

SAP's BI Accelerator: Search & Retrieval Meets Analytics

This paper describes how SAP has leveraged similarities and synergies between search and multi-dimensional reporting to create the BI Accelerator (BIA), an engine that works as an add-on to SAP BW (aka NetWeaver BI). After motivating this approach some insights are given on the BIA's architecture and its underlying query processing.

1 Introduction

With the recent release of its technology and integration platform¹, NetWeaver, SAP will ship a new BI Accelerator (BIA²) engine as part of its business intelligence (BI) component. The BIA engine combines technologies and infrastructures from SAP's own search and retrieval engine TREX and SAP's Business Warehouse (BW; a business intelligence and data warehousing platform). This approach is interesting in two ways: first, it shows how innovation can be created by transferring technologies, algorithms and infrastructure that have proved to be viable in one area – search and retrieval in this specific case – to another area – structured, multi-dimensional data. In a second, subsequent step, the BIA engine can be considered as a successful instance of the general concept of bringing together generic business software components in one platform.

»Taking the power of Google to a bunch of enterprise data« seems to be a straightforward exercise to the outsider. However, we will give some insight into the technology problems that have to be overcome. We will do this by comparing the domain space of analytics to that of search and retrieval (section 2) and then by diving into the specific details of the BIA engine processing (section 3). The paper is concluded in section 4 and summarized in section 5.

1. Sometimes this is also referred to as an applistructure (application infrastructure) platform.
2. For the record and references: BIA has also been referred to as High Performance Analytics (HPA) or Euclid.

2 Analytics and Search & Retrieval

In this section, we compare the areas of analytics and search & retrieval thereby elaborating similarities and differences.

2.1 Analytics

Let's consider an example of a typical analytical query. Figure 1 shows an analytical view on some sales figures. Essentially, there is a result table showing sales figures for a number of selected countries and months. On the left hand side, a navigation block allows to arbitrarily slice and dice the data along various dimensions. In addition, there is a bunch of tools like highlighting exceptional values, presenting the data graphically as charts, distributing the analysis via e-mail or an intranet portal, converting currency or unit values to other currencies/units etc.

Naturally, the user assumes the result to show exact figures rather than approximations. Typically, he will start with an initial, aggregated view and then dive into more detail analysis, pivoting the data, e.g. around the exceptional value for Italy in 09.2003 (highlighted cell for Italy in figure 1) by drilling down along cities or product groups, in order to find out origins of failures or success. Filters are based on standard boolean expressions such as month = 09.2002 OR month = 09.2003 OR month = 09.2004.

The two dominating processing steps in that context are (a) filtering the data (on a detailed level) and then (b) aggregating that data to a less granular level. As initial views on data typically start on a highly aggregated level and drill down to a finer granularity the focus shifts from a huge amount of aggregation and less selective filtering at the beginning (e.g. sales figures by months of a specific year and by a list of countries) to little aggregation and highly selective filtering (e.g. sales figures by days of a certain month and cities of a specific country).

The performance of analytical queries is highly critical. On one extreme, there are queries being executed routinely, e.g. to watch certain business processes like supply chains, inventories, failure rates, sales pipelines etc. Frequently, such queries form part of, so called, analytical dashboards or cockpits³. Performance-relevant properties are that such queries can be calculated prior to their routine usage (possibly in many variants for a huge number of users) and that slicing and dicing takes place only whenever exceptional circumstances arise. On the other extreme, there is ad-hoc querying with arbitrary slicing and dicing, albeit mostly by a small number of users. In both cases, the amount of data processed can be huge: typical retail scenarios involve from several hundred million and up to a few bil-

3. The names dashboard and cockpit hint at the role that those devices play in complex technical environments like planes where the pilot routinely checks whether all technical processes run as expected.

