferred from the database into the sphere of the application, which usually comes with severe performance impact. The two following examples demonstrate this effect: if currency conversion[2] is implemented on application level, analytic queries have include the currency field in the "group by" list in order to retrieve the currency information. Depending on the currency distribution, this can multiply the data to be transferred between the layers. An even harder problems consists in dealing with hierarchical data structures. Since many database systems are not providing core support for hierarchies, complex hierarchy resolution algorithms are implemented at application layers. For counting the transitive child nodes of a given node for example, the whole subtree of the respective hierarchy has to be moved from the database to the application. With appropriate hierarchy functionality in the database system, only the number of nodes needs to be communicated to the calling application. Further examples among others are unit conversion, manufacturing calendar support, financial planning and optimizations in combination with comprehensive simulations. With HANA we started to systematically push functionality down into the database and build business application specific libraries/extensions in the DB layer with significant positive impact for the runtime. We argue, that these kinds of extensions are needed for other application types as well.

Other optimizations of the application/database interface are not adding novel functionality but allow better use of application knowledge in the database layer. One quite simple example is the maintenance of dictionaries of table columns. Domain or dictionary encoding is a common technique to compress the columns: For a string column, all distinct values are inserted into a sorted dictionary and the column itself just stores the references to the dictionary. One of the major challenges in this context is to efficiently keep the dictionary sorted in the case of inserts and deletes. In HANA this is guaranteed by having a buffer structure called delta store which records all changes. A merge phase incorporates these data sets into the main column. In order to maintain the sorting of the dictionary within this merge process, the dictionary must potentially be resorted which forces the references within the main columns to be updated accordingly [8]. On the application level, large tables with obviously high-cardinality key columns are created from a vast amount of transactional data, which is mainly created outside of the system. Very often, the keys are generated by concatenating some information from application context plus an incremental counter to achieve uniqueness. By knowing the mechanism of how the keys are generated, the dictionary maintenance and merging can be done much simpler and more efficiently. Incorporating application knowledge, a stable sort order without resorting can be achieved in some situations, improving the merge process.

An even more important integration topic is data aging: There are multiple ways of implementing data aging, which are purely based on database statistics. By letting the application define the aging rules and storing them in the metadata of the database, the aging mechanism acquires a semantic meaning which allows for much better partition pruning than any

[2]Currency conversion is a highly complex business process and consists of 100s of lines of code.

Final edited form was published in "2015 IEEE 31st International Conference on Data Engineering. Seoul 2015", S. 1259-1267, ISBN 978-1-4799-7964-6
http://dx.doi.org/10.1109/ICDE.2015.7113374

approach purely based on access statistics. Statistical methods can be used to propose new application rules to the application developer or system administrator. Consider an example with orders and invoices, two independent objects in a business application. The sales order aging rule could be defined as "age a sales order, if it is closed and closing date is older than 3 months and the sales order is not from this year". For invoices, the aging rule could be defined as "if the invoice is paid for at least 3 months and the invoice is not from this year". When asking for all open invoices of the last 3 months, it is clear, that the older (already aged) partitions of the invoice object can be pruned. A more interesting case is, if somebody wants to see all open orders and the corresponding invoices–a standard query in this context. Given the independent aging rules of the objects, the join has to be calculated between the "non aged" orders and the complete invoice table. When analyzing the aging rules, the query could be split into two parts, one specifies that all open sales orders from the current year have to be joined with the "non aged" invoice partition, the second one requests that open orders from the previous years must be joined with the complete invoice object table. Assuming that there are hopefully no open orders from the previous year, that optimization could be beneficial. We could extend the given aging rules by "... and if the corresponding sales order is in the aging set". This rule would enforce that an invoice can only be aged, if the corresponding sales order is also aged. Applying this additional rule would mean that the join can be executed only on the "non aged" partitions. Logically, a dependency graph for aging is created. There must be rules and checks, which ensure that despite of the dependencies still many objects can be aged, and there is no cycle in the dependency graph.

IV. SAP HANA Scale-out option

In addition to the core HANA system, we outline the SAP HANA Scale-Out Extension. The core idea of SAP HANA SOE is to extend the "big" Scale-up oriented SAP HANA system with massive scale-out capabilities. In this section, we briefly sketch the general approach, the scale-out architecture, and integration into Hadoop-based data management infrastructures.

