In-database Analytical Systems: Perspective, Trade-offs and Implementation

Executive summary

TIBCO Spotfire® is a visualization-based data discovery tool. It has always held its data in memory; this allows for very rapid processing but does place a restriction on the volume of data that can be analyzed. TIBCO has always acknowledged that for some customers this can be an issue. As an earlier white paper says: "However if you need to analyze TBytes of data in a single analysis, and people do, then in-memory systems are probably not for you."

The significant change, which this paper addresses, is that Spotfire can now run its dimension-free data exploration environment against data held in Teradata, Oracle’s Exadata, SQL Server 2008 and 2012 and in SQL Server Analysis Services (SSAS). Since these engines can comfortably handle data volumes far in excess of that which can be held in memory, it is now possible to use Spotfire’s unsurpassed analytical capabilities against some of the largest data sets in the world. This provides much greater flexibility and also the ability to use the same tool for both classes of data provides some unexpected synergies.
How has Spotfire traditionally worked?  
In-memory vs. disk-based approaches

Data discovery tools, such as Spotfire, tend to work in one of two ways – they can either hold and manage the data themselves in a purpose-built, optimized store or they can connect to an existing database engine (relational or multi-dimensional) and work with the data it holds.

Of those that hold and manage the data themselves, some hold the data in memory and some hold it on disk. Both forms of storage have their pros and cons, which is why both continued to be used. So what are the tradeoffs and how do they interact?

Price
Disk is much, much cheaper than RAM. How much cheaper depends on a large number of variables, notably the huge variety of memory and disk types, but it is probably safe to say that disk storage is between 30 and 50 times cheaper than RAM. Some sample prices gathered at the end of 2012 are:

<table>
<thead>
<tr>
<th></th>
<th>RAM $ per Gbyte</th>
<th>Disk $ per GByte</th>
<th>Approx. Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Laptop</td>
<td>4.5</td>
<td>0.09</td>
<td>50:1</td>
</tr>
<tr>
<td>Server</td>
<td>15.6</td>
<td>0.45</td>
<td>35:1</td>
</tr>
</tbody>
</table>

So storage cost, considered alone, tends to favor disk-based systems.

Data volume
Data volume is intimately allied to price; the larger the volume of data you want to analyze, the more likely you are to favor a disk-based system.

We tend to say that in-memory is suited to modest volumes of data, but one person’s ‘modest’ is another person’s ‘massive’. To try to provide a sense of perspective, let’s assume you want to analyze a single table with 20 columns. Half of the columns are text (limited to, say, 50 characters but with an average length of seven characters), five are dates, three integers and two reals.

A 1.5 million row table saved as a CSV file is 236 Mbytes – about a quarter of a GByte. If we double the number of columns (40) and rows (3 million), that’s about 1 GByte. Double those two again, to 80 columns and 6 million rows, and it is still only 4 Gbytes. Most laptops now have 8 GBytes of RAM so even without compression this relatively large number of complex rows can be analyzed comfortably in memory.
So in-memory systems are clearly favored for the analysis of modest quantities of data, and disk-based systems where the volume of data is huge and simply won’t fit in memory.

**Speed, and the trade-offs necessary to achieve it**

However there is a third variable to consider. There is very good evidence that the productivity of the users of a system is highly influenced by the speed with which it can deliver answers to their questions (see the box out on ‘Train of thought analysis’).

---

**‘Train of thought’ analysis**

The ability to interact rapidly with an analytical system is paramount for generating results efficiently. It’s not just the absolute time the querying session takes, it’s the fact that sometimes the answers you get pose other questions and you’re able to follow this new train of thought to a point where a new aspect of your business is revealed. A rapid response has been shown to be vital; if responses are too slow, thoughts tend to wander, to think about coffee or tonight’s game… and the will to follow a novel route is quashed by the undesirability of all the time ‘wasted’ in waiting. The gain from such rapid responses can be impressive: research in the 1980s by IBM showed that productivity increased by a dramatic 62% if the response time was improved.

---

We also tend to collect data as individual transactions but analytical questions tend to examine the data at reasonably high levels of aggregation.

So, for example, we might collect the data like this:

<table>
<thead>
<tr>
<th>CustFName</th>
<th>CustLName</th>
<th>Item</th>
<th>Date</th>
<th>Store</th>
<th>Units Sold</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>Smith</td>
<td>Canned Peaches</td>
<td>8/5/2012</td>
<td>Walla Walla</td>
<td>2</td>
</tr>
<tr>
<td>Sally</td>
<td>Jones</td>
<td>Candy</td>
<td>8/5/2012</td>
<td>Greenwich</td>
<td>1</td>
</tr>
</tbody>
</table>

but want to analyze the unit sales month by month across different stores, or state by state across different years. It is apparent that most of the analytical queries will be returning answers that are made up of aggregations of thousands or millions of rows. This isn’t a problem for in-memory systems because aggregation calculations performed in memory are so fast.
Disk-based systems, on the other hand, would struggle to return these aggregations fast enough if they had to read each individual transaction from a specific, single location on a disk and perform the aggregations in real time. So disk-based systems adopt a variety of strategies to provide faster response times.

