Paper 312-2009

The SAS® Scalable Performance Data Server® – Controlling the Beast!

Abdel Etri, SAS Institute Inc., Cary, NC

ABSTRACT

Achieving maximum scalability in order to fully exploit symmetric multiprocessing and parallel I/O hardware has always been a challenge. The SAS® Scalable Performance Data Server®, a multi-threaded server, answers that challenge by using the capabilities of a Symmetric Multiprocessing (SMP) platform to provide scalable and high-performance data storage and access.

SAS Scalable Performance Data Server is designed to scale across multiple processors, such that adding additional processors and resources improves performance seamlessly. The application also provides tuning parameters that help you maximize performance by taking the best advantage of system resources.

This paper discusses how to effectively tune your SAS Scalable Performance Data Server parameters and to configure your I/O subsystem for optimum performance. The paper includes the following topics:

- organizing your data to minimize I/O constraints
- configuring threads to ensure that they are adequately busy
- configuring memory to maximize performance for sorting and index creation
- influencing parallel WHERE planning and SQL join performance

INTRODUCTION

The SAS® Scalable Performance Data Server® (SPD Server) is designed to meet the storage and performance needs for processing large amounts of SAS data. As the size of data increases, the demands for processing that data quickly increase. The scalability of the hardware plays a critical role in the overall performance of SPD Server. SAS SPD Server software uses SMP technology to optimize WHERE clause processing, read tables, load data, build indexes, and sort data.

SPD Server is used at hundreds of customer sites worldwide and hosts some of the largest known SAS data warehouses. SAS SPD Server provides high-performance SAS data storage to customers in banking, credit-card, insurance, telecommunications, healthcare, pharmaceutical, transportation, and brokerage industries; and to government agencies.

SYMMETRIC MULTIPROCESSOR (SMP) ARCHITECTURE

A symmetric multiprocessing environment has the following attributes:

- supports multiple CPUs that share the same memory, storage, and input/output resources
- provides a thread-enabled operating system
- spawns and processes multiple threads simultaneously using multiple CPUs
- enables the application to coordinate threads from the same process

ORGANIZING SAS SPD SERVER DOMAINS FOR SCALABILITY

The SAS SPD Server software allows partitioning of data and separation of components across different file systems. See Figure 1.

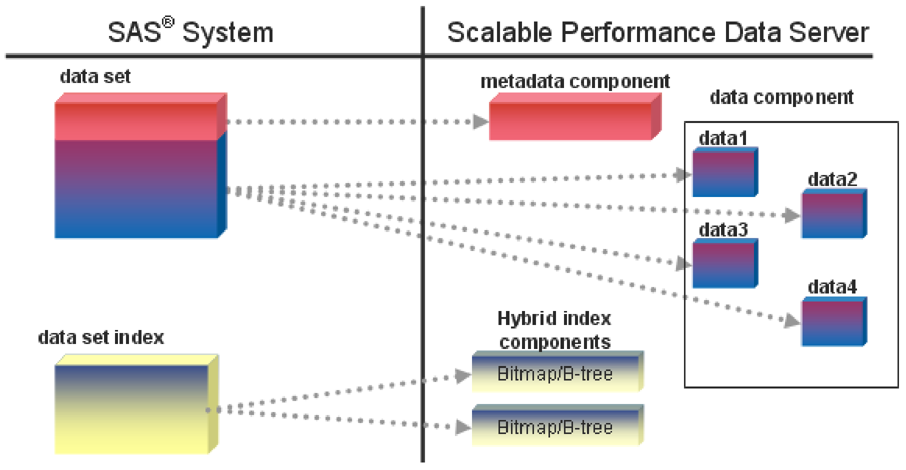


Figure 1. SAS SPD Server Storage Model

SPD SERVER LIBNAME PARAMETER FILE

The SAS SPD Server software uses the parameter file LIBNAMES.PARM to define the directory paths for metadata, data, and indexes for each SAS SPD Server LIBNAME domain.

On start-up, the SPD Server reads the information stored in the LIBNAMES.PARM file. The LIBNAMES.PARM file establishes the names and file storage locations of SPD Server domains during the server session. SPD Server administrators can use the LIBNAMES.PARM file as a central tool to control SPD Server domain resources and user access to SPD Server domains.

These are examples of LIBNAMES.PARM file domain declarations:

```
LIBNAME=simple PATHNAME=/data/simple;
LIBNAME=spd123 PATHNAME=/spdmeta/spd123
        ROPTIONS="DATAPATH= ('/data1/spd123'
                            '/data2/spd123'
                            '/data3/spd123'
                            '/data4/spd123')
                INDEXPATH= ('/index1/spd123'
                            '/index2/spd123')";
```

SAS SPD Server may use these four different areas to store data and other information:

- metadata area
- data area
- index area
- work area

METADATA AREA

The metadata area keeps information pertinent to a LIBNAME domain, its data tables, and its indexes. SPD Server also uses this area to store other resources, such as catalogs and views.

The physical metadata location is determined by the PATHNAME= option of the LIBNAME domain definition in the SPD Server LIBNAMES.PARM file. It is vital not to lose any metadata. Although the space required for the metadata is small, the set-up and configuration of the disk space is very important. The disk space must be expandable, mirrored, and backed up. Therefore this area needs disk set-up, which features primarily redundancy, such as Redundant Arrays of Independent Disks (RAID) 10 or RAID 5 array.

DATA AREA

The data area is where the data partition files (DPFs) are stored. The data area requires some thought and planning in order to provide high I/O-throughput as well as scalability and availability. The physical data location is determined by the DATAPATH= option of the LIBNAME domain definition in the SPD Server LIBNAMES.PARM file.

The DATAPATH= option specifies a list of file systems (under UNIX systems) or disk drives (under Windows) where the data partitions are stored. SAS SPD Server randomly chooses a data path, as a starting point for round-robin partitioning, to receive the first data partition. The second data partition is stored in the next data path in sequence, and so on. SAS SPD Server is shipped with the option RANDOMPLACEDPF, which is set by default in SPDSSERV.PARM file (discussed later in “tuning SPD Server parameters”). The random placement strategy manages large tables efficiently and balances data partitions across all data paths.

To ensure redundancy, use either a mirrored stripe-set (RAID 10) or a RAID 5 array. RAID 10 features the best performance while maintaining redundancy at the same time. RAID 5, also referred to as striping with rotating error correction code (ECC), has the best ratio of redundancy and performance versus cost on the other side, as shown on Figure 2.