A. Goals and Architectural Principles

Over the next few years, we will see hardware improvements which will massively change the way database systems are built and used. Hardware transactional memory, which got and will get global availability with Intel's integration into its Haswell systems, helps to develop scalable algorithms and data structures. In particular, Neumann et al. [9] have shown that transactional systems can significantly benefit on executing global database transactions by splitting them into multiple hardware transactions and getting rid of explicit locks. With the upcoming non-volatile memory there are many new options to optimize the design of data management systems. Oukid et al. showed in [10] how recovery of a database can be accelerated by a careful design of the underlying data structures and an optimized redo/undo log design. With new network fabrics we expect a significant improvement of network latency and bandwidth, which allows a new software design of scale-out systems. Another trend is the move to mobile devices with new energy efficient processors, which build an immense network of calculation power. This network will get even larger with the billions of sensors and collectors which are to be expected in the near future. All these changes create new opportunities and challenges for data management systems, which are of massive scale, have small and flexible single servers, provide energy efficient execution, having a main memory based design and the deep integration with a backend system which provides data and services of already existing and consolidated data.

In this context we developed an extension to HANA with low footprint , extreme performance and designed for high scalability (in the range of thousands of nodes). The SAP HANA SOE sticks to the HANA storage model paradigm which is based on a main memory column store. Some of the compression requirements are relaxed to allow more energy efficient calculation. This is for example true for compressing the references and for resorting the tables during merge. In addition, during runtime the engine compiles the SQL statement into C code and translates it into an executable binary format. As Dees and Sanders described in [11] there are significant performance advantages with this approach. The compiler framework LLVM with Clang does the compilation from C code into native code. A similar approach is followed in [12], however, we generate C code instead of LLVM byte code in order to support more sophisticated maintenance and debugging functionalities. On the operator level we use many algorithms and features which are already optimized in the HANA backend engine. In the current version the engine is heavily optimized for read requests with an intensive use of join indexes. Additionally, we generate plans for a distributed landscape. These plans can lead to strong speedup results compared to single machine execution as shown in [13] if the plans are specifically tailored for a clustered execution in combination with efficient communication algorithms. A first version of SAP HANA SOE was delivered to customers at the end of 2014 as part of SAP's analytic solution (Lumira Desktop and Lumira Teamserver). As a next step, the engine will obtain the capabilities described above, like GIS, text search, graph support, etc., which allows to use these functionalities in an extreme scale-out landscape.

B. System Components and Relationships

Our distributed transaction mechanism favors availability at the expense of consistency (e.g. CAP theorem [14]). As already outlined, SAP HANA SOE is composed of a set of services, packed into executables, and deployed on a cluster of nodes. Figure 3 shows the different system components as well as their relationship in some detail. At the core is the SAP HANA SOE local query processing executable (`v2lqp`) which contains a query and a data service. The query service is a wrapper around the core engine and operates on horizontal table partitions which are created during data import. These prepackaged partitions allow for a fast distribution of the data when scaling out or for data recovery. A data service takes care of retrieving and storing the horizontal table partitions.

The execution of distributed queries is controlled by a distributed query coordinator service (`v2dqp`) which translates each query to a directed acyclic graph of tasks. The tasks are being sent to the query service instances where they are compiled and executed. A transaction broker service executes,

Final edited form was published in "2015 IEEE 31st International Conference on Data Engineering. Seoul 2015", S. 1259-1267, ISBN 978-1-4799-7964-6
http://dx.doi.org/10.1109/ICDE.2015.7113374

serializes, and persists transactions to a distributed shared log (`v2transact`). Similar to the Corfu approach described in [15], the log stores all changes in a transactional consistent way. We introduced several changes and optimizations to the original proposal including information on how we distribute the log across multiple locations and how we update the different database nodes.

