Big Data Management in SAP HANA

Tina Philip
Computer Science Department
San Jose State University
San Jose, CA 95192
408-924-1000
talk2tinaphilip@gmail.com

ABSTRACT

Enterprise applications have become quite complex and demanding in the past years and many applications require processing of analytical transactions or running reports, while many users consume and update the data. The growth in capacity of main memory enabled a completely new database paradigm called in-memory columnar database. SAP High Performance Analytics Appliance (HANA) was made with the goal of combining analytical and transactional data into the same database. This is made possible with three main factors namely, multicore parallel processing, availability of faster and cheaper main memory and the use of columnar architecture. The columnar database uses effective compression techniques, maximum parallelization of database kernels and specialized data structures to support the complete data lifecycle which involves modeling, provisioning and consumption. In this paper, we delve into the scenario for the emergence of SAP HANA, its architecture and advantages. We will also provide an in-depth analysis of in-memory columnar database technology. Finally we look at SAP HANA Vora, the in-memory engine that connects SAP HANA to Hadoop, Spark and other ecosystems for handling big data.

1. INTRODUCTION

In the last few decades, with the advent of multicore processors and exponential increase in memory capacity with a significant drop in price, it has become possible to build memory systems in which all or most of the data can be stored in the main memory itself. To quote Jim Gray, "Memory is the new disk, disk is the new tape" [1]. We are overlooking a trend where disks are sidelined to archival duties while memory takes center stage for storing. As shown in Figure 1, accessing data from memory is around ten thousand times faster than accessing it from the disk [2]. By the late nineties, it was clear that Moore's law was largely attained because of improvements in the single core CPU itself and to meet the mounting performance requirements, multicore processors were a necessity. The advent of multicore processing, massively larger and cheaper main memory and columnar database structure paved the way for SAP HANA and in-memory computing, which brought a drastic change to the existing database technology landscape. In-memory databases use the main memory not just as a cache, but to store, compute and retrieve huge volumes of data by using row and column storage. This eliminates the need for indexes which in turn acts as a key area of compression.

In the following sections, we will cover the background and working of SAP HANA. Section 3 covers the architecture and Section 4 covers the features and advantages that SAP HANA offers. In section 5 we will cover SAP HANA Vora and its architecture, components and features. We will also cover the sample code for the integration of Vora with Spark and Apache Hadoop in the following subsections. Finally we look at partitions and hierarchies in Vora.

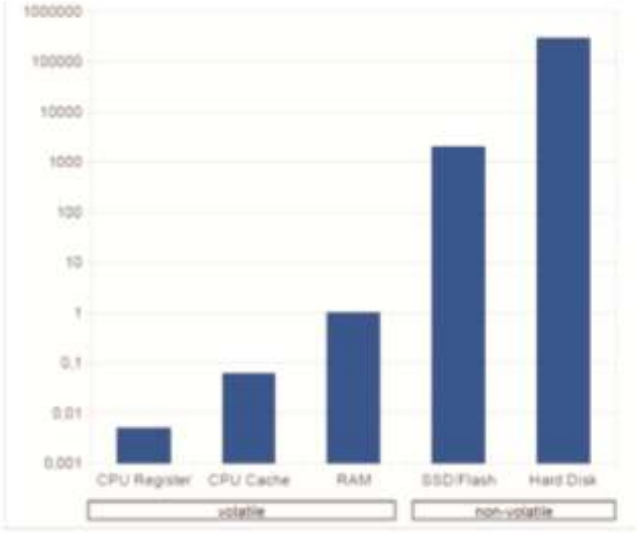

Figure 1. Data access times of various storage types, relative to RAM (logarithmic scale) [2]

2. BACKGROUND

In the early days, disks were the primary storage source and all operations required accessing the disk, making disk-seek time a key parameter in raising the execution time. As the operations became more complex and execution time needed to be improved, companies designed specialized databases to support and accelerate a specialized workload such as, analytical, graphical or transactional data. But the disadvantage of this method was that, the specialized databases required a separate copy of data to perform complex operations. This created problems with duplication and synchronization of the multiple copies of data. Even though extract, transform and load tools commonly known as ETL tools were employed to synchronize the data, the result could not be obtained in real-time to provide actionable insights. The push to accelerate processing time and the breakthrough advancements in hardware technology has resulted in in-memory databases. SAP HANA leverages in-memory database technology

with multi-core processors for advanced analytical processing, faster response time and immediate decision making.

