Paper 114

Large Scale Data Warehousing with the SAS® System

Tony Brown, SAS Institute Inc., Dallas, TX
Leigh Ihnen, SAS Institute Inc., Cary, NC
Jim Craig, SAS Institute Inc., Austin, TX

Abstract

Aspects of design, modeling, and physical implementation of data warehouse structures are driven by the volume and data access patterns that are involved. The SAS® System provides excellent scalability for data warehouse structures. There are several key architectural and physical implementation and access issues that must be dealt with to enable the best possible performance. This paper will discuss large data task issues, scalability in data architectures, and how the data can be modeled, stored, and physically I/O managed to achieve the best performance with the SAS® System.

This paper should be of interest to experienced data warehouse practitioners and users, who currently or will work with large scale systems. An assumption is made that the reader has very basic knowledge pertaining to disk I/O subsystems.

Introduction

A crucial ability of any data warehouse system is to scale to the enterprise level. Large scale data warehousing introduces more complex engineering challenges for design, modeling, and physical implementation to sustain reasonable performance levels. When the warehouse begins to encompass multiple terabytes of data, the underlying design, architecture, hardware, and software system it is built upon must be robust enough to support it. The SAS® System can readily meet multi-terabyte data warehousing requirements. Appropriate modeling, architecture, and physical performance implementation on the hardware system, can take advantage of the power and functionality of the SAS® System.

This paper will briefly discuss the nature of large volume data tasks, and how their characteristics affect architecture and the use of different models. Physical implementation issues will then be addressed to explain how to implement an efficient and practical I/O plan to surface the large amounts of data needed in a timely manner for the solutions required. Finally, some tuning tips will be covered, followed by a brief discussion on parallel solutions.

Large Volume Data Usage

Nature of Tasks for Large Volume Operations

Tasks involving large amounts of data are largely comprised of query and reporting, decision support, and time series analysis functions to support predictive modeling. These tasks are different in nature, with different processing and access patterns.

Query and Reporting

Query and reporting tasks in large data environments are often of a dimensional reporting nature, e.g. "How many Gold tier clients in the Northeast Region bought our premium brand ice cream in July?" The big difference in this query in a large data environment is the volume of data encountered to solve the query, and the result set size. The goal of query and reporting is to obtain experiential "counts and amounts" type of information, sliced by available descriptive variables, including time. The queries are very ad-hoc, and "query by example" in nature, and are experienced by the data warehouse in a high volume. These types of data access do not typically re-constitute the time dimension as a series, or if they do, only at a very high level of aggregation, and not across large dimensions. In addition, these types of queries are usually aimed at well defined levels of granularity. Query and reporting tasks are much the same in the in the large data environment as smaller environments, though the underlying data architectures can be very different.

Decision Support and Time Series Analysis

Larger scale data operations also involve the tasks of measuring and tracking changes in customer behavior, personal demographics, experiential and lifestyle attributes, etc. over time. These operations are utilized to enable decision support and time series analyses to predict future experience based on a concrete understanding of past experience as reconstituted in time series of data. The time series aspect of this process is critical to capture actual point-in-time changes crucial to understanding when and why changes took place, and what future changes are likely. These types of queries tend to be very data and resource intensive. The data captured in these time series are subsequently used to feed several types of predictive models to help make decisions about future activity.

A good example of the typical decision support/time series analysis query involves selecting a population of customers via a dimensional query, and then accessing that customer populations demographic data, reconstituting the time series to determine changes in behavior at specific points in time. As stated earlier, these time series specific data points, once compiled, can then be used by predictive models (regression analysis, neural nets, etc.) to predict future behavior changes and build profiles.

Large Scale Design Performance

Query and Reporting - Influence on Architecture and Modeling

The use of very large data stores for query and reporting alter the architecture and modeling decisions that would normally be made on smaller data structures. The sheer size of the data being loaded into the system, or queried from it dictate new considerations in terms of how to model and store the data for best performance.

Summarized data marts and sub-warehouse structures are obvious alternatives to keep as many queries out of the large granular tables as possible. But they cannot service all data requirements.

Many warehouse modelers use the Star Schema to store data used for granular level dimensional querying and reporting . This tends to work fairly well until a table size grows large enough to present performance problems. Those problems surface in load and query operations.

