

Managing, Storing and Utilising Data for Business and Analytical Intelligence

Author: Philip Howard

Date Published: October 2004



Executive Summary

SAS believes that in business intelligence environments it makes sense to adopt different database approaches, according to the circumstances. In addition, it further thinks that it is advisable to implement such approaches specifically for the task in hand as opposed to relying on a general-purpose product that will do anything.

To be more specific, the company's view is that it is better to have a database that has been specially designed to support data warehousing along with a second database that has been developed just for on-line analytical processing (OLAP) than it is to have a generic database. The latter is not just capable of supporting these business intelligence capabilities but is also intended to support transaction processing and other front-office activities.

This white paper examines this contention both in general terms and with respect to SAS Intelligence Storage. In the latter case, the structure offered by SAS is taken to be no more than an example of the approach as adopted by SAS, and is not intended as a recommendation of that company's technology over anyone else's. It should also be noted that we are not considering alternative technologies such as column or vector-based databases, or other exotica, which might be used in business intelligence environments under appropriate circumstances. On the contrary, our emphasis in this report is on the familiar, tried-and-tested techniques represented by relational technology and OLAP.



Transactions and Business Intelligence

Introduction

There has long been a debate about the best structure for storing data. In particular, should databases be designed for general-purpose use or for specific purposes? In terms of processing transactions that debate has long been won by the relational database vendors, whose products have overcome those of alternative approaches and subsumed their capabilities. However, that is not to say that other approaches do not offer superior performance. There are environments where hierarchical and object databases, for example, continue to outperform their relational counterparts for the processing of particular types of transactions. Nevertheless, leaving marketing considerations aside, there are three main reasons why relational databases have enjoyed this success: first, they are more flexible than rival technologies; second, they are now very familiar, which creates a virtuous circle; and third, their performance is good enough, even if it is not actually as good as those offered by object, hierarchical or other database approaches.

When it comes to business intelligence, the situation is somewhat different: not in type but in scale. When it comes to analytic processing, the performance issues raised are orders of magnitude greater than those that pertain to processing transactions. Typically, transactions need to be completed in times measured in fractions of a second. One particular solution might offer a standard user response time of 1½ seconds, another might offer 1 second. Certainly this is significant in comparative terms: a 50% improvement, but it pales into insignificance when compared to business intelligence processing times, where a change in technology can easily result in 20 times performance improvements or more. Indeed, we have seen and spoken with companies that, for particular tasks, have achieved performance improvements that can be measured in hundreds and even thousands of times. Moreover, we should not forget that there are many business intelligence functions where processing time is so long (perhaps measured in days) that it is not practical to even ask the question in the first place, since it will get killed by the system's query governor or, even if the query is run, then the opportunity to make a decision based upon the result may have already passed by.

Thus there is very much more to be gained by improving the performance of business intelligence applications than there is with respect to transaction processing, which is why the debate over database storage methods in this area remains very much alive.

In other words, while conventional approaches to transaction processing generally provide at least 'good enough' processing times, this is by no means the case when it comes to business intelligence. It is therefore pertinent to consider whether it makes sense to use a general-purpose database in a business intelligence environment, as opposed to specialised facilities that have been specifically designed to cater for the needs of analytic processing.

Take a conventional relational database. Many of these are designed to provide across-the-board capabilities, with support for transactions, XML, object and



content processing, business intelligence, data warehousing, multi-dimensional slice and dice, and so on, and so forth.

In theory that sounds great: one database that does everything. However, in practice it doesn't work that way—in practice what happens is that you have multiple instances (or partitions) of the same database, some of which are used for transactional purposes and others of which are employed to support business intelligence. But what you get is the whole database on each occasion.

Now we will consider what is typically delivered with a relational, general-purpose database that you don't need in a purely business intelligence setting. Once we have done that we will go on to discuss what you still need but might do differently, or more easily, in a business intelligence only environment. However, we need to begin by considering the basic differences between the two environments.