The distributed log is initially designed to work on top of Non-Volatile Memory, but multiple implementation variants will be provided (also on top of HDFS). With the distributed log approach we decouple the transaction mechanism from the query processing which allows easier scale and provides a clear system design. Updates in the log are incrementally communicated to the different data services. If the resulting isolation level is not sufficient for a given query, the distributed query coordinator service can ask the transaction broker service directly for additional updates to be considered. Therefore we are able to achieve different transactional behaviors by distinguishing two types of database nodes. On the one hand, an OLAP node updates itself in a transactionally consistent way but not necessarily synchronously to the update request coming from the application. The updates can either be incorporated by regularly polling the log or by retrieving the latest snapshot of the data hosted by a particular node. On the other hand, OLTP nodes allow real time transactional update of the data by incorporating the log during the update transaction.

We use different MVCC implementations to optimize multiple workloads. A coordinator service accepts queries from a user or application and coordinates distributed query processing. The cluster manager provides cluster services like statistics, monitoring and orchestration. The discovery service keeps track of all available components in the landscape. All these services (including global transaction blocker and database services) can be isolated by a container infrastructure like Docker[3]. As part of the HANA system the scale-out extension supports multi-level horizontal partitioning (range and hash), with the capability to handle huge of amount of partitions. High availability is achieved by supporting multiple replicas with the log replication mechanism described above. By defining OLAP or OLTP database nodes, different levels of consistency are supported.

The overall supervision and configuration of the cluster is done by a cluster management service (`v2clustermgr`). This service can dynamically start and stop other query processing services as well as orchestrate data movement. It can access statistical information about the current cluster usage in order to identify hotspots or to monitor performance goals. An authorization and a cluster discovery service are bundled together (`v2disc&auth`) to store cluster access rights and keep track of availability of services across the cluster. A catalog service stores and provides schema and metadata information, a data discovery service keeps track of the location of the corresponding horizontal table partitions (`v2catalog`).

C. Hadoop Integration

With the growing importance of the Hadoop ecosystem, databases and other engines have to provide versions which will run natively inside the Hadoop ecosystem [16]. With HANA backend/scale-out combination we are currently building a deep Hadoop integration, which allows to read Hadoop data with standard HANA SQL and Hadoop mechanisms and combine them with data in HANA like SAP ERP data. The most simple way of integration is a federated approach which is pushing down SQL statements from HANA into Hive or similar frameworks. The queries on HDFS data are executed on Hadoop and the results are combined in the HANA layer. While this combination is already delivered with SAP HANA, the scale-out option provides a significantly deeper integration into the overall database system stack. First, we allow to install the low footprint SAP HANA SOE on each Hadoop node. As standard we provide a file-based connector for the SAP HANA SOE, which allows to combine SAP HANA SOE data processing with standard MapReduce jobs. Figure 4 shows the bigger picture of the integrated SAP HANA data management ecosystem comprising the HANA in-memory system, the SAP HANA SOE (shown as codename "Velocity") running within YARN stack as well as streaming (ESP) and dynamic tiering for cold data. In order to also play the role of a central integration hub to other data sources, SDA ("Smart Data Access") enables federation to a huge variety of different data sources.