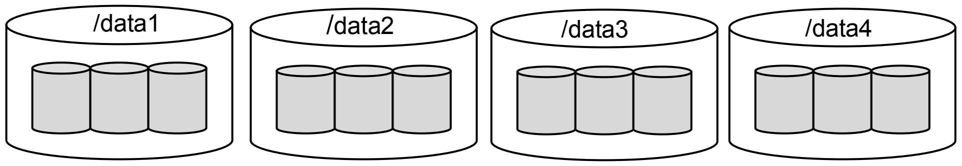


Figure 2. Four RAID 5 arrays, each striped across three disks

DATA PARTITION SIZE

The default setting for partition size is 128 megabytes. The partition size might need to be adjusted to maximize throughput. For tables that are stored in the SAS SPD Server, the default setting for the minimum partition size is controlled by using the MINPARTSIZE= parameter (see section “tuning SPD Server parameters”) in the SPDSSERV.PARM file. This parameter sets a minimum partition size before a new partition is created. SPDSSERV.PARM file is shipped with a setting of 256 megabytes. However, it is not uncommon for sites to increase the setting to 512 megabytes, 768 megabytes, or 1024 megabytes. Users commonly create tables in the range of 250 gigabytes or higher, so when they do not explicitly set the partition size, the MINPARTSIZE= setting controls the number of partitions that large tables have.

For a more complete discussion about data partitions, see the SAS Institute white paper “Partitioning of SAS® Scalable Performance Data Server® Tables.”

The goal is to choose a partition size so that the tables DPFs are distributed among all of the data partition paths while avoiding a DPF explosion due to a partition size that is too small.

Too small partition size creates many files that could run into file system or user limitations on the number of open files (file descriptors). Too large partition size can cause I/O contention and impact performance because DPFs will not distribute across all data paths.

INDEX AREA

The index component files are stored in the index area. The physical index location is determined by the INDEXPATH= option of the LIBNAME domain definition in the SPD Server LIBNAMES.PARM configuration file:

```
INDEXPATH= ('/index1/spd123'   '/index2/spd123')
```

The INDEXPATH= option specifies a list of file systems (under UNIX) or disk drives (under Windows) where the index component files (HBX and IDX files) are stored. The use of multiple index paths should lessen I/O contention in multi-user environment as shown in Figure 3. The I/O characteristics are the same as the data area ones.

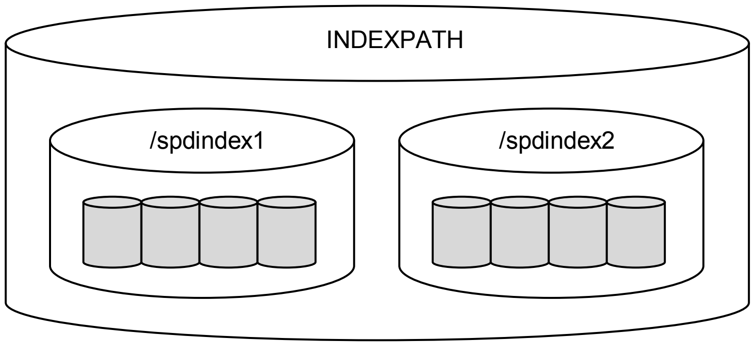


Figure 3. Index area spread across two RAID 5 arrays, each striped across four disks

WORK AREA

The work area is where temporary files are created. For example, temporary utility files can be generated during SPD Server operations that need extra space, like parallel index creation, or sorting operation of very large files.

Normally a stripe-set of multiple disks (RAID 0) should be sufficient to gain good I/O-throughput. However, high availability could require choosing redundancy (RAID 5), as a loss of the work area could stop SPD Server from functioning entirely.

The physical location where SPD Server stores temporary files is determined by the WORKPATH= option specified in the SPD Server SPDSSERV.PARM configuration file:

```
WORKPATH= ('/spdwork1'  '/spdwork2')
```

The SAS SPD Server supports the use of multiple work paths and many of the parallel processes will distribute utility files they create across the multiple work paths allowing the system to scale.

DISK STRIPING AND RAIDs

A defining feature of all RAID levels is disk striping. Striping is the process of organizing the linear address space of a volume into pieces or chunks that are spread across a collection of disk drive partitions. The stripe size is the largest amount of data written to disk before moving to the next disk. For example, a RAID 5 consisting of four data disks and one parity disk (4+1) uses a stripe size of 64KB. With a stripe width of only 256KB (64KB x 4 disks), even small files (< 1MB) will be present on all four disks in the volume. Stripe 0 lives on disk 1, stripe 1 lives on disk 2, and so on as shown in Figure 4.

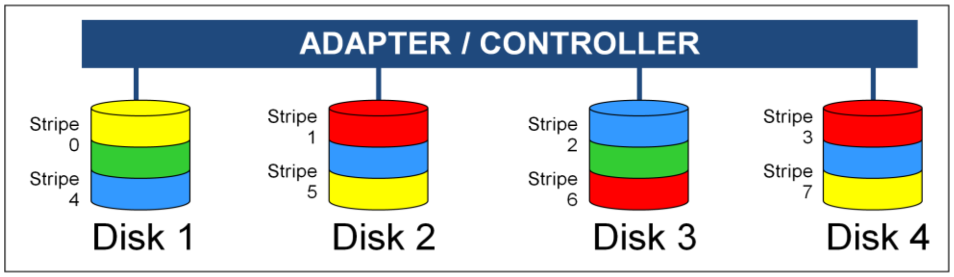


Figure 4. Disk striping

By distributing the stripes of a volume across multiple disks, it is possible to do the following:

- achieve parallelism at the disk I/O level
- boost I/O throughput

This also reduces contention and data transfer latency for a large block I/O because the physical transfer can be split across multiple disk controllers and drives.

Regardless of RAID level, disk volumes should be hardware striped when using the SPD Server software. This is a significant way to improve performance. Without hardware striping, I/O will bottleneck and constrain SPD Server performance.

The following is a brief summary of RAID levels relevant to the SAS SPD Server.

RAID 0 (also referred to as striped set)

High performance with low availability. I/O requests are chopped into multiple smaller pieces, each of which is stored on its own disk. Physically losing a disk means that all the data on the array is lost. No redundancy exists to recover volume stripes on a failed disk. A striped set requires a minimum of two disks. The disks should be identical. The net capacity of a RAID 0 array equals the sum of the single disk capacities.

RAID 1 (also referred to as mirror)

Disk mirroring for high availability. Every block is duplicated on another mirror disk, which is also referred to as shadowing. In the event that one disk is lost, the mirror disk is likely to be intact, preserving the data. RAID 1 can also improve read performance because a device driver has two potential sources for the same data. The system can choose the drive that has the least load/latency at a given point in time. Two identical disks are required. The net capacity of a RAID 1 array equals that of one of its disks.

RAID 5 (also referred to as striped set with rotating ECC)

Good performance and availability at the expense of resources. An error-correcting code (ECC) is generated for each stripe written to disk. The ECC distributes the data in each logical stripe across physical stripes in such a way that if a given disk in the volume is lost, data in the logical stripe can still be recovered from the remaining physical stripes. The downside of a RAID 5 is resource utilization; RAID 5 requires extra CPU cycles and extra disk space to transform and manage data using the ECC model. If one disk is lost, rebuilding the array takes significant time because all the remaining disks have to be read in order to rebuild the missing ECC and data. The net capacity of a RAID 5 array consisting of N disks is equal to the sum of the capacities of N–1 of these disks. This is because the capacity of one disk is needed to store the ECC information. Usually RAID 5 arrays consist of three or five disks. The minimum is three disks.

