Data Mining in Life Sciences: A Case Study On SAPs In-Memory Computing Engine

Joos-Hendrik Boese[1], Gennadi Rabinovitch[1], Matthias Steinbrecher[1], Miganoush Magarian[1], Massimiliano Marcon[1], Cafer Tosun[1], and Vishal Sikka[2]

`<firstname>.<lastname>@sap.com`
[1] SAP Innovation Center, Prof.-Dr.-Helmert-Str. 2-3, 14482 Potsdam, Germany
[2] SAP, 3410 Hillview Ave, Palo Alto, CA 94304, USA

Abstract. While column-oriented in-memory databases have been primarily designed to support fast OLAP queries and business intelligence applications, their analytical performance makes them a promising platform for data mining tasks found in life sciences. One such system is the HANA database, SAP's in-memory data management solution. In this contribution we show how HANA meets some inherent requirements of data mining in life sciences. Furthermore, we conducted a case study in the area of proteomics research. As part of this study we implemented a proteomics analysis pipeline in HANA. We also implemented a flexible data analysis toolbox that can be used by life sciences researchers to easily design and evaluate their analysis models.

Topics: Data mining and data analysis in real-time, Case studies

Submission type: Industry paper

1 Introduction

The amount of data generated by life science research has been rapidly increasing over the last years. Examples include genome sequencing and mass spectrometry for proteomics. This data requires extensive and complex analysis, thus demanding high performance from data management systems. Today these analysis processes are typically implemented using heterogeneous systems, applications and scripts. The data sources are also diverse, with data distributed over different files or systems and being in different formats. Hence supporting heterogeneity of application logic and data sources is a critical requirement on data management systems in this domain.

While Business Intelligence (BI) and other analytical enterprise systems have traditionally operated on pre-aggregated data, recent technologies are enabling analysis of highly structured multi-dimensional data directly on the raw data sources in real-time [1]. Moreover, modern analytical systems include interactive tools allowing the data scientists to explore data using relational operations and interact with the data at the "speed of thought". At the same time, these systems evolved from pure SQL query engines to multipurpose execution engines that can execute arbitrary logic directly on the data.

Such a system is the SAP HANA in-memory database [2]. In contrast to traditional disk-based relational databases, SAP HANA keeps its primary data permanently in memory in order to achieve high analytical performance without pre-computed index structures. This enables fast execution of complex ad-hoc queries even on large datasets. Furthermore, HANA includes multiple data processing engines and languages for different application domains. These characteristics make HANA a promising platform to meet the data management requirements arising in life sciences.

In the scope of our life science projects we work in close collaboration with research centers, universities and bioinformatic companies. Our goal is to provide the life science research community with fast and flexible data management tools based on HANA. In one of our projects, in collaboration with the Freie Universität Berlin, we concentrate on proteomics research. Proteomics is the branch of molecular biology that deals with the analysis of the tens of thousands of proteins present in a living organism. A very important application of proteomics research is the detection of biomarkers associated with different diseases including cancer. In order to detect these biomarkers, researchers must be able to flexibly analyze terabytes of data about the distribution of proteins in blood samples.

In this paper we present our case study in the field of proteomics and describe how SAP HANA in-memory database technology is used to support proteomics research. Moreover we present a flexible and extendible framework for the real-time analysis of proteomics data that we developed based on HANA. We conclude that although HANA is initially designed for enterprise analytics, but it is also well-suited for data management and alaytics in life sciences.

The rest of this paper is structured as follows. In Section 2 we describe the SAP HANA in-memory database architecture and its distinctive features. A set of requirements for life science database systems that we identified during our collaboration with lifes cience researchers is summarized in Section 3. In Section 4 we then provide a brief introduction to the field of proteomics, sketch the analysis pipeline used in this case study and describe the implementation of our HANA-based proteomics data analysis framework. Finally, Section 5 concludes the paper and presents some of our future work.

