

Version 9: Scaling the Future

Diane Olson, SAS Institute Inc., Cary NC
Robert Ray, SAS Institute Inc., Cary NC

ABSTRACT

In Version 6, SAS introduced the SPDS (Scalable Performance Data Server) as a technology to exploit SMP (Symmetric Multi-Processor) hardware for speeding up data services. In Version 9, BASE SAS introduces the SAS Scalable Architecture (SSA). SSA makes parallel processing and partitioned I/O constructs available to the entire SAS System for the first time.

What does that mean to you? It means that some elements of your jobs will take full advantage of SMP architectures to reduce time-to-solution for critical tasks. This paper presents an overview of this new strategy to improve your compute and I/O bound tasks, including specifics on how you may get a significant performance increase for your SAS jobs.

INTRODUCTION

The data processing and warehousing industry is bombarded with immense volumes of data from both the “bricks and clicks” sides of the business; however, the data processing time remains unchanged. In response to this data onslaught, SAS has a strategy that will be incorporated in an evolutionary manner over the next several releases, starting with Version 9.

Changes in Version 9 include performance enhancements for both I/O and compute-bound problems. These performance enhancements are realized by a combination of strategies:

- partitioned I/O
- multi-threaded I/O
- parallel multi-threaded computation

This paper begins with an overview of SSA and enumerates the problems Version 9 starts to address. Features to remedy these problems are explained, along with how you can make use of the new features. Preliminary performance benefits resulting from these changes are included.

SAS SCALABLE ARCHITECTURE

Successfully scaled performance is not obtainable by simply installing faster processors or I/O devices. Achieving true scalability is a balancing act involving the choice of scalable

hardware and some choices about the software that is specifically designed to leverage it. For the Version 9 SAS user, these choices include selecting optional enhanced procedural algorithms, choosing the best SAS engine to create and access data, how (or indeed, if) data sets are partitioned, as well as the number of threads launched to best process those partitions.

SSA combines the legacy strengths of SAS with the ability to create a highly scalable solution in order to meet the evolving market demands, taking advantage of hardware advances. This involves embracing elements of the classic MVA (Multi-Vendor Architecture) server, SPDS data server, and technologies for lightweight threading. Integration of SPDS-like technology means employing threads to distribute I/O requests across multiple controllers. These threads may in turn create other threads to manage the processing of blocks of data. Thus, SAS flows from a mostly synchronous model to one that allows I/O functions to run asynchronously across multiple processors in an SMP environment. Properly used, an SMP environment can employ the CPUs in the system to process data simultaneously, avoiding bottlenecks that slow the computation of results.

When trying to speed a single task or problem, there are two types of scalability to consider - the inherent scalability of the problem in question and the scalability of the software solution for that problem. Problem scalability can vary greatly. The problem of sorting, for example, generally scales computationally on the order of $N \log_2 N$, where N is the number of records to be sorted. However, if the I/O device cannot keep pace with the CPU, then the scalability will be linear with the size of the file (N). A full SQL join will scale computationally as $N * M$, where N and M are the table row counts. Therefore, doubling the number of observations for both tables in a full join could consume four times the CPU resources. However, if the process were I/O bound, the actual scalability would be linear with the combined size of the tables ($N+M$). Therefore, reducing time-to-solution is a complex problem involving both CPU and I/O optimizations.

With software scalability, the goal is to apply additional physical resources (CPUs or I/O channels) and have the real time-to-solution be lowered by a proportional amount. The real time is the focus here, and not the combined CPU time. It is a foregone conclusion that extra CPU cycles are consumed to manage a set of process threads across multiple CPUs. The portion of the original problem that can actually be processed in parallel governs the amount of scalability achieved from the software solution. For instance, although a partitioned data set can be read in parallel for SORT, the sorted data is written out in a linear fashion to preserve the sorted state. If writing the data takes 50% of the original time, then only 50% of the process is scalable; this represents the limit of scalability for this problem. Further improvements in time-to-solution can only be achieved by increasing the speed of the output I/O devices.

EVOLUTION

Rewriting the entire legacy MVA code to be thread-safe (i.e. where the code runs with no side effects in a threaded environment) much less thread-hot (i.e. where the code is thread-safe and exploits the threaded environment for faster execution) would be a major undertaking. To avoid delays in delivering releases to the customer, the move to thread-hot code has been planned in evolutionary stages.