RAID 10 (also referred to as striped mirrors, RAID 1+0)

Many RAID systems offer a combination of RAID 1 (pure disk mirroring) and RAID 0 (striping) to provide both redundancy and I/O parallelism. This configuration (also referred to as RAID 1+0). Advantages are the same as for RAID 1 and RAID 0. A RAID 10 array can survive the loss of multiple disks. The only disadvantage is the requirement for twice as many hard disks as the pure RAID 0 solution. Generally, this configuration tends to be a top performer if you have the disk resources to pursue it. If one disk is lost in a RAID 10 array, only the mirror of this disk has to be read in order to recover from that situation.

STORAGE AREA NETWORK (SAN) – STORAGE ARRAY

A Storage Area Network (SAN) is a networked architecture that provides I/O connectivity between host computers and storage devices as shown in Figure 5.

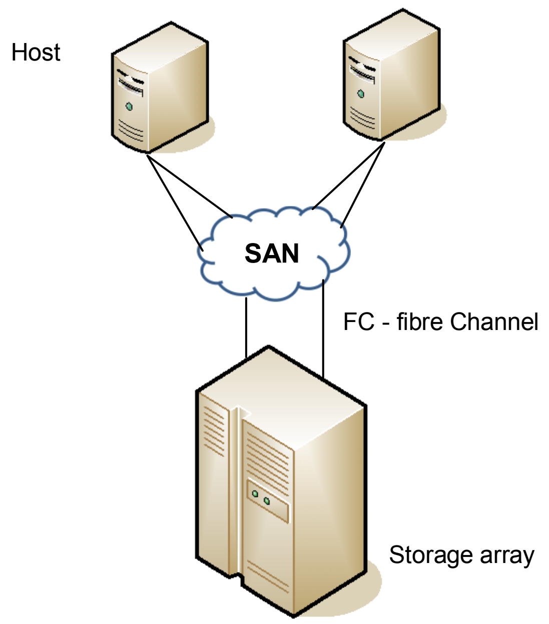


Figure 5. SAN Architecture

In a SAN, one or more storage devices are connected to servers through fiber channel switches to provide a central location for disk storage. Physical disks are split into logical partitions or Logical Units (LUNs). See Figure 6.

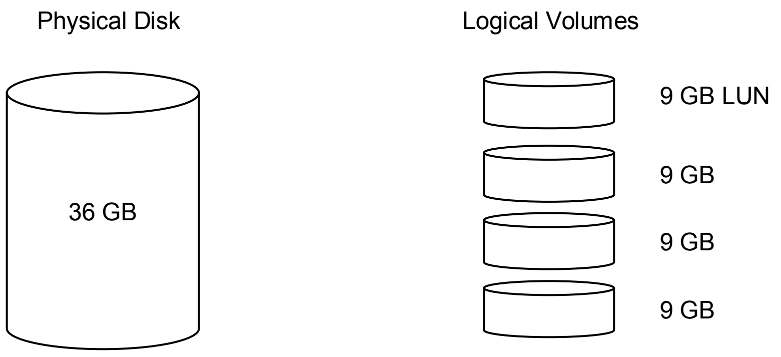


Figure 6. Physical disk partitioned into four 9 GB LUN

The stripe unit size is configurable on a per LUN basis. To an extent, the stripe unit size determines the performance of the LUN. Smaller stripe units are most efficient for random I/Os. Large stripe units are more efficient for sequential I/Os. Usually a stripe unit size of 64K is optimal for most application.

RAID GROUPs AND LUNs

One RAID group (e.g., RAID 5 or RAID 1) can span multiple LUNs. The advantage of multiple LUNs per RAID group is the ability to divide a large disk into logical partitions.

A number of logical partitions, one from separate physical disk, are combined together (concatenated or striped) to form a single logical volume. Logical volumes are then presented as disk devices to your host. Your system administrator can carve up the storage and allocate disks to the Volume Groups for your host. For example, a logical volume that is spread over four 9 GB chunks of physical disk provides 36 GB of usable storage. The host will just see a pool of 36 GB of "disks" as shown in Figure 7.

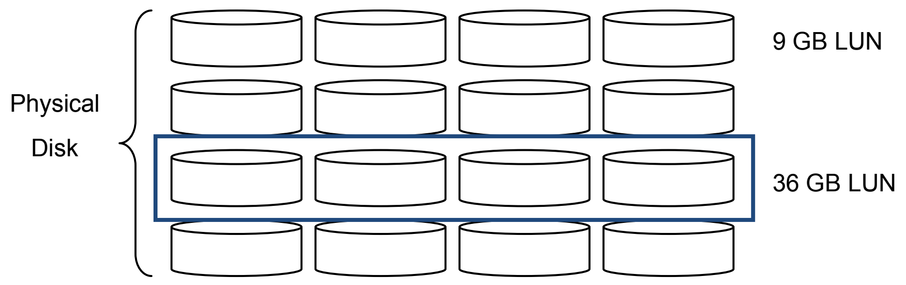


Figure 7. Logical volume spread over four 9 GB LUN

One disadvantage of multiple LUNs on a RAID group is that I/O to each LUN may affect I/O to the others in the group. If one LUN is constantly busy, I/O performance with other LUNs may be degraded.

The choice of the number of disks (LUNs) in a RAID group is a tradeoff between performance and the amount of time it takes to reconstruct a disk after a disk failure. For example in a RAID 5 volume, the number of spindles/LUNs is sometimes dictated by the hardware (a RAID 5 array has support for 3+1, 4+1, 7+1 disks, etc). If a volume (file system) with higher I/O bandwidth is required, then software striping via a Logical Volume Manager (LVM) or Volume Manager can be used to spread the I/O across more spindles in multiple RAID 5 LUNs.

Details on how to set up the disks in your storage arrays is beyond the scope of this paper. We will give you some guidelines on the I/O subsystem set-up recommendations and you can work with your storage administrator and/or storage vendor to determine how to accomplish this with the hardware and operating system you will be using with SAS SPD Server.

SAS SPD SERVER I/O SUBSYSTEM RECOMMENDATIONS

The following items are general guidelines for setting up the file systems required by SAS SPD Server. The number of file systems needed depends on the size of the data warehouse and the I/O throughput rate (measured in megabytes per second, MB/s) required to process the data (see diagram below). SAS SPD Server can be both bandwidth intensive for sequential activities (e.g., long reads and writes, sorts, etc.) and require a high number of I/Os per second (IOPs) for random activities (e.g., indexed reads, by sorts that utilize the index, etc.).

