Data Management with SAPs In-Memory Computing Engine

Joos-Hendrik Boese
SAP AG
j.boese@sap.com

Christian Mathis
SAP AG
christian.mathis@sap.com

Cafer Tosun
SAP AG
cafer.tosun@sap.com

Franz Faerber
SAP AG
franz.faerber@sap.com

ABSTRACT

We present some architectural and technological insights on SAP's HANA database and derive research challenges for future enterprise application development.

The HANA database management system [1] was developed to meet changed requirements of modern business applications. Nowadays, these require fast and complex analytical data processing coupled with traditional transactional data management. Additionally, fast and agile decision processes that take operational data as well as structured and unstructured information into account are the current key driver in enterprises for business success. In conventional system landscapes currently found in enterprises, dedicated systems are used for analytical and transactional data processing. In contrast, HANA follows a more holistic data management approach by integrating OLTP and OLAP functionality in a single system and by adding features beyond traditional database management systems, such as graph or text processing for semi- and unstructured data. While in common three-tier architectures, compute-intensive applications run at the application server layer and data is loaded into the main memory of application servers, enterprise applications developed for or moved to HANA are more tightly integrated with the database. The main principle of application development for HANA is to execute data-intensive computations in the database close to the raw data in order to prevent expensive data movement. This shift in application design poses new challenges to the application developer: in order to utilize HANA efficiently, he has to think differently about how to design his application. We'll address these challenges and present some open questions in this area in the second part of the talk.

Categories and Subject Descriptors

H.2 Database Management [**General**]: Systems

EDBT 2012, March 26–30, 2012, Berlin, Germany.

General Terms

Design, Languages

1. SAP IN-MEMORY DATABASE

The SAP HANA database management system [1] provides the data management infrastructure for current and future SAP enterprise applications. The technical architecture and design of HANA was driven by the significant change of requirements on data management in modern enterprise applications, as well as recent developments in hardware architectures:

Over the last decades, enterprise applications could be classified as either OLAP or OLTP centric. Today, in most cases, this distinction cannot be made anymore, as more and more analytical applications require the immediate availability of operational data to support accurate decision making, while transactional applications also tend to touch large amounts of data, e.g., to calculate aggregates like sums of already delivered orders per client [4]. In addition, modern enterprise applications demand for non-standard features such as planning, optimization or predictive analysis functionality, which sometimes rely on non-relational data models, such as graphs or semi-structured data. As speed is a key factor for business success, response times of query execution for decision making must be in sub-second to seconds. This requires all data to be kept constantly online for fast querying and analytics. Therefore, HANA keeps the primary copy of its data in-memory to provide up-to-date data for fast ad-hoc processing in-memory at any time.

To provide a solution for the heterogeneous needs of complex enterprise applications, HANA embeds multiple storage and query engines supporting different domain-specific languages and targeting different data models:

1. To support the standard SQL features for enterprise applications, HANA provides a relational storage and query engine that allows access to relational data via SQL. The relational store can physically organize data in a column or row-oriented fashion, depending on expected data access patterns. Organizing data along columns bears the advantage of high compression rates and cache-efficient processing of aggregation and scan operations [3]. Thus, relations that are used for data-intensive operations are stored in a column-oriented manner. In fact, scan operations on compressed columns are so fast that in the majority of cases

there is no need to create and maintain indexes, which reduces the memory required to manage the data.

2. Since valuable insights can be gained from enhancing structured information with unstructured or semi-structured data, HANA embeds a text engine that supports text indexing and common text search features, such as fuzzy or phrase search. Such data can then be "joined" to relations of the relational store.

3. The graph engine provides access to graph structures, as commonly required in SAP's planning and supply chain applications. Domain-specific graph languages are supported to query and manipulate stored graph data.

To access and process the information stored by the different storage engines, domain-specific languages and extensions to SQL are provided. Besides SQL, supported languages at the moment are: (i) MDX for multi-dimensional expressions, (ii) proprietary languages for planning applications, and (iii) various extensions to SQL, e.g., for text search. Functionality that cannot be expressed in SQL or one of the domain-specific languages can be implemented using a procedural extension to SQL called SQL Script. SQL Script is a flexible programming language with imperative elements such as loops and conditionals allowing to define the control flow of applications. SQL Script can be coupled with standard (declarative) SQL and operators defined by HANA's domain-specific languages. Functionality and business logic that is frequently used in SAP's enterprise applications, such as currency conversion, is implemented in a function library natively in the database kernel. These functions can be programmed with a maximum degree of parallelism, since the library developer is in full control and can implement parallel execution on a lower level. Therefore, these operators are much more tightly coupled with the database kernel in contrast to classical stored procedures implemented in SQL Script. These native functions can then be called in procedural SQL Script code. Stored procedures and domain-specific languages are translated into an internal query processing structure called the "Calculation Model" [2]. Calculation Models may contain native operators implemented by different query engines or operators defined in the native function libraries.