3. SAP HANA ARCHITECTURE

At its core, SAP HANA is an in-memory, massively parallel, relational data platform. It is common knowledge that minimizing the data movements can improve performance significantly. SAP HANA implements this by keeping all or most of the data in-memory next to the processor to avoid performance penalty of disk I/O and allowing users to work directly on the data. As shown in Figure 2, the in-memory processing engines comprising of Column/Row engine, Graph engine and Text engine offers different processing capabilities and forms the heart of SAP HANA database server. Data replication and synchronization is not required since they all run on the same environment. In HANA, tables can be configured as a column store or a row store and can be restructured later if needed. As the name suggests, the column based engine stores relational data as columns and is useful for storing huge volume of data to be combined and used in analytical operations. The row based engine stores relational data in rows and can perform write operations (insert/update) with lower compression rates. The column engines fare better for query performance and data can be stored in whichever engine required as per the operation to be performed. The tables stored in the row store are loaded at startup whereas the tables in column store are either loaded at startup or on demand. Both these engines rely on the Persistence layer which provides backup and recovery in the case of a power failure and the system has to restart.

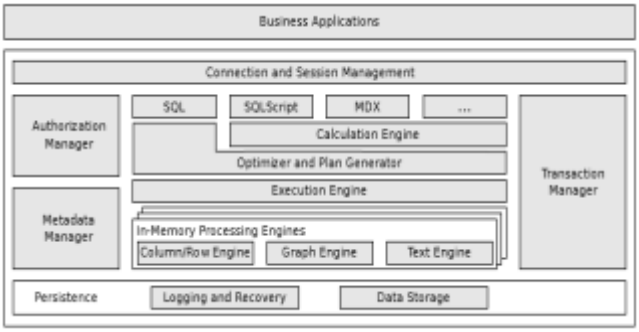

Figure 2. SAP HANA database architecture [3]

Data is logged frequently by the Logger in the persistence layer for durability and atomicity so that the system can roll back to the last committed state of each transaction during recovery. Disks or solid state drives are used for this activity but it does not hamper performance in any way because the backup operations to the disk take place asynchronously as a background task. The Session manager performs the connection and session management between the application layer and the underlying database layer. The Authorization manager has an important role to authorize the user permissions. The Transaction manager has the extremely important task to maintain concurrency control/transaction management. In-memory systems with multicore processing require a complex mechanism to provide concurrency, but the age-old centralized lock manager and deadlock detection adds significant overhead. Instead, Very Lightweight Locking (VLL) is used, which compresses all locks of one record into a pair of integers. The Metadata manager consists of data about the tables and other data structures.

4. SAP HANA FEATURES

In this section, we go through the various features/advantages of SAP HANA and how it is achieved.

4.1 Eliminate Indexes with Column oriented structure

In-memory data access is very fast when compared to disk access. A major factor for this fast access is the lack of indexes which allows for more compression. In a database system, indexes allow us to efficiently look up a value or record that satisfies some criteria. Scans can iterate over each record one by one and is used for queries that access many records. It may seem that indexes might be the best way to access records, since it access only the specific records that is needed, but it comes with and overhead. Indexes can be expensive to create and difficult to maintain. Hence analytical queries which access a huge number of records mostly involve scans. Companies use these kinds of queries to understand sales trends, buying and selling patterns and so on. Column oriented databases make these analytical queries run faster when compared to a traditional row oriented database. The data in a disk is laid out sequentially, and so, even though we need to access only certain columns, we have to read/pass through all the data to gather these values. This means that even though only a few columns are accessed in the query, the total time to read it would be the total time to read the entire set of data.