Differences

There are three basic differences between transaction processing and business intelligence. The first is that transactions are small whereas queries are often large. That is, each transaction usually involves only a small amount of data, typically processing a record at a time. Queries and reports, on the other hand, often scan entire tables dealing with, perhaps, millions of rows. As a corollary, transactions typically take place very rapidly because there is relatively little actual computer processing to do, whereas the opposite is often the case with respect to queries and reports.

Secondly, transactions are usually concerned with updating information, while queries are mostly about reading data; and thirdly, there are usually lots of people (or processes) entering transaction data but only a relatively small number of people (or processes) that look at those transactions. The reverse is true in the case of business intelligence, where often reports are triggered by a single process but are delivered to very many desktops.

In other words, both transactions and queries are both big and small, but usually in the opposite places. It is not unreasonable, then, to ask whether it makes sense to have a technology to cover this spectrum of capabilities or if it would be more sensible to have one processing engine for transactions and another for queries. However, the issue is not quite as simple as that: people do want to query their transactional systems and they want to report directly from them. Thus it behoves the vendors of databases that support transaction processing to provide query support. The question is whether this can ever be good enough to support a pure business intelligence environment rather than just hybrid functionality?

Things you can do without

While we do not need to go into detail about every feature that you need for transaction processing that is not required for business intelligence purposes, it is



worth providing a partial list. All of the following are wholly or partially unnecessary in a query-specific environment:

- Locking and contention—virtually all of the issues surrounding locking and contention (row level locking, deadlocks and so on) disappear, since you are reading information rather than writing it.
- Similarly, there is no requirement for commits. You do not have to implement features such as deferred commits because you need to improve performance and you do not need to support two-phase commits in distributed environments.
- Recovery is much simpler. Because the database is not being updated except through external feeds there are not the same requirements for rollback, roll-forward and so forth. The logging required is greatly simplified as well because it is primarily for audit purposes rather than also being required for backup.
- Transactions require lots of small buffers to operate efficiently. Queries need a few large buffers. This is one of the reasons why transactional systems that incorporate query processing are so difficult to tune. In a database designed just for running a data warehouse you only need large buffers. Similarly, small page sizes are best for transactions and large pages for queries. Again, conventional databases have to support both.
- Another reason why tuning is so difficult is that different index types are more suitable, depending on whether you are reading data or writing it. If you optimise an index to support transactions then it is likely that that index will perform poorly if used for queries, and vice versa. The only alternative to this compromise is to have multiple indexes for different purposes, but this adds to the size and complexity of the database, which has its own adverse consequences.
- There are a variety of specialised facilities that regularly appear in transactional databases that are not required in a query-only environment. Examples include support for storage in XML format (as opposed to the ability to simply read and write XML) and the hierarchical indexing that goes along with it, nested arrays, special datatypes to support large objects of various types (spatial data, text, audio, video and so forth), support for user-defined datatypes and functions, SQL extensions to allow the processing of all these additional capabilities, and so on.
- Replication is not always necessary in a data warehouse, since this should be provided by the source environment. Note, however, that a combined transaction/query environment has an advantage here, since the movement of data is reduced—anything within the database does not have to be moved at all and this also has advantages when it comes to drill-down. However, there is an important proviso here. Normally, when you move data from a transactional system to a data warehouse, you extract the data, profile and analyse it, cleanse and match it, transform it, profile it again (in case the



transformations have introduced any errors) and then load the data into the target data warehouse. While having all of the data in one place means that you do not, theoretically, have to move the data, it does not mean that the quality of that data is any more assured than it would be in any other scenario. In other words, you still need to go through all the relevant data quality processes before you can rely on this data. This either means that you have to extract the data anyway in order to profile it (because that's the way your profiling tool works) or you do all of that processing in situ. But some of the tasks involved in profiling and cleansing are long and complex and put a significant load onto the system. Do you really want that load imposed on your operational platform? Assuming that is not the case then you will have to move the data out of the transactional system, perform the necessary data quality functions, and then reload the data back again, which eliminates almost all of the potential advantages of having the data in one place.

