SAP HANA –
From Relational OLAP Database to Big Data Infrastructure

Norman May, Wolfgang Lehner, Shahul Hameed P., Nitesh Maheshwari,
Carsten Müller, Sudipto Chowdhuri, Anil Goel
SAP SE
firstname.lastname@sap.com

ABSTRACT

SAP HANA started as one of the best-performing database engines for OLAP workloads strictly pursuing a main-memory centric architecture and exploiting hardware developments like large number of cores and main memories in the TByte range. Within this paper, we outline the steps from a traditional relational database engine to a Big Data infrastructure comprising different methods to handle data of different volume, coming in with different velocity, and showing a fairly large degree of variety. In order to make the presentation of this transformation process more tangible, we discuss two major technical topics–HANA native integration points as well as extension points for collaboration with Hadoop-based data management infrastructures. The overall of goal of this paper is to (a) review current application patterns and resulting technical challenges as well as to (b) paint the big picture for upcoming architectural designs with SAP HANA database as the core of a SAP Big Data infrastructure.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Query Processing

General Terms

Big Data, Streaming, Map-Reduce

1. INTRODUCTION

The event of the "Big Data" hype has triggered a significant push within the data management community. On the one hand, new systems following unconventional system architectural principles have been developed. On the other hand, traditional data management systems have incorporated requirements usually voiced within the context of a "Big Data" discussion. For SAP, the Big Data strategy is of tremendous relevance because of the opportunity to extend traditional as well as reach out to novel application scenarios. A premium example for extending existing applications can be seen with respect to traditional data-warehouse solutions. Many studies show a significant growth in terms of numbers of installations as well as the requirement to embrace non-traditional data sources and data formats [5]. Novel application scenarios address primarily a mixture of traditional business applications and deep analytics of gathered data sets to drive business processes not only from a strategic perspective but also to optimize the operational behavior.

With the SAP HANA data platform, SAP has delivered a well-orchestrated, highly tuned, and low-TCO software package for pushing the envelope in Big Data environments. As shown in figure 1, the SAP HANA data platform is based in its very core on the SAP HANA in-memory database system accompanied with many functional and non-functional components to take the next step towards mature and enterprise ready data management infrastructures. In order to be aligned with the general "definition" of Big Data, we outline the SAP HANA data platform capabilities according to the criteria volume, velocity, and variety.

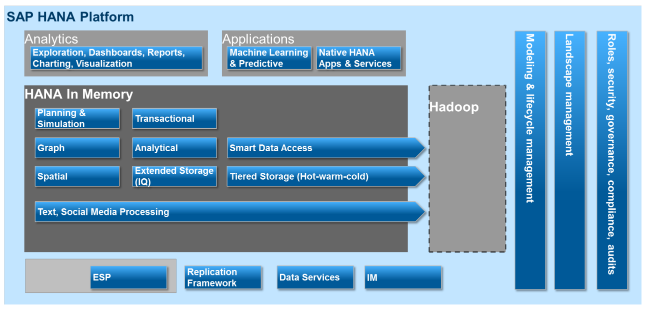

SAP HANA Platform
Analytics
Exploration, Dashboards, Reports, Charting, Visualization
Applications
Machine Learning & Predictive
Native HANA Apps & Services
HANA In Memory
Planning & Simulation
Transactional
Graph
Analytical
Smart Data Access
Spatial
Extended Storage (IQ)
Tiered Storage (Hot-warm-cold)
Text, Social Media Processing
Hadoop
ESP
Replication Framework
Data Services
IM
Modeling & lifecycle management
Landscape management
Roles, security, governance, compliance, audits

Figure 1: SAP HANA Data Platform Overview

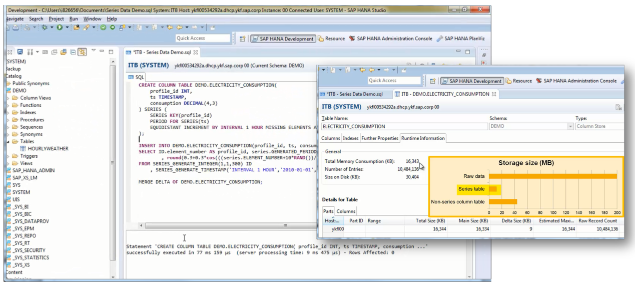
Variety

Since the SAP HANA database system has its origins partially in document processing, support for semi- and unstructured text is part of SAP HANA's DNA [8]. Recent extensions to cope with the aspect of variety additionally consists in comprehensive support for geo-spatial data and time series data as seamless extensions of the traditional table concept. Moreover the pure relational concept is relaxed and extended within SAP HANA within two directions. First, SAP HANA offers so-called "flexible tables" to extend the schema during insert operations allowing applications to extend the schema on the fly without the need to explicitly trigger DDL operations. Second, SAP HANA provides a native graph engine next to the traditional relational table engine to support schema-rich and agile data settings within one single sphere of control and based on the same internal storage structures [22]. Such an architecture reduces TCO by operating only one single system and–at the same time–allows for cross-querying between different data models within a single query statement.

Figure 2 shows an example of time series support for a rich set of

10.5441/002/edbt.2015.59

series data style application scenarios e.g. within monitoring manufacturing equipment or analyzing energy meter data in the large. As can be seen, the system does not only provide the opportunity to explicitly model the semantics of the data set (e.g. certain characteristics or missing value compensation strategies) but also provides an optimized internal representation to increase query performance and reduce the memory footprint. As indicated in figure 2 it is quite common to compress the data by more than a factor of 10 compared to row-oriented storage and more than a factor of 3 compared to columnar storage.



```
CREATE COLUMN TABLE DEMO.ELECTRICITY_CONSUMPTION(
    profile_id INT,
    ts TIMESTAMP,
    consumption DECIMAL(4,3)
) SERIES (
    SERIES KEY(profile_id)
    PERIOD FOR SERIES(ts)
    EQUIDISTANT INCREMENT BY INTERVAL 1 HOUR MISSING ELEMENTS A
);

INSERT INTO DEMO.ELECTRICITY_CONSUMPTION(profile_id, ts, consum
SELECT ID.element_number AS profile_id, series.GENERATED_PERIOD
        , round(0.3+0.3*cos(((series.ELEMENT_NUMBER+10*RAND())/
FROM SERIES_GENERATE_INTEGER(1,1,300) ID
    , SERIES_GENERATE_TIMESTAMP('INTERVAL 1 HOUR','2010-01-01',

MERGE DELTA OF DEMO.ELECTRICITY_CONSUMPTION;
```

```
Statement 'CREATE COLUMN TABLE DEMO.ELECTRICITY_CONSUMPTION( profile_id INT, ts TIMESTAMP, consumption ...'
successfully executed in 77 ms 159 µs  (server processing time: 9 ms 475 µs) - Rows Affected: 0
```

Total Memory Consumption (KB): 16,343
Number of Entries: 10,484,136
Size on Disk (KB): 30,404

Host:...	Part ID	Range	Total Size (KB)	Main Size (KB)
ykfl00			16,344	16,334

Figure 2: SAP HANA timeseries support

Velocity

In order to cope with the requirements coming from the "velocity" dimension of the Big Data definition, the SAP HANA data platform addresses velocity from an end-to-end perspective. With SQL Anywhere, the platform provides a mature instrument for distributed data capturing, local processing, and efficient propagation into central database installations. As can be seen in figure 1, external data can be imported using Data Services instruments [2] and replicated using SAP Replication Server [3]. In order to cope with extreme low-latency requirements, the mature event stream processor (ESP) solution is integrated and coupled with the in-memory database system. Section 3 will provide a deeper look into this technical solution.

Volume

While volume is fundamental to "Big Data", the SAP platform implements the full spectrum ranging from being able to work with large main-memory installations as well as providing a disk-based extension based on Sybase IQ technology. While the core in-memory HANA engine has successfully proven to work on systems with 12 TByte in a single appliance configuration (e.g. HANA-Hawk [18]), Sybase IQ still holds the world record for the largest DW installation with 12.1 PByte [21]. The transparent integration of IQ technology into the HANA core database system yields an unprecedented combination of high performance as well as the ability to handle large volumes, well beyond of any of today's enterprise-scale requirement.