2 The SAP HANA Database Management System

The SAP HANA database management system [2, 3] provides the data management infrastructure for SAP's future enterprise applications. The technical architecture and design of HANA was driven by the increased demand of modern enterprise applications for analytical performance as well as recent developments in hardware architectures. Requirements of modern enterprise applications include complex data mining and machine learning functionalities, increased data volumes, and being able to answer ad-hoc queries on large datasets within seconds or subseconds.

In modern enterprise applications data analysis tasks go beyond classical reporting such as OLAP queries and call for non-standard features like planning, optimization, or predictive analysis features. Often, these tasks rely on non-relational data models, such as graphs or semi-structured data. To efficiently support these tasks, HANA keeps the primary copy of the data constantly in memory. Moreover, HANA supports a high-level of parallelism and cache conscious execution of queries. This enables answering ad-hoc queries quickly and at any time without costly pre-computation of special index structures as usually done in traditional data warehouse systems. HANA follows a holistic approach to data management by consolidating transactional (OLTP) and analytical (OLAP) workloads in a single system.

To meet the requirements of a number of heterogeneous and complex data-intensive applications, HANA includes multiple storage and query engines:

Relational engine: The majority of enterprise applications rely on relational data accessed via SQL. To support the standard SQL features, HANA provides a relational storage and query engine. Depending on the expected workload, the relational store can physically organize data column-wise or row-wise. Column-oriented data organization favors analytical processing of the data, since it allows cache-efficient processing of analytical operators and high compression rates [4]. In practice it turned out that scan operations on compressed columns are so fast that in most cases there is no need to create and maintain indexes, thus reducing the memory consumption.

Text engine: To discover information hidden in unstructured text and combine it with structured information, HANA embeds a text engine that supports text indexing and a comprehensive set of text analysis features. Supported features range from various similarity measures to entity resolution capabilities and allow phrase or fuzzy search on text indexes.

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

Besides the ubiquitous SQL, HANA supports several domain-specific languages to process data managed by the different storage engines. These languages include: (i) MDX for multi-dimensional expressions, (ii) a proprietary language for planning applications called FOX, (iii) SQL extensions for text search, and (iv) a proprietary language called WIPE to access and manipulate graph structures.

Functionality that cannot be expressed in SQL or one of the above languages can be implemented using a procedural extension to SQL called SQLScript. SQLScript adds imperative elements to SQL such as loops and conditionals allowing to define the control flow of applications. SQLScript can be coupled with operators defined by HANA's domain-specific languages. If performance is paramount, functionality can be directly implemented in C++ using native database interfaces within the database kernel. Since in this case the developer can implement parallel execution at a very low level, these native functions can be programmed with maximum control on parallelism during execution.

HANA ships with multiple preinstalled C++ libraries that implement frequently required functionality in enterprise applications. Examples of such li-

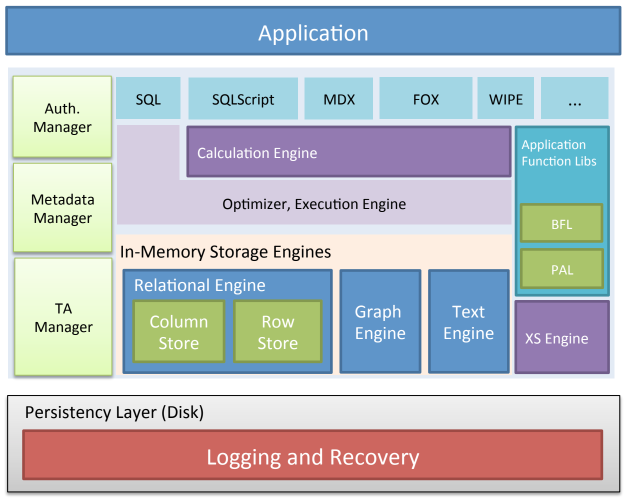

Fig. 1. Overview of the HANA architecture [2]

braries include the business function library (BFL) and the predictive analytics library (PAL). The PAL implements standard machine learning algorithms, while the BFL provides functions frequently used in enterprise applications such as currency conversion etc.