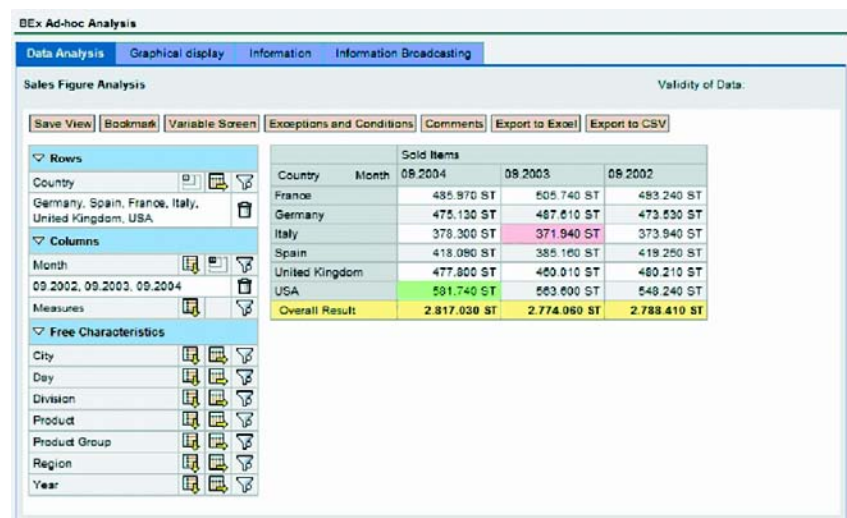


Fig. 1: Analytical view

lion fact records originating in point-of-sales (POS) data.

Finally, there is the issue of latency, i.e. the delay between the moment an update on the (transactional) data takes place and the moment that update is visible via the analytical tool. The latency is typically used to build up secondary and redundant data structures that heavily improve the analysis performance. Examples are indexes, materialized views, automatic summary tables, de-normalized tables and the like. The rule of thumb is that the more latency one can afford the better the query response time performance can get. Therefore, the question of latency is highly dependent on the application: the analysis of slow selling items in a supermarket does not require last minute information but can be based on data that might reflect only last week's situation. In contrast, a call centre operator cannot afford to miss the data provided by a customer earlier on the same day. An extreme situation arises with planning tools that use historical data (the so called »actuals«) to plan the near future business key figures. Such a process is highly interactive and requires newly created plan data to be shown immediately via an analytical tool, i.e. in real-time. To address this wide range of affordable latencies, the more general and somewhat sagacious term of right-time analytics has recently come up to subsume real-time, near-real-time or almost-real-time. In summary, it is fair to say that latency times between minutes and weeks are the prototypical case.

2.2 Search & Retrieval

A typical example of an Internet search is to look for pages with words »near«, »realtime« and »analytics« via Google. The result is potentially huge and does not have to be calculated entirely; the search engine therefore gives an estimate – such as »results 1-10 of about 82000«. Due to the result size, the ranking is very important, i.e. the most relevant pages should be displayed first. Modern ranking functions are based on a variety of implicit constraints, like position of the search terms within the page, proximity to each other, number of inbound links (»recommendations«) of the page and many more.

Once, the initial result shows up, a user is either satisfied with it or alternatively modifies the query by adding, removing or replacing search terms. This stands in contrast to the multi-dimensional case discussed in section 2.1: there is only one single dimension, namely text, and the only option is to modify the filter. Due to the fuzzy nature of searching, the process is highly interactive and response times in the range of seconds are mandatory. As the result cannot be exact anyway, it is even possible to compromise on the result, i.e. to abandon query processing after having passed a certain threshold of processing time and to display what has been found so far.

Data volumes are enormous and in the range of billions of pages for Internet search engines. Indexing technology is used to handle these volumes and to provide an acceptable query performance.

The indexes do not reflect the current state of the data but there is always some latency involved. The latter is – like the data itself – also kind of fuzzy: intuitively, the user expects frequently changing sites (like news, financial information or market places) to be polled more frequently than more static pages (like museum catalogues, article archives etc.) and this is what probably happens – »probably« actually translates into »undefined« or maybe the range of days to weeks.

2.3 Similarities and Differences

Figure 2 above generalises and compares the scenarios described in the previous two sections. While the differences mainly originate in the nature of the data (structured vs. unstructured data) there is a wide range of similar requirements around the issue of performance: mass data has to be processed, thereby allowing acceptable query response times. Latencies between data updates and visibility are common in both scenarios and are used to create secondary data structures like indexes or pre-calculated summaries.

3 BIA Engine

3.1 Overview

In this section, we briefly describe how the worlds of analytics and search & retrieval have been brought together to form SAP's BI Accelerator (BIA).