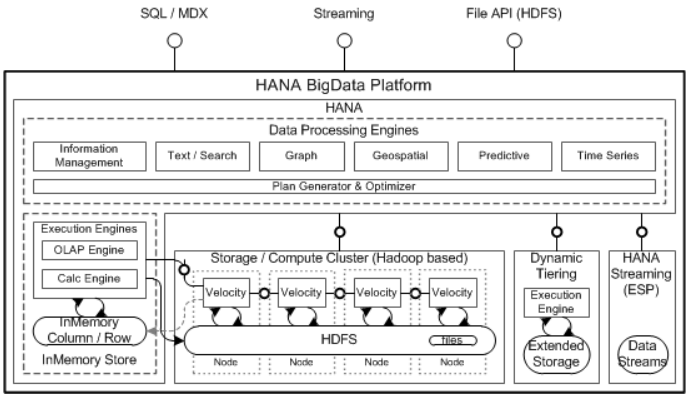

Figure 4. SAP HANA Ecosystem

With this approach, any kind of SAP data can be accessed via SQL or MapReduce-style frameworks. In addition data from local HDFS nodes can be loaded into the local SAP HANA SOE nodes. Rules can be defined which data should be loaded and how the update mechanism is configured. There are three ways to integrate these data: First–as already outlined–data can be loaded via a standard file reader. Second, integration is performed into the Spark framework as RDD objects by utilizing SAP HANA SOE for relevant operations like join, filters, aggregation etc. By wrapping SAP HANA SOE in RDD objects customers can still use all Spark functionality such as Spark SQL or MLib. Therefore, customers can combine the capabilities of the Spark framework with the speed and data integration capabilities of HANA. The third integration is planned by using the database scale out features of the HANA scale out option. Distributed SQL statements can be sent to HANA, which creates one single execution plan for querying HDFS and SAP HANA data in one database execution mechanism including one optimizer.

With SAP HANA SOE all the capabilities which are

[3]http://www.docker.com/

Final edited form was published in "2015 IEEE 31st International Conference on Data Engineering. Seoul 2015", S. 1259-1267, ISBN 978-1-4799-7964-6
http://dx.doi.org/10.1109/ICDE.2015.7113374

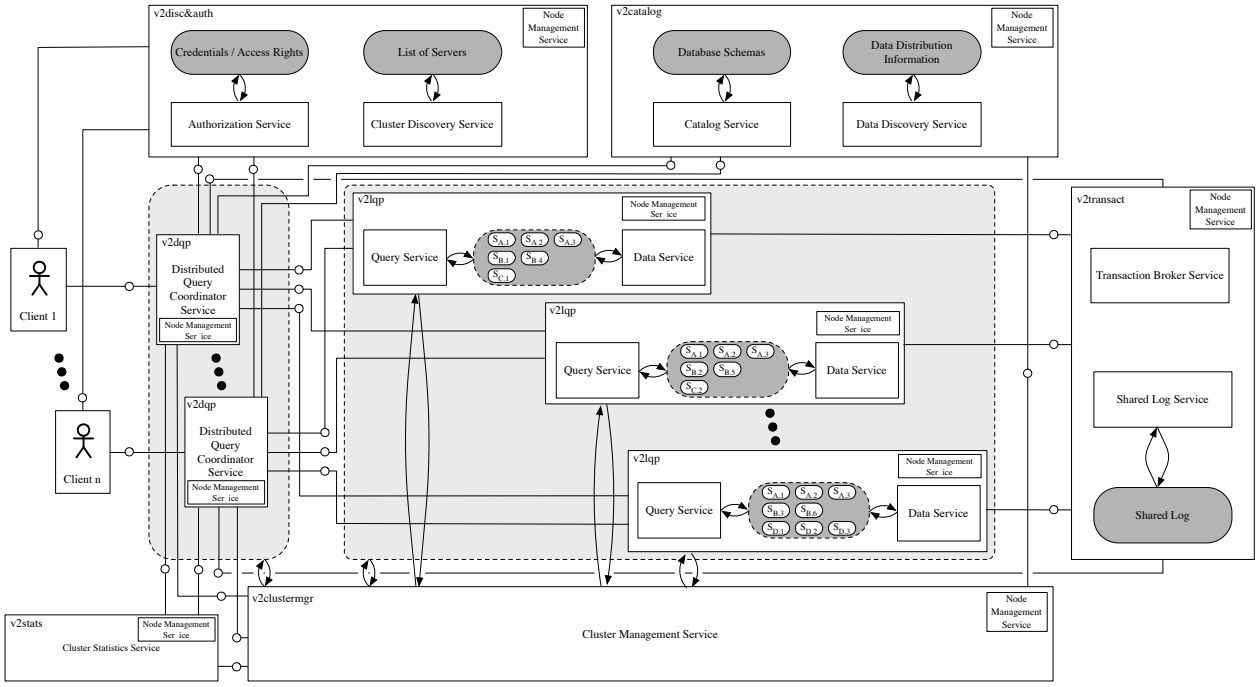

Figure 3. Architectur Components of the SAP HANA SOE

described above can be used on HDFS data and can be combined in a flexible way. Data can be stored on HDFS in three different ways: the first one is by standard HDFS mechanisms as it is used today. Secondly HDFS is used as an aging store for HANA, where aged data (for aging see III) is stored on a cheap storage mechanism. The third one is that HDFS is used as a log persistence it is stored as a log for the SAP HANA SOE. As previously described we implement one version of the distributed log on top of HDFS. The data of this log can either be consumed by SAP HANA SOE API, the distributed log API or the HDFS file reader.