As an interim summary: there are a lot of capabilities in a transaction-oriented database that you do not need if you are only interested in business intelligence. What would happen if you took all of these features out? There are two conclusions:

1. The footprint of the database would be much smaller, it would be simpler, and it would be easier to manage. In particular, this should mean a lower administrative overhead.
2. The database will be much easier to tune, which should mean that it is more efficient in performance terms.

There is also a third point, not related to features per se, but relevant to the current topic, which relates to the way that the database is used. Very few companies have a single database for transactions and, in particular, the data warehouse is usually fed from multiple sources. If you attempt to combine the data warehouse with one of the transactional databases then you will also have to load substantial amounts of data from external sources into the warehouse. If these volumes are at all significant, this will have a significant impact on operational performance. Even where this is not the case, it is worth bearing in mind that there are functions of a business intelligence environment (such as large multi-way joins) that would impose a severe threat to the operational performance of a transactional system.

Finally, we should consider the question of OLAP engines. Does it make sense to embed these within the transactional environment, as opposed to leaving them as stand-alone capabilities? Taken in isolation, we do not feel particularly exercised about this. The major potential advantage is that you do not have to physically move the data in order to put it into the OLAP cube, which has the corollary that you can drill-down directly to source transactions without having to drill across the network. However, as we discussed with regard to data warehousing, the movement advantage is illusory, because you still have data quality issues to check, which means that only the corollary is an advantage and that will be offset by the performance disadvantages imposed by one system trying to be too many things at once.



Conclusion

While there are evident advantages to having a single database that performs all functions, there are also significant disadvantages. In our opinion, based both on the features discussed above and the exigencies of performance within a business intelligence environment, it is clear that a dedicated business intelligence storage approach has significant theoretical benefits when compared to the all-encompassing RDBMS alternative. However, theory is one thing and practical application is another, and the quality and design of the implementation will be important in deciding what advantages may accrue from any particular solution. In the case of SAS we will discuss this solution in the following section.

The bottom line is, of course, that this is all about cost of ownership: you can throw extra hardware resources at your solution in order to overcome any performance problems that may arise, and you can devote additional administrative resources to the management and tuning of your database if that is necessary. It is the balance between features and benefits on the one hand and the cost of providing that functionality on the other, that is the determining factor. This equation will depend upon the solution being considered as well as the individual requirements of each potential user and, as such, is too specific for consideration here.



Product description

Introduction

The idea behind SAS Intelligence Storage is that it is better to have different types of database storage, each of which has been tailored to the needs of a particular type of business intelligence, rather than a single data store that attempts to do everything. As we have discussed, this is a plausible and sensible approach, in that this minimises the additional functionality that is required in each database, thereby reducing its footprint and making it easy to administer and tune. Specifically, SAS offers two types of database: a relational database that has been designed specifically to support data warehousing, and an OLAP database for multi-dimensional analysis.

However, SAS Intelligence Storage is just one part of the SAS infrastructure (the SAS9 Intelligence Platform) and it is important to appreciate this fact. In particular, there is a separate metadata repository that is shared by the various SAS databases as well as the company's other products, such as its ETL (extract, transform and load) tool, and its front-end business intelligence applications. In other words, although this is not a major discussion point within this document, SAS Intelligence Storage is a part (albeit a key part) of a wider picture, rather than merely representing stand-alone functionality.

Another major feature of the SAS9 Intelligence Platform is that there is a single management console, which is used to administer the whole environment. In other words, all administration, whether for either or both of the databases, for ETL, or for any other part of the environment, is done through this interface. In terms of database support, this puts SAS on a par with vendors that have adopted an all-in-one database approach but ahead of anyone that requires an additional environment to support ETL or other processes.

Products

In SAS9, the company has significantly re-engineered its database products. In particular, its OLAP engine has been re-designed while its relational storage offering has been significantly extended through the introduction of multi-threading capability and partitioning.

SAS Intelligence Storage consists of both relational and multi-dimensional storage options. While there is only one option in the latter category, there are four in the former, as follows:

1. Base SAS Engine—as illustrated in Figure 1, is a single threaded product with single user/process update capability. With data stored in a single file and metadata stored along with the data, this solution is most suitable for small data marts or to support data mining models with small sample sizes.