Most tables (especially the Fact table, and large Dimensions) in a Star Schema usually contain multiple composite indexes to support critical retrieval times of sub-populations of data. The indexes often utilize different variables, not in the same left-to-right variable order as the primary index or table sort order. As the table grows, the differently ordered indexes can take longer and longer to build, even in a data append situation. Testing on high volume tables shows the processing growth associated with these types of index builds to be non-linear.
Note: Version 7 of the SAS® System is introducing an indexing feature that will alleviate some of this problem. In version 7, as long as the Base and appending datasets have the exact same variables, an internal sort will be performed on the values required to complete the append process for each index in the Base dataset. This improvement is added along with some other internal short cuts (more efficient appends for observations that are added with the same value of an indexed variable) ,and showed good results in internal SAS® benchmark tests.

Sometimes simply using a step-wise approach to delete indexes, append to such a table, then create the indexes from scratch can have significant savings. For example an internal test on a process updating household and person tables from PUMS Census data from 10 states (representing 1,162,741 households and 2,611,538 individuals) was conducted.
The tests showed an enormous reduction in CPU time in table load (over 800 percent) by destroying the indexes, loading the table, then rebuilding the indexes, rather than appending to the table utilizing the index. *Note: Version 8 of the SAS® System will introduce a parameter to the index delete request that will allow the use of an option _ALL_ that will delete all indexes without the internal cleanup step between each index delete being executed. This will help speed up the index deletes portion of this alternative, but the primary expense is in rebuilding the indexes.*

Even these approaches have a practical limit though. Given continuous data growth, at some point, partitioning a table for both load and query performance may become necessary.
Table partitioning should be explored when either the table update time is too long or table query performance begins to become unacceptable. Partitioning Star Schema FACT

tables on a time based scheme can improve both load times and query performance. The load (append) process has only the current time element table to update – reducing it's workload.

By partitioning extremely large fact tables by time, only the table or tables of interest need be searched for the appropriate observations to satisfy a query, negating the wade through an extremely large table, with large indexes. A helpful factor is that most dimensional reporting queries are time-based, and many compare one time period to another (e.g. this month's counts vs. this month last year).

Large Dimension tables can also be partitioned by hashing algorithms, demographic, or other means.

When table size grows significantly in a Star Schema, table partitioning can reduce both Load and Query execution time. On very large data, more normalized schemas are not usually recommended because they exacerbate the performance problems mentioned earlier. They do have a place in instances where the queries involving highly normalized tables are infrequent, and space savings is required.

Query and Reporting - Query Management Considerations

When a Star Schema relational structure grows to a terabyte level, query performance must be carefully studied and managed. The sheer size of the Cartesian product resulting from many joins (especially across large Dimensions) can disable practical performance. There are strategies to avoid Cartesian product detailed in Data Warehouse Efficiency Techniques with the SAS® System , Tony Brown, SUGI 23 Proceedings. These techniques can be used to aid query performance in both partitioned and non-partitioned table schemas. These strategies, and others can (and in many cases should) be encompassed in a utilization management facility for very large warehouses.

Utilization management of the data warehouse becomes a more important issue when complex queries joining terabytes of data can literally bring the warehouse to its knees.

The problems encountered here involve numerous uncontrolled queries launched by end-users, sometimes very poorly written, and without regard to the performance impact on the system. Utilization management of such queries can provide a step-wise facility to help users build more intelligent, efficiently written, and bounded queries. An application front-end can enforce query syntax, and SQL operations efficiency requirements such as bounding very large queries, and can enable the maximum system performance for the community at large. Such an application can also relegate extremely resource intensive queries to special run times or after hours, freeing the peak time usage for more manageable queries. While this type of application places some restrictions on users, it can provide better overall query performance for the community as a whole.

Decision Support and Time Series Analysis - Influence on Architecture and Modeling

Because the demographic data (time series) is voluminous, and stored in time differential data sets, modeling it in a relational format such as a Star Schema is impractical. To attempt to re-constitute a time series of data from a large group of customers for example, would create a huge Cartesian product between the joined demographic tables. Such a query is highly impractical, and can easily take an unacceptable amount of time to complete, especially if you pull a time series of data points on your entire customer database. The nature of the time series data and their processing dictate that they be stored in a similar fashion as operational data stores (ODS). A preferable model is to use SAS® datasets partitioned by a time period appropriate to achieve an acceptable file size given the granularity. Monthly partitions usually work well. If the volume is very high in monthly datasets, they may need to be partitioned not only by time period, but by another scheme as well (e.g. geography, hashing algorithm, etc.). An advantage to this model is that it only requires use of the time period datasets of interest.