Figure 1 depicts an architectural overview of HANA with its different languages, storage engines, and libraries. In order to execute logic implemented in different domain-specific languages or available in function libraries, HANA generates complex programs called “calculation models” [5, 3]. A calculation model defines the data-flow graph of a program executed by HANA’s calculation engine. The edges of the data-flow graph connect data sources (i.e. tables, views, or results of other calculation models) and operator nodes. Operator nodes are functions implementing arbitrary logic defined in a domain-specific language and can have multiple input edges. Calculation models can also be exposed as database views and thus be easily reused as data sources for other calculation models or standard SQL queries. Current types of operator nodes in a calculation model are:

Relational operator nodes. All common relational operators (such as join, union, projection etc.) that handle classical relational query processing can be embedded.

R operator nodes. R nodes allow to embed logic defined in the R [6] language into the calculation model. In this case the data is copied into an R execution environment outside of the HANA database kernel. Results are then moved back into the calculation engine and integrated into the data flow of the calculation model.

L operator nodes. L is the internal runtime language of the HANA database and a subset of C. Usually it is not accessible for end users, but used as an intermediate language into which all domain-specific languages are compiled.

Custom nodes. Custom nodes allow access to native functions implemented in C++ and are tightly coupled with the database kernel. Examples include functions defined in PAL or BFL.

Split and merge nodes. In addition to functional operators, split and merge operators are provided to define application specific data-parallelization. Split nodes partition the data for parallel processing, while merge nodes are used to combine results from different data partitions.

While SQLScript programs are transparently compiled into optimized calculation models, these models can also be defined explicitly using either an XML description or a graphical modeling tool which is part of HANA Studio. HANA Studio is a client application for creating data models and stored procedures, or manage tables and issue queries interactively. It provides graphical editors for calculation models, thus allowing to easily define and manipulate analysis pipelines for enterprise applications as well as for life science research.

To conveniently manage the lifecycle of calculation models, HANA includes a built-in repository that is used to manage calculation models and other development artifacts, such as views and procedures. The built-in repository provides the basis for concepts like namespaces, versioning, and software component delivery.

Besides defining and managing calculation models, visualization and consumption of results is crucial. The Extended Scripting Engine (XS Engine) is a HANA component that supports consumption of results via an HTTP REST interface, e.g. for visualization purposes. Therefore, in most cases, developers do not need to integrate additional software components such as web servers or write application-specific code for data access. However, if the application requires some very specific mapping functionalities, these can be easily incorporated into the XS Engine in the form of additional software modules.

3 Requirements of Data Management in Life Sciences

In life sciences the ability to manage and analyze large data sets is crucial. Data analysis in life sciences is often carried out through a diversity of scripts and applications developed in different languages. Data sources are also heterogeneous, with data spread across different systems and stored in different formats. This combination of heterogeneous applications and data sources requires costly data transfers and conversions, as well as the knowledge of different programming languages and models. Maintaining and processing all scientific data in a single centralized system would eliminate the need for expensive data transformation, thus enhancing performance and enabling researchers to concentrate on analysis algorithms rather than on data transformation tasks.

The database community recognized the mentioned problems and studies have been carried out to identify requirements for data management in life sciences [7].

Specialized scientific database systems, such as SciDB [8], have been developed to support life science applications. Other approaches like [9] focused on integrating data from multiple databases into a federated database system allowing for a unified query interface for heterogenous life science data.

In the remainder of this section we describe some of the important requirements we identified while supporting life science research groups, and explain how HANA addresses these requirements. Some of the identified requirements are in line with the general requirements for scientific databases mentioned in [7].

3.1 Tight integration of scientific data and analysis algorithms

Consolidating different data sources and applications into a more homogeneous, centralized system removes the need for most data transfers and conversion across system boundaries. Nevertheless, data still needs to be transferred between the database's persistency layer and the application for computation, thus causing a significant performance loss. To reduce data movement between database and application the database needs to be able to execute user-defined analysis algorithms directly on the data.

As mentioned in Section 2, HANA addresses the heterogeneous needs of complex enterprise applications by embedding multiple storage and query engines that support different domain-specific languages.