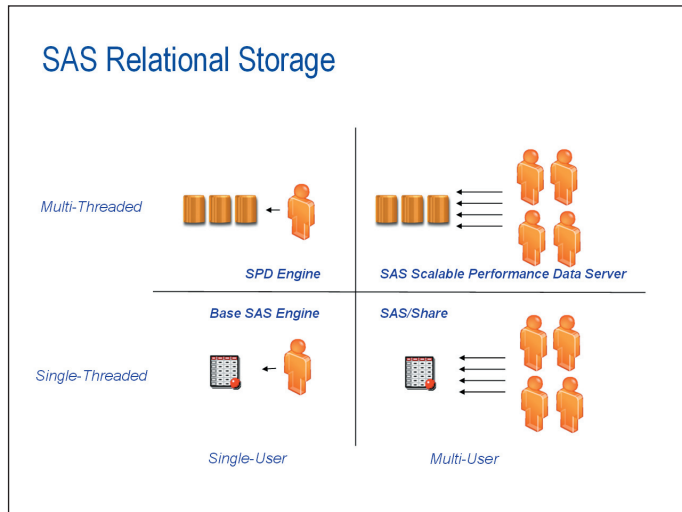


Figure 1: SAS Relational Storage

2. Base SAS Engine with SAS/Share—if specific applications require multi-user write capability then SAS/Share adds this capability to the Base SAS Engine.
3. SAS SPD Engine—this supports multi-threading, and partitioning and parallel processing, as well as SAS’ hybrid indexing capability (see later). It is deemed to be most suitable for medium to large data marts or to support data mining with large samples. Note that the storage method used is distinctly different from that used in Base SAS.
4. SAS Scalable Performance Data Server (SPD Server)—as the top end product in this range it is SPD Server that we will be focusing on in this white paper; it includes

all of the features of the SPD Engine but extends these by providing SQL optimisation, a separate security (name) server, and automatic aging rollout. This last, which is not discussed in this report, provides automatic rollout of data from the data warehouse on an aged basis, which is particularly useful for the very largest data warehouses.

SAS Scalable Performance Data Server

In this document we do not intend to provide a complete description of SPD Server (nor, in the section that follows, of SAS Open OLAP Server). Instead, we will be focusing on the major elements of these solutions as they relate to performance. In the case of SPD Server this means the partitioned architecture, extended multi-threading and parallel capabilities, the hybrid indexing used by SAS, and the use of Name and Proxy Servers for security purposes.

Architecture

There are two important aspects of the architecture of SPD Server, which is illustrated in Figure 2, to discuss. The first of these is the partitioning scheme shown, while the second is the use of the Proxy Server.

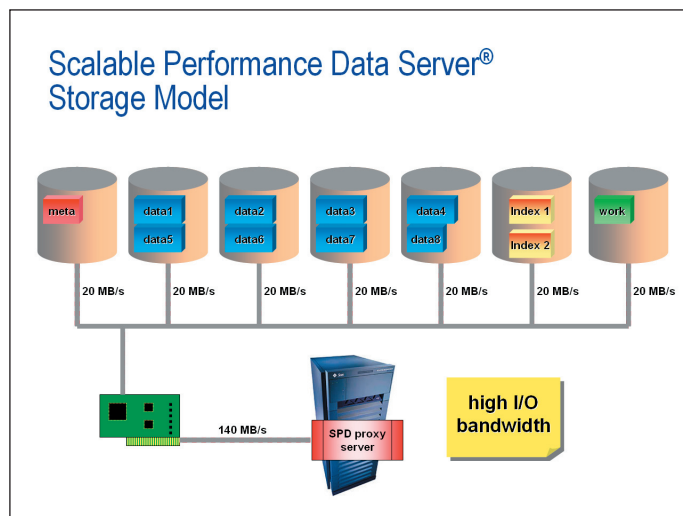


Figure 2: I/O-Architecture of SPD Server

In terms of partitioning, this is organised as illustrated. As can be seen, SPD Server partitions its data across multiple disks with separate disks also being used to store metadata, indexes and current work. The partitioning of data is organised on a round robin basis, which means that data is assigned to disk 1, disk 2 and through to disk 8 and then back to disk 1 again. The actual number of disk processors is, of course, not limited to four.