By integrating support for multiple data models and languages, HANA establishes a holistic data management platform for SAP's enterprise applications that allows to speed up existing applications, and enables the development of completely new types of applications. To illustrate this, we briefly present two examples:

- Availability-to-Promise (ATP) Check: ATP is a classical SAP application in logistics as well as sales and distribution. ATP checks are typically run during sales-order processing, where a customer or an agent tries to confirm, whether or not an ordered quantity of some product can be delivered at a specific future point in time. The ATP check is carried out on database tables representing the current stock situation, planned goods issues and planned goods receipts. To compute the ATP check, this information has to be transformed into a time series (representing goods movements). In the classical three-tier R/3 solution the database had to be alleviated from compute-intensive processing. In consequence, ATP time series are kept in a redundant form as pre-computed persistent aggregate tables. The code responsible for updating and keeping the redundant data in sync runs outside the database system in the application server, to reduce the workload on the backend. With SAP HANA and its on-the-fly calculation, we have the opportunity to abolish persistent aggregates and to calculate the ATP time series at query runtime. Furthermore, we can express the ATP check logic in SQL Script and push it into SAP HANA [5]. This simplifies code, allows greater flexibility (e.g., adjustable time buckets) and avoids maintenance and consistency checks against persistent aggregates.
- Analytics of patient's records in cancer treatment: documentation of diagnosis and therapies of cancer patients in hospitals includes various records in natural language such as doctor's notes, manually entered diagnoses texts or therapy plans. Various analysis and planning operations in daily operations of hospitals require to identify cohorts of patients with similar diagnoses, therapies or other similarities, such as survival status, tumor types etc. HANA's text engine and its entity recognition and fuzzy search features enable a large german hospital to compute flexible reports on their patients that include relations transparently generated from semi-structured patient records.

While the ATP application is an example for an enterprise application that was revised and redesigned for SAPs new in-memory computing engine, the second application is an example for a new application enabled by HANA. The possibility of running flexible ad-hoc analytics on operational data is expected to enable numerous new applications in the realm of enterprise software, as well as other application areas.

To leverage the performance of this new database layer, applications have to be revised to push application logic down to the data, i.e., from the application layer into the database. This poses some new challenges for application development described in the second part of this talk.

2. DEVELOPING THE NEXT GENERATION OF BUSINESS APPLICATIONS

Pushing application logic down to the database contrasts the classical three-tier pattern, which has been widely implemented in practice. In the classical architecture the database was considered as the computational bottleneck and the middle-tier was responsible for processing compute-intensive operations. The rationale behind this design pattern was that the application server layer could be easily scaled-out by adding additional machines, while the database management systems could only be scaled-up, which was uneconomical. Thus, the database server had to be protected from compute-intensive operations. With the ever increasing computing power and main memory sizes of modern hardware architectures, scaling-up the database server is technically and economically feasible today as proved by the HANA appliance.

Having a database management system that integrates heterogeneous domain-specific features in addition to significant computing power, changes the way SAP's enterprise applications can be designed. To make use of the full potential of SAP's in-memory computing engine, the developer has to decide about what parts of his application are data-intensive. This part can then be pushed down into the database, where it is executed close to the primary data structures of the storage engines. This eliminates data movement from the database into application servers, which, in fact, is the main bottleneck for data-intensive applications in traditional three-tier architectures today. Identifying data-intensive operations is sometimes trivial and sometimes tricky. Application developers could for example be guided by best practices, design patterns, tools for code analysis, profiling, etc. Another approach is to decide at execution time, whether for a given setting, either data is moved from the database to the application server or whether code should be shipped to the database to be executed there.

To utilize the computing power of modern server platforms, user-defined logic must exploit parallelism provided by the abundance of compute cores. Also, data-specific code- and runtime optimizations must be applied to execute application code efficiently. Since SAP's in-memory database technology is evolving from a classical database system to a multi-purpose data analysis engine, runtime optimizations and transparent parallelization must also be applied for non-SQL (procedural) parts of the application code. Focusing on well understood SQL query optimization is not sufficient anymore. Automatic optimization and parallelization of arbitrary procedural parts of application code is required. Algorithms implemented in domain-specific languages or in SQL Script must scale over multiple cores.

One solution is to provide language features, that allow application programmers to describe how the in-memory execution engine can optimize and parallelize user code: to address different types of application developers, the system can allow for coding at different levels of abstraction. For example, some application logic can be implemented using graphical tools such as SAP's HANA modeler. Encorporating parallelism hints on this level can be done by providing split/merge operations that split/merge data streams for parallel processing. Again, other parts of the application can be directly implemented in SQL Script, where parallelism can be formulated, for example, using `parallel for` operations or functional patterns like map/reduce. At the level below, we can provide an infrastructure that generalizes and modularizes existing database-internal algorithms and data structures for easy re-use and re-combination in application code running in the database system.

For convenient development of arbitrary data-intensive logic in SQL Script, such a language will need to evolve from a pure procedural SQL extension to a more complete programming model, supporting features such as modularization, standard libraries, debugging tools, exception handling, etc. The way these challenges are addressed and solved are fundamental, since they will define the way we program future enterprise applications for SAPs in-memory engine.

3. REFERENCES

[1] F. Färber, S. K. Cha, J. Primsch, C. Bornhövd, S. Sigg, and W. Lehner. SAP HANA database: data management for modern business applications. *SIGMOD Rec.*, 40(4):45–51, Jan. 2012.

[2] B. Jaecksch, F. Faerber, F. Rosenthal, and W. Lehner. Hybrid data-flow graphs for procedural domain-specific query languages. In *Proceedings of the 23rd international conference on Scientific and statistical database management*, SSDBM'11, pages 577–578, Berlin, Heidelberg, 2011. Springer-Verlag.

[3] H. Plattner. A common database approach for OLTP and OLAP using an in-memory column database. In *Proceedings of the 35th SIGMOD international conference on Management of data*, SIGMOD '09, pages 1–2, New York, NY, USA, 2009. ACM.

[4] H. Plattner and A. Zeier. *In-Memory Data Management: An Inflection Point for Enterprise Applications*. Springer Verlag, Heidelberg, 2011.

[5] C. Tinnefeld, S. Müller, H. Kaltegärtner, S. Hillig, L. Butzmann, D. Eickhoff, S. Klauck, D. Taschik, B. Wagner, O. Xylander, A. Zeier, H. Plattner, and C. Tosun. Available-to-Promise on an in-memory column store. In *BTW*, pages 667–686, 2011.