Furthermore, HANA is highly extensible and enables users to implement their domain-dependent application logic at various abstraction levels, ranging from high-level SQLScript procedures to native function libraries. This allows for more flexibility than traditional stored procedures or user-defined functions. As described in Section 2, application logic can be pushed down to be executed on primary copy of data which is hold in memory resulting in high performance. Since all application logic is executed by the calculation engine directly on the data, there is no need for any data transfer beyond system boundaries.

3.2 Intuitive interface for the design of analysis pipelines

Life science researchers continuously design and test new analysis pipelines. This involves changing data sources, operations and parameters. Furthermore, researchers need to compare the results generated by different pipelines. Therefore, providing an easy-to-use and intuitive framework to interactively design analysis pipelines would greatly improve researchers' productivity.

In HANA, analysis pipelines are defined through calculation models, which can be designed in HANA Studio through a graphical interface. This interface can also be extended with domain specific operator nodes, e.g. custom nodes organized in toolboxes for different application domains.

3.3 Versioning of Algorithms and Data

Analyzing scientific data and comparing different data mining models is a complex task. Depending on some intermediate results, the requirements for further data evaluation and analysis may change over time. This often results in modifications of the algorithms or even whole data analysis processes. Discarded approaches as well as all the intermediate results are usually thrown away. If this intermediate data is needed at some later point, its reconstruction is often difficult and time-consuming. Versioning of algorithms and results helps addressing this problem. More specifically, in the case of a database system, both database objects (e.g. tables, stored procedures, user-defined functions etc.) and the stored data need to be put under version control.

Because every calculation model activated in HANA is stored and registered in a repository (see Section 2), it is easy to re-activate previously executed analysis steps. Also, because data in HANA is modified in a transactional context, every data record is associated with a transaction identifier. A mapping of transaction identifiers to revisions of calculation models can be used to easily identify the calculation model used to generate a specific record.

3.4 Support for non-relational data structures and operations

Processing of scientific data often requires domain-specific data structures and operations. While relational models are widely used to store scientific data and the associated metadata, emulating data structures such as graphs or hierarchies on top of pure relational tables results in poor performance. In order to achieve adequate performance, basic data structures such as graphs should be supported natively. HANA's holistic data management design follows this approach by integrating multiple in-memory storage engines for non-relational data such as graphs and including specific operators to access and manipulate non-relational data.

3.5 Big data support

Scientific datasets are steadily increasing in size, thus requiring tools to store, access, and mine terabytes or petabytes of data. While large amounts of data must be stored and managed, the more complex and specific analysis tasks often touch only a fraction of this data. For example, data is often filtered or aggregated before an analysis model is run on the filtered data. For such filter and aggregation tasks Map-Reduce frameworks are frequently used. These frameworks scale transparently to large numbers of commodity machines, thus allowing to compute simple pre-processing, filter, and aggregation tasks in a massively parallel manner.

To exploit Map-Reduce implementations such as Hadoop for pre-processing or filtering of data sets that are too big to be loaded into HANA or that are already residing in a Hadoop cluster, some extensions need to be made to HANA. According to SAP's recently announced Unified Database Strategy [10], HANA is going to support the access of such “big data” sources and offer a deeply integrated pre-processing architecture, allowing it to efficiently incorporate pre-processed data and to initiate post-hoc analyses with a Map-Reduce implementation.

4 Case study: Proteomics research

In this section we describe a proof-of-concept implementation that shows how HANA supports life sciences applications. Specifically, our contribution focuses on proteomics research. In Section 4.1, we first provide some general information about proteomics research and the analysis pipeline that we target in our HANA-based framework. In Section 4.2 we describe the implementation of this analysis pipeline in HANA.

4.1 Proteomics research

Proteomics is the area of life sciences which focuses on the study of the proteome, a term referring to the totality of proteins expressed by a genome, cell, tissue or organism at a given time.