- Avoid the use of NFS mount points when accessing large data. Transferring large volumes of data over the network may degrade SAS SPD Server performance and cause network congestion. For sequential reads and writes, we recommend the following file systems – on Solaris 8 or 9 – VxFS, on Solaris 10 – ZFS; on AIX - JFS2, on HP-UX - JFS; on Windows NTFS.

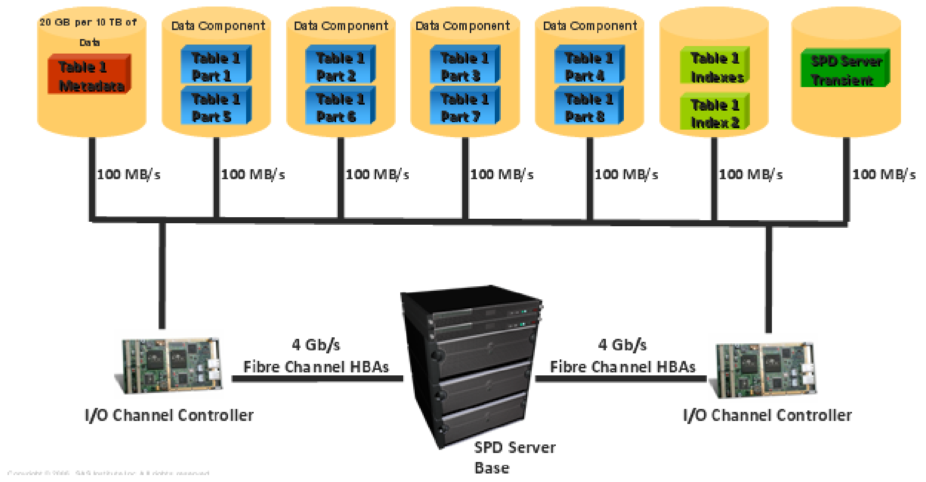
- To achieve I/O scalability, set up the I/O subsystem with as many physical disks/LUNs as possible. In fact, many small disks striped across are better than a few large disks. If you are constrained by the number of available disks/spindles, place the work area on separate volumes from the data area.

- When mounting the file systems, make sure that Read Ahead and Fast Writes (this term differs on various hardware platforms) are enabled. The data is read once but written later at a configurable interval. Acknowledging the write to the host at the time the data hits the cache could result in performance improvement and reduce the server load in multi-user environment.

- The set-up of SAS SPD Server storage area (data path, index path, and work path file systems) is crucial to the performance that will be achieved. It is recommended to have SPD Server file systems on their own independent set of physical disks. If possible, you should avoid sharing disks assigned to SAS SPD Server storage with other applications (whether these applications are performing heavy I/O or random access to the data) to avoid I/O contentions.

- Consider using direct I/O for the work area if CPU and memory resources are constrained. Mounting the WORKPATH file systems for direct I/O would free up the CPU and memory resources used for file caching. For example, temporary utility files once read, the data will not be read again or large tables that are read sequentially that are bigger than the cache (they will just clean out your cache for nothing). The cache is only useful for random reads that re-read sections of the table, or for components of tables that are continually being used regardless of where you are reading the rows from.

- When attaching a storage device via multiple fiber channels, you need to enable dynamic multi-pathing in your storage management software to balance I/O workload across all available paths to the storage array.



TUNING SPD SERVER PARAMETERS

The SAS SPD Server is shipped with a sample SPDSSERV.PARM configuration file. This paper will cover the important parameter options in this file that effect system resources (I/O, CPU, and Memory) and SQL Options. We will also provide general guidelines for setting them. This paper is based on SAS SPD Server Version 4.4 tsm7. Earlier versions of the software may not include some of the parameters listed in this paper.

SPDSSERV.PARM FILE

The SPDSSERV.PARM file is a parameter file used to control resources and tune the behavior of the SAS SPD Server. The SPDSSERV.PARM file is read by the server at start up and provides means of setting global and resource parameters for all users who make connections to the server. There will be a process created for each user of the SAS SPD Server. The resource parameters are per-process parameters. So the values given to the resource parameters need to keep in consideration the number of users that can be concurrently using the SAS SPD Server to avoid over consumption of system resources.

The install of the software should take under 30 minutes to complete. Once the install is finished, the SPDSSERV.PARM file will require further tuning and configuration for your hardware environment.

SAMPLE SPDSSERV.PARM file:

```
SORTSIZE=512M;
INDEX_SORTSIZE=512M;
GRPBYROWCACHE=512M;
BINBUFSIZE=32K;
INDEX_MAXMEMORY=30M;
WORKPATH= ('/spdwork1' '/spdwork2');
```

```
NOCOREFILE;
SEQIOBUFMIN=64K;
RANIOBUFMIN=4K;
MAXWHTHREADS=64;
MAXSEGRATIO=75;
WHERECOSTING;
RANDOMPLACEDPF;
MINPARTSIZE=256M;
TMPDOMAIN=TMP;
SQLOPTS="reset plljoin";
```

PARAMETERS REQUIRING CONFIGURATION

The server has seven parameters that require setting. The first three parameters control memory usage during sorting, creation of indexes, and SQL summarization. The fourth parameter in the list, WORKPATH, controls the location where utility files are written to. The fifth parameter specifies a domain that the server will use to optimize certain type of SQL queries (e.g., correlated queries). This domain will be used to store temporary tables created during the optimization process and then deleted when the processing is finished. The sixth parameter, MAXWHTHREADS, controls the number of threads on a per-process basis for algorithms that use parallel processing (e.g., sorts, WHERE clause filtering, summarizing data). The seventh parameter, MINPARTSIZE, globally sets a minimum data partition size for all table partitions that are created. The list of parameters is shown below:

SORTSIZE

INDEX_SORTSIZE

GRPBYROWCACHE

WORKPATH

TMPDOMAIN

MAXWHTHREADS

MINPARTSIZE

Memory Parameters

The first three options are parameters that control memory usage during process execution. The types of processes that can be memory intensive are sorts, index creation, and summarization of data. There are two purposes to setting the memory parameters: prevent the OS from paging and sharing of system resources so processes can accomplish the tasks required of them. The three parameters need to be set based on the total number of jobs that will be running concurrently and how much memory they are expected to consume.

Incorrect settings of the memory parameters can impact the server performance. Memory on SMP machine provides CPUs fast access to data it requires for processing. SMP machines often use hard disk space for overflow of memory, an area located on disk drive called swap space. If a process requests for more memory than what is available, the OS will select parts of memory and store it in the swap space. Performance problems can occur if the OS has to constantly read and write between swap space and memory. The process of reading and writing between memory and the overflow area is called paging. The constant writing and reading between the swap space and memory degrades the server performance.