V. Scenarios for a Data Management Ecosystem

To complete the bigger picture of the intended SAP HANA data management ecosystem, we describe some real world use cases and scenarios to motivate our statements and architectural decisions above. In a nutshell, we see a clear demand for polyphonic data management infrastructures. More and more, data management landscapes will comprise multiple systems with highly specialized systems. The added-values of SAP's ecosystem is to (a) cover the full spectrum reaching from traditional ACID-compliant via streaming systems to massive scale-out systems supporting a huge variety of functionality beyond the relational model. From a non-functional perspective, the SAP HANA data management ecosystem strives (b) to provide one single and coherent operational environment: one central repository for business objects with consistent deployment procedures into all SAP systems, seamless migration from development via test to active systems, single interface for a central administration of all components. As shown below, the functional depth as well as the non-functional breath is required to cope with future data management challenges in real-world business scenarios.

1) Combination of database data with numerical algebra systems: In business environments, data analysts often load data from a relational database into a numerical algebra system to perform their analysis by means of complex computations using linear algebra methods. For example, financial analysts storing stock price data within a RDBMS require on the one hand the business context of stock values, e.g., an excerpt for recent news or complete history of economical figures of the different companies. On the other hand, the analysts use statistical algorithms for example to identify correlations of stocks and derivatives.
   As a consequence, the ecosystem has to provide a transparent integration of external systems and orchestrate a distributed query processing taking the burden of explicit data exchange between the systems from the user.
2) A customer institution collects massive sensor data within a large Hadoop installation by measuring a large number of parameters of huge production infrastructures. In addition, the ERP system of the customer shows the state of the current production as well as production problems (i.e. unexpected degradation of output). The overall challenge now is to correlate the sensor data with events in the production process in order to analyze and predict machine failures or trigger pro-actively maintenance activities. As a consequence, the ecosystem has to provide a seamless integration of Hadoop infrastructures and traditional ERP application systems.
3) A producer of soap for washrooms wants to plan the routes for their service teams to fill the dispensers. Sensors in each dispenser measure the fill grade and indicate the need for a refill. In addition, the company wants to pro-actively fill them even without

Final edited form was published in "2015 IEEE 31st International Conference on Data Engineering. Seoul 2015", S. 1259-1267, ISBN 978-1-4799-7964-6
http://dx.doi.org/10.1109/ICDE.2015.7113374

an immediate need, if they have notice (gathered from different websites) of a big event in the specific location. Routing comprises to identify optimal paths to different locations as well as to give directions within huge locations in order to find the correct washrooms. Based in washroom usage patterns, the company now offers a service for the local facility teams to automatically locking (and opening) some washrooms, if the number of people visiting a washroom for an event is significantly lower or higher than expected in order to reduce the maintenance costs. Again, sensor data are stored in a Hadoop system, location data is stored in GIS information system. The ERP system holds the company's master data, and performs the resource planning, route planning as well as billing and other business functionality. Different data sets as well as business service invocations to optimize routes and compute models predicting the use of washrooms have to be combined within a single perspective to the application.

4) An insurance company wants to calculate their insurance rates based on probabilities of hurricanes and the route of hurricanes. They have stored the huge amount of data about the past hurricanes on a Hadoop like storage. Their current customers and their current rates are stored in their ERP system and the locations of the customers are kept in a geospatial storage. The goal is to generate a prediction model of future hurricanes and to map it to the locations of the customers to generate a risk profile for the different locations. Computed models have to go back to the ERP for consumption.
5) A customer is responsible of a gas pipeline which is stored as a huge graph. In addition to the logical perspective of the pipeline, the location information for the graph is stored. One out of many use cases for the customer is the development of an evacuation plan in real time if a leak in the gas pipeline is detected.