Application Patterns

Looking at the different requirements, the SAP HANA infrastructure is designed to cope with different application patterns ranging from traditional decision support systems to web-scale operational recommendation applications as shown in figure 3. Typically, data from online sources is captured and pre-filtered using the SAP HANA ESP component and then forwarded to the central SAP HANA system. In addition, low-level log data stored in Hadoop infrastructures are tapped and used for statistical algorithms (e.g. recommendation algorithms). Staying within the single system, the outcome is directly forwarded to the operational system (e.g. SAP ERP) to trigger a subsequent business request.

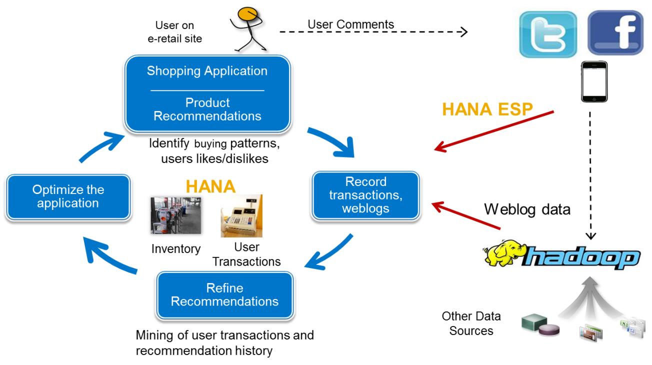


Figure 3: SAP HANA Big Data Infrastructure Core Components

In order to cope with such very typical application patterns, a comprehensive solution acting as an umbrella for individual systems and algorithmic tasks is required, which may consist of different components but presenting itself as a single point of control for the application as well as administration.

2. BEYOND THE TRADITIONAL Vs

While the SAP HANA data platform is well setup to cope with the traditional V-requirements, SAP especially focuses on delivering additional **V**alue to customers and therefore goes beyond the traditional Vs.

Value

The SAP HANA platform provides added **V**alue scenarios for customers by representing not only a core database system but an enterprise-scale data platform with additional services like:

- integrated repository of application artifacts for holistic life cycle management; for example application code in combination with database schema and pre-loaded content can be atomically deployed or transported from development via test to a production system.
- single administration interface and consistent coordination of administrative tasks of all participating platform components; for example, backup and recovery between the main-memory based SAP HANA core database and the extended IQ store is synchronized providing a consistent recovery mechanism.
- single control of access rights based on credentials within the platform; for example, a query in the SAP HANA event stream processor (ESP) may run with the same credentials as a corresponding query in the SAP HANA core database system.

More technically, SAP HANA defines **V**alue of data with respect to relevance distinguishing the low and high density data being handled with different technologies embedded into the SAP HANA platform. Figure 4 outlines the interplay between age (on the x-axis) and **V**alue (on the y-axis). Very recent data may come into the system at a high speed and high volume and is directly digested by the HANA Streaming component (ESP); from there, enriched and

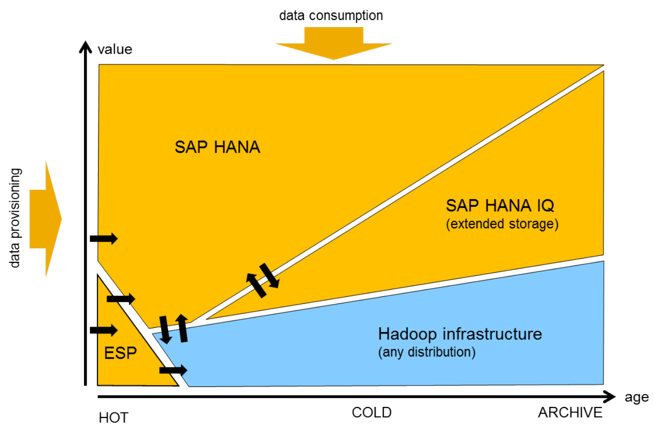


Figure 4: SAP HANA Big Data Infrastructure Core Components

potentially aggregated data can be propagated into the core SAP HANA database engine. In low-velocity scenarios raw data may directly be loaded into the SAP HANA database or handed over to a Hadoop installation as cheap background store for low-density data. High **V**alue is defined as enriched data after usually complicated transformation and cleansing steps. For example "golden records" within a master data management system or statistically corrected and improved fact data within a typical data warehouse scenario represent data sets with high relevance and customer data. As this intensively used data usually fits into the main-memory of a smaller cluster of machines, this data can be managed by the SAP HANA core database engine. HANA low-density data or data with low business value reflect fine-grained log or sensor data, (frozen) archives for legal reasons or extracted images of source systems required to feed complex data transformation processes. For economical reasons but also because of its high volume archival data is stored on disk. **V**alue therefore does not directly correlate with volume, i.e. even cleansed business data may show–especially considering historic data–significant volume but require extremely high performance. Hence, depending on its value we store high-volume data with high-value data in the extended storage and low-value data in Hadoop.

SAP Big Data = Data Platform + Applications

As mentioned before, the SAP HANA platform provides a single point of entry for the application as well as reflects a single point of control with respect to central administration for backup/recovery or life cycle management. Therefore, SAP defines Big Data infrastructures not only via core technology components but as a comprehensive application suite with support for data capturing, cleansing, transformation, and finally consumption using a large variety of transactional and analytical applications. From that perspective, the SAP business warehouse (SAP BW) solution may be considered a foundation and starting point for Big Data Analytics using the SAP HANA platform. The SAP BW solution comprises a complete tool chain for data provisioning and consumption in the traditional data warehouse sense and is based (in addition to other database systems) on the SAP HANA core database engine. Sybase IQ may be used (alternatively to other approaches) as nearline archive controlled by the SAP BW suite. Also, as shown in figure 5, SAP BW (starting with version SPS08) may exploit the extended storage to transparently move cold data to the SAP HANA extended storage. The SAP Big Data Strategy is now pushing the envelope in an evolutionary way with respect to streaming and extended storage as well as supporting Hadoop installations and without compromising the industry-strength tool chain of data provisioning as well as data consumption.

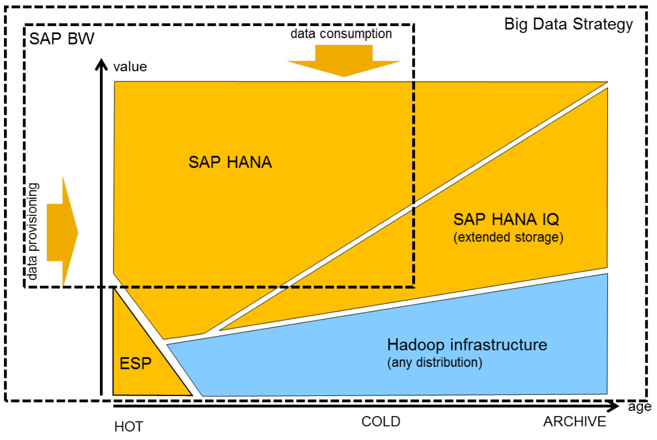


Figure 5: Extension of SAP HANA Business Warehouse

SAP Big Data–The Native and Open Strategy

Figure 5 sketches the positioning of the SAP data platform from a specific perspective of the overall Big Data design space. Obviously, the data platform is designed to deploy the components required to solve a specific Big Data problem using SAP proprietary as well as embracing open systems. In general, this leads to the following SAP Big Data strategy:

- **HANA Native Strategy:** For any Big Data installation, a data platform with only native SAP HANA components is able to cope with any number of volume, velocity, and variety requirements ([8]).
- **HANA Open Strategy:** Any existing Hadoop installation[1] can be embedded into a SAP HANA ecosystem leveraging SAP HANA added value capabilities like life cycle management and integrated application development.

Following these statements, we will detail the technical implications in the remainder of this paper. Section 3 sketches the integration of SAP Sybase IQ as extended storage (HANA IQ) natively into the SAP HANA core database engine and outlines the native support of SAP Sybase ESP as streaming component with SAP HANA (HANA Streaming). Following this native integration concepts, we will outline challenges and opportunities of the HANA open strategy and show the integration of data residing in the Hadoop Distributed File System (HDFS) and pushing processing logic to Hadoop (code to data).

