



Partitioning of SAS® Scalable Performance Data Server® Tables



Table of Contents

Abstract	1
Introduction	1
Partitioning SAS SPD Server Tables	1
History of Partitioned Tables	2
Example Table and Server.....	2
Threaded Model	2
Kernel Settings.....	3
How SAS SPD Server Tables Are Partitioned	3
Data Paths	4
Round-Robin Partitioning	5
Starting Point for Round-Robin Partitioning	5
Controlling and Setting Partition Size	6
Controlling Partition Size with the MINPARTSIZE= Server Parameter	7
Setting a Specific Partition Size for a Table.....	8
Basic Approaches for Controlling Partition Size	10
Creating Multiple Partitions per Data Path.....	12
Estimating Rows Counts	13
Example 1: Using an Estimate That Is Higher than the Actual Number of Rows ..	13
Example 2: Using an Estimate That Is Lower than the Actual Number of Rows ...	14
Partitioning a Dynamic Cluster Table	15
Partitioning Dynamic Cluster Tables Further with Table Column Values	16
Metadata and Index Files in Dynamic Cluster Tables	17
Conclusion	18

Abstract

This paper is an overview on setting and controlling the data partition size of SAS Scalable Performance Data Server (SAS SPD Server) tables. Setting and controlling the partition size are the most basic functions for managing data tables that are stored with the SAS SPD Server. The paper discusses partitioning standard SAS SPD Server tables and then uses the discussion as a basis to develop an understanding of how to partition dynamic cluster tables.

Introduction

SAS SPD Server is designed to meet the storage and performance needs that are required for processing large amounts of SAS data. As the amount of data grows, the demands for processing that data quickly increase. Therefore, the storage architecture must change to keep pace with business needs.

Hundreds of sites worldwide use SAS SPD Server. Some of these sites host data warehouses in excess of 30 terabytes. SAS SPD Server provides high-performance data storage for customers in industries such as banking, credit card, insurance, telecommunications, health care, pharmaceuticals, and transportation as well as for brokerages and government agencies.

This paper assumes that the reader has a basic understanding of SAS SPD Server tables and the concepts of data partitioning, symmetric multiprocessing, and disk configuration for I/O scalability. SAS SPD Server performs best with multiple CPUs and parallel I/O, and the hardware must be set up correctly to obtain the maximum benefit.

Partitioning SAS SPD Server Tables

When you create a table for the first time, one of the most basic tuning parameters is the partition size. Partition size can influence performance of the server as well as impact the management of the data. The discussions and examples that follow present

- the history of partitioning
- general concepts for setting the partition size
- some of the ways partition size can affect performance.

The paper also discusses partitioning *dynamic clusters* (structures that consist of multiple tables).

Partitioned tables are most important in that they provide the ability to scale the reading of the data table to levels greater than what can be accomplished with a single file system. This environment supports algorithms in the software that take advantage of the partitioning to process the table faster than a table that is stored on a single file system. You will also see scalability benefits in multiple-user environments. Prior to the advent of partitioned tables, multiple users often experienced I/O constraints when they tried to process the same table concurrently on a single file system.

History of Partitioned Tables

SAS SPD Server was developed in 1996 when many operating systems were limited to 32-bit address spaces, which imposed a 2-gigabyte limit on the size of files. Customers that required larger tables used strategies that divided the table into a set of smaller tables and then wrote SAS code to process them. The rise of Symmetric Multiprocessor (SMP) architecture in the marketplace and the need for faster data processing compelled SAS to develop a method of storing tables beyond the 2-gigabyte limit and for processing those tables in parallel. The method developed by SAS enabled customers to avoid the size limitation by creating a table structure that consisted of multiple 2-gigabyte files. This structure appeared to customers in a logical view as a single table. This method also helped reduce run time by incorporating parallel processing on the table.

SAS also had to address another critical factor that, imposed by the 32-bit operating systems, only allowed a process to open 1024 files at a time. In the SAS SPD Server processes, this limitation included all input files, output files, utility files, and other system files that are required by the software to operate. This limitation has been removed. Even though these limitations no longer exist, it is important for you to understand how to set and control partition size for SAS SPD Server tables because of the performance and data management implications.

Example Table and Server

The examples in this paper are based on a hypothetical table that contains 120 million rows of data.

Although table width is not a concern for the purpose of the discussions in this paper, the examples use a table that has 120 million rows to help you understand how the SAS SPD Server partitions a table and how to develop good data-partitioning practices.

The example server has 12 processors, each of which can each process 10 million rows every 5 minutes. This hypothetical server, which uses multiple I/O controllers, has an attached disk array with 12 independent file systems. The I/O throughput for each file system and the distribution of file systems across the I/O controllers enables the 12 CPUs to read data fast enough for each one to process 10 million rows of data every 5 minutes.