In a column oriented database, each column is stored in a separate file on disk. So if we need to read just three columns, we do not have to read any additional data. Consider the sample query "Select SUM (Sales) from Sales;" given in Figure 3. In a column oriented database we need to access only the Sales column from the Sales table, rather than reading all the data like Country and Product from Sales row by row.

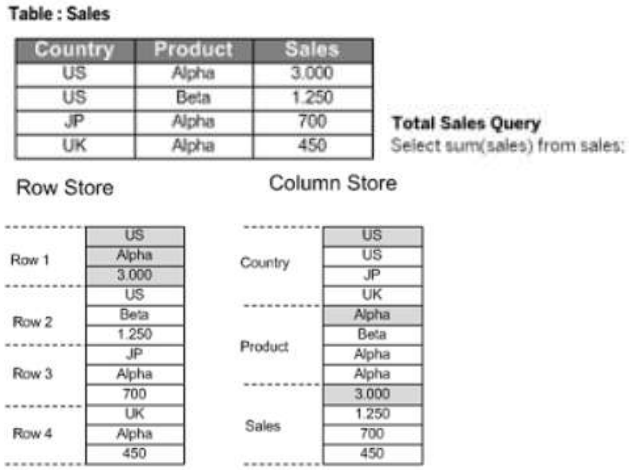

Figure 3. Row and Column oriented table structure

In reality, databases are hundreds of columns wide, but queries hardly require more than five attributes to be accessed at a time from these tables. This is the concept that columnar databases have used to speed up access time. Columns compress well compared to rows. In the columnar representation of Sales table, we can find the country "US" repeating. So instead of writing the name twice, we could have the value US once, followed by the

count of the number of times it repeats. This technique of compression is called Run Length Encoding (RLE) and is very effective for this kind of data. Even though this example might not look very effective, in a table where there are a few billions of rows, we can effectively reduce the Country column to a few thousands of records, which is a dramatic reduction in space. However, this technique may not work for some quantities such as Sales Prices. But we can use different compression techniques like Sparse encoding or Cluster encoding for different type of data in columnar databases, allowing for very aggressive compression of data. In effect, each column works as an index and thus saves the space for storing the index and the overhead to maintain the index during each operation. Column stores generally follow the "append-only" policy in which, when there is an update to an existing record, the new version is simply appended and the old one is invalidated without any complex reordering or in place updates. This improves the overall query throughput significantly.

4.2 Process more data with less memory

Partitions in table exist either as row store or column store. Row and Column store each have their own merits and weaknesses. SAP HANA overcomes it by having a three-level column oriented structure with L1-delta, L2-delta and main store to provide efficient support for analytical and transactional queries [4]. The lifecycle of data stored as tuple in HANA has 3 major levels as shown in Figure 4. The data is first stored in the L1-delta in the row format. Then it is moved to the L2-delta in column format and finally undergoes more compression and is stored in the Main store. Since updates have a cost involved, each table will have delta tables which will store the data in between updates. The data from the delta stores will then be transferred to the main store. Sorted dictionary encoding or Substitution encoding converts each value to bit-coded integers in order to save space. This helps to process more data faster with less memory and also store more data in a single node. SAP HANA automatically chooses the right compression for the data, allowing for maximum compression.

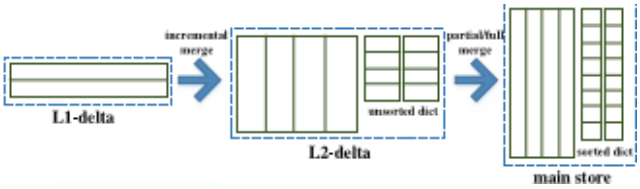

Figure 4. SAP HANA Hybrid store

4.3 Parallel Processing for more efficiency

SAP HANA was designed to perform a variety of operations like scans, calculations, analytical queries and aggregations all in parallel, utilizing the multiple cores and available computing resources in a distributed system. The benefit of parallel processing with multiple cores in columnar structure is that each core can work on a column or a portion of the column parallelly without conflict as in Figure 5. This means that each column can have different processors executing operations on it in parallel, and if needed, the column itself can be partitioned to have the look and feel of multiple columns so that different processors can work on it. This leads to optimal hardware utilization and high-speed access. If this was row oriented storage, the data would have been spread out in memory and the same operation would have taken much longer due to cache miss in the processor.