The human proteome is highly dynamic, changing constantly as a response to the needs of the organism. Depending on factors such as sex, age, diet and stress, the proteome differs widely between different individuals. Other factors, like cancer or other diseases, can also change the composition of the proteome. These changes in the proteome are indicated by the presence or absence of certain proteins or protein combinations. An important application of proteomics is to analyze the proteome of different groups of people, i.e. male/female or healthy/cancer patients, and to search for patterns within these groups that clearly assign a single patient to the appropriate group. These pattern can then be used as biomarkers for the respective group, i.e. in cancer diagnosis.

A prominent way of analyzing the proteome is to use mass spectrometry, a technique that measures the masses and concentrations of proteins in a biological sample and exports the data in form of a spectrum. An example of mass spectrum is shown in Figure 2. In our case study, blood samples are analyzed using the MALDI-TOF[1] mass spectrometry technique which is illustrated and described in Figure 3. The process is carried out multiple times per sample to increase the coverage, leading to a multitude of spectra per blood sample.

Spectrum data. The analysis of a single blood sample results in 1,500 to 2,000 spectra with approximately 100,000 data points (mass/count pairs) each. Thus, each sample results in 150-200 million data points. Because these data points are generated by external tools, they are usually provided in CSV[2] or specific mass spectroscopy data formats such as mzXML[3]. The raw data we use for our case

[1] Matrix-assisted Laser Desorption/Ionization, Time-of-Flight mass spectrometry
[2] Comma-separated values
[3] `http://tools.proteomecenter.org/wiki/index.php?title=Formats:mzXML`

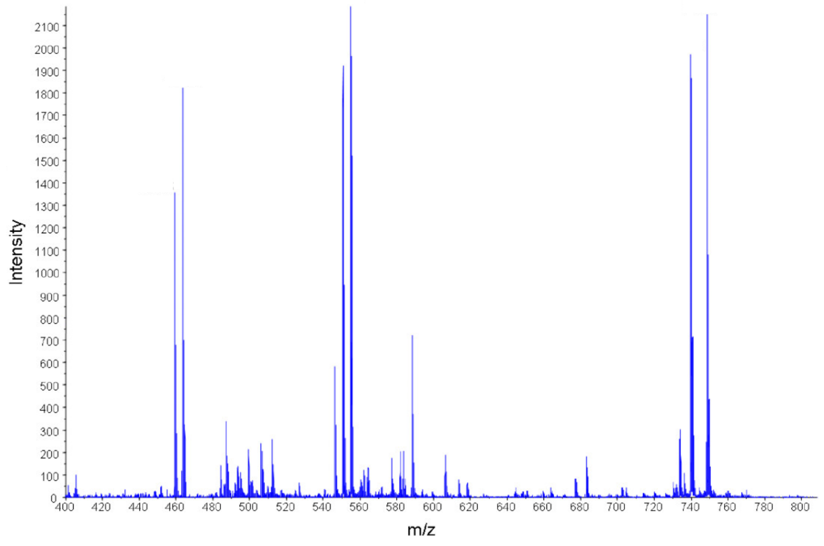

Fig. 2. Portion of a raw protein mass spectrum showing pronounced peaks in the concentration (y-axis) of proteins corresponding to certain mass-to-charge ratios (x-axis). Peaks in the concentration of certain proteins may indicate the presence of a disease, e.g. cancer.