Here is a basic example of a system that would page. A server has 8 gigabytes of memory and there are three jobs running concurrently: one job performing a sort, one job building indexes, and one job performing a Parallel Group By. Each job is using 4 gigabytes of memory, which in total is 12 gigabytes and exceeds the system memory. Because the memory being used, 12 gigabytes, is greater than the memory available, the OS must move memory pages to swap space. The information, held in memory, can be constantly written to and read from the swap space until one or more processes finished and frees enough memory, so the swap space is no longer required.

If the memory parameters had been set low enough, the paging would have been avoided. For example if each of the parameters were set to 2 gigabytes, then each job would have been limited to 2 gigabytes. The total memory used would have been 6 gigabytes. Setting the three memory parameters should be done with an understanding of the type of processing that the parameter influences.

GENERAL SORT

General sorts are performed on tables when ordering rows of the table is required for processing. The majority of general sorts will be invoked through the software in one of three ways. One way is by an SQL execution method that has chosen to perform the join of the tables using a sort merge method. Another is an implicit sort to order the rows of a table for BY-processing in a DATA step if the tables are not already sorted. A third would be an explicit sort, using PROC SORT, to order the rows of a table. The sorts used by the SQL are done implicitly by the software to order rows before match merging of rows can occur when executing the SQL join.

When sorting a table, the width of the row being sorted includes the columns that make up the sort key, an 8-byte row identifier (RID), and all columns that the user requires for the sort. This means the rows being sorted could be very wide or very narrow depending on the sort key and other columns involved in the sort. Often sorts are performed on a subset of rows from a table because of WHERE clause filtering and may include a select list or all columns of the table. Therefore sorts being performed are not necessarily predictable in terms of the number of rows or the width of the rows.

Sorts required for creating an index are predictable as they consist of the columns being indexed plus an 8-byte row identifier (RID). The creation of indexes require sorts to be performed on only the columns of the table that will be included in the index, however all rows of the table must be included in the sort. When multiple indexes are being created in parallel, there will be multiple sorts occurring, one for each index.

SORTSIZE

The SPD Server by default uses a threaded pointer sort to order rows in a table. A Pointer sort orders the rows of a table by creating an array consisting of the sort key and a pointer to row location in memory. The array gets sorted and then rows are flushed from memory using the pointer to access the memory location and writes these rows to temporary files on disk. This process continues iteratively until all the required rows have been through this process. Then the rows that have been sorted to disk additional threads will read the temporary files collating the rows into the final sorted order. The sort only moves the pointers around, so you are minimizing what you are moving (generally the pointer is shorter than the row). For this type of sort, each thread gets a fractional amount of the memory allocated by the SORTSIZE setting.

For example when SORTSIZE=2048M; and MAXWHTHREADS=8; each sort thread will receive 256 megabytes to sort with. The SPDSSERV.PARM option SORTSIZE=; controls the amount of memory used to order data. By default all rows are written to utility files on disk before the final ordering/merge of rows takes place. This can occur with tables being sorted by PROC SORT, implicit sort with a BY clause in a DATA step, or SQL execution that requires a sort. The sort algorithm used is a pointer sort.

The SPDSSERV.PARM option SORTMODE=MOVE forces the server not to spill to disk sorted rows in memory if we determine the sorted file can fit into a single sort bin (determined by the SORTSIZE= option). Then we will read all of the rows to be sorted into that bin. The sort algorithm used to sort the rows will be a MOVE sort, so the entire rows will be moved around.

SORTMODE=MOVE does not assure better performance and could also cause thrashing. Consider a sort bin size of 4 gigabytes, with SORTMODE=MOVE, any part of the 4 gigabytes can be involved in a row swap, resulting in writing to any of the 4 gigabytes of memory. If the OS decides it needs to do some paging, it could flush the written page to swap space. This page could then be needed in a subsequent row swap. Therefore, the SORTMODE=MOVE "could" still be more expensive than a "pointer sort/flush/merge-read" operation depending on the overhead of moving the full rows in the MOVE sort, versus moving only the pointers in pointer sort, then doing a flush of the bin in sorted order, and reading it back in.

INDEX_SORTSIZE

The INDEX_SORTSIZE parameter limits the amount of memory used to sort data when creating indexes. The process of creating indexes in parallel is a multi-threaded process that can be CPU, I/O, and memory intensive. For each index that is being created, a parallel sort on the index columns will be performed along with an additional 8 bytes, which is used as row identifier (RID). The memory specified by the INDEX_SORTSIZE will be divided by the index creation threads. For a setting of 2048M (2 gigabytes) and a process that is creating four indexes in parallel, 512 MB of memory will be used to sort the combination of columns and RID for each index.

When indexes are created in parallel, there will be multiple sorts but the input table is only read once. Serially creating indexes will cause the input table to be read for each index created.

GRPBYROWCACHE

GROUPBYROWCACHE is the maximum memory that can be held for the Group BY. Memory used by Parallel Group BY is allocated incrementally and the process will use up to this memory. If the data for the GROUP BY cannot fit in memory, the process will write to utility files. Sorts and sorts for index creation are designed by default to write to utility files on disk, which frees memory for other processes to utilize.

One of the factors that influences memory used by Parallel Group By is the duplication of Group By values within the table, which is often referred to as cardinality. Low cardinality column, with few distinct values, will have small memory foot print. In a large table a high cardinality column, with values near unique, will most likely utilize the full memory allocated.

For example, consider a billion row table that is being summarized on a column that has only 50 values. The process will utilize memory based on the number of values being summarized. For the 50 values the memory utilization will be very small. With a multi-billion row table of a very large telecommunication company that has over 100 million customers. If a query was to use Parallel Group By to summarize by customer, the memory allocated could be easily reached and the process will spill memory to utility files.

Setting Memory Parameters

Setting the three memory parameters requires knowledge of the peak usage of users that will be performing these types of processing on the server concurrently. Users may be performing sort, others may be performing Parallel Group By, and others could be loading tables and building indexes. Each requires memory space to perform the tasks required and the space allocated can influence performance.

To set these parameters, the following information is required: maximum number of concurrent sorts, index builds, and Parallel Group By processing that will occur at the same time and the total amount of memory on the server. When this analysis has been done, a determination of how much memory to reserve can be made. You will also need to reserve memory for the Operating System and other types of processing that will occur on the server that requires memory.

For example a server that is dedicated to SPD Server has 16 gigabytes of memory, and the OS requires approximately 1 gigabyte of memory. Then the remaining memory can be allocated for sorting, index creation, and Parallel Group By processing. The total memory on the system is then reduced by this amount and the remainder is divided by the number of concurrent processes that will be running. This number is the most basic setting.

For example during peak time there are eight jobs running concurrently, all performing sort, index build, or Parallel Group By. If the server has 16 gigabytes of memory, then each job can use close to 2 gigabytes of memory without causing the server to page. If each job was to use a full 2 gigabytes of memory, the server would be required to page because the OS system is also a consumer of memory.

WORKPATH

The parameter WORKPATH controls the location where temporary files are created. Temporary utility files can be generated during SPD Server sorts, index creation, or Parallel Group By. The SAS SPD Server supports the use of multiple work paths and many of the parallel processes will distribute utility files they create across the multiple work paths allowing the system to scale.