3. HANA NATIVE INTEGRATION POINTS

As outlined, the native Big Data strategy of SAP addresses the full spectrum of Big Data use cases using the native SAP HANA in-memory storage together with a native integration of the IQ storage and ESP event streaming systems. The SAP HANA core in-memory engine is optimized for processing large volumes of data in an OLTP and OLAP-style allowing to serve as a consolidation vehicle for transactional as well as decision support systems [19].

[1]Currently, distributions of Hortonworks are preferred.

Within a Big Data installation, the SAP HANA core database system is usually positioned to keep high value data, i.e. hot and warm data for high-performance access. Although main memory prices went down dramatically in the recent past, it is still not cost-effective to keep bulks of cold data or archival data in main memory [9]. The SAP HANA data platform addresses this economical aspect through native integration points with the extended storage concept which is based on the IQ storage engine. However, in order to provide an end-to-end solution for Big Data scenarios, the SAP HANA data platform integrates tightly with the HANA event stream processor (ESP) of SAP for high-velocity data of (yet) low value.

Other database vendors also attempt to optimize the storage technology for the access patterns of the data. For example, the DB2 multi-temperature feature [1] allows to assign hard disks or SSDs as storage technology on the level of table spaces. Also, DB2 BLU can be used to opt either for conventional row-based and disk-based processing or column-oriented (and mainly) in-memory processing on the table level [20]. In a similar fashion, Microsoft SQL Server also features the in-memory engines Hekaton [7] and Apollo [13] for in-memory processing of data. In [23] the authors discuss a prototype for the in-memory row store, Hekaton, for identifying tuples that can be stored in cold storage. Overall, for both DB2 and SQL Server the decision for in-memory processing currently seems to be done on the table level while SAP HANA supports hot and cold partitions within the same logical table.

As we discuss in this section, both extended storage and ESP are integrated into the SAP HANA query processing. On the application level, a common repository is used to manage the life cycle of development artifacts. Administrators and developers use the SAP HANA Studio as the main interface for development and administration tasks. Meta data is centrally managed by SAP HANA and exploited during query optimization and query execution. Finally, aspects like backup and recovery are tightly integrated. In the following, we will outline technical details of this integration.

3.1 HANA IQ extended storage

SAP HANA offers a variety of options to store data: row-oriented storage in main memory is used for extremely high update frequencies on smaller data sets and the execution of point queries. Column-oriented storage is the basic option for both read- and update-intensive workload, especially for scan-intensive OLAP workloads. In this section we discuss a new storage option, the *extended storage*, which addresses use cases where massive data sets are mainly inserted and infrequently used for reporting. According to [9], it is still the case that rarely accessed data shall primarily reside on (cheaper) disk storage. In order to address this requirement, the extended storage option in SAP HANA offers a tightly integrated disk-based storage based on the Sybase IQ storage manager [15].

For example a concrete scenario for using the extended storage related to the SAP BW application includes the management of the persistent staging area (PSA) where massive amounts of data from a source system are directly mirrored into the BW infrastructure. The main purpose of a PSA is to keep the extracted data sets as sources for refinement processes within BW as well as for provenance and archiving reasons. Since objects of a PSA are (after being used to derive higher value data sets by different BW processes) only rarely used, a disk-based storage reflects an economically reasonable solution without compromising the overall in-memory approach of SAP HANA.

A similar use case with respect to the management of a PSA are so-called write-optimized DataStore objects (DSO) which serve as a *corporate memory*, i.e. data must be kept for very long durations for legal reasons. Similar to PSA objects, DSOs are rarely accessed and performance is not of paramount concern. Overall, using the extended storage SAP HANA customers transparently enjoy the benefit of low-cost storage on disk, but they can still expect reasonably short response times for infrequently accessing data stored on disk.

As a natural extension for hot and cold data stored in a single table, *hybrid tables* allow one or more partitions of a table to be represented like a SAP HANA in-memory column table and other partitions as extended storage. Figure 6 outlines the architectural setup. The IQ engine is completely shielded by the SAP HANA environment and therefore not accessible to other applications. This restriction allows for a tight integration with SAP HANA as the only “application” for IQ. In addition to query shipping to external partitions, SAP HANA provides an integrated installation and administration using SAP HANA Studio. Additionally, the tight integration allows to provide a consistent backup and recovery of both engines. Furthermore, the system allows to directly load mass data into the extended storage and register the data at the orchestrating SAP HANA instance. This direct load mechanism allows therefore to support Big Data scenarios with high ingestion rate requirements.

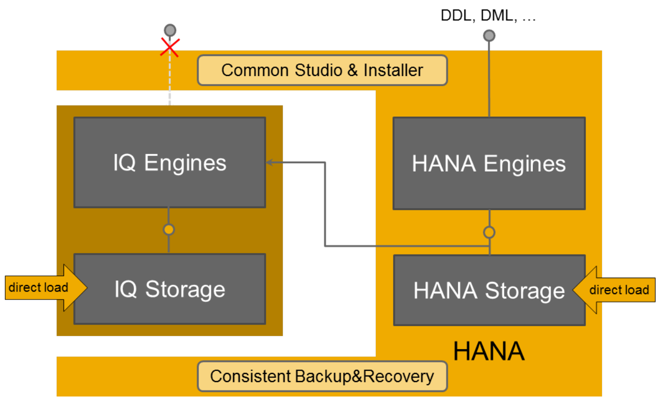


Figure 6: SAP HANA IQ Extension Overview

In summary, the concept of hybrid tables spanning SAP HANA and the Sybase IQ storage manager has the following benefits:

1. It provides a simplified system landscape as SAP HANA is the only interface to both hot and cold data including simplified setup or upgrade as well as integrated backup and recovery.

2. Seamless growth of the data can be managed in the data warehouse far beyond available main memory.

3. Integrated query processing with the extended storage including function shipping to the extended storage exposes the native query performance of the IQ storage engine even when using as extended storage orchestrated by SAP HANA.

In addition to extending SAP HANA for dealing with cold data, the extended storage deployment scenario also provides a seamless migration path from standalone IQ installations to the SAP HANA data platform. Without moving data out of an IQ system, an IQ instance can be registered at a SAP HANA system, and the customer may take advantage of the overall SAP data platform properties.

Extension on Table and Partition level

In using the extended storage, two different levels of working with the extended storage option can be differentiated: In the first scenario, an existing Sybase IQ table is directly exposed and therefore accessible in SAP HANA using the following syntax:

```
CREATE TABLE table_name table_definition
USING HYBRID EXTENDED STORAGE
```

In this syntax the HYBRID clause is optional, because the extension is based on a table level and no mixture of IQ and SAP HANA tables is configured [4]. The extended storage technology performs data type mappings between the engines as well as provides full transactional support. Additionally, a data load issued against such an external table directly moves the data into the external store without taking a detour via the in-memory store followed by a subsequent push into the extended store. Moreover, the extended storage technique supports schema modifications like any other table in SAP HANA to complete the hybrid table concept.

In the second scenario of partition level extension, an SAP HANA table may be partitioned into one or multiple hot partitions, which are realized by regular in-memory column-oriented storage and one or multiple cold partitions which live in a table of the IQ storage. Such a resulting table is considered a SAP HANA hybrid table. The residence of hot and cold data is decided on the partitioning criteria and can be controlled by the application. Additionally, the SAP HANA data platform provides a built-in aging mechanism, which periodically moves data from the hot storage of the in-memory partitions into the cold storage. The decision is based on a flag in a dedicated column of the hybrid table.

Configuration Scenarios

In order not to interfere with the memory management of the in-memory engine, the IQ engine is usually deployed at a separate host/cluster. This allows for a more specific sizing of the involved nodes in the SAP HANA distributed landscape. For example, the extended storage may rely on a more powerful I/O subsystem than the server where the SAP HANA database is running and usually requires less main memory. The notion of a data platform ensures that the overall system landscape is kept manageable because of a unified installer and integrated administration tool chain.