In Version 9, some shared code modules used by selected high profile procedures are being reworked to be thread-hot. In the I/O arena, a new subsystem was created to enable reading blocks of data versus record-by-record reading as in previous SAS versions. Several multi-partitioned engines will allow simultaneous block-mode reading of data from multiple partitions in parallel. A small number of procedures are being converted to exploit this new subsystem in Version 9. These include SORT, SUMMARY, DMREG and REG, with more procedures expected to come aboard in following releases.

BASE SAS procedures have a set of incremental goals for leveraging SMP architectures from the traditional MVA development environment. First,

some key procedures' algorithms are being modified to decrease time-to-solution using the traditional single-partition data set. In these cases, both parallel and pipeline algorithms are being applied, as appropriate for the application. The opportunity to decrease the time-to-solution for single data partitions exists when the problem is somewhat CPU bound; in these cases, the procedure can use multiple threads, and thereby CPUs, to match the data processing rate with the I/O read rate. Once the read rate of a single partition is matched, additional increases in speed must be achieved by using either a faster single I/O channel or partitioning the data so that multiple partitions can be read simultaneously. Many BASE SAS procedures exhibit this "fence straddling" between being I/O and CPU bound, depending on the nature of the data being processed and the procedure options being used.

Not all procedures will be able to exploit the new parallel I/O system, which is capable of reading blocks of data across partitions simultaneously. PROC PRINT, for example, expects to print the observations of a data set from the first observation through the last; reading the data set simultaneously across threads does nothing to speed ordered access. PROC SUMMARY, however, does not necessarily depend on ordered access and so could leverage parallel data access.

In Version 9, the new I/O subsystem will have limitations that may prohibit parallel data access under certain circumstances. These include BY processing and some engine-dependent capabilities. In addition, there will be no parallel data writing capabilities. These exceptions will be addressed in forthcoming releases. See Table 1 for a per-engine description of factors that prohibit the use of the new I/O subsystem. Note that this table is expected to change in subsequent releases. This evolutionary plan allows quick delivery of performance enhancements to the customer as well as laying groundwork for future advances.

FEATURE	BASE ENGINE	ACCESS ENGINE	SPDS ENGINE
Compression	No	Yes	Yes
Encryption	No	Yes	Yes
BY processing	No	Yes	Yes
CEDA	No	No	No

Table 1 - Feature support per engine with new I/O subsystem

BENEFITS

The large data sets of today have changed significantly from that of a decade ago. The size of today's large data sets would have been unthinkable in the Version 6 timeframe; even so, results from processing those gigabytes of data are still expected to be obtainable in a reasonable amount of time. Advances in hardware technology help improve throughput, but without software advancements, processing time can still be unacceptably slow. The migration to a threaded SAS architecture will lead to faster processing of not only today's data sets, but also those of the future.

Faster processing is possible because SSA provides the tools necessary to exploit multiple CPUs concurrently, to read data through multiple I/O channels simultaneously, and to overlap I/O and data processing. SAS applications will have the opportunity to utilize these tools to increase the total throughput of a single SAS server session. Of course, the degree of time-to-solution improvement gleaned is directly related to how CPU bound or I/O bound the problem is. SSA works in tandem with the MP-CONNECT technology; MP-CONNECT enables parallelism via multiple concurrent SAS invocations.

PROBLEMS

There are several potential bottlenecks standing in the way of a full solution. Which bottleneck that is restricting your performance depends on both the operation requested and the nature of the data itself.

Depending on the type of user request, some applications will be I/O bound, that is, CPU time to actually process the data is negligible in comparison to the time required to read the data. Data sets with many variables and many observations are more likely to create this situation, but it is still dependent to a great extent on the operation requested. For example, requesting the means of a numeric variable is not CPU intensive, and when run on a large data set the process will be I/O bound.

Conversely, PROC REG of such a data set is likely to be compute-bound since the CPU time to process the observations is greater than the time to read the observations.