The partitioned SAS® datasets model utilizes the MERGE process to join the files. This process is efficient if the files are maintained in the correct sort order as operational data stores and do not have to be sorted on the fly. The resultant merged product represents a collection of time series snapshots that are normally used to feed a numerical modeling process. These resultant files or extracts, are usually saved permanently in a staged area, or on a separate server.

In a very large system, if time series data are partitioned by time period and another scheme, the result is having to read and combine a potentially large number of datasets. It is usually not feasible to keep all of the time series data online, so only recent data is kept online with historical data stored offline for immediate restore on demand.

Parallel processing should be explored as an option when the time of the linear process combined with the extract volume begins to grow outside expected performance boundaries. This subject will be discussed later in this paper.

Decision Support and Time Series Analysis - Query Management Considerations

Decision Support/Time Series Analysis queries are typically executed by modelers. When a large population of modelers is accessing the time series data stores to pull modeling extracts, or a small group is demanding large volumes of extracts, the I/O subsystem can become overloaded quickly. The same utilization management techniques discussed for Query and Reporting access need to be evaluated. This type of system is even more important for an extract system because of the large volume of the time series operational data stores. An advantage to working on this economy of scale is that it quickly becomes economically feasible to build and operate a query/extract utilization management system.

By forcing time series users to go through a management system, crucial system performance can be maximized through scheduling of requests, and bundling of like requests (that require the pull of the same time periods of data in the series). With careful metadata management of online and offline data, the management system can determine if requested time series data is available online, in a current “restore” state, or needs to be restored from tape.

Understanding some data model and access issues, the focus must be turned to actual physical implementation of the data architectures in the hardware system. These physical implementations must meet the business requirements for performance, therefore careful attention must be paid to their design and construction.

Physical Performance Scalability Issues

System Throughput and Processing

As data volumes scale, system throughput becomes critical to performance. As stated earlier, it is a different engineering feat to move one or more terabytes through a system in an operation as opposed to several gigabytes. The entire processing system must be analyzed from a physical standpoint to determine what maximum sustained throughput can be achieved. This will determine fitness of the system to perform the required tasks, as well as give information as to how the tasks should be engineered to perform best on a given system. This analysis must pertain to the CPU capacity, system bus bandwidth, RAM, NVRAM, cache capacity, file system chosen, controller arrangement to disk arrays, array setup and management, etc. This analysis must also consider what optimization is needed for the SAS® System to perform at its maximum for the given situation.

Due to limited space, this paper will focus on I/O related issues associated with moving data to and from the disk array.

Physical Storage Arrangements

Operations performed against data on a storage system consist of read, write, and update. These activities can take place in a random or sequential manner. Understanding the nature of which of these categories is most important in terms of performance, and most prevalent (random or sequential), is helpful in making decisions about hardware setup and tuning.

Once your operations are understood, some of the decisions involving setting up a storage facility involve:

- Deciding which file system to use
- Storage system organization
- Considerations for using memory, file system and controller cache
- Physical strategy of controller usage, striping, and data placement for data surfacing

File System Choice

There are many choices in file system selection, depending on which brand of hardware you are on. The material discussed in this paper will pertain to comparing two of the Unix file systems only, since that is the environment in which the majority of enterprise level data warehousing is performed. The types of file systems compared will be indirect-block-based and extent-based file systems.

The choice of selecting an indirect-block-based or an extent-based file system depends on several issues. Some of these involve how the file system caches reads and writes (read aheads, write behinds), how that caching can affect memory, and also about file system tuning flexibility.

An indirect-based file system breaks files into 8K blocks that can be spread over the disk. Additional 8K indirect blocks keep track of the data block locations. For files greater than a few megabytes, double indirect blocks are used to keep track of the location of the indirect blocks. When a file is read sequentially at high speed, the system has to keep seeking to pick up indirect and scattered blocks. This seek time limits the maximum sequential read performance to about 30 MB/s, even with fast CPU and disk performance. An example of an indirect based file system is the Unix File System (UFS).