Transactions

From an application perspective, both extended tables and hybrid tables appear like regular row or column tables in SAP HANA. This implies that they can participate in update transactions that insert, update or delete data in the extended or hybrid table. As a consequence, database operations on SAP HANA extended tables participate in (normal) distributed HANA transactions. In such a scenario, SAP HANA coordinates the transaction, e.g. generating the transaction IDs and commit IDs to integrate extended storage. As a seamless integration, we use the improved two-phase commit protocol described in [14] also for the extended storage. Consequently, SAP HANA will be able to recover the data in case of failures as any other SAP HANA table, including point-in-time recovery. In case of an error of the extended system, every access to a SAP HANA table may throw a runtime error. In particular, any query that touches objects located on the extended storage will be aborted. Additionally, if that access is part of a transaction that also touches in-memory column tables in SAP HANA, the entire transaction will be aborted. Finally, the recovery of an extended storage instance is recovered jointly with SAP HANA. Without this integrated recovery any transaction that had touched the extended store but not committed will be marked as "in-doubt". Clients will have the ability to manually abort these "in-doubt" transactions.

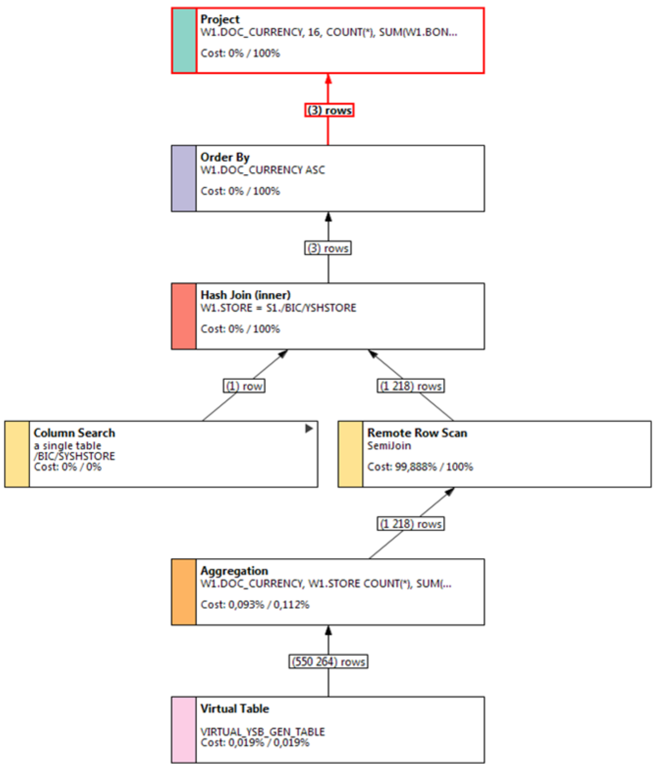


Figure 7: Federated query processing

Query Processing

Due to the tight integration of the IQ engine into the SAP HANA distributed query processing framework, the SAP HANA engine is able to exploit a huge variety of query processing tweaks, especially push-downs to the IQ storage manager. Capabilities include the ability to perform inserts, updates or deletes, order by, group by, different kinds of joins, or execution of nested queries. The cost-based query optimizer of SAP HANA either uses q-optimal histograms based on values for cardinality estimates on the extended storage [16]. The query optimizer considers communication costs for the data access to the extended storage. The *distributed exchange* operator is used to mark the boundary between HANA-local processing and remote execution in the IQ query processor. Sub plans below this operator are executed on the remote data source. During query optimization different alternatives exists to evaluate subplans in the extended storage:

- **Remote Scan:** Process a complete sub query independent from SAP HANA in the extended storage. The optimizer decides if the result should be returned in columnar or row-oriented format for further processing in SAP HANA.

- **Semijoin:** In this alternative data is passed from SAP HANA to the extended storage where it is used for filtering either in an IN-clause or a join using a temporary table. The optimizer picks such a strategy for example if parts of the fact table are sitting in IQ and require a join with smaller (and usually frequently updated) dimension tables.

- **Table Relocation:** This alternative considers SAP HANA tables as remote tables for IQ and pulls data on demand from SAP HANA.

- **Union Plan:** Do local processing in SAP HANA and the IQ storage engine, and use a union to integrate the partial results. Such a strategy is usually picked, if the dedicated aging flag allows for partition-local joins and partial group-bys.

Figure 7 shows the plan for a query that joins a columnar SAP HANA table with a selective local predicate with a table in the extended storage. In this scenario, the semijoin strategy is the most effective alternative because only a single row is passed from SAP HANA to the extended storage where it can be used to filter the large remote table. In this example it is even possible to push the group-by operation to the remote data source.

3.2 HANA Stream Extension

The HANA stream extension is based on the SAP Sybase Event Stream Processor (SAP Sybase ESP) and addresses the specific needs of data acquisition and responding with actions in realtime in a Big Data scenario. The data acquisition part deals, for example, with filtering out irrelevant events or immediate aggregation of the captured data for further processing in the SAP HANA core engine or Hadoop. The SAP Sybase ESP may also detect predefined patterns in the event stream and trigger corresponding actions on the application side.

To illustrate these use cases consider a telecom company that captures live data from its mobile network, for example information on the health status of its network or unexpected drops of customer calls (figure 8). In this figure, multiple components of the SAP HANA platform interact; each component is shown in a separate box with distinct color. In this telecom scenario, data is generated in high volume in the telecom network, but not all data has high value. Sensors in the telecom network capture various events, e.g. information about started or terminated phone calls. As long as the network is in a healthy state, no specific action has to be taken, and it is sufficient to store aggregated information on the status of the network. Hence, the SAP Sybase event stream processor uses the Continuous Computation Language (CCL)[2] to analyze the raw event data, instantly analyze it, and aggregate events over a predefined time window for further processing.

Furthermore, the raw data may be pushed into an existing HDFS using a dedicated adapter such that it is possible to perform a detailed offline analysis of the raw data. The resulting network data archive is then analyzed using map-reduce jobs in the Hadoop cluster. Such an advanced analysis might attempt to optimize the network utilization or to improve the energy efficiency of the network. In this scenario we use Hadoop for archiving purposes as the incoming data has highly variable structure. For more strictly structured data using the extended storage would be more appropriate.

In a development system the raw events collected in the network data archive may be replayed to the event stream processor to verify the effectiveness of improved event patterns. If the patterns derived through the map-reduce jobs in the Hadoop cluster prove useful, they are deployed in the productive ESP system.

However, if an outage of the network is detected, immediate action is required, which could be directly triggered by the SAP Sybase ESP and forwarded to the SAP HANA database using the SAP Data Services [2]. While an alert is immediately sent to the operations staff, a detailed analysis of the imported event data of

[2]CCL is a SQL-like declarative language to process events over windows of data, see [12] for details

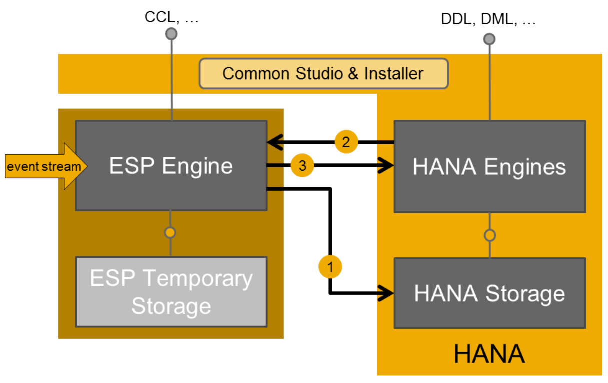


Figure 9: Example Scenario for Complex Event Processing in SQL HANA data platform

the exceptional event may also trigger reports to be prepared for service engineers. In a similar way, business data, e.g. information about the amount of transferred data but also standard business processes like billing or marketing are processed in the SAP HANA database.

Moreover, pre-aggregated and cleansed data may be loaded into the SAP HANA database for online processing. For example, the sensor data of antennas may be normalized into equi-distant measures of the network state and loaded into a time series table of SAP HANA. This allows advanced analysis on that data, e.g. perform correlation analysis between different sensors. Furthermore, the sensor data may trigger business actions because information about established connections collected from the sensor data must be matched with customer data for accounting purposes. This calls for an integrated processing runtime of sensor data and relational data stored in the SAP HANA database.

Use Cases

In general, the integrated ESP engine tackles the following three main use cases, also depicted in figure 9:

1. **Prefilter/pre-aggregate and forward:** In this use case, incoming data streams are filtered and/or aggregated. The result is then forwarded and directly stored within native SAP HANA tables. Although the ESP provides some temporary storage capability to reconstruct the content of a stream window, the forward use case allows to permanently store the window content under the control of the database system.