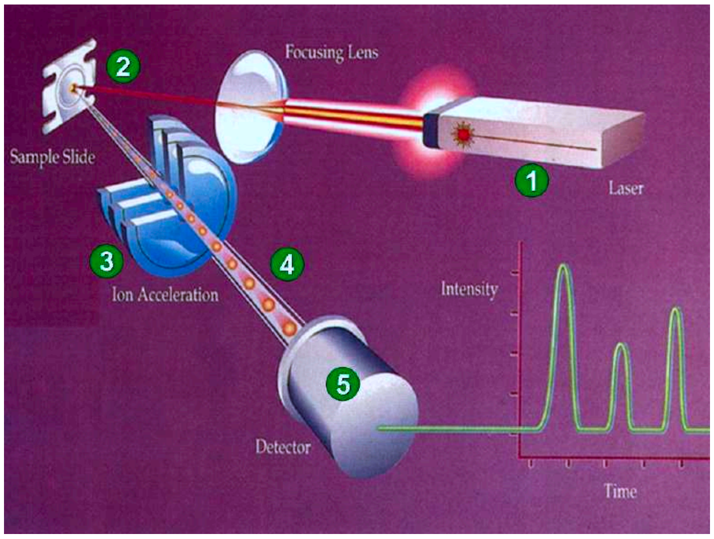

Fig. 3. Operation of a modern MALDI-TOF mass spectrometer. A laser beam (1) vaporizes and charges the molecules in the sample (2). These are accelerated in an electrical field (3) and their time of flight in the drift region (4) is measured by a detector (5). (picture taken from [11], therein modified and cited from [12]).

study amounts to two gigabytes per sample. When this data is imported into the HANA database, in-memory compression reduces its size to about one gigabyte. More details on the compression ratio are provided in Section 4.2.

Analysis Pipeline. In this study, we consider the analysis pipeline proposed in [11]. This pipeline takes as input raw mass spectra and produces a predictive model that can be used to classify patients across relevant groups, such as healthy or diseased. The pipeline consists of four major steps:

1. **Preprocessing**
   Removal of noise and other systematic errors from the mass spectrum.
2. **Peak-Seeding**
   Identify peaks within the preprocessed spectra. This includes the identification of potential peaks and the derivation of features describing each peak.
3. **Peak-Picking**
   Examine the peaks across a group of samples (e.g. healthy patients) and group similar peaks occurring across the samples into clusters relevant for further analysis. These clusters of peaks are called master peaks.
4. **Analysis**
   Find master peaks that are able to separate the group of healthy patients from the group of diseased and therefore contribute to a corresponding biomarker.

The peak-seeding phase finds relevant peaks in a single raw spectrum. Each peak in the spectrum is described by the parameters of a Gaussian function which is fitted to the raw data points contributing to this peak. This process also includes deconvolution of overlapping peaks. The result of the peak-seeding phase is a set of peaks from all spectra of all samples, where each peak is described by a distribution and additional parameters such as start mass, end mass, contributing points, raw height, and fit quality.

In the peak-picking phase, for each group of samples (e.g. healthy patients) similar peaks occurring across the samples of the group are then grouped into clusters of peaks. A cluster of similar peaks defines a so-called master peak.

Finally, the set of master peaks of two different groups (e.g. healthy/diseased) is analyzed to determine a subset of master peaks that is successfully able to distinguish between the two groups. This subset of master peaks can then be used to classify new samples and is the base for developing novel medical tests e.g. for cancer diagnosis.

As the data flows through the analysis pipeline, its size diminishes: The raw unprocessed spectrum in step 1 constitutes the largest data volume. The output of the peek-seeding phase are only those parts of the spectrum that represent actual peaks, thus introducing some degree of semantic meaning and reducing the data (10%–20% less than the raw spectrum size). The remaining peak-picking and analysis steps further reduce the size of data passed on in the analysis pipeline.

The reduction of data within each processing step depends on the resolution of peak-seeding and picking, i.e. whether small peaks are either considered noise or are passed on to subsequent analysis steps. Since master peaks with a relatively small height in the raw data might make the difference in identifying relevant biomarkers, researchers change thresholds for peaks to be considered in peak-picking and analysis. Identifying biomarkers is, therefore, an interactive process that requires re-running of the analysis pipeline in an iterative process. Therefore, fast and predictable responses are crucial to support the workflow of proteomics researchers operating this pipeline.

4.2 Proteomics on HANA

We implemented the proteomics analysis pipeline described in Section 4.1 in the HANA database using multiple calculation models that together define the proteomics analysis pipeline. Moreover, we provide a graphical editor that enables interactive execution and manipulation of the analysis pipeline. Besides integration of proteomics-specific operators, the editor also enables an easy integration of operators from the PAL.