The point about this approach is that the throughput from each disk system is additive rather than based a lower common denominator. Thus, with the architecture



illustrated, throughput should be a total of 140 Mb/sec rather than 20 Mb/sec. This isn't quite true, of course. For any particular query you need to address the indexes first and then retrieve data but, given that index access is only a very small part of any particular query, then these figures can generally be taken as accurate.

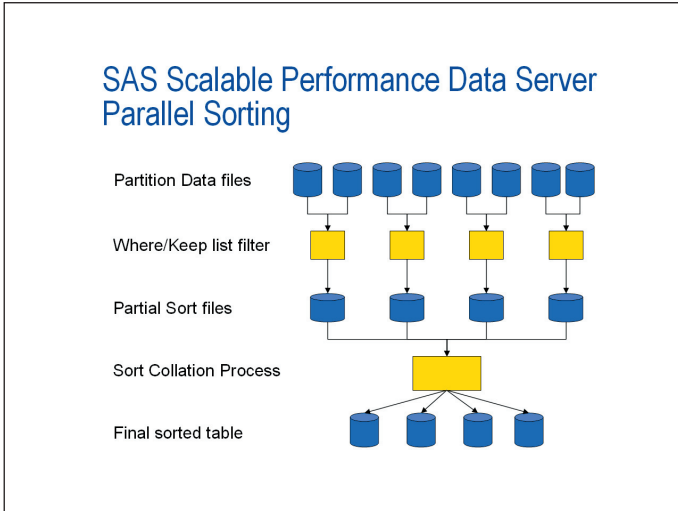


Figure 3: Sorting in SPD Server

In practice, the largest live installation SAS has seen, at a UK bank, has demonstrated 800 Mb/sec, which is not just the practical limit for the hardware concerned (running HP-UX with a Symmetrix SAN) but is also the theoretical maximum. This level of match between actual and theoretical throughput is very impressive.

Partitioning, of course, enables parallelism. While a number of the parallel features of SPD Server will be discussed in the section on Indexing, one that is relevant to consider here is with respect to sorting. How this is accomplished is illustrated in Figure 3.

What is happening here is that multiple threads are reading data from the various partitions and then these are filtered into subsets using the appropriate SQL functions. Sorting then continues in parallel on the partial sort files shown, prior to the final collation process, after which the results will be written back to disk, again using the product's parallel capabilities.

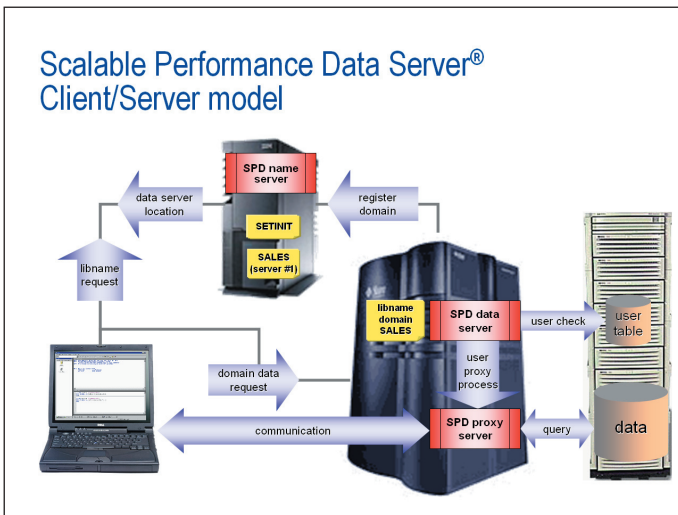


Figure 4: Using the Proxy Server in the Client/Server model

Note that the important thing about all this is that the performance of this whole operation is not dependent on the size of the initial table but only on the size of the individual subsets of the data.