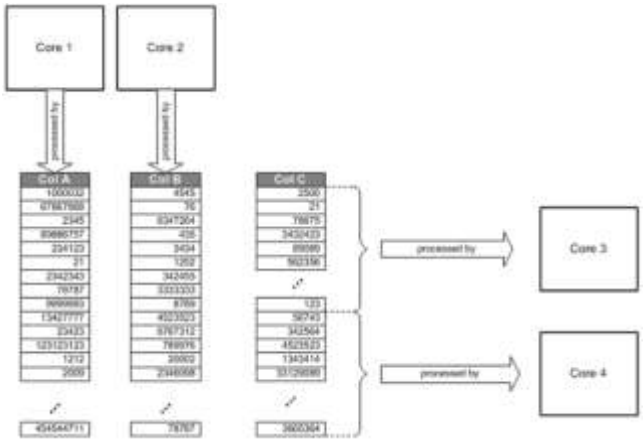

Figure 5. Multicore processors utilizing columnar architecture

4.4 Single Instruction Multiple Data (SIMD) processing

Single Instruction Multiple Data (SIMD) processing is a key concept to enhance performance. As per Moore's Law, the number of transistors in chips has doubled every eighteen months or so and the current generation of chips have over 2.6 billion transistors which have helped to widen the registers from 8 bits to over 256 bits. Traditionally, computations were performed by loading bit values to a register, making the necessary modifications and then storing them back again as shown in Figure 6. One instruction would produce one result. But since the expansion of registers, more space is available and with the help of a bit of added circuitry, we could load 4 values at a time instead of one, operate on them and store them back again 4 at a time. i.e., we could operate on multiple data values at once with a single computer instruction and hence the name SIMD.

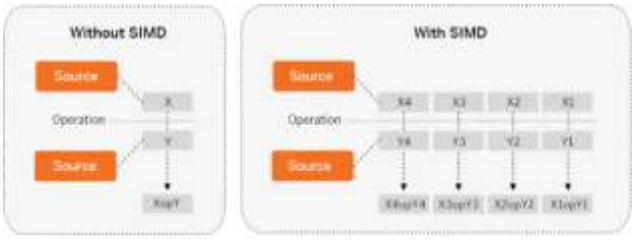

Figure 6. Faster processing in SAP HANA with SIMD

4.5 Dynamic tiering to manage large volumes of data cost effectively

SAP HANA can handle huge volume of data in its memory effectively. But it is not necessary to have all the data in-memory at all the time and HANA strikes a balance between price and performance by a technique known as Multi-Temperature Data Management. Multi-temperature data management refers to storing frequently used "Hot" data on faster storage medium such as in-memory and less frequently accessed "Warm" data on

slightly slower storage medium such as the disk. This is the core concept of Dynamic Tiering. To achieve this, tables are defined as "extended tables" which are no different from the normal tables, but have the exception that they are created on the disk and not in memory. Dynamic tiering allows tables to span across memory and disk.

5. SAP HANA VORA

Taking big data management one step forward, SAP introduced SAP HANA Vora -a critical component of their Digital Enterprise Platform, which provides seamless integration between SAP HANA and Hadoop [7]. SAP HANA Vora is an in-memory query engine, built to run natively on Hadoop and Spark ecosystems. It provides a highly-distributed SQL engine to process different types of workload such as analytical processing, time series and graph processing on huge volumes of data in a distributed environment. The existing solutions for handling big data included Hadoop, distributed SQL databases like Google F1, Facebook Presto and of course SAP HANA, and NoSQL databases such as MongoDB, Apache Lucene, Cassandra and Neo4j. Most of these solutions were to process a specific workload. When it came to spawning data across tens of thousands of nodes and processing petabytes of data, the existing SAP HANA portfolio did not scale well. Some of the customers already had their data stored on Hadoop due to cost efficiency, but required the analytical capabilities of SAP HANA to take meaningful actions from the data. While Hadoop provides a cost-efficient way of storing scalable data and disk-based distributed computing, HANA provides real-time data analysis capability in an in-memory platform. This is when SAP HANA Vora came into the picture to combine the best of both worlds and make it possible to process data within the open source framework in a cost efficient way. Spark has a rich capability for SQL queries using SparkSQL which also supports data hierarchies and SAP HANA Vora extends Spark to accelerate query performance.