2. **ESP join:** In this case, slowly changing data is pushed during CCL query execution from the SAP HANA store into the ESP and there joined with raw data elements. For example, city names are attached to raw geo-spatial information coming from GPS sensors.

3. **HANA join:** In the opposite of a HANA join, a native HANA query may refer to the current state of an ESP window and use the content of this window as join partner within a relational query. Such a setup is particularly interesting, when dynamic content, e.g. the current state of a machine or environmental sensors, is required to be merged with traditional database content.

As common in stream environments, no transactional guarantees are provided; This however is usually accepted in high-velocity scenarios.

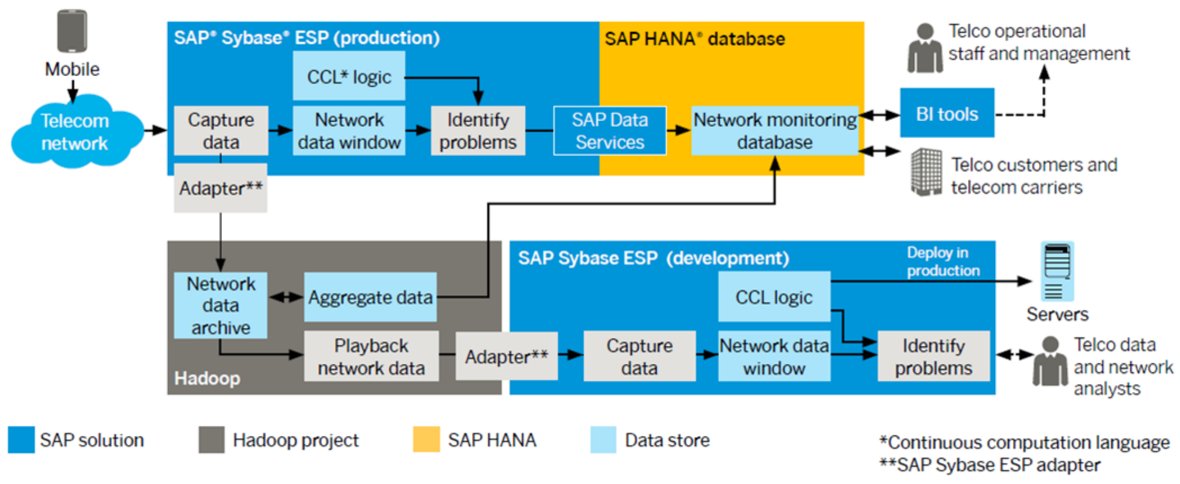


Figure 8: Example Scenario for Complex Event Processing in SQL HANA data platform

3.3 Summary

The overall strategy of SAP to rely on a native SAP HANA Big Data story as well as embracing (usually existing) open source infrastructure components requires to provide the concept of a data platform with an integrated set of different components. In this section we outlined the native extension points of SAP HANA with respect to IQ as the extended storage as well as ESP as the integrated stream engine allowing cross querying between the table-based database and the stream-based event processing world.

4. HANA INTEGRATION POINTS WITH HADOOP

Huge amounts of data are already stored in HDFS, and significant investments were made to analyze the data stored in the HDFS using either custom-made map-reduce jobs or using more declarative languages like HiveQL [25]. Clearly, for SAP customers want to tap into this ocean of data and generate higher-value information from it using these map-reduce jobs or Hive. In many cases, the resulting information must be integrated with the enterprise data warehouse or operational applications, e.g. for a detailed customer analysis.

In this section, we explain the goals we want to achieve with the Hadoop integration based on some customer scenarios. After that we survey technical details of the Smart Data Access (SDA) framework which is used to integrate SAP HANA with Hadoop. We focus on the Hadoop-side caching which results in significantly lower response times of map-reduce jobs and more effective use of the Hadoop cluster.

The integration of SAP HANA with any Hadoop distribution illustrates how easy and powerful it is possible to link almost any external data source with SAP HANA in a loosely coupled way. This open platform strategy is mainly realized with the Smart Data Access (SDA) technology, SAP HANA's capability-based adapter framework.

4.1 Goals and scenarios

Within the previous section we already pointed out that SAP ESP may push raw events immediately into the Hadoop Distributed File System (HDFS) so that it can be analyzed further in a rather offline fashion. In addition, significant amounts of data today are already stored into the HDFS. Usually, the shear amount of data (volume) but also its loose structure (high variety) makes it unattractive to load this data immediately into a relational database. Since raw data is of low value, freshness is not critical, and it is often sufficient to thoroughly analyze this data in batch jobs and then exposing the results to the SAP HANA database for reporting, planning etc.

In general, such a setup implies a co-existence of a conventional database side by side with data stored in HDFS which is analyzed in Hadoop [26, 17, 24, 6]. Map-reduce jobs are periodically executed to derive higher-level information in a more structured form which may be stored in a data warehouse but also to correlate the content stored in a HDFS with data already available in a database.

In this loosely coupled setup two main scenarios exist to link the database with Hadoop:

- Delegation of ad-hoc queries to Hadoop using an ODBC adapter and Hive [25].
- Exposing the content in the HDFS by calling existing map-reduce programs.

Having SAP HANA as the federation platform with Hadoop then leads to a setup with two separate systems. Standard tools for HANA, e.g. the eclipse-based administration and development tool SAP HANA Studio, can be used to develop Big Data applications. With these development tools at hand, SAP HANA can also handle life cycle tasks, e.g. transporting map-reduce programs from a development system to the productive system. As data now resides both in SAP HANA row/column in-memory tables and HDFS, query processing is distributed over both SAP HANA and Hadoop. In such a setup, SAP HANA serves as the main interface for ad-hoc queries and orchestrates federated query processing between these stores. If Hive is used, parts of a query may even be shipped to Hive based on the capabilities registered for Hive and Hadoop. Hadoop returns its result in a structured form that is ready to be consumed by HANA for further processing.

Having an interface from SAP HANA to Hadoop exposes several features of Hadoop which are typically not available in a relational database. For example, one can reuse libraries implemented on top of Hadoop like Mahout for machine learning, or custom social media analysis [11]. These kinds of tasks are typically a weak side of relational databases and can be softened using a Hadoop infrastructure. Moreover, one can use Hadoop as a scalable and very flexible way to implement user-defined functions which are capable to access schema-flexible data without the need to transform them into the relational model beforehand. Having SAP HANA as federation layer also allows us to combine relational query processing and Hadoop with the capabilities of statistical analysis with R [10].

To illustrate these use cases consider a project with a large SAP customer in the automotive industry with the objective to predict warranty claims. The data sources stored in SAP HANA on the one side included condensed information on the production, assembly and sales of automobiles but also facts about customers and past marketing campaigns. The raw data stored in HDFS on the other side included diagnosis read-outs on cars, support escalations, warranty claims and customer satisfaction surveys. Overall, the Hadoop cluster used twenty servers with 250 CPU cores, 1500 GB RAM and 400 TB Storage as aggregated compute power available in the Hadoop cluster. The HANA server was equipped with 40 cores, 512 GB RAM and 2 TB disk storage. Using Hive, we extracted data from twelve months data for a specific car series and made it available to the SAP HANA database server. With the SAP predictive analysis library using the apriory algorithm thousands of association rules were discovered with confidence between 80% and 100%. The derived models then were used to classify new readouts as warranty candidates in real-time in the SAP HANA database.

4.2 SDA-based integration - Query shipping

An important building block for the integration of Hadoop with SAP HANA is the Smart Data Access (SDA) framework implementing an access layer for a wide variety of different remote data sources like Hadoop, Teradata or other relational and non-relational systems. Thereby, SAP HANA realizes an open strategy to integrate and federate remote data sources.