The second aspect of the architecture that it is appropriate for us to discuss is the use of the Proxy Server. This is part of the client/server architecture that is implemented by the SPD Server product, as illustrated in Figure 4. It can be seen here that SPD Server in fact uses three servers: namely a name server, a proxy server and a data server. All client requests are addressed to the name server (using Access Control Lists) and this is the only instance known to the client. Moreover, each client is assigned its own proxy, which has the significant advantage that if there is a failure then only that user is affected.

Indexing

The indexing used by SAS is a hybrid method that combines a Btree with bit maps. Specifically, a Btree structure is used when values are unique (which is determined automatically by the software) and bit maps are used when there are non-unique values. Arguably this is not ideal since it would be reasonable to use Btrees for all low cardinality values (that is, when there aren't many duplicates) but that would add to the complexity of the system and particularly to index tuning



requirements, since you would have to decide the best cardinality to use. Overall it is therefore moot as to any advantage that might accrue. However, what is not moot is that this combination should provide both good indexing performance and a reduced footprint.

This last factor is worth explaining further. The size of most data warehouses is typically several times the size of the raw data—around five or six times is commonplace though we have heard of implementations where this has been as much as ten times. The reason for this increase is primarily due to the proliferation of indexes that are required in order to improve query performance. While SAS does not do anything to reduce the number of indexes required, its hybrid indexing is significantly more efficient than other methods and, according to SAS, only takes up around a third of the footprint of alternative approaches, where Btrees and bit maps are typically treated as separate rather than combined approaches.

Another important aspect of SAS indexes is that they are segmented. These segments (which are 8Kb each) have been designed specifically to support parallel processing and provide superior performance for a number of processes, including join optimisation, WHERE clause sub-setting and table scans. This last is particularly important since large table scans can take a very long time and it is worth reporting on SAS' own tests in this regard. In-house it has run a 577Gb table (this is not the largest known—one customer has a single table of over 1Tb with 2bn rows) with 740 million rows and 84 columns whereby a full table scan returning 4,000 rows took just 15 minutes. While not as fast as some data warehouses that use specialised techniques (that is, not standard relational approaches) this figure is still very good: we are aware of implementations where full table scans of this sort of magnitude can take times measured in hours rather than minutes.

Another feature of indexing within SPD Server is that the metadata related to indexes is stored separately from the indexes themselves. The big advantage that this provides is that it minimises index I/O and, in particular, it enables parallel processing during index scans, index builds and append (refresh) performance. These last procedures can be very time consuming when it comes to maintaining the data warehouse. For example, in Base SAS, appending information to an index can be slow (and this is not an aspersion on Base SAS, it is merely typical); as a result of which the company recommends that you simply delete the existing index and create a new one. This is not the case with SPD Server as it is much faster to add new references to an index than to create a new one, even bearing in mind that the procedures to create a new index are much faster with the product's latest features anyway.

SAS OLAP Server

SAS supports MOLAP (multi-dimensional OLAP), ROLAP (relational OLAP) and HOLAP (hybrid OLAP), where the former allows you to store OLAP data within the company's own SAS OLAP Server and the second allows you to store the same data in either a flat file or within a star schema using SAS' relational storage or a third party relational database. HOLAP, on the other hand, allows you to mix these different storage methods. So, for example, you might use a star schema



in a data warehouse (SPD Server) for high cardinality dimensions and the OLAP Server for low cardinality dimensions. As the name of this section implies, we will be focusing on the OLAP Server in this section though the ability to support other approaches should be borne in mind.

In many ways, the developments in the OLAP Server are similar to SPD Server. In particular, the software makes use of other servers when it is appropriate to do so, and it makes extensive use of parallel capabilities. In the first case, this use of specialised services is exemplified by the use of the SAS Workspace Server, which has the specific function of creating the multi-dimensional structure, which is written to the SAS OLAP Server. A number of advanced facilities are supported by the structure including parallel drill hierarchies (for example, year – month – week as opposed to year – quarter – week – day), member properties (useful additional information that you cannot drill from), and support for multiple languages within a single cube, which is enabled by the fact that SAS' multi-dimensional storage is Unicode compliant.