The space of the work area must be large enough to accommodate all the concurrent processes that are performing tasks that need temporary space. When there is more than one work path (file system) specified and a multi-threaded process is writing utility files, one of the paths will be chosen, at random, as the path for the first thread, then the subsequent path will be used for the second thread, and so on, until all threads have been assigned a path. For example say there are three paths and six threads. The first thread is assigned the second path, the second thread is assigned the third path, and the third thread is assigned the first path. The forth thread will be assigned the second path, and so on, until all threads have been assigned a path. This round robin assigning of paths is similar to the way the partitions of a table are distributed across the file systems for the DATAPATH of a domain.

TMPDOMAIN

The parameter is used to control the creation of temporary LIBNAME domain. This domain provides storage for temporary created tables during SQL Query Rewrite. The SQL Query Rewrite facility in SAS SPD Server examines SQL queries in order to optimize processing performance.

MAXWHTHREADS

MAXWHTHREADS is one of the most important parameters. It controls the number of threads a single process can create when performing a task in parallel. A multi-threaded task that performs sort, index creation, Parallel Group By, full table scan, and index scan will all create threads. On a typical system the thread count is set based on the number of CPUs and expected concurrent users of the system. We highly recommend you perform the scalability test (see I/O scalability tests below) to determine optimal MAXWHTHREADS value for your environment.

On most systems the benefits from additional threads occur when the value of MAXWHTHREADS matches the number of CPUs. It is possible that a well configured system will continue to yield performance benefits when the numbers of threads are set higher than the number of CPUs. However, gains are most often marginal. Perfect scalability is when *N* x threads achieve a speed factor of *N*. The overhead of managing threads and other factors such as I/O latency will prevent perfect scalability. Keep in mind that this is a per-process parameter. Therefore, multiple users will each get MAXWHTHREADS for sorts, WHERE clauses, and so on. Creating too many threads can often cause performance degradation due to "thread starvation." This is when threads cannot get CPU resources to do their work, or the work done by the thread is not sufficient to keep it busy on the processor to justify the creation and scheduling of the thread. You should avoid the temptation of setting a high MAXWHTHREADS to increase concurrency to improve performance, which can often do just the opposite.

SAS SPD Server uses parallel threading algorithm to ensure units of work will be divided among system resources such as I/O, memory, and CPU. See Figure 8.

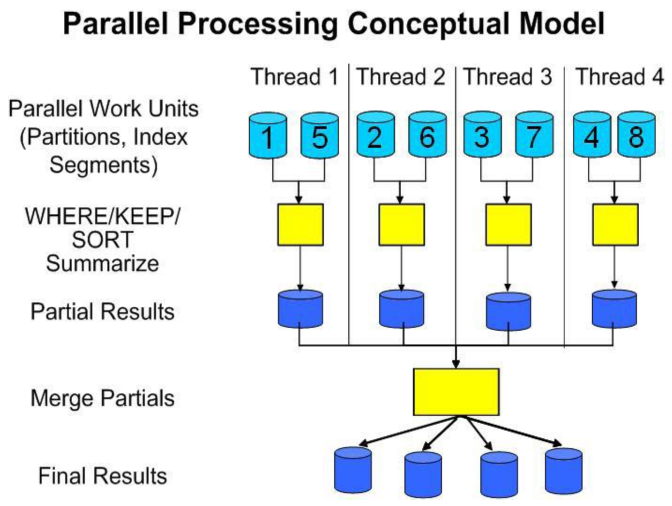

Parallel Processing Conceptual Model
Thread 1
Thread 2
Thread 3
Thread 4
Parallel Work Units (Partitions, Index Segments)
1
5
2
6
3
7
4
8
WHERE/KEEP/ SORT Summarize
Partial Results
Merge Partials
Final Results

Figure 8. Parallel Processing Conceptual Mode

I/O SCALABILITY TESTS

The purpose of the scalability testing is to be able to identify I/O bottle-necks, to determine what the optimal thread count (MAXWHTHREADS) is, and to determine what maximum practical throughput the I/O subsystem will deliver.

This is achieved by creating a physical table, in a domain specified in LIBNAMES.PARM file, of a size greater than the physical memory of the machine, thus avoiding query caching by the SAS SPD Server.

To perform the I/O scalability test, first, create the test data:

```
/* the macro below sets the size of the SPD Server table in gigabytes */


%let gigfile=n;

data testdom.scale;
    length a $248;

   do i = 1 to (4194304 * &gigfile);
      output;
   end;
run;
```

Before you run the %IOSCALE macro, ensure that no other programs are running on your machine. Run the %IOSCALE macro few times, noting the time for each run.

```
%macro ioscale(maxth);

   %let spdstcnt=1;
   %do %while(&spdstcnt le &maxth);
      data _null_;
         set testdom.scale;
         where i=0;
      run;
      %let spdstcnt=%eval(&spdstcnt*2);
   %end;
%mend;

%ioscale(64);
```

Note:

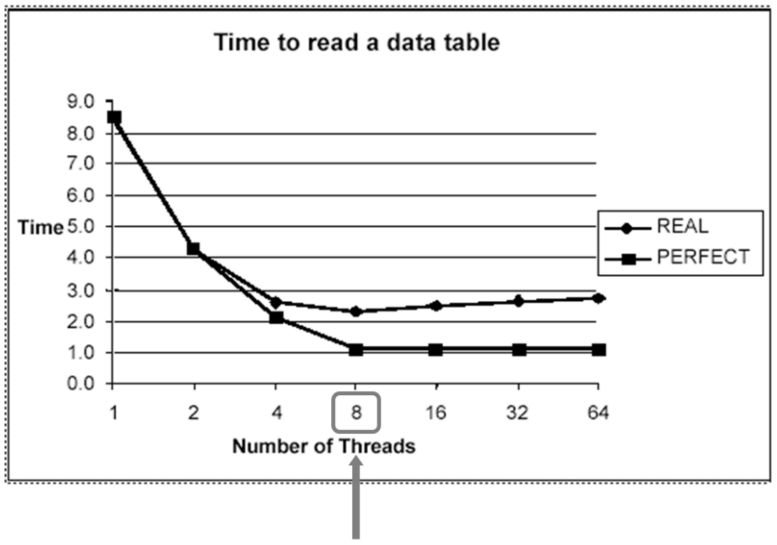
If the SPDSTCNT (thread count) value exceeds your MAXWHTHREADS setting, the value of MAXWHTHREADS will override the SPDSTCNT value. You may want to temporarily set MAXWHTHREADS value to 64 threads for the purposes of running this test.