SSAS, for example, typically pre-aggregates the data after loading and before the users start to query it. This has the disadvantage of introducing a delay between data loading and querying, but it does provide a level of performance that can approach in-memory systems.

Teradata doesn’t pre-aggregate the data; instead it parallelizes the query across multiple nodes where each node potentially only handles a subset of the data.

Both of these are excellent solutions in that they provide very high performance against large/huge data sets but each comes at a cost - either in terms of a delay or the requirement for additional hardware resources.

So, when speed of response is important (and it usually is) and the data volume is reasonable, in-memory systems are typically favored. However disk-based systems have the advantage that they allow you to analyze truly massive quantities of data – we are talking terabytes and even petabytes of data. (This is not an exaggeration. Teradata currently has 35 customers in its so-called ‘Petabyte club’.) But that then begs the question, why would anyone want to even think of analyzing that quantity of data?

**Data warehouses and Big Data**

**The value of consolidating data**

Many people, particularly those at a high level in a company, often ask deceptively simple analytical questions. A classic example is the CEO who needs to know if the company is making a profit. It’s very easy to ask the question, it takes just five words: ‘Are we currently in profit?’ The answer is even simpler, a single word from a choice of two – ‘Yes’ or ‘No’. The question and the answer are simple, but providing that answer is, for companies of any size, mind-bogglingly complex. The problem is that almost all organizations run multiple transactional systems - for finance, HR, sales, manufacturing, CRM and so on. Data from all these systems must be taken into consideration before the simple question can be answered accurately.

The solution is to:

- Extract the data required for analysis from the transactional systems
- Transform it into a consistent format and quality
- Load it into a centralized store for analysis

The centralized store of data is then referred to as a data warehouse.
The volume of data handled by a warehouse is typically huge for two reasons: firstly it pulls a great deal of data from multiple transactional systems and secondly the warehouse often stores history going back two, five, perhaps ten years at the transactional level of detail. It is now common to refer to this data as 'big data' (see box out). This historical data, although very bulky, is of enormous value to the business for the simple reason that we can often use past behavior to give insights into the future.

**Big Data**
Big data is a term with multiple meanings. If you gather five analytical specialists in one room, there you will find six definitions of big data. In this paper we are using it simply to mean very large quantities of transactional data – the kind of volumes that you find in the data warehouse of a reasonably sized company. This definition has two advantages. It is straightforward and it also describes, very neatly, a common class of data that needs to be analyzed.

Constructing a data warehouse takes a huge amount of time and effort, but once that data has been consolidated and is in one location, it becomes enormously valuable which is why most organizations have, by now, made the investment of creating a data warehouse.

**Summary so far**
Analysis of data can either be performed on data that is held in memory or on data held on disk. The in-memory approach gives much faster analysis but RAM is significantly more expensive than disk. There are ways in which disk-based analysis can be speeded up, for example by pre-aggregating the data.

The analytical requirements (and the regulatory reporting requirements) of many companies have driven them to create data warehouses that contain big data. This data is far too large to fit in memory, so it has to be disk-based.

The three major points that any system used for data discovery needs to address are the speed with which analyses can be performed, the clarity with which the results can be displayed to non-specialist users and the ability to analyze any measure against any dimension and any other measure (see box out on Measures and Dimensions).
Measures and Dimensions

Analysis is often described in terms of measures and dimensions. A measure is simply a numeric value: units sold, number of customers, etc. KPIs (Key Performance Indicators) are all typically measures. But measures, no matter how many you have, are almost always meaningless without reference to at least one so-called dimension. For example, it may be the case that your company has sold 431,763,273,812 items since its inception in 1924. This might have some ‘wow’ value but it only begins to reveal information about your business if you take that measure and split it out so, for instance, it shows the unit sales of canned peaches in two stores, displayed as monthly totals over the past year.

This is far more meaningful: the measure now is shown with the context of a product dimension (canned peaches), a location dimension (the store) and a time dimension (the last twelve months).

Some analytical queries require a measure to be analyzed against another measure. For example, in analyzing the effectiveness of loss leaders in supermarket sales, it is common to plot sales price against cost price. Spotfire natively supports this kind of analysis; some other analytical systems do not.

Spotfire addressed all three of these but the brutal truth is that most of the users who want to perform analysis don’t care about the difference between in-memory and disk-based systems, they don’t care about data volumes or pre-aggregation. They simply want to have a fast, friendly, intelligent interface into their data wherever it resides and whatever volume it happens to have, so that they can extract useful information from it.