Vora acts as an extension to the Hadoop platform and has the following features: very fast in-memory processing, support for languages such as Scala, Python and Java, very fast query processing and support for hierarchies. The most useful benefit of Vora is to allow existing SAP HANA customers to use Hadoop to store "Cold" data which is accessed very less frequently. Vora also provides data tiering which allows storing data in two distinct physical mediums, yet providing a single logical view in a business perspective to analyze data efficiently.

5.1 Architecture of SAP HANA Vora

The main components involved in Vora are shown in Figure 7. Relational in-memory stores data into memory for processing. Relational disk engine process the data that may not fit into main memory. Time series engine compresses time series data and provides algorithms on the compressed data. Graph engine performs graph operations and handles complex analytical queries on very large graphs. Spark shell command line Scala extension runs Spark programs. Spark driver provides Spark SQL context and Spark SQL query processor that compiles SQL queries into Resilient Distributed Datasets (RDDs) [7]. SAP Vora Spark Extension communicates with the transaction coordinator to execute queries on data stored in various Vora engines and with the Catalog Server to store/fetch metadata for tables. Kubernetes dashboard runs all Vora services on nodes [8].

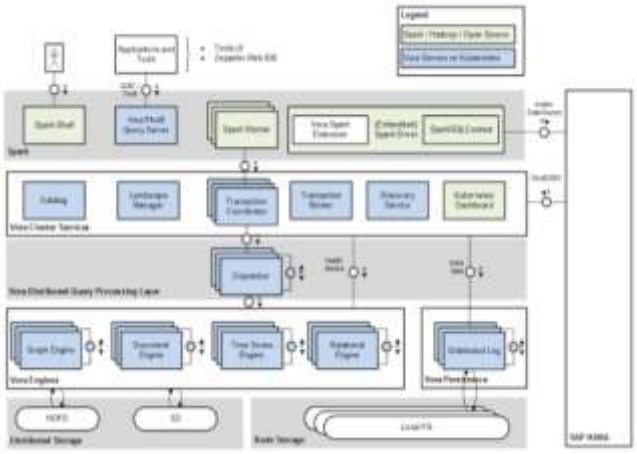

Figure 7. System Architecture of SAP HANA Vora

5.2 Apache Spark Integration to Vora

The SAP Vora API is integrated to Apache Spark SQL and can be used to write Spark programs with the SAP Vora Spark extension. Figure 8 shows the sample code to create a new table in the relational in-memory engine using data from a CSV file. Spark SQL can be used to query the table and SAP Vora Spark Extension analyzes and pushes the query into SAP Vora engines. There are two ways to push queries across SAP HANA and SAP Vora. Either from SAP Vora to SAP HANA via the Spark extension or from SAP HANA to SAP Vora via SAP HANA Smart Data Access (SDA) and SAP Vora remote source adapter (Vora ODBC).

```
We assume a CSV file on HDFS that contains some data:
John,10
Jane,20
John,20
Jane,40
*/

import org.apache.spark.sql._
val sqlc = new SapSQLContext(sc)
sqlc.sql(
s"""
            CREATE TABLE testTable (name string, score integer)
            USING com.sap.spark.engines.relational
            OPTIONS (
               files "/path/to/file.csv",
               hdfsnamenode "<host>:<port>"
    )""".stripMargin)

/* query for entries with score > 10 */
sqlc.sql("SELECT name, age from testTable where score > 10").show

/* query for average score */
sqlc.sql("SELECT name, AVG(score) AS avgScore from testTable GROUP BY name").show
```

Figure 8. Sample code to create table in Vora using Spark

5.3 Partitions in Vora

Partition function defines how many table partitions are created and how they are created. Each partition is to be placed on one host and so the number of partitions must be less than or equal to the number of hosts. There are various types of partitioning such as range partitioning, hash partitioning, and block partitioning.