Figure 3 gives an architectural overview of BIA. On the left side, there is the traditional BI platform providing all kinds of capabilities to extract, transform, load data into a variety of persistency containers, like data store objects and InfoCubes¹, for flat and multi-dimensional reporting respectively. BIA focuses on the multi-dimensional case. An SAP BI InfoCube is physically represented by a star schema on a relational database. The traditional approach is to improve query performance by pre-computing and materializing aggregations; the latter being stored also in relational database tables

1. This is the SAP term for a data container and model for multi-dimensional scenarios. In many ways, it corresponds to what is commonly known as a cube. However, as the BIA capability is tailored to and takes advantages of the many properties of the InfoCube it is appropriate to use the SAP terminology.

	Analytics	Search & Retrieval
differences		
nature of result	exact (missing data is fatal)	fuzzy (missing or superfluous data can be acceptable)
nature of data	structured (mostly numbers)	unstructured (mostly text documents)
dimensionality	multi-dimensional	one dimension (→ unstructured)
filters	boolean expressions	regular expressions, patterns
implicit constraints	none	textual constraints (language, position, proximity, ...)
similarities		
reporting environment	read-mostly	read-only
latency with respect to data updates	minutes – weeks	days – weeks
relationship: data updates - read	mostly disconnected	disconnected
mass data	yes	yes
expensive operations	sorts (→ grouping, aggregation), joins	sorts (→ ranking function), filters

Fig. 2: Analytics vs. Search & Retrieval

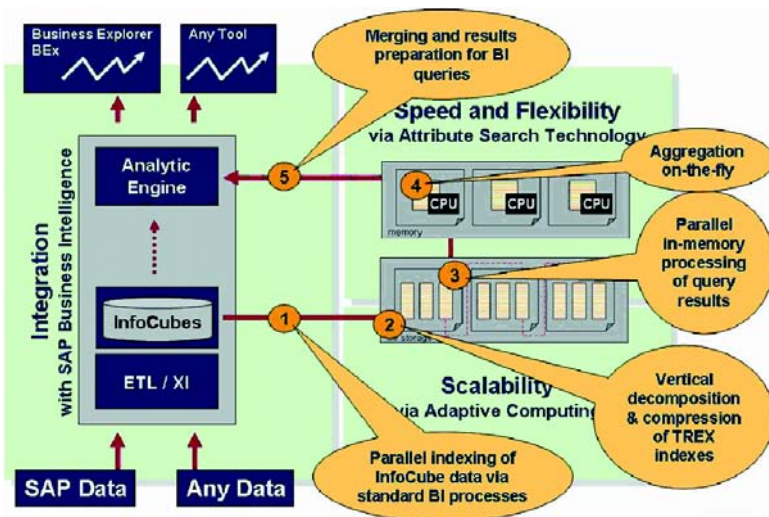


Fig. 3: BIA Architecture

organized as a star schema. They are referred to as aggregates. Alternative, but very similar, approaches could be to use features provided by RDBMS platforms like materialized views, automatic summary tables, indexed views etc. which are similar albeit more generic (and therefore sometimes semi-optimal) features. The challenge is to synchronize aggregates with changes within an InfoCube and its associated master data, i.e. rolling up new data and realigning aggregates along structural changes within dimensional hierarchies. Furthermore, administrators are permanently forced to trade off the query performance (which is better in the presence of many aggregates) and the time consumed for aggregate roll-ups and realignments during the daily or weekly data load window (i.e. less aggregates translate into smaller load windows). BIA presents an alternative to this. The BIA engine runs on a separate server comprising a set of blade servers (see right hand side of figure 3). Data residing in the tables that form the star schema of an InfoCube can be moved to indexes in the BIA engine (step 1 in figure 3). Similar to aggregate roll-ups, those indexes are updated every now and then with data that has been recently loaded into an InfoCube. In fact, the same infrastructure is leveraged to handle this process. During the upload into the BIA, data gets heavily compressed – experiments on productive scenarios have shown compression rates between 7 and 12. Furthermore, data is vertically partitioned (step 2 in figure 3). Both techniques are prerequisites to allow efficient main memory processing of the

indexed data. While steps 1 and 2 are executed in the context of a data load process, query execution is covered by steps 3, 4 and 5. Once an analytical query is issued from a front-end tool it reaches the analytic engine, more specifically a calculation layer that handles formulas, security, exceptions, alerts, hierarchies and hierarchy-related logic and the like. The calculation layer requests aggregated data from an aggregation layer and this is where BIA provides an alternative: rather than issuing SQL statements to a relational database, equivalent queries are pushed to the BIA server. The latter processes those queries in parallel and in memory (step 3 in figure 3). One major extension to the original search engine has been to incorporate the capability of aggregating data in memory (step 4 in figure 3). Parallel processes send their local results back which are then merged and handed back to the analytic engine (step 5 in figure 3) from whereon standard processing continues.