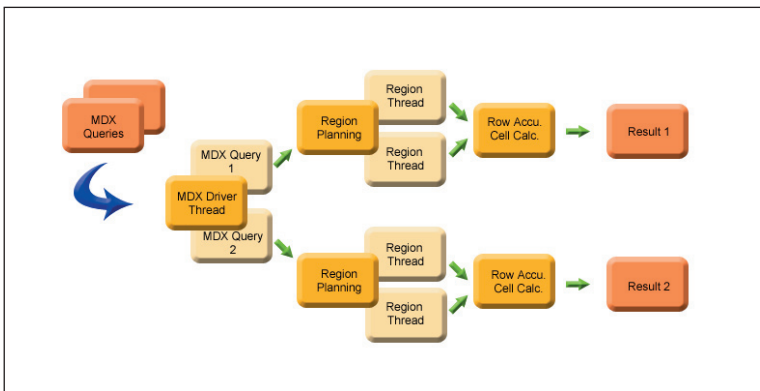


Figure 5: Main features of the OLAP Server

The other big thing with the OLAP Server (like SPD Server) is that it makes extensive use of both parallel and multi-threaded facilities. In Figure 5, which illustrates the main features of the OLAP Server, for example, there is one thread for each MDX query (where MDX is an OLAP standard) so that queries are executed in parallel. In addition, storage is also parallelised, through use of the OLAP Plug-in, which allows you to spread data across multiple systems. This allows you to store fundamental cube data on one system,

indexes and other similar data on a second system, and additional aggregations on further systems, for example. In particular, high level (fundamental) data can be stored in a flat table, while more low-level detailed information can be stored using partitioning, thus optimising performance regardless of the sort of data that you want to look at.

A number of other points about this diagram should be noted: first, an axis is SAS terminology for a dimension; second, a region is a slice through the OLAP cube and, as can be seen, region working is also parallelised, with each region having its own thread; and third, note the (configurable) caching option.

Summary

In this white paper we have considered the arguments in favour of having dedicated and specialised storage mechanisms for different tasks, especially in the business intelligence and analytics areas. Secondly, we have considered whether SAS Intelligence Storage delivers a solution that qualifies it as a leading specialised provider of data warehouses and data marts.

We do not think that there is any doubt about either of these questions. With regard to SAS Intelligence Storage it should be evident from the preceding dis-



cussions that the company has taken great strides in optimising the performance it can provide. As far as the principle of the matter is concerned, the argument between specialised and general-purpose storage is the same one that says that a surgeon specialising in neurology is better equipped to operate on your brain than a general practitioner.

However, database structure is not quite such a life and death issue. Moreover, there is a difference between theory and execution—just because one approach may be better than another in theoretical terms, does not mean that the former will outperform the latter, since the quality of the implementation is also an issue. However, from our discussions about SAS Intelligence Storage it should be clear that we believe that the implementation has been well done and it is therefore difficult to envisage a general-purpose database (as opposed to another specialised provider) providing comparable storage efficiency and performance across the whole spectrum of business intelligence requirements, from reporting to data mining, without throwing a lot of extra hardware at the problem. The task for a vendor of such a database would be to make this extra hardware cheap enough, including the additional administrative and management costs required by such solutions, so that it can compete with SAS Intelligence Storage or similar products. We do not say that this is impossible but it is certainly a tall order.

Copyright & Disclaimer

This document is subject to copyright. No part of this publication may be reproduced by any method whatsoever without the prior consent of Bloor Research.

Due to the nature of this material, numerous hardware and software products have been mentioned by name. In the majority, if not all, of the cases, these product names are claimed as trademarks by the companies that manufacture the products. It is not Bloor Research's intent to claim these names or trademarks as our own.

Whilst every care has been taken in the preparation of this document to ensure that the information is correct, the publishers cannot accept responsibility for any errors or omissions.



Suite 4, Town Hall, 86 Watling Street East
TOWCESTER, Northamptonshire, NN12 6BS, United Kingdom

Tel: +44 (0)870 345 9911 – Fax: +44 (0)870 345 9922
Web: www.bloor-research.com – email: info@bloor-research.com