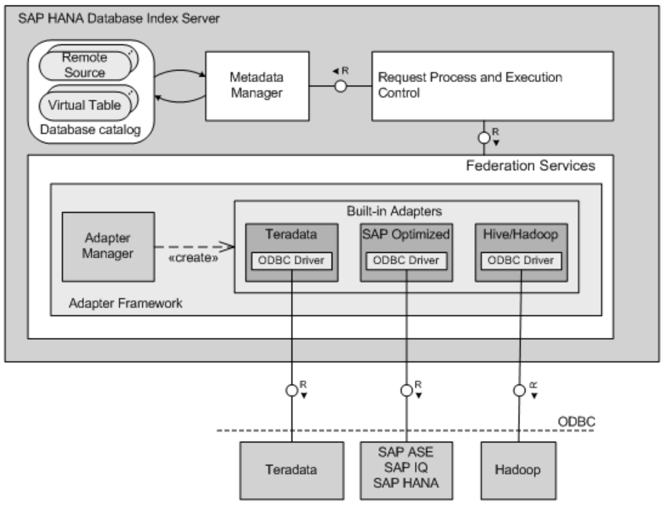


Figure 10: SAP HANA SDA Overview

Figure 10 gives a high-level overview of the generic SDA framework. In the SDA framework, remote resources are exposed as a *virtual table* to SAP HANA. Consequently, these resources can be referenced like tables or views in SAP HANA queries or views. The communication to remote resources is realized by adapters which are usually specific to the data source. There are various SDA adapters already available, e.g. Hadoop[3], any SAP database, IBM DB2, Oracle, or Teradata. This means that by providing an SDA adapter for a specific type data source makes this data source accessible for SAP HANA. As each SDA-adapter exposes capabilities specific to this data source, the SAP HANA query optimizers is able to forward parts of a query execution plan to the remote data source. In the remainder of this section we focus on the integration of Hadoop using SDA adapters as one prominent example, but most concepts also apply to other data sources.

[3]With HANA SPS09 we support the Hadoop distributions Intel IDH, Hortonworks, and Apache Spark.

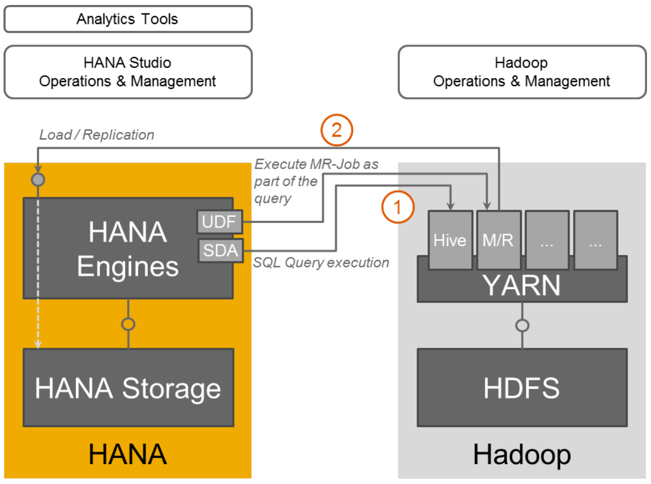


Figure 11: SAP HANA / Hadoop side-by-side

Registering a Remote Data Source

We use SDA to communicate with Hadoop via Hive, especially to address ad-hoc data analysis on data stored in HDFS. As indicated in figure 10, we use ODBC to establish the connection to the Hadoop system. In this setup SDA passes partial or complete queries or DDL statements to Hive for execution in Hadoop. The SDA framework and its integration into the HANA query compiler takes care that only queries are passed to Hadoop that are also supported by Hive and Hadoop. SDA also applies the required data type conversions with Hadoop. Below we show a typical workflow to first create the remote access to a Hive-based Hadoop distribution, wrap the remote source as a virtual table, and finally query the content of the data:

```
CREATE REMOTE SOURCE HIVE1 ADAPTER "hiveodbc"
       CONFIGURATION 'DSN=hive1'
 WITH CREDENTIAL TYPE 'PASSWORD' USING
       'user=dfuser;password=dfpass';
CREATE VIRTUAL TABLE "VIRTUAL_PRODUCT"
       AT "HIVE1"."dflo"."dflo"."product";
SELECT product_name, brand_name
  FROM "VIRTUAL_PRODUCT";
```

Query Processing

As mentioned above, SDA relies on a description of the capabilities of a remote server. For example, transactions for some database servers e.g. updates and transactions, are supported. However, for Hive and Hadoop only select statements without transactional guarantees are supported. For Hive on Hadoop it is possible, e.g. to push predicates or joins to Hadoop, but also to use semi-join reduction for faster distributed joins between Hadoop data and HANA tables. In the capability property file one finds, e.g. `CAP_JOINS : true` and `CAP_JOINS_OUTER : true`, to denote that inner joins and outer joins are supported. It is even possible that complete queries are processed via Hive and Hadoop. When only parts of a query can be executed in Hive and Hadoop, the query compiler generates a transient virtual table that represents the result of the subquery which will be processed by Hive and Hadoop. It also adds the needed operations to integrate this subquery into the plan executed in SAP HANA. In the simple-most case the results are only integrated via joins or union, but in more complex cases compensating operations might be required, e.g. mapping of data formats and data types.

To estimate the costs for accessing the Hadoop data, we rely on the statistics available in the Hive MetaStore, e.g. the row count and number of files used for a table. These statistics and also estimated communication costs are considered for generating the optimal federated execution plan. The plan generator attempts to minimize both the amount of transferred data and the response time of the query.

4.3 HANA Direct Access to HDFS

Besides the ad-hoc query capabilities via Hive discussed above, SAP HANA can also invoke custom map-reduce in Hadoop, see figure 11. This allows customers to reuse their existing map-reduce codebase and to access the HDFS files directly, i.e. without the additional Hive layer. The basic workflow is shown below: The first statement registers an Hadoop cluster with SAP HANA. The subsequent statement declares a virtual remote function that exposes an existing map-reduce job as a regular table function in SAP HANA. Finally, this virtual table function can be used in SQL statements like any other table function in SAP HANA.

```
CREATE REMOTE SOURCE MRSERVER
    ADAPTER hadoop CONFIGURATION
      'webhdfs=http://mrserver1:50070;
       webhcatalog=http://mrserver1:50111'
       WITH CREDENTIAL TYPE 'password'
       USING 'user=hadoop;password=hadooppw';

CREATE VIRTUAL FUNCTION
       PLANT100_SENSOR_RECORDS( )
    RETURNS TABLE (EQUIP_ID VARCHAR(30),
                   PRESSURE DOUBLE)
    CONFIGURATION
      'hana.mapred.driver.class =
    com.customer.hadoop.SensorMRDriver;
     hana.mapred.jobFiles =
              job.jar, library.jar;
     mapred.reducer.count = 1'
     AT MRSERVER;

SELECT A.EQUIP_ID, A.LAST_SERVICE_DATE,
       B.PRESSURE
FROM   EQUIPEMENTS A  JOIN
       PLANT100_SENSOR_RECORDS() B
       ON A.EQUIP_ID = B.EQUIP_ID
WHERE  B.PRESSURE > 90;
```

It is worth noting that this workflow is also embedded into SAP HANA's development environment and landscape management. For example, views accessing the map-reduce job as a table function keep their connection when the models are transported from test to productive system.

4.4 Remote Materialization

The map-reduce jobs in Hadoop are typically used to turn large volumes of rather low-value data into smaller data sets of higher value data. As the runtime of these map-reduce jobs tends to be significant, SAP HANA offers to cache the result of map-reduce jobs on Hive side. As customers have a sound understanding of their business data they will know where absolute data freshness is not needed, e.g. for low velocity data. The freshness of the cached results of map-reduce jobs is configurable.

Extended Query Processing

When an application submits a query that includes a Hive table as a data source, it can request the use of the caching mechanism. More precisely, the application appends `WITH HINT (USE_REMOTE_CACHE)` to the query string to enable the cache usage. During query processing it is checked if caching is requested for the query. If this is the case, a hash key is computed from the HiveQL statement, parameters, and the host information. With this hash key we can can ensure that the same query is cached at most once. If a value is cached for this hash-key the corresponding query result is returned immediately. If either no caching is requested, no cache key was found or the cached value is older than a configurable parameter, the query is evaluated from scratch. Evidently, the cached data is not related to specific transactional snapshots, but we expect this to be of little concern in a Hive setup. The cached data is stored in temporary tables in HDFS. Large caches are possible because the HDFS often has enough space to keep materialized query results in addition to the raw data. Overall, caching leads to a more effective use of the compute resources in the Hadoop cluster.