Veritas VxFS (Veritas File System) is an extent-based file system. An extent based file system keeps track of data by using extents. An extent is a set of contiguous disk blocks that is treated as a unit. They can vary from a single block to many megabytes in size. Each extent contains a starting point and size. If a 2 gigabyte file is written to an empty disk, it can be allocated as a single 2 gigabyte extent. When the file is read, there are no indirect block reads, just the extent record read, then the data for the entire extent. By using extents, file systems like Veritas VxFS can do fewer I/O's, in multiple blocks, than the block-at-a-time operations performed in UFS. VxFS can also do snapshot backups of a file system while it is writing data, without stopping the application doing the writing.

In addition, VxFS does not require a unified buffer cache for read-aheads (the kernel maps files directly in to its virtual address space and allows the memory management units to take care of the rest). Unlike UFS, VxFS allows individual file systems to be set up with separate parameters. It provides tunable write throttles, read-ahead (caching the next several blocks of data to read ahead of the time it is needed) and write-behind (caching disk writes, then organizing the write process in the most efficient manner possible to reduce seek time) parameters for separate file systems. UFS allows this, but the same parameters will apply to all file systems created. The primary drawbacks to using VxFS is that it requires a considerably larger amount of system memory than UFS, and if large extents are used, they can fragment the disk. VxFS generally does allow shorter recovery times, faster file system checking, and higher performance with large volumes of data on some platforms.

Compounding the decision on which file system to use is the fact that VxFS on a Hewlett Packard platform is not the same as VxFS on a SUN Microsystems platform. The HP version is their adaptation of a journaled file system based on Veritas VxFS, and doesn't have the same characteristics as the SUN Microsystems implementation of the Veritas VxFS file system. *Note: HFS has proved optimal for large sequential writes of SAS datasets over other file systems for the HP platform.*

Knowing the frequency/latency of the reads is important. If the system is going to experience high frequency/low latency reads in concert with low frequency/high latency reads, they can't be interleaved well if the file system extents are extremely large.

Disk Organization

When setting up disk arrays for usage, the balance between cost, performance, and safety must be considered. The adage of "Fast, cheap, safe; pick any two" applies here. Some of the ways to achieve the combinations are listed below:

Fast and Safe - duplicate the disk contents onto a “mirror” set of disks. Pros – This option is easy to manage, and provides fast and safe access to data. Cons - This option can be prohibitively expensive in terms of hardware and write processing for very large data volumes.

Cheap and safe – RAID5. Use a Redundant Array of Independent Disks, with a parity stripe to act as a mechanism for immediate recovery to a hot spare.

Pros - When there is sufficient memory, and blocks are equal in size or larger than most I/O requests, RAID-5 has very good random read performance because drives can respond independently to requests, and simultaneously to multiple requests. When block sizes in a stripe are smaller than the average size of an I/O request, forcing multiple drives to work together, RAID-5 systems can have good sequential performance.

Cons - Small writes performance is affected here, because in a single-block write command, the system must complete several I/O operations and parity calculations in a read-modify-write sequence. This hurts random write performance (although this is not often a big issue for data warehouses). The parity disk itself cuts down on available storage. RAID controllers require a lot of non volatile memory (NVRAM) to perform write-behind safely, and to coalesce adjacent writes into single operations. Without a lot of NVRAM, write performance could be extremely poor. In the event of a disk loss, read performance is degraded until the lost stripe portion is recovered from parity. *Note: Never stripe SAS WORK space with a parity stripe as in RAID-5. It will severely hamper SAS performance, as all writes to the work area are calculating and writing parity – you don’t want to do that! It is highly recommended to configure SAS Work in a RAID-0 configuration, on an I/O path that is independent from other applications. If it becomes necessary, you can use multiple file systems and spread users across them by using multiple config files.*

Fast and cheap – multiple disks, no parity or mirror, used as raw partitions. Pros - This can be very fast for individual disk operations. Cons – Managing raw partitions of data can be a headache. Volume Managers are worth their money. Extending logical volumes across multiple physical volumes can have some serious performance penalties. In the event of a disk failure, recovery is dependent on backup speed. For extremely large volumes of data, this method can be risky and painful if the system availability does not allow a long interruption.