Interpreting the Results

First, average the results from the few %IOSCALE macro runs. If your I/O subsystem is well configured you should expect to see these results:

- The real time for each successive DATA step should ideally follow the curve 1/SPDSTCNT. That is, It should take half as much time to process with each successive doubling of the thread count
- The real time for each successive DATA step should decline and then after some point the curve should begin to flatten

If you are not getting the results above, there might be an I/O bottleneck that you need to investigate with your system or storage administrator. Once you get a curve with the characteristics listed above, set the value of MAXWHTHREADS= in the SPDSSERV.PARM file to the value of SPDSTCNT= where the curve flattens. (See the graph below.)



The scalability test not only helps you determine MAXWHTHREADS value but also the overall throughput in megabytes per second the step took. To calculate this number, take the size of the test data table in megabytes and divide it by the real time the step took in seconds. This number should come as close as 70 to 80 percent to the theoretical maximum of your I/O bandwidth, if the set-up is correct.

MINPARTSIZE

MINPARTSIZE parameter sets a default minimum partition size for any table that is stored by the SAS SPD Server, which reduces the need having to calculate and set a partition size when you create a table in the SAS SPD Server domain. However, it is a good practice to determine and set specific partition size to override the default whenever possible. For example, a production job that loads a 240-gigabyte table might be best suited for a partition size of 4 gigabytes. If the MINPARTSIZE= parameter is set to 256 megabytes, the table will have 960 data partitions. If the partition size for this table is changed to 4 gigabytes, the table will contain only 60 partitions. Setting a higher minimum partition size reduces the distribution of smaller tables.

For example, if you have one CPU and one file system that processes 50 megabytes of data per second, processing a 1000 megabytes table in a single partition would require 20 seconds. If you have four CPUs and four independent file systems, with a partition size of 256 megabytes, processing the same 1000 megabytes table across four file systems would reduce the time to 5 seconds. This example demonstrates that the threading model will achieve good results if your data is spread out on independent file systems and your system has sufficient processors to allow the threads to concurrently process the data.

PARAMETERS THAT MAY INFLUENCE PERFORMANCE

The parameters discussed in this section may have impact on the overall performance of SAS SPD Server. They are set by default in SPDSSERV.PARM file based on some experience in the field. However, understanding your hardware environment, your application I/O profile, and how these parameters interact with your system resources could help you maximize performance.

SEQIOBUFMIN

RANIOBUFMIN

WHERECOSTING

SQLOPTS

SEQIOBUFMIN

SEQIOBUFMIN sets the minimum I/O buffer size that is used by SAS SPD Server when it performs sequential I/O. Sequential I/O does not imply that a table is being read single-threaded. It implies that each thread that is working on a table is given a “work unit’s” worth of work that is independent and contiguous. Sequential I/O is utilized by the following types of SPD Server queries:

- full single-threaded table scans, utilized by queries that do not contain any WHERE clause
- multi-threaded parallel indexed WHERE clause evaluation
- multi-threaded parallel non-indexed WHERE clause evaluation

For Sequential I/O, SPD Server threads will be given either a work unit of a data partition file for full table scans, or sections of a data partition for a parallel indexed WHERE clause evaluation. The amount of the work unit that can be read in and processed by a single read is limited by SEQIOBUFMIN.

For sequential read/write of non-compressed tables, the read and write buffer size is computed based off of the NETPACKSIZE. The default NETPACKSIZE is 32K. It can be increased using either SAS SPD Server macro variable SPDSNETP or table option NETPACKSIZE. The actual read buffer size will be based on the block factor, which is computed using NETPACKSIZE divided by row length so that SAS SPD Server always read in complete rows. SAS SPD Server will always try to fill the full read buffer size on the read request, so if it encounters a partition break while doing the read, it will do a second read on the next partition to fill the buffer.

For writes, SAS SPD Server can only write what is passed to it from the client, but it is similar to the read. If it calculates that the write buffer will cross a partition boundary, then it will split the write (based on the most full rows it can write without going over the partition break) to fill the first partition and do a second write to the next partition with the remaining buffer. SAS SPD Server will take the MAX (SEQIOBUFMIN, NETPACKSIZE) when computing the block factor for sequential reads. The default for SEQIOBUFMIN is 64K, which is set in the SPDSSERV.PARM.

Hence the read size will be based off of the larger of these two parameters. You can sync up SEQIOBUFMIN with your disk subsystem stripe size or logical volume manager (LVM) stripe size. This can result in fewer physical I/O requests, which improves performance. Keep in mind that larger buffer means more memory. We recommend you use the scalability test (after you have set MAXWHTHREADS value), increment SEQIOBUFMIN in a multiple of 64K, run the %IOSCALE macro, and see if that yields significant performance improvement.

RANIOBUFMIN

This parameter controls the buffer size for random reads. The larger the buffer, the more SAS SPD Server reads for random reads, which is good if you are getting a lot of rows from a single random read. A smaller buffer is good if your data is widely dispersed and you only need a smaller buffer to read in random rows.

Random I/O is used by the following types of queries that utilize an index to globally read the RIDs (row identifiers) of a table that satisfy a query:

- Index points utilize the table index to retrieve the RIDs (Row Identifiers) for a particular value. An example of an index point would be an index join, where the Outer table is “pointing” to the Inner table to retrieve the rows of the Inner table to join with the current row of the outer table.
- An Indexed By clause without a WHERE clause will utilize the index to return the table by value by RID. As each value of the index is queried, the RIDs for that value are read and returned.
- An EVAL3/EVAL4 WHERE clause evaluation will evaluate the true RIDS of the WHERE clause, and then read the RIDs to retrieve the rows.

For random I/O, SAS SPD Server will group the selected RIDs together that can be read into a single buffer whose maximum size is controlled by the server parameter RANIOBUFMIN. The server will optimize the read by only reading from the first RID to the last RID in the “group.” Therefore the largest read via random I/O would be RANIOBUFMIN bytes, and the smallest could be a single row. Generally RANIOBUFMIN is configured in the SPD Server Parameter file to be fairly small (default is 4K). This is to avoid reading in larger sections of the table where a large percentage of the rows read would not satisfy the query.

For longer records you could increase the default value, since longer records will decrease the number of records that can fit in the RANIOBUFMIN buffer. Setting RANIOBUFMIN depends on how random the reads actually are. If, for example, you are reading in every other record, make RANIOBUFMIN large. If, however, you are sparsely reading records, then RANIOBUFMIN should be small.

WHERECOSTING

SAS SPD Server will either use either a parallel or sequential WHERE clause strategy to evaluate a WHERE clause. The parallel WHERE clause strategy will use multiple threads to evaluate concurrently different sections of the table. The sequential WHERE clause strategy will use a single thread to evaluate the entire WHERE clause.

SAS SPD Server, by default, will use WHERECOSTING to determine whether to use the parallel WHERE clause strategy or sequential WHERE clause strategy. WHERECOSTING is set by default in SPDSSERV.PARM parameter file. There are two key factors WHERECOSTING uses to evaluate WHERE clause indexes:

- duplicity
- distribution

Duplicity refers to the proportion expressed by the number of rows in a table divided by the number of distinct values in the index. Distribution refers to the sequential proximity between observations for values of a variable that are repeated throughout the variable's data set distribution.