To illustrate these concepts, consider the following query on the TPC-H schema which references two virtual tables `CUSTOMER` and `ORDERS` from Hive:

```
SELECT c_custkey, c_name,
       o_orderkey, o_orderstatus
FROM   customer JOIN orders
       ON c_custkey = o_custkey
WHERE  c_mktsegment = 'HOUSEHOLD'
```

Under normal execution, i.e. without any caching in Hive, when the federated query is executed at the remote source, the Hive compiler generates a DAG of map-reduce jobs corresponding to the federated query. Map-reduce jobs from this DAG are triggered thereafter, upon completion of which the results are fetched back into HANA for further operations. The query execution plan for the normal execution mode of the example query is shown in figure 12. It shows two Virtual Table nodes corresponding to the tables `CUSTOMER` and `ORDERS` with their corresponding predicates, and a `Nested Loop Join` node which is also computed at the remote source. Since, there are no other tables accessed locally inside SAP HANA, the data received in the `Remote Row Scan` node is projected out. If the query has references to other local HANA tables, the data received in the `Remote Row Scan` node will be used for further processing along with data from the local tables.

Under the enhanced mode with remote caching enabled, the optimizer identifies the hint to use remote materialization and materializes the results, after executing the DAG of map-reduce jobs generated by the Hive compiler, to a temporary table at the remote site. It then modifies the query tree to read everything from this temporary table. It should be noted that this materialization process is a single-time activity and every subsequent execution of this federated query will fetch the results from the materialized copy stored in the temporary table instead of executing the corresponding DAG of map-reduce jobs. The modified query execution plan for our example query under the enhanced mode is shown in figure 13. It shows one `Virtual Table` node from which all the interesting

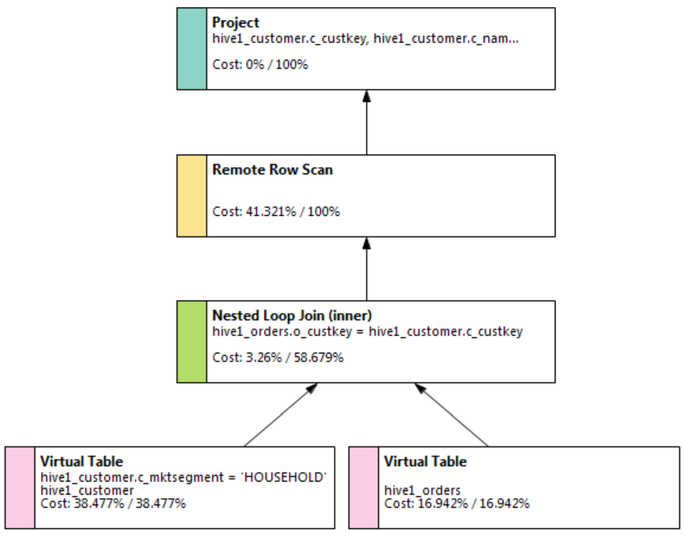


Figure 12: Query plan without remote materialization

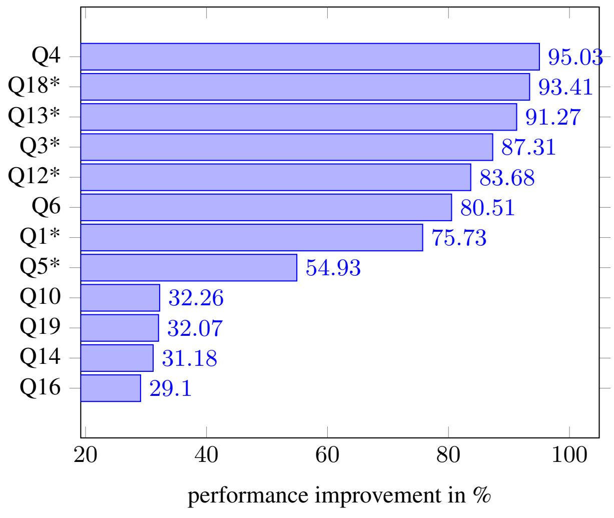


Figure 14: Runtime benefit of remote materialization

data required to answer the query can be retrieved. For the current example, this virtual table node corresponds to the joined data from the tables CUSTOMER and ORDERS with all the necessary predicates already applied. We can see from the execution plan in figure 13 that the Remote Row Scan node is directly fetching the necessary data from the temporary table, with no additional predicates being applied on the temporary able.

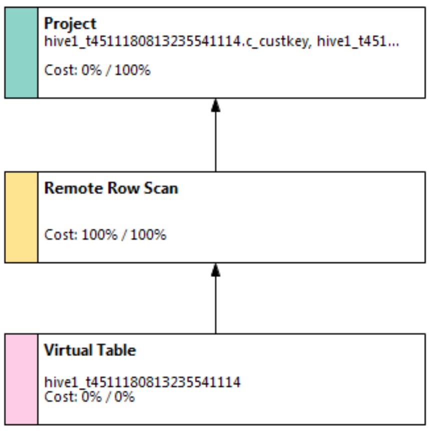


Figure 13: Query plan with remote materialization

In addition to these basic caching techniques the remote materialization implements further improvements: First, we only materialize queries with predicates. This ensures that we do not replicate the entire Hive table as a materialized result set as this will not add any value to the performance of the system. Second, the duration for which a materialized result set is valid and is persisted in the remote source is controlled via a configuration parameter, called remote_cache_validity. When the query optimizer identifies the hint USE_REMOTE_CACHE, it checks if the materialized data set on the remote source is valid based on this parameter setting before actually using it. If it discovers that the data set is outdated, it discards the old data set and materializes the result set to a new copy at the remote source. Finally, the remote materialization enhancement in SAP HANA is disabled by default and can be controlled using the configuration parameter enable_remote_cache. This parameter is useful in scenarios when the customer needs to completely disable the feature, for example, low storage availability in HDFS or when the tables in Hive are being frequently updated.

Performance Analysis

We demonstrate the difference between using Hive and SAP HANA with and without remote materialization. This small experiment uses the TPC-H dataset with scale factor 1. The experiments were performed using SAP HANA SPS07 as the primary database running SUSE SLES 11.2 on a server with 16 physical cores and 256GB RAM. The 7-node Hadoop cluster was accessed from SAP HANA via Hive's ODBC driver as remote source. We used a Hadoop cluster configuration with Apache Hadoop 1.0.3, Hive 0.9.0 on an HDFS with 21.5TB capacity, 240 map tasks, 120 reduce tasks, and 6 worker nodes. The following tables from the TPC-H schema were federated remotely at Hive: LINEITEM, CUSTOMER, ORDERS, PARTSUPP, and PART. The tables present locally in SAP HANA were: SUPPLIER, NATION, REGION (, and PART only for Q14 and Q19). Such a small scale-factor is ridiculously small for a typical Hive and Hadoop setup, but for large data sets the positive impact of remote materialization would be even more pronounced. In that sense, this somewhat unrealistic setup is a very conservative analysis of the expected performance improvements of remote materialization with SAP HANA and Hive.

We used slightly modified versions of the benchmark queries. In particular, we removed the TOP and ORDER BY clauses from the TPC-H queries, with the exceptions being those queries for which the sorting was done inside SAP HANA. This is desirable as we cannot make any assumptions about the ordering property of the datasets fetched from Hive which were materialized earlier.

In figure 14 we present the runtime benefit achieved when using SDA with remote materialization enhancement using the normal execution mode with SDA as the baseline. We also demonstrate the materialization overhead incurred in materializing the results on the remote system; this is shown in figure 15. We have marked the modified queries discussed above with an asterisk (*).