The CREATE PARTITION FUNCTION command is used to create a partition as shown in Figure 9.

```
<partition_function> ::=
  CREATE PARTITION FUNCTION <name> <pf_definition>

<pf_definition> ::= <column_type_list> AS <partition_function_type>

<column_type_list> ::= '(' {<identifier> <type>},... ')'

<partition_function_type> ::= (

    <range_partitioning>

 || <hash_partitioning>

 || <block_partitioning>

 || <system_partitioning>
 )
```

Figure 9. Sample code to create partition in Vora

5.4 Hierarchies in Vora

Hierarchical data structures define a parent-child relationship between different data items which makes it easier to perform complex queries and aggregations on different levels of data separately [9]. Vora data source has extended the Spark SQL to create a hierarchical data frame and performs custom hierarchical user-defined functions on it. This will push the execution down to the Vora Engine. To create a hierarchy, we need a table which defines relations between predecessor and successor in each level. If the predecessor value is 0, then it will be a root node.

6. SUMMARY/CONCLUSION

SAP HANA platform provides a comprehensive and breakthrough technology with in-memory columnar database that has capabilities for processing analytical and transactional queries and data integration on large volumes of data to provide actionable insights. It can process more data with less memory and employs multicore processors and deep parallelization on SIMD architecture to achieve efficiency in utilizing resources. As a next step, SAP HANA Vora the in-memory query engine for big data processing with its support for various types of data was introduced. SAP HANA will benefit customers who want to integrate multiple data sources into one common place and create mash-up queries for deep dive and interactive analysis.

7. REFERENCES

[1] S. Robbins, “RAM is the new disk,” InfoQ News, Jun. 2008.

[2] C. Brockman, and T. Mirzoev, (2012, Dec.), i-*managers Journal on Information Technology*, Vol. 2, No. 1. Available: https://arxiv.org/abs/1404.2160v1

[3] F. Farber, N. May, W. Lehner, P. Große, I. Mller, H. Rauhe, and J. Dees, “The SAP HANA database – an architecture overview,” in *IEEE Data Eng. Bull.*, vol. 35, no. 1, pp. 28–33, Mar. 2012.

[4] H. Zhang, G. Chen, B. C. Ooi, K. L. Tan and M. Zhang, "In-Memory Big Data Management and Processing: A Survey," in *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 7, pp. 1920-1948, July 1 2015.

[5] V. Sikka, F. Farber, A. Goel, and W. Lehner, “SAP HANA: The evolution from a modern main-memory data platform to an enterprise application platform,” Proc. VLDB Endowment, vol. 6, pp. 1184–1185, 2013. Mackay, W.E. Ethics, lies and videotape... in *Proceedings of CHI* '95 (Denver CO, May 1995), ACM Press, 138-145.

[6] V. Sikka, F. Fa¨rber, W. Lehner, S. K. Cha, T. Peh, and C. Bornh¨ovd. Ecient transaction processing in SAP HANA database: the end of a column store myth in *SIGMOD Conference*, 2012.

[7] P. Jifroodian and B. Krishna, (2016). openSAP: Big Data with SAP HANA Vora [Online]. Available: https://open.sap.com/courses/hsvo1

[8] B. Krishna, (2012, Aug. 27). Solving Big Data with SAP HANA and Hadoop [Online]. Available: https://blogs.saphana.com/2012/08/27/solving-big-data-with-sap-hana-and-hadoop/?q4654483=1

[9] Hewlett Packard Enterprise. (2016, Sep.). HPE Reference Architecture for SAP HANA Vora with Spark and Hadoop [Online]. Available: http://h20195.www2.hpe.com/V2/getpdf.aspx/4AA6-7739ENW.pdf