Implementation of the proteomics analysis pipeline. Data received from the MALDI-TOF mass-spectrometer is pre-processed outside of HANA to remove systematic errors and noise. The pre-processing step is executed outside of HANA because it requires only one pass through the data and can thus be easily performed while sequentially reading the raw data from files. The resulting pre-processed spectra are then loaded into HANA's column-store in a single import operation. The product of the import operation is a single column table containing the pre-processed mass-spectra for all samples.

Storing the spectrum data in a column table allows for compression of the data at a ratio of approximately 1:2. Although a mass spectrum contains mainly double and integer values, the main memory required to store a single sample is about 1.4 GB compared to 2.4 GB required in a CSV file. Studies in proteomics research rarely consider more than 1,000 samples due to expensive data acquisition, so we assume that a HANA appliance with 2 TB of main memory has sufficient capacity for most studies. In case the number of samples exceeds the available main memory, HANA can be scaled out among multiple machines.

The peak-seeding phase requires the determination of Gaussian components using standard minimization methods. Since minimization is an iterative process, this algorithm was implemented in C++ within the database kernel for a maximal performance. The peak-seeding function operates on single spectra of a sample and is, therefore, easily parallelizable.

The peak-picking step is not implemented in C++ but in L and SQLScript since the data sizes the peak-picking step operates on are much smaller than the spectrum data processed during the peak-seeding step. The peak-picking step also makes use of functions implemented in the PAL.

Subsequent analysis steps of the pipeline aggregate master peak data further, e.g. by computing statistical measures of master peaks and comparing peaks between two sample groups (e.g. between healthy and diseased patients) to select

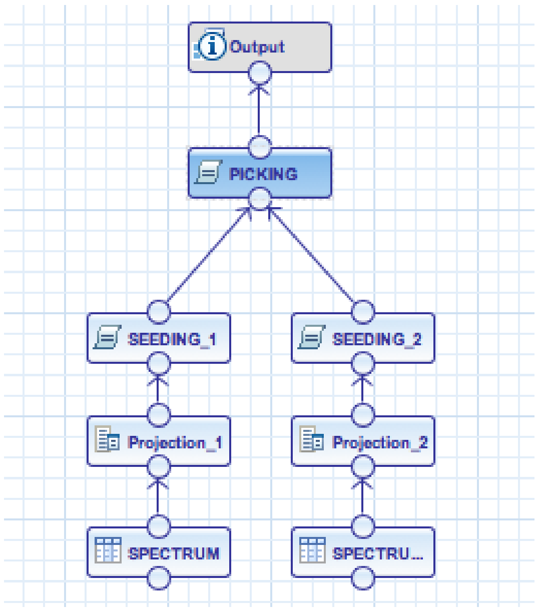

Fig. 4. A fragment of the proteomics analysis pipeline, that includes peak-seeding and peak-picking steps

appropriate biomarkers. These steps work on master peak data and can, therefore, be expressed efficiently in SQL, SQLScript, and PAL operators.

Calculation model editor. We used the calculation model described in Section 2 to implement the proteomics analysis pipeline. To design the calculation model we extended the HANA Studio modeling tool with a data-mining plugin that enables integration of domain-specific L and Custom operator nodes. For our case study we implemented a domain specific library for proteomics research. In addition to domain-specific operator nodes, the data-mining plugin contains nodes for standard machine learning algorithms, such as k-means and C.4.5, which are implemented in the PAL. The user can simply drag and drop operator nodes and orchestrate them in the data-flow graph of the calculation models.

Figure 4 depicts a calculation model created in HANA Studio using our data-mining plugin. The depicted model is part of the analysis pipeline that includes peak-seeding and peak-picking steps. The peak-seeding and peak-picking operator nodes are provided by our proteomics library. The data sources are database tables with sample spectrum data. To implement the peak-seeding step, we connect each data source to a peak-seeding operation node to detect the peaks within the spectra of the data source. We also interpose a projection operator between the data source and the peak-seeding step, its purpose is to map data source attributes to the signature of the operator node. Finally, peak-picking is performed by grouping the peaks identified in both spectra sources during the peak-seeding phase. The final results are provided by an output operator that projects the relevant attributes of the table containing the results of the peak-picking phase. The results can then be used as a data source in other calculation models.