3.2 Query Example

Let us consider a simple example, namely the query behind figure 1. In SQL terms, SAP BI's analytic engine would translate this into something like

```
SELECT country, month,
       SUM (sold_items)
FROM <some joined tables>
WHERE country IN ('DE', 'FR',
                 'UK', 'IT', 'US', 'ES')
AND month IN (09.2002,
              09.2003, 09.2004)
GROUP BY country, month
```

Figure 4 shows the situation graphically: there is a fact table that contributes the key figure value SUM(sold_items) and there is a bunch of dimension tables (D_{Time} , D_{Region} and D_{Prod}) that have foreign key – key relationships with master data tables (product, product group, city, region, ...). There are two filter, one on *country* and one on *month* and the latter are also the items to be used for the grouping. The BIA uses a similar setup as depicted, tables translate into indexes but a star schema approach is still employed.

Query processing within the BIA follows a few simple paradigms:

1. outside in, i.e. start with the master data tables and process the fact table at the end;
2. only process columns are relevant to the query;
3. start with the branches on which filters are defined;
4. parallelize as much as possible.

It is fair to consider this as star join technique. However, there a few details that distinguish the BIA from standard relational processing. Let us consider figure 4: processing would start with the value columns of indexes *month* and *country*

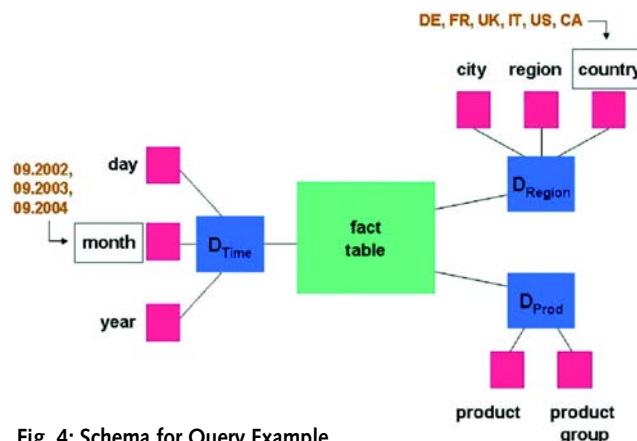


Fig. 4: Schema for Query Example

and apply the respective filter (paradigms 2 and 3). This results in sets of keys that can be applied to the respective dimension index. If additional data was required, e.g. the cities, then index *city* could have been joined to index D_{Region} in parallel to the above (paradigm 4), even at the expense that the selectivity imposed by the filter on countries would not have been available at that time (as that processing happens simultaneously). Reducing sizes of intermediate results is less beneficial than the advantages gained from additional parallelism.

There is a synchronization of the parallel processes before the fact table index is processed, i.e. once all filters applicable to the fact table index are available processing continues again by parallelizing over the (horizontal) partitions of that index. Typically, fact tables have a considerable number of key figure columns. Here, only those that are used within the query are processed – column *sold_items* in our example – which significantly reduces the memory consumption. Processing the fact table index not only means applying the filters but also aggregating the key figure values – in terms of SQL: performing the GROUP BY.

Finally, the query result is assembled in an internal representation by a master process, translated into an external format and then sent to SAP BI's calculation layer. The latter would convert the line-oriented result into a cell-oriented one, check for exceptional values (highlighted cells in figure 1), calculate totals according to the user's request (sums, averages, ...) and elaborate the visual grid representation.

4 Conclusions

There is a number of interesting observations that have been made during the development and the first trial experiments of the BIA. Firstly, a huge share of the search engine infrastructure and technology could be reused. Secondly, memory bandwidth is the bottleneck in most situations. Thirdly, query processing is mostly based on very simple processing techniques and plans (e.g. in-memory scans of data). While this looks like the worst case you can do it is still ways faster than the traditional alternatives, including MOLAP, in most situations due to large-scale parallelism and sophisticated usage

of multi-level caches in modern micro-processors. Furthermore, it gives very stable, controllable and predictable query performance because it cannot get worse than the worst case. While traditional benchmarking focuses on speed, many customers also look for stability and some even rank stability over speed. This is accompanied by a fourth observation, namely that the BIA engine speeds up almost linearly with the hardware. Simple processing algorithms, parallel and main memory processing are the main assets and allow more hardware resources to easily translate into better performance. This gives a customer the confidence that they can »buy themselves out of performance bottlenecks« by adding inexpensive hardware (blade racks). Even improvements in the hardware sector and the general trust in Moore's law contribute in this context. Finally, there is no need for realignment processes as all aggregations are computed at query execution time and do not rely on pre-calculated and materialized aggregates.