We can see from figure 14 that using remote materialization, some queries can benefit as high as 95% with respect to query re-

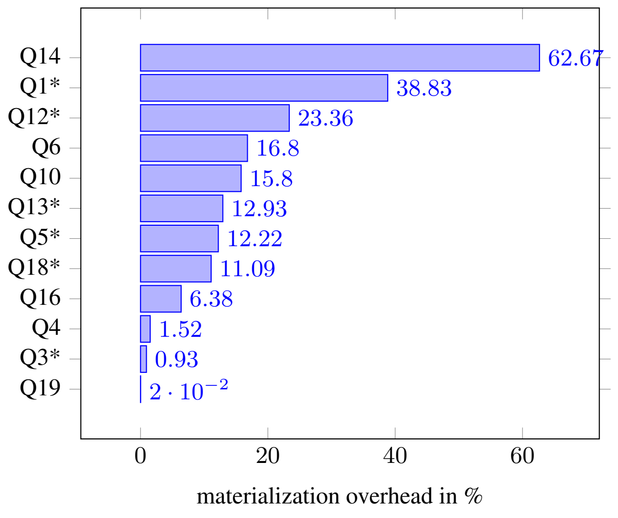


Figure 15: Materialization overhead of remote materialization

sponse time. There are two aspects to the overall query execution time when we have a remote system in tandem with SAP HANA: time taken to fetch the data required from the remote source, and time taken to join the fetched data with local data in SAP HANA. We can infer from the results that the queries can be divided into two segments based on their respective performance gains.

The data displayed in figure 14 has been sorted based on the maximum runtime benefit. The top seven queries for scale factor 1 demonstrate high gain of more than 75%. This is expected because all the tables accessed in these queries are federated tables, and there are no local tables with which the results fetched from the remote source are joined. For the remaining queries, the performance gain is on the lower side because the results fetched from the remote source are joined with local tables in HANA. This is also expected since we demonstrate the percentage improvement in the overall query execution time, and not just the time taken to fetch the data required from the remote source. We can conclude from the results of our experiment that our enhancement provides maximum benefits for the cases when the majority of the computation is performed on the remote system and minimum data is read back into HANA. In an analytical workload, several queries are executed multiple times and with remote materialization enhancement, every execution of the query can benefit by skipping this computation and directly fetching the already materialized data. This enhancement is specifically useful in case of Hive, because a user may not have exclusive access to the Hadoop cluster and may only get a limited share of the Hadoop cluster's overall capacity.

The materialization process at the remote source has an associated overhead with it, which is also demonstrated in figure 14. This overhead is the additional time required to materialize the results in Hive and is a single-time cost, which is incurred when the query is first executed with the optimizer hint `USE_REMOTE_CACHE`. As long as the data in the Hive tables is not modified, SAP HANA can be configured to continue reusing the materialized results from Hive. This can have huge benefits with low velocity data stored in Hive, depending on the frequency of queries that use these materialized results. This overhead in materialization process is partly because `CREATE TABLE AS SELECT` (or `CTAS`) in Hive is currently a two-phase implementation: first the schema resulting from the `SELECT` part is created, and then the target table is created. We make use of this `CTAS` infrastructure provided by Hive to create the temporary tables. The materialization overhead demonstrated in our experiments is set to go down when the `CTAS` implementation in Hive gets optimized.

5. SUMMARY

The SAP HANA data platform reflects SAP's answer to the ever-increasing requirements and opportunities in management of data. It was specifically designed as a single point of access for application logic by providing a HANA native as well as HANA open strategy:

- SAP HANA core database can serve real time, complex queries and multi-structured data needs.
- SAP Sybase IQ (HANA IQ) can provide highly concurrent OLAP workload in combination with large scale, disk-based storage
- SAP Sybase ESP (HANA ESP) can provide high velocity on-the-fly analysis with native hand-over to other SAP HANA components.
- Hadoop can provide cheap data storage and pre-processing for large scale unstructured/unformatted data and compiled into the HANA data platform by allowing query/code push-down and part of HANA's life cycle management capabilities.
- SAP BW and SAP EIM [2] – as some examples on an application level – can provide consumption, modeling, and integration capabilities (including Hadoop)

This paper gives some insight into the overall picture of Big Data applications in SAP's perception as well as diving into technical details of the HANA native and HANA open integration strategy.

Acknowledgment

We would like to thank all colleagues working on the Big Data strategy and productization of SAP HANA. We especially thank the team of Sudipto Chowdhuri which did the main part of the implementation presented in this paper.

6. REFERENCES

[1] *Delivering Continuity and Extreme Capacity with the IBM DB2 pureScale Feature*. `http://www.redbooks.ibm.com/abstracts/sg248018.html`.

[2] *SAP Enterprise Information Management*. `http://help.sap.com/eim`.

[3] *SAP Replication Server Documentation*. `http://help.sap.com/replication-server`.

[4] *SAP SQL and System Views Reference*. `http://help.sap.com/hana_platform/`.

[5] S. Chaudhuri, U. Dayal, and V. Narasayya. An overview of business intelligence technology. *Commun. ACM*, 54(8):88–98, Aug. 2011.

[6] D. J. DeWitt, A. Halverson, R. V. Nehme, S. Shankar, J. Aguilar-Saborit, A. Avanes, M. Flasza, and J. Gramling. Split query processing in Polybase. In *SIGMOD Conference*, pages 1255–1266, 2013.

[7] C. Diaconu et al. Hekaton: SQL Server's memory-optimized OLTP engine. In *SIGMOD*, pages 1243–1254, 2013.

[8] 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.*, 35(1):28–33, 2012.

[9] G. Graefe. The five-minute rule 20 years later: And how flash memory changes the rules. *ACM Queue*, 6(4):40–52, July 2008.

[10] P. Große, W. Lehner, T. Weichert, F. Färber, and W.-S. Li. Bridging two worlds with RICE – integrating R into the SAP in-memory computing engine. *Proc. VLDB*, 4(12):1307–1317, 2011.

[11] P. Große, N. May, and W. Lehner. A study of partitioning and parallel udf execution with the SAP HANA database. In *SSDBM*, page 36, 2014.

[12] S. Inc. *Stream, Schema, Adapter, and Parameter CCL Language Extensions*. `http://www.sybase.de/detail?id=1080119`.

[13] P.-Å. Larson, E. N. Hanson, and S. L. Price. Columnar storage in SQL Server 2012. *IEEE Data Eng. Bull.*, 35(1):15–20, 2012.

[14] J. Lee, Y. S. Kown, F. Färber, M. Muehle, C. Lee, C. Bensberg, J. Y. Lee, A. H. Lee, and W. Lehner. SAP HANA distributed in-memory database system: Transaction, session, and metadata management. In *ICDE*, 2013.

[15] R. MacNicol and B. French. Sybase IQ multiplex - designed for analytics. In *VLDB*, pages 1227–1230, 2004.

[16] G. Moerkotte, D. DeHaan, N. May, A. Nica, and A. Boehm. Exploiting ordered dictionaries to efficiently construct histograms with q-error guarantees in SAP HANA. In *SIGMOD*, pages 361–372, 2014.

[17] F. Özcan, D. Hoa, K. S. Beyer, A. Balmin, C. J. Liu, and Y. Li. Emerging trends in the enterprise data analytics: Connecting Hadoop and DB2 warehouse. In *SIGMOD*, pages 1161–1164, 2011.

[18] J. Prabhala. The future is now for SAP HANA, last accessed Jan. 17th 2015, June 2014. `http://www.advizex.com/blog/future-now-sap-hana/`.

[19] I. Psaroudakis, F. Wolf, N. May, T. Neumann, A. Boehm, A. Ailamaki, and K.-U. Sattler. Scaling up mixed workloads: a battle of data freshness, flexibility, and scheduling. In *TPCTC*, page accepted for publication, 2014.

[20] V. Raman et al. DB2 with BLU acceleration: So much more than just a column store. *Proc. VLDB Endow.*, 6(11):1080–1091, Aug. 2013.

[21] G. W. Records. Largest data warehouse, last accessed Jan. 17th 2015. `http://www.guinnessworldrecords.com/world-records/5000/largest-data-warehouse`.

[22] M. Rudolf, M. Paradies, C. Bornhövd, and W. Lehner. The graph story of the SAP HANA database. In *BTW*, pages 403–420, 2013.

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

[24] X. Su and G. Swart. Oracle in-database Hadoop: When mapreduce meets RDBMS. In *SIGMOD*, pages 779–790, 2012.

[25] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy. Hive: A warehousing solution over a map-reduce framework. *Proc. VLDB Endow.*, 2(2):1626–1629, Aug. 2009.

[26] Y. Xu, P. Kostamaa, and L. Gao. Integrating Hadoop and parallel DBMs. In *SIGMOD*, pages 969–974, 2010.