And if we can give them that, we give them a hugely powerful tool. Printed reports are all very well, but they stifle the insight that comes from allowing a highly knowledgeable business user, with years of experience, to freely browse the historical data of the company looking for trends, anomalies and patterns. A good data discovery tool allows that unique marriage of experience and data to flower into information and ultimately business advantage.

Given this background, TIBCO felt it was important to expand the capabilities of Spotfire – on the one hand leaving its capability to analyze significant volumes of data in memory completely untouched (in fact, enhanced), on the other providing the capability to hook directly into the disk-based data held in the major database engines.

The remainder of this paper is about how this was achieved and explains some of the implications of being able to analyze both in memory and on disk.
How does it work?
Spotfire establishes a database connection to the database engine that is hosting the disk-based data warehouse. (The example we are using is a data warehouse. In practice, Spotfire can establish a database connection to any Teradata, Oracle's Exadata, SQL Server or SSAS database.)

The data remains held by that engine, but it appears in Spotfire and the user is free to manipulate the data as normal using the familiar Spotfire interface. So, for example, you might be looking at ten years’ worth of sales data and use the mouse to filter the data down to just white goods, month by month, for 2012. As the filter is applied, the data in the interface updates.

However it is important to understand exactly how this is achieved because an understanding of the mechanism reveals both the strengths and limitations of this approach. (What!? You mean it has limitations? Of course it does; all engineering solutions have their pros and cons.)

The raw data remains at all times in the data warehouse, under the control of the database engine (Teradata, Oracle Exadata etc.). As choices are made in Spotfire’s interface, Spotfire generates queries in SQL (or in MDX in the case of SSAS). Those queries are sent to the data warehouse. They are executed there, both in terms of extracting the correct data and aggregating it where necessary. The processed data is then returned to Spotfire where it is rendered into whatever presentation has been requested – bar chart, scatter plot etc. Notice the clear separation of responsibilities here. Spotfire’s job is to display the data, not only making it easy for the user to visualize that data but also to allow them to form their questions and data manipulations very intuitively. Finding the data and aggregating it is the responsibility of the engine, and the data returned by the external engine is displayed in Spotfire’s data discovery interface.

The pros here are huge; users of Spotfire have just acquired the capability to connect to multi-terabyte or even petabyte data stores and analyze them in an intuitive, well understood way.

The limitations are somewhat more subtle. Spotfire is capable of a huge range of data visualizations and manipulations, more than most comparable systems. In order to be able to work with this huge volume of disk-based data, Spotfire is relinquishing the actual handling of the data to another engine; inevitably some of Spotfire’s specific manipulations aren’t possible simply because some of the engines cannot perform these manipulations. In practice, by far the majority of the traditional Spotfire functionality is available when connected to one of these disk-based systems. The exact set of Spotfire-specific manipulations that can/cannot be performed depends on the database engine to which the connection is made.
And there are a few cases where a specific engine offers, for example, an aggregation function that is not offered by Spotfire (or perhaps Spotfire offers a similar function but it doesn’t map exactly onto the one offered by the database engine). In that case the function is usually made available to the user in the Spotfire interface.

Of course, describing this is relatively straightforward “Spotfire establishes a database connection to the database engine that is hosting the data warehouse”. In practice it is complex to achieve because the different engines use different languages and/or dialects of the same language to describe the analysis required.

**SQL – Teradata, Exadata, SQL Server**

Three of these engines (Teradata, Oracle’s Exadata and SQL Server) use SQL as their communication language, so Spotfire has to translate any request formed in its user interface into SQL before the query can be sent to the engine.

Imagine several years of sales data held in a set of relational tables. One is an Invoice table, with SaleDate and Amount as columns. Suppose the Spotfire user requests the aggregated total for the invoices in Q1 2012.

One SQL statement that could be used to extract the data necessary to deliver this information might read as:

```
SELECT Invoice.SaleDate,
SUM(Invoice.Amount) FROM Invoice
WHERE Invoice.SaleDate >= '1/1/2012'
AND Invoice.SaleDate < '4/1/2012'
GROUP BY Invoice.SaleDate;
```

(This query includes a GROUP BY because Spotfire always brings back aggregated data to reduce the volume of data returned. Indeed Spotfire intelligently controls the amount of data pulled from the on-disk store to reduce network traffic and optimize the speed of response.) This returns the daily invoice totals for the period of Q1 2012 which will number around 90 individual values. These can then be aggregated to give the requested total in a variety of ways – either by increasing the complexity of the SQL statement shown or by chaining several SQL statements together. However, to the Spotfire user this process would be transparent; they would use the interface to request the data and it would appear.