The behavior of raw disk partitions versus stripes above, must be balanced with the physical I/O characteristics of raising large amounts of data off the disk through the controllers to the system memory. This will substantially affect the equation of whether to stripe or not, and will be discussed in detail later.

Memory Usage

Earlier, we stated the importance of understanding which of your particular operations was performance critical – and whether the operations were more random or sequential. An example of a random operation would be a query against a few rows scattered throughout a table, a sequential would be a full table read.

Understanding these operations and access patterns is important, because anything that can be done to turn random accesses into sequential accesses, or increasing the transfer size of the accesses can have a positive effect on performance.

Throughput can be greatly increased on a disk subsystem by increasing I/O size and reducing the number of seeks. Buffer and memory usage are two components in this operation.

<u>Balancing Buffer Cache and Memory on UFS.</u>
The UFS maintains a large cache of I/O buffers. This buffer cache allows the file system to optimize read and write operations as discussed earlier. When a program writes data, it is not written immediately, but stored in the buffer, and later flushed by a daemon which gathers a group of buffers, looks at the transfers in the group, and figures out the physical location of each data block to make the write transfer in the best possible order. This buffer cache is also used to hold “read-ahead” blocks in memory for more efficient read operations. So it sounds like you would want to increase this buffer cache by as much as you could, doesn’t it? That is not always the case. Increasing the buffer cache puts

the squeeze on main memory. Squeezing main memory can in some cases force paging and swapping which could overload the CPU and I/O subsystem. In addition, the buffer cache's size becomes less important with extremely large files.

In some UFS implementations (SUN OS4, SVR4) there is no special buffer cache allocation, but main memory is used to cache pages of code, data, or I/O.

Memory on RAID Controlled Systems.
Earlier, statements were made about RAID controlled systems requiring sufficient amounts of NVRAM to achieve good performance on disk writes. The amount of NVRAM placed on the controller itself does not have to be much (newer systems have 64 to 128 Mbytes), if additional NVRAM is placed in the memory system on an I/O bus. Since system memory is cheaper than cache memory, and much faster to access from the CPU (doesn't require disk I/O calls like cache memory), it makes more sense to add it to the system memory.

While disk controller cache is helpful (especially with the addition of system NVRAM) for write performance, it doesn't help read performance. It is usually too small to be an effective read cache, and adding NVRAM here is a waste. The first read will miss the cache, and further references to the data in main memory will never involve the disk subsystem (and the cache). Even for writes, the optimal place to add memory is on the main system.

So whether you are on a UFS or VxFS file system, adding more NVRAM to main memory can help both reads and writes.

Physical Implementation and Striping

A disk stripe consists of logical contiguous blocks of data spread across multiple independent disks (spindles). The amount of data written to each individual disk is the stripe depth. *Note: Large SAS datasets tend to exhibit the best performance when a stripe depth of 64KB is used.*

When a disk striping implementation is selected for use on a disk cabinet, there are a couple of ways to handle it, and the affect on I/O can be significant. The first way to implement a stripe set is to stripe "down the controller", in a hardware stripe. This stripe is placed across multiple disks in an array, all governed by the same hardware controller (hence "down the controller"). This effectively limits the stripe to one hardware controller as its "I/O highway" to transfer data through.

By utilizing a software controlled stripe, and striping "across controllers", a stripe is placed across multiple disks in an array that are governed by different controllers. This is like expanding to multiple highways.

What are the advantages of a software stripe? First, let's consider the practical limitations of a single disk controller. If you utilize a very large file, striped across an array down the controller, you are sequentially trying to surface that table through a single controller, and the maximum rate you can sustain is the rate of the controller. By striping across disks on different controllers, the table is surfaced by multiple controllers at once (each surfacing the data on its disks). This essentially aggregates the throughput of several controllers for a single file system. It "parallels" the surfacing process, with the transfer time now being divisible by the number of controllers used (actual transfer time will be held to the rate of the slowest controller in the system).

The throughput measurements used to make such decisions as how to stripe should be based on sustained performance rather than peak performance measures. Sustained performance measures tend to range between 50 – 65% of peak performance measures. Experience shows that a SAS process can sustain in excess of 50 MB/s performance using aggregated I/O subsystems (striping across controllers). Using that rate, table sizes can be used up to the point that the sustained throughput measures won't surface the table in the required time (even using controller aggregation). Then dataset partitioning strategies (perhaps across more volumes) need to be employed.