The goal of adding threaded access is to alleviate both types of bottlenecks, solving the problems of

- Speeding I/O
- Speeding Computation

Some procedures, like PROC SORT, are not solely I/O bound or solely compute bound, but contain processes that are. Some of these processes may be I/O bound, while some may

be compute bound. These procedures benefit from a combination of the solutions for speeding I/O and computation.

Of course, you are **always** going to have some bottleneck, but the software developer's ultimate goal is to have it be the hardware that constrains the time-to-solution.

SOLUTIONS

That ultimate goal is certainly a desirable one, but it seems esoteric without a structured implementation strategy. For Version 9, that strategy is enumerated by attacking problems on three fronts, including those applications that are I/O bound, those that are CPU bound, and those that are a mixture of each.

SPEEDING I/O

For I/O bound processes, the goal is to read as quickly as possible, keeping the computation process supplied with data. The solution for increasing data set access speed is two-fold; first, optimize the read rate for each data partition, and second, allow multiple partitions to be read simultaneously using multiple I/O controllers.

Traditionally, SAS has always read data sets record-by-record, a style that is extremely flexible and lends itself to noting, pointing and indexing. However, speed-reading is more important for I/O bound applications. Thus, the new I/O subsystem reads data sets a block at a time, resulting in lower function call overhead. Delivering a block of data is somewhat less flexible than record I/O, as it is not as easy to address a particular record in the data set; however it fulfills the need of the I/O bound problem – more data in less time.

The second part of the I/O solution is to simultaneously read partitioned data sets via multiple threads. Partitioned data sets are data sets that are located in different physical locations, but still comprise a logical data set. With this strategy, not only are the block reads done in parallel threads, but processing of the data can also be done in separate threads. Each thread reads and processes data independently; when all partitions have been read and processed, the application folds the result from each partition (or set of partitions) into a final solution. Note that more than one partition may be read and then processed in a single thread.

The degree of increased throughput still depends heavily on hardware considerations. If there is only one I/O controller, for example, it does not matter how many threads are launched; they would all serialize waiting on the controller. Assuming no hardware constraints, the limiting factor becomes the number of partitions

comprising the data set and the number of threads launched to process those partitions.

There has never been an engine that processed partitioned data sets shipped with BASE SAS. Beginning with Version 9, however, it is planned that the partitioning SASSPDS engine will be shipped with BASE SAS, available on all platforms. ACCESS users will also have the choice of Oracle, Sybase, DB2 or Teradata as their partitioned engine. This allows all SAS customers to employ partitioned data sets if they so desire.

SPEEDING COMPUTATION

When I/O is not the only limiting factor, parallel computation using modern thread constructs allows further speedup to occur. The two basic models of parallel computation are the boss-worker model and the pipeline model. Both models allow a computationally intensive task to be divided and distributed to multiple CPUs within a shared memory model using lightweight process threads (LWPs). Each SAS procedure that is reworked to improve scalability will use one or both of these techniques along with parallel I/O where appropriate in order to reach its scalability limit.

In order to utilize threads, the traditional SAS MVA procedure will create process threads that essentially run outside of the MVA environment, synchronizing back to the MVA process once some portion of work is complete. Since these new process threads do not execute in the current SAS MVA process space, they do not detract from and may even enhance the interactivity of the SAS environment during the execution of long-running procedures on SMP machinery.

SCALING PROC SUMMARY

One BASE SAS procedure that can leverage both the parallel and pipeline techniques is PROC SUMMARY. When a CLASS statement is used with PROC SUMMARY, the rate at which data can be classified, sorted and aggregated is often not as fast as the maximum data read rate. In order to keep pace with input and minimize amount of redundant aggregate groups in memory, we choose to pipeline this process by dividing the summarization process into distinct steps and assigning each step to a separate thread. Data is processed in blocks by each stage and passed on, much like an assembly line. Memory resources required by each step are not duplicated; there is only one thread per step. It is possible that some stages in the pipeline may be able to make use of multiple threads. In these cases, the memory domain of the stage is divided and assigned to separate threads, thus avoiding memory duplication within a stage.