Within what is called ‘standard SQL’ it is entirely possible to express the same query in multiple ways, some efficient and some woefully inefficient. And the very nature of truly intuitive data discovery means that it is impossible to know in advance which questions are going to be posed. So in practice it is an enormously complex job to write a translator that not only produces the functionally correct SQL but highly optimized, functionally correct SQL.
And if that wasn’t complex enough, it turns out that SQL is a non-standard standard. In other words, whilst all of these engines will understand and execute standard SQL, they all have different dialects. True, the Spotfire engineers could have ignored these variations and just sent standard SQL to the engines. However, using the specific dialect for a given engine often provides huge performance benefits, so the connections to each of these engines have been highly optimized for that particular engine.

**MDX – SSAS**

And just to keep the engineers on their toes, the fourth engine (SSAS) uses a totally different language called MDX. It uses MDX because the data in SSAS is not structured as relational tables, but as a multi-dimensional structure. In other words, the data is not held as traditional two-dimensional tables but rather it is structured as a set of measures (such as Invoice Amount) held at the intersections of a set of dimensions (such as Date, Customer, Product etc.). Microsoft (along with other manufacturers) has elected to use MDX to query these structures because the language is specifically designed to express multi-dimensional queries.

In MDX the query described above is very different from SQL because date information will already be stored in the data structure within a date dimension. Assuming that the Date dimension (DateDim) includes a level for Quarter, the query might look like this:

```sql
SELECT
    {[DateDim].[Calend].[2012 Q1]} ON Columns,
    {[Measures].[Invoice Amount]} ON Rows
FROM [SalesCube]
```

Note that while the SQL query above returns the individual values which still need to be aggregated, this MDX query is returning the aggregated total for Q1 2012.

MDX makes some analytical queries much easier to express, for example, obtaining the four aggregated quarter values from 2012 can be expressed very easily:

```sql
SELECT
    {[DateDim].[Calend].[2012].Children} ON Columns,
    {[Measures].[Invoice Amount]} ON Rows
FROM [SalesCube]
```

The SQL required to perform the same operation is considerably more complex.
The bottom line is that the new features in Spotfire allow it to connect to four of the most powerful data warehouse engines seamlessly, no matter what language they use.

**How this can be used to your advantage**

Until now, Spotfire has been able to provide visualization-based data discovery against data that could be held in memory, now it can do the same against in-memory data and truly massive sets of data held on disk. And the ability to move seamlessly between these two produces a huge synergy.

Suppose that your data warehouse holds the daily sales totals for 10,000 products, over ten years in 2,000 different stores. This equates to a little over 73 billion rows of data. Suppose that you start using Spotfire to visualize the data and uncover a fascinating trend. You can continue to analyze against the main data warehouse but you might find that the focus of your interest is a certain set of fifty products in one hundred stores over the last three years. That’s about five million rows and even with 80 columns that will fit comfortably into 4 GBytes of RAM. So you can extract that set from the warehouse and analyze it in memory. This has three potential advantages:

- it may be faster than your data warehouse (depending on the power of the warehouse and how many simultaneous users it has currently)
- you can use the complete set of functions and visualizations in Spotfire
- you can work off line from the warehouse

You don’t have to subset the data in this precise way of course; you could take 100 products from ten stores over the entire time period or any other combination that yields a sub-set that will fit in memory.

And this can work the other way around, of course. You may start off analyzing a sub-set of the warehouse in memory, find an interesting trend for a certain set of products and then want to know if that is true for other products. You can simply connect to the data warehouse and find out.

The bottom line is that both in-memory and disk-based data discovery have their advantages, Spotfire now allows you to do both and it allows you to combine them.
And it isn’t just data warehouses. If you have multiple database engines (perhaps a Teradata data warehouse and a SQL Server finance system) it is now possible to use the same data discovery tool to analyze data from multiple database engines. You can even analyze data from different engines in the same Spotfire report or analysis.

Summary

We are visual animals. Most people find visualization-based tools by far the most intuitive way of understanding their data. Both in-memory and disk-based systems have their pros and cons. In the past we have had to choose between them. TIBCO’s Spotfire now allows us to use one tool to perform both types of analysis. This has the obvious advantage that we can use one tool across all of our sets of data, but it goes further than that.

Remembering that both in-memory and disk-based systems have their own unique advantages, it means that in any given instance, we can elect to use the one which has the pros we need at that point in our analysis.

We can essentially have our cake and eat it.

About the Author

Professor Mark Whitehorn specializes in the areas of data analysis, data modelling, business intelligence (BI) and Data Science. Mark is an author, academic researcher (he holds the Chair of Analytics at the University of Dundee in Scotland) and also works as a consultant designing analytical systems for national and international companies.

Additional Contributors: Ben McGraw and Erik Petterson of TIBCO Software, Inc.

1 In-memory Analytical Systems: Perspective, Trade-offs and Implementation. TIBCO White Paper