In a RAID-5 disk array, hot spares can still work off of a different controller if this type of stripe method is used. For heavy large files access, RAID-5 stripes across controllers can be a very advantageous method. Coupled with intelligent table layouts, two of which are discussed below, this method can enable significant parallel advantages.

Data Placement

How data is placed on a disk architecture can enable or stagnate performance. There are two primary task separation concepts that are important to consider to maximize performance when many large files are being accessed in a system. The most critical is physically separating read and write tasks between files, and the second is physically separating large read tasks between files that are processed together. Separating these tasks is handled by placement of data in the system.

For the first example we will use the scenario of separating reads and writes in updating a very full system of many large SAS datasets (such as a time series ODS).

For the second example we will consider processing a few files that are read (such as joining two tables in a Star Schema).

Separating Reads and Writes

In earlier discussions, the benefits of separating data onto different controllers was entertained.

When many extremely large datasets (comprising 1/2 to 1+ terabytes) are being written to (updated) in a data warehouse, performance is significantly enhanced if the reads and writes are separated. This is especially true for an ODS type data store, where the data are partitioned in time series, or other formats (hashing group, geographic, demographic, etc.). This can be done on a large scale by synchronizing the file systems onto separate read and write file systems on different I/O paths.

By having files read out of one file system (read-from) in an I/O subsystem, updated, and then written to a separate mated file system (write-to), head conflict and to and from data movement through one file system are avoided. This type of operation predicates that there is a read-from and write-to file system available. This is best accomplished by mating two file systems on separate I/O paths (in separate cabinets if available) and moving the data from one file system to the other in a flip/flop fashion in update cycles.

When this activity is expanded across multiple sets of disk cabinets, the benefits of synchronizing the update of multiple terabytes of datasets become obvious. The read and write heads are not bounced around, and the process has the efficiency of a fluidly, synchronized ballet.

This type of operation requires a dynamic metadata management system to keep track of where the starting and ending points of the fixed location data are moved to and from, and where the data is in it's current update state. Such a highly synchronized process only works well for situations where the additional administrative management cost of the data synchronization is economically outweighed by the performance cost of updating terabytes of data. Such systems work well with large, time series operational data stores.

Separating Multiple Reads

In addition to allowing a large dataset to be surfaced by multiple controllers at once, is the idea that placing data on physically separate file systems can further reduce the sequential process of reading multiple datasets.

If two datasets are read for use together (similar to joins or merges in a Star Schema operation) and are placed on separate file systems on separate I/O subsystems, they can be accessed simultaneously. If enough volumes are available in the subsystem, separating reads for datasets used together is the second important thing to do to avoid serialization of a process.

Tuning SAS® for Performance

There are several tunable parameter settings in the SAS System. They can be tuned to allow maximum performance in a variety of situations. They will be listed below with some brief information about each, and further reading should be done using the appropriate SAS manuals and sources listed in the bibliography of this document.

MEMSIZE – the soft limit on the amount of memory that the SAS process will allocate and does not include the SAS image size. This can be set for the session, but is limited by the kernel parameter MAXDSIZE. This should be set for sessions based on experiential need.

SORTSIZE – the soft limit on the amount of memory that the SAS SORT procedure will allocate. This is limited by the MEMSIZE parameter. This should be set for sessions based on experiential need.

Notes: Sorting and index building are significantly faster if performed in memory. The SORTSIZE and MEMSIZE parameters determine whether this can happen or not. If the sort will not fit into the allocated memory and disk I/O is required, setting it higher will have very little effect.

Summary and FREQ procedures build B-trees in memory and spools to disk if the MEMSIZE parameter is too small to allow the operation to complete in memory.

BUFSIZE – buffer size of SAS I/O for SAS datasets. The SAS system dynamically chooses a value that matches a small number of observations per page (usually 4K – 6K). This parameter can be specified at a datasets level. *Note: For large files that are read sequentially, 64K is recommended.*

UBUFSIZE – the buffer size of SAS I/O for utility files. The SAS system defaults to 8K. *Note: Sorting elapsed time can be improved by increasing to 64K or 128K.* The parameter is adjustable by overriding with an OPTIONS statement.