Once parity with the maximum read rate of a single partition of data is achieved, the model must be replicated across multiple partitions to achieve further speed-up. If classification is performed from separate input partitions, the elements of the pipeline could be replicated for each partition. However, this could result in a great expansion of memory required to solve the problem, as each domain could contain partial aggregation results for identical output levels. If the aggregation space is shared across multiple partitions/thread, excessive shared memory access overhead is a possible result. Clearly there is a tradeoff involved for parallel multi-partitioned summarization.

When the CLASS statement is not used, only the aggregation step of the summarization process is left; this causes the problem to be I/O bound. In this case, scalability is achieved by allowing parallel aggregation across multiple partitions. The results of each partition aggregation are merged together and the input scan is complete.

RESULTS TO DATE – PROC SUMMARY

Thus far the work on PROC SUMMARY has focused on summarizing from a single partition data set using multiple CLASS variables. Using multiple CPUs, for some cases the time-to-solution is within fifty percent of the time it takes to simply read the data set and write the results. This is approaching the best that can be done with a single partition and represents a significant improvement over the current summary logic.

SCALING PROC SORT

Another BASE procedure that can benefit from multi-threaded design is PROC SORT. Sorting is generally I/O bound, but thread technology provides the opportunity to overlap I/O and processing in some steps. The boundness of internal sorting can be a function of the ratio of the sort key length to the record length, as this affects the number of CPU cycles used per key comparison. Therefore, to ensure that key processing keeps up with the maximum read rate, a process pipeline with fan-out is being used. When the size of the sort exceeds the available memory, the sort must shift to an external algorithm, spooling partially sorted results to one or more utility files. These partial results are finally merged together to produce the final result.

External sorting adds many complications, as well as many opportunities for performance improvements. By employing multiple threads in the V9 design, the phases of multi-way merging can be isolated and have dedicated threads for each phase. These phases include runs creation, utility file read-ahead, merging records and writing output. Effective utility file read

scheduling is critical to reduce head seeks during external merging. Anyone sitting near a hard drive during an external sort merge-back can attest to the "weed whacker" sound produced by heavy disk seeking. Seek elimination makes for more efficient (and quieter) external sorting. By assigning this read-ahead function to a separate thread, records flow smoothly into main memory for a merging thread to process.

RESULTS TO DATE – PROC SORT

Measurements thus far for Version 9 PROC SORT have focused on internal sorting. For sorts where the key size is less than fifty percent of the record length, it is relatively easy to sort as fast as the I/O rate since the job is primarily I/O bound. However, if the key increases beyond eighty percent of the record length, as many as four sorting threads can be effectively employed in order to keep pace with the input rate. The additional work comes from the fact that each key comparison takes longer than the time to read each record. Thus, as the I/O rates improve, the multi-threaded sort on SMP machinery will keep pace; this will provide the performance improvements the customer expects from their hardware investment.

USER'S RESPONSIBILITIES

To scale performance using Version 9 SAS, the customer's input is of utmost importance. First and foremost are the hardware decisions. Because some processes serialize, the maximum throughput of any given I/O device may become the limiting factor for the time-to-completion. Lots of slow disk drives are not necessarily equivalent to a couple of really fast ones. However, having only one really fast disk drive will not allow partitioning to decrease throughput time. SMP hardware appropriate for your I/O bound and/or CPU bound applications is necessary to SSA's effectiveness.

Another user responsibility is the SAS engine choice. If you wish to have multi-threaded access across partitions, you must choose an engine that supports partitioned data sets. It is planned that you will have the choice of using the SASSPDS engine or one of the partitioning ACCESS engines. The default BASE engine will not partition files.

Certain engines or attributes of a particular data set may limit your data to record I/O access. See Table 1 to determine if any of those limiting factors will affect your application. For example, the BASE engine is not planned to support block I/O reads when the data set is encrypted. If encryption was less important than performance, then the file could be rewritten without encryption and block I/O reads could be used.

SUMMARY

Partitioned I/O, parallel I/O, parallel processing and algorithm changes are all a part of the Version 9 strategy of speeding your time-to-solution. With these evolutionary changes, expect to see performance gains from selected applications using SMP hardware. The gains seen in Version 9 are merely the beginnings of the plan for scaled performance increases across all applications.

Your comments and/or questions are very welcome. Please contact the authors at:
Diane Olson - Diane.Olson@sas.com
Robert Ray - Robert.Ray@sas.com