5 Summary

In this paper, we have discussed the similarities between the areas of analytics and search & retrieval. There is number of differences, mainly based on the different nature of the underlying data, but also a number of similarities with respect to mass data handling and the resulting performance requirements. Based on this notion, technology and infrastructure from a traditional search engine has been applied and extended to address the specific requirements of analytics. SAP's BIA capability is an instance of that approach. It has been described in some detail, thereby outlining the difference it makes in comparison to traditional approaches that have been used in the BI arena during the last decade. In that respect, it is interesting to see how innovation can be created not only by completely new ideas but by transferring, combining and extending well-known technologies to address the requirements of a new domain.

References

- [Akhter & Roberts 2006] *Akhter, S.; Roberts, J.*: Multi-Core Programming. Intel Press, 2006.
- [Boncz 2002] *Boncz, P. A.*: Monet: A Next-Generation DBMS Kernel For Query-Intensive

Applications. Ph.D. Thesis, Universiteit van Amsterdam, Amsterdam, May 2002.

- [Cannane et al. 1999] *Cannane, A.; Williams, H. E.; Zobel, J.*: A General-Purpose Compression Scheme for Databases. Data Compression Conference 1999.
- [Fotakis et al. 2003] *Fotakis, D.; Pagh, R.; Sanders, P.; Spirakis, P.*: Space efficient hash tables with worst case constant access. 20th International Symposium on Theoretical Aspects of Computer Science, Springer-Verlag, 2003, S. 271-282.
- [Legler et al. 2006] *Legler, T.; Lehner, W.; Ross, A.*: Data Mining with the SAP NetWeaver BI Accelerator. To be published in Proceedings of the 32th Int. Conf. of Very Large Data Bases (VLDB), 2006.
- [Manegold et al. 2002] *Manegold, S.; Boncz, P. A.; Kersten, M. L.*: Optimizing Main-Memory Join On Modern Hardware. IEEE Transactions on Knowledge and Data Eng., 14(4):709-730, July 2002.
- [Monash 2005a] *Monash, C.*: Maximizing analytics performance III – SAP's HPA. Computerworld, July 2005; www.computerworld.com/blogs/node/577.
- [Monash 2005b] *Monash, C.*: Faster and Cheaper Means Better Data Management. Computerworld, November 2005.
- [Ross 2005a] *Ross, A.*: Business Analysis an Order of Magnitude Faster. SAP Info Online Issue, January 2005; www.sap.info.
- [Ross 2005b] *Ross, A.*: Speed, Scalability, and Flexibility – All at Once with New High Performance Analytics. SAP Insider, October 2005; www.sapinsider.com.
- [Stonebraker et al. 2005] *Stonebraker, M.; Abadi, D.; Batkin, A.; Chen, X.; Cherniack, M.; Ferreira, M.; Lau, E.; Lin, A.; Madden, S.; O'Neil, E.; O'Neil, P.; Rasin, A.; Tran, N.; Zdonik, S.*: C-store: A column-oriented dbms. VLDB, 2005.
- [Witten et al. 1999] *Witten, I. H.; Moffat, A.; Bell, T. C.*: Managing gigabytes: compressing and indexing documents and images. 2nd ed., Morgan Kaufmann, 1999.
- [Zurek 2005] *Zurek, T.*: Comparing the BI Accelerator to Traditional RDBMS Technology. SDN Weblog, October 2005; <http://www.sdn.sap.com/irj/sdn/weblogs?blog=/pub/wlg/2553>.



Thomas Zurek is a development manager in the area of BI with the SAP NetWeaver platform. His group builds NetWeaver's analytic engine. He has 10 years of experience in analytics and over 15 years with RDBMS. He holds a PhD in computer science from Edinburgh University.

Dr. Thomas Zurek
SAP AG
Dietmar-Hoppe-Allee 16
69190 Walldorf
thomas.zurek@sap.com
www.sap.com