The SAS SPD Server uses duplicity and distribution in several evaluation strategies when determining how to process the WHERE clause. Generally, WHERECOSTING will provide good results for a variety of WHERE clauses, and it is the recommended approach to determine the WHERE clause strategies. However, WHERECOSTING is not free. There is a cost involved to cost the WHERE clause. A sophisticated user that can determine the best strategy for their WHERE clauses can turn off WHERECOSTING and directly influence the WHERE clause strategy using the following options:

- SPDSWCST=NO macro variable setting to turn off WHERE clause costing. The default WHERECOSTING strategy will be parallel WHERE clause.
- SPDSWSEQ=YES macro variable with SPDSWCST=NO will force the sequential WHERE clause strategy.

Please refer to the SAS SPD Server User's Guide section on WHERE Planning for a detailed description of the parallel and sequential WHERE clause strategies.

SQLOPTS

The parameter SQLOPTS overrides default options for SAS SPD Server SQL. For example the syntax SQLOPTS=”reset plljoin” in the SPDSSERV.PARM will globally set the parallel join facility. The parallel join facility executes SQL joins in parallel, thus reducing the total elapsed time required to complete the query. Other SQL reset commands can be added with the “reset plljoin” option to control parallel join behavior.

CONTROLLING PARALLEL JOIN FACILITY

The Parallel Join Facility has a set of default behaviors that are used to control the execution methods, which control the degree of parallelism employed. A full set of options are available to turn the Parallel Join Facility on or off, to turn the Parallel GROUP BY feature on or off, and to control the types of parallel join (for complete discussion please see white paper “Parallel Join with Enhanced GROUP BY Processing”).

By default, the Parallel Join Facility sets concurrency to a value equal to half the number of processors on the machine. To set the concurrency to another value, use the SAS SPD Server SQL reset option CONCURRENCY=. This option controls the number of threads that the parallel join facility can utilize. SAS recommends not increasing the concurrency setting higher than the Parallel Join Facility default settings. Increasing concurrency can degrade performance because competing parallel join threads compete for resources.

For example setting SQLOPTS=”reset plljoin concurrency=4” in SPDSSERV.PARM will use four threads to execute SQL joins in parallel.

The following Parallel Join execution methods illustrate the effect of concurrency.

PARALLEL INDEX JOIN

Concurrency sets the number of parallel join threads. A parallel join thread work unit is a range of like values determined by the join index. The number of work units is determined by the “join parts” parameter when the parallel join index is created. Each parallel join thread will join a “join part” from the two tables concurrently. The join method used for each parallel join thread for its “join part” is a merge join. The rows from each table of the pair wise join are selected based on the value range of the join key, sorted by the join key, and then joined via a merge join on the join key.

This will result in (concurrency * 2) concurrent sorts being done. To assure that sort memory is not over consumed, the SORTSIZE memory parameter is divided by the number of concurrent sorts, and each sort is given their resulting fair share of the SORTSIZE memory. Therefore, each sort is given (SORTSIZE / (concurrency * 2)) memory.

PARALLEL MERGE JOIN

Parallel merge join does a full sort on the smaller of the two pair wise join tables. The larger table is divided up evenly by “concurrency” parallel join threads. Each parallel join thread will join their portion of the larger table with the smaller table. Therefore, parallel merge uses “concurrency + 1” threads. One thread will sort the smaller table and one thread for each part of the larger table. To assure that sort memory is not over consumed, the SORTSIZE memory parameter is divided up evenly for each concurrent sort. Therefore each sort is given (SORTSIZE / (concurrency+1)) memory.

For either Parallel index join or parallel merge join, each concurrent sort can use up to MAXWHTHREAD threads to sort the table. Therefore parallel index join can use up to (concurrency * 2 * MAXWHTHREAD) sort threads in addition to its parallel join threads. Parallel merge join can use up to ((concurrency+1) * MAXWHTHREAD) sort threads in addition to its parallel join threads.

This can result in an explosion of threads if concurrency is set too high, or MAXWHTHREAD is set too high. Therefore, the “recommended” maximum concurrency is no more than 4. Performance tends to level out after this, and can actually degrade if there are too many threads. The default concurrency is number of CPUs divided by 2. MAXWHTHREADS should be set such that the calculations above do not exceed the number of processors in the system. For example, if the system has 16 processors and concurrency is set to 4, then MAXWHTHREADS should be set to 4.

NOTE that currently the parallel join planner is rules based. We are looking at heuristics that will dynamically set the “concurrency” and number of sort threads to maximize parallel joins performance.

CONCLUSION

The recommendations covered in this paper are a good starting point for a successful SAS SPD Server implementation. The best possible performance is achieved if all the resources (CPU, Memory, and I/O) within the system are being equally utilized. Planning starts with correctly sizing the system and understanding the workload and characteristics of your SAS SPD Server application. The understanding can assist in the set-up of your I/O subsystem and help you tune your SAS SPD Server parameters to achieve optimal performance. The Management of the Data is also a key part of achieving performance for SAS SPD Server. More information about this subject can be found in the SAS Forum paper “Managing large Data with SAS SPD Server.”

REFERENCES

Sargent, Daniel. 2008. “Managing large Data with SAS SPD Server®.” Proceedings from the SAS Global Forum 2008 Conference. Cary, NC: SAS Institute Inc. Available at http://support.sas.com/resources/papers/sgf2008/spds.pdf.

SAS Institute Inc. 2006. SAS Institute white paper. “Partitioning of SAS® Scalable Performance Data Server® Tables.” http://support.sas.com/resources/papers/partitiontables.pdf.

ACKNOWLEDGMENTS

Many thanks to my colleagues, Daniel Sargent and Guy Simpson, from SAS R&D for their contributions of expertise and putting up with my endless questions.

RECOMMENDED READING

Best Practices for Configuring your IO Subsystem for SAS® Applications

support.sas.com/rnd/papers/sgf07/sgf2007-iosubsystem.pdf

SAS SPD Server Technical Papers

http://support.sas.com/resources/papers/tnote/tnote_spds.html

SAS SPD Server documentation

http://support.sas.com/documentation/onlinedoc/91pdf/index_913.html#scalable

Hardware Tuning Guides can be found on the external SAS Web site. Please see our Hardware Vendor Partner white papers at:

http://www.sas.com/partners/directory/hp/papers.html

http://www.sas.com/partners/directory/ibm/papers.html

http://www.sas.com/partners/directory/intel/papers.html

http://www.sas.com/partners/directory/sun/papers.html

http://www.sas.com/partners/directory/unisys/papers.html

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author:

Abdel Etri
SAS Institute Inc.
Cary, NC 27513
E-mail: abdel.etri@eur.sas.com
Web: www.sas.com