After designing the analysis pipeline, the model can be activated and executed on the HANA calculation engine and results can be previewed in the form of diagrams and tables. Therefore, the tool constitutes a flexible design interface where the user can customize and adjust the pipeline steps, e.g. peak-picking, by providing different values for input parameters or by easily change the operators in the calculation model. In this way, the user can modify and re-run analysis pipelines and conveniently compare their results.

5 Summary and Conclusion

In this paper, we identified a set of requirements for data analysis and management in life sciences research and described how HANA meets these requirements. We made the point that data management tools like HANA, although initially designed for enterprise analytics and BI, are also a good fit for data management in life sciences.

As a concrete example of a life sciences application on top of HANA, we presented a proof-of-concept implementation of a proteomics analysis pipeline within the HANA database kernel and described how different analyses were implemented and how the pipeline was orchestrated in calculation models. The concept of calculation models in HANA and the high analytical speed allow life sciences researchers to interactively analyze data using graphical modeling tools and to easily design and modify their analysis tasks. We showed how an extension to the HANA Studio can be used for such explorative domain-specific data analysis.

Although our work is a promising first step, much remains to be done in order to successfully use in-memory databases for life sciences. While our work thus far focused on flexibility and extendibility, our next goals are visualization of results and performance evaluation. With respect to visualization we plan to extend our framework with interactive visualization components that allow users to easily inspect the analysis results. Moreover, we plan to make it easy for applications to consume the results by exposing them through standard HTTP using HANA built-in XS Engine. With respect to performance, we plan to benchmark our solution against DBMSs currently used for life sciences research.

References

1. Plattner, H., Zeier, A.: In-Memory Data Management: An Inflection Point for Enterprise Applications. Springer-Verlag (June 2011)
2. Färber, F., Cha, S.K., Primsch, J., Bornhövd, C., Sigg, S., Lehner, W.: SAP HANA Database: Data Management for Modern Business Applications. In: SIGMOD Rec. Volume 4 of 40. (2012) 45–51

3. Sikka, V., Färber, F., Lehner, W., Cha, S.K., Peh, T., Bornhövd, C.: Efficient transaction processing in SAP HANA database: the end of a column store myth. In: Proceedings of the 2012 international conference on Management of Data. SIGMOD '12, New York, NY, USA, ACM (2012) 731–742
4. Plattner, H.: 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, ACM (2009) 1–2
5. Jaecksch, B., Faerber, F., Rosenthal, F., Lehner, W.: 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, Berlin, Heidelberg, Springer-Verlag (2011) 577–578
6. Venables, W.N., Smith, D.M.: An Introduction to R. Notes on R: A Programming Environment for Data Analysis and Graphics Version 2.15.0. R-project.org. (March 2012)
7. Stonebraker, M., Becla, J., DeWitt, D.J., Lim, K.T., Maier, D., Ratzesberger, O., Zdonik, S.B.: Requirements for Science Data Bases and SciDB. In: CIDR. (2009)
8. SciDB.org: The Open Source Data Management and Analytics Software for Scientific Research. http://www.scidb.org/Documents/SciDB-Summary.pdf
9. Haas, L.M., Schwarz, P.M., Kodali, P., Kotlar, E., Rice, J.E., Swope, W.C.: Discoverylink: a system for integrated access to life sciences data sources. IBM Syst. J. **40**(2) (February 2001) 489–511
10. SAP: SAP Unveils Unified Strategy for Real-Time Data Management to Grow Database Market Leadership. SAP News. http://www.sap.com/corporate-en/press.epx?PressID=18621 (April 2012)
11. Conrad, T.O.F.: New Statistical Algorithms for the Analysis of Mass Spectrometry Time-Of-Flight Mass Data with Applications in Clinical Diagnostics. PhD thesis, Freie Universität Berlin (2008)
12. MALDI-TOF Mass Analysis: `http://www.protein.iastate.edu/maldi.html`