As can be seen, many installations of modern data-intensive applications require to transparently merge traditional ERP systems and classical relational data sets with graph-structured, geospatial or time-series potentially residing in different systems with different performance and TCO requirements. Moving logic to data which reside in a specifically designed engine and transparently fusing results sets is a core capability of a data management ecosystem.

VI. Summary

Within the paper, we have outlined the fundamental pillars of a web-scale data management ecosystem. We argue that such an environment will (1) exhibit a rich set of specific functionality represented in specialized data structures with sophisticated algorithms potentially residing in specific data management engines. Furthermore, (2) we see huge potential in communicated application knowledge to the data management layer. The more the database system knows about opportunities and/or constraints, the better executions plans can be generated and more efficient data structures can be used. Finally, we argue that (3) a powerful orchestration is needed to provide a single point of entry as well as a single semantic understanding for modern business applications. The SAP HANA data management ecosystem strives to achieve exactly these requirements.

Acknowledgment

We would like to thank the HANA Research and Development team for building up the HANA core engines as well as the overall ecosystem with the necessary infrastructure to provide a web-scale data management foundation for modern business applications. As always, only working as a team can yield such a success story.

References

[1] F. Färber, N. May, W. Lehner, P. Große, I. Müller, H. Rauhe, and J. Dees, "The SAP HANA database – an architecture overview," *IEEE Data Eng. Bull.*, vol. 35, no. 1, pp. 28–33, 2012.

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

[3] ——, "A common database approach for OLTP and OLAP using an in-memory column database," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2009, pp. 1–2.

[4] M. Paradies, M. Rudolf, C. Bornhövd, and W. Lehner, "GRATIN: accelerating graph traversals in main-memory column stores," in *Second International Workshop on Graph Data Management Experiences and Systems, GRADES 2014*, 2014.

[5] J. Finis, R. Brunel, A. Kemper, T. Neumann, F. Färber, and N. May, "Deltani: an efficient labeling scheme for versioned hierarchical data," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2013, pp. 905–916.

[6] D. Kernert, F. Köhler, and W. Lehner, "SLACID - sparse linear algebra in a column-oriented in-memory database system," in *Conference on Scientific and Statistical Database Management, SSDBM '14*, 2014, p. 11.

[7] R. Appuswamy, C. Gkantsidis, D. Narayanan, O. Hodson, and A. I. T. Rowstron, "Scale-up vs scale-out for hadoop: time to rethink?" in *ACM Symposium on Cloud Computing, SOCC '13*, 2013, p. 20.

[8] 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," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2012, pp. 731–742.

[9] V. Leis, A. Kemper, and T. Neumann, "Exploiting hardware transactional memory in main-memory databases," in *IEEE 30th International Conference on Data Engineering*, 2014, pp. 580–591.

[10] I. Oukid, D. Booss, W. Lehner, P. Bumbulis, and T. Willhalm, "SOFORT: a hybrid SCM-DRAM storage engine for fast data recovery," in *Tenth International Workshop on Data Management on New Hardware, DaMoN 2014, Snowbird, UT, USA, June 23, 2014*, 2014, p. 8.

[11] J. Dees and P. Sanders, "Efficient many-core query execution in main memory column-stores," in *29th IEEE International Conference on Data Engineering, ICDE 2013, Brisbane, Australia, April 8-12, 2013*, 2013, pp. 350–361.

[12] T. Neumann, "Efficiently compiling efficient query plans for modern hardware," *PVLDB*, vol. 4, no. 9, pp. 539–550, 2011.

[13] M. Weidner, J. Dees, and P. Sanders, "Fast olap query execution in main memory on large data in a cluster," in *IEEE Big Data*, Oct 2013, pp. 518–524.

[14] E. Brewer, "Pushing the cap: Strategies for consistency and availability," *Computer*, vol. 45, no. 2, pp. 23–29, 2012.

[15] M. Balakrishnan, D. Malkhi, J. D. Davis, V. Prabhakaran, M. Wei, and T. Wobber, "CORFU: A distributed shared log," *ACM Trans. Comput. Syst.*, vol. 31, no. 4, p. 10, 2013.

[16] J. M. Emison, "Big data, pick your platform," *InformationWeek, Tech Digest*, vol. September 2014, pp. 1–12, 2014.