IBUFNO – an optional number of extra buffers that the user can obtain from main memory for the use of speeding up index navigation. The system automatically sets up a minimal number of buffers, so this can help, especially in a SHARE environment. Do not use more buffers than performance improvement indicates as they do take away from main memory.

IBUFSIZE – the buffer size for use with indexes. These should be set based on testing in your particular situation and environment.

Parallel Implementations

When the amount of data, and/or CPU processing becomes extremely large, parallel processing can offer benefits in many situations. There are multiple ways to approach parallel processing.

One method is to use a software product that recasts the sequential query task into parallel processes through the SQL optimizer. Essentially the software takes a sequential process, and based on internal optimization techniques the SQL process uses, launches parallel processes to divide up the workload on separate CPUs, recombine the finished work, and deliver the output result.

Pros – You don't have to know anything about the underlying physical architecture of the data to use this method. These types of products work best with query and reporting types of tasks as opposed to time series manipulations described earlier in this paper.

Cons – The methods the optimizer uses to handle the parallel work division and process are not explicitly manifest outside the SQL optimizer. They are essentially "black boxes" running on a serial thread, launching parallel processes. For small result sets, these work well, but on larger result sets they can represent throughput bottlenecks on serial surfacing of data to the process that does the parallel thread launching.

The second major method of implementing parallel processing is use explicit data physical parallel processing through programmatic techniques that launch multiple physical processes. This method essentially is explicitly "human controlled". Essentially it involves launching multiple simultaneous SAS sessions to accomplish the work. This type of parallel method has significant performance benefits for very large result sets. In order to accomplish this, a thorough knowledge of the data, access patterns against it, etc. are necessary. Unlike the SQL optimizer method described earlier, the practitioner needs to handle the work division across the physical parallel threads, managing the processes, and handling the result integration from each of the processes. This sounds design intensive and it is. These systems are usually "hard-wired" in nature, with a fixed paradigm of how processes, datasets, and results are handled.

Pros – Handles large result sets that benefit from parallel process flows well. There is more control and flexibility to exploit data and take advantage of architectural structures for performance. This method can also be engineered to automatically take advantage of utilizing near-line and off-line stored data. One of the biggest performance advantages is that it can eliminate serial data throughput issues, because it uses separate physical process on different disk I/O subsystems and file systems as described in previous examples in this paper.

Cons – It takes an extremely thorough knowledge of the data, and the processes that will be run against it, and the access patterns that will be used in those processes. It is predicated by a thorough knowledge of disk subsystem performance architecture issues. Also required is an intelligent plan to author, manage, and implement the processes completely under the auspices of metadata management. Without metadata management to automate the intricacies involved, the system will not be maintainable, especially if near-line/off-line storage is used.

Parallel processing is a resort that is entertained when serial data movement through a system will no longer deliver the required performance results. It can be expensive and complicated and should be practiced by knowledgeable individuals. SAS Consulting Services has experience implementing both of these types of parallel implementation with our products, and is available for companies that require help in implementing these solutions.

Summary

The SAS® System provides a scalable data repository, processing engine, management and exploitation tool set that can warehouse and transform multi-terabyte volumes of data into information with effective performance. Utilizing intelligent data models, I/O subsystem design, and data surfacing strategies, advantage can be taken of SAS® System engine formidable throughput capabilities.

References

SAS Institute Inc., "SAS® Language, Version 6, First Edition", order number 56076. Available through SAS Institute's Book Sales.

Mike Loukides, *System Performance Tuning*, O'Reilly and Associates, 1990

Tom Madell, *Disk & File Management Tasks on HP-UX*, Hewlett-Packard Company/Prentice Hall, 1997

Adrian Cockcroft, Richard Pettit, *Sun Performance and Tuning, Java and the Internet*, Second Edition, Sun Microsystems Press/Prentice Hall, 1998.

Adrian Cockroft, *Increase System Performance By Maximizing Your Cache*, http://www.sunworld.com/sunworldonline/swol-02-perf.html;

For Additional Information:

Tony Brown
SAS Institute Inc., Dallas, TX
214/977-3916 sasabr@wnt.sas.com

Leigh Ihnen
SAS Institute Inc., Cary, NC
919/677-8000 saslai@wnt.sas.com

Jim Craig
SAS Institute Inc., Austin, TX
512/258-5171 saswjc@wntunx.sas.com