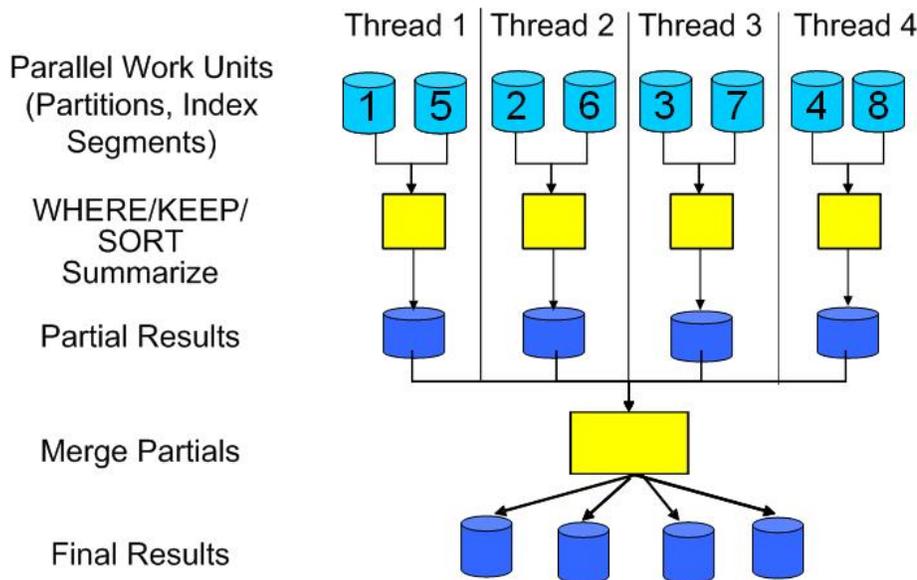
For this example, SAS SPD Server is set up with 12 data paths; one data path for each of 12 file systems in the disk array. When the 120 million rows of this table are divided evenly across the data paths, the entire table will be read in 5 minutes because each file system contains 10 million rows that feed concurrently into the 12 processors. The example discussions do not explain other factors (such as interaction with indexes, metadata, or work areas) that can impact performance. The discussions focus only on the partitioning of data in the SAS SPD Server tables. The combination of table and server is used to create the examples in this paper in order to help you understand the relationship between SAS SPD Server data partitioning and performance.

Threaded Model

The examples in this paper use a *threaded model* in which the maximum thread count is set to 12, and each thread processes only one partition at a time. In examples that have fewer data partitions than there are data threads, unused threads remain idle. In examples that have more data partitions than data threads, only 12 of the data partitions can be processed concurrently by the threads. As mentioned previously, the examples given here will process 10 million rows every

5 minutes (a partition with 9 million rows will process in 4 minutes and 30 seconds; a partition with 8 million rows will process in 4 minutes; and so on). Using these measures as a standard helps to illustrate the effects of partitioning on performance. The following diagram illustrates the threaded model that is used in the examples.

Parallel Processing Conceptual Model



This paper does not consider sections of the threading model that require sequential processing after multiple threads initially read and process the table. A threaded sort is an example of this type of processing. While the threads might be able to create multiple, partial sort results after quickly reading and sorting individual partitions, the process must include a final step in which the partial sort results are coalesced into a final result. Often, this final step is single threaded, which can greatly impact the total run time for the sort.

Kernel Settings

Kernel settings in the operating system (which are just as important as table partitioning) control the number of files a process can open. These settings enable a single process to open thousands of files, which minimizes the risk of reaching the limit of your operating environment. You should check with your UNIX administrator to make sure these settings are configured appropriately for your environment. Later in this paper, you will see examples that show different sizes of tables and the number of files these tables can contain. These examples will help you understand how to set the kernel.

How SAS SPD Server Tables Are Partitioned

SAS SPD server tables are made up of three components: metadata, indexes, and data. These components can be distributed across multiple file systems for the purpose of improving the I/O throughput when the server reads a table. When a SAS SPD Server table is opened, all of the

component files are opened to ensure that the table is complete. The earlier section “History of Partitioned Tables” discusses the operating system’s limitation related to the number of files that a process can open in a table.

While this paper discusses all of the table components, it emphasizes

- the data component , which contains the table rows
- how to set the data component partition size
- how SAS SPD Server organizes, distributes, and processes the data components.

The data component is the only component in a SAS SPD Server table that users can control.

To understand how data is distributed, you need to understand three concepts:

- data paths
- round-robin partitioning of data files
- the starting point for the round-robin partitioning.

The next three sections discuss these concepts and provide basic examples of how you can use them to provide a scalable environment for data tables.

Data Paths

A *data path* is a location in a file system to which SAS SPD Server can write the data-component files of a table. The list of data paths is defined in the LIBNAMES.PARM file, which is the central point of control for defining the locations for writing different component files.

Note: Instructions about how to set up the LIBNAMES.PARM file are found in the SAS SPD Server Administrator document that is shipped with the media.

SAS SPD Server supports the use of multiple data paths that can be located on multiple file systems. This configuration can achieve three important goals.

- It provides a manageable environment in which to apply the system’s back-up and recovery tools in parallel. Critical operations in a data warehouse or a data mart require a divide-and-conquer approach for backing up and recovering data within a specified time frame. If all data is contained on a single file system, utilities that rely on scalability might contend with each other for the same resource, thereby creating I/O constraints. The ability to perform timely back ups provides a more secure environment in which to maintain the data.
- Multiple independent file systems provide an architecture that enables the maximum throughput for reading a table in parallel with threading. Such systems can store a single data table across a scalable I/O subsystem. Using a distributed model for the data component, the SAS SPD Server distributes the table across multiple threads, which enables simultaneous processing on the different parts of the table.

- The data-path architecture provides the ability to add new disk resources to the system without having to reconfigure existing file systems. For example, if the data that is stored in a system grows to the point that additional disk resources are required, you can add these resources. You can expand a domain that has 18 data paths to 24 by modifying the LIBNAMES.PARM file. You will have to do some work to redistribute data that is already stored in the current data paths, but the process is not difficult.

Round-Robin Partitioning

Another important concept to understand is round-robin partitioning. *Round-robin partitioning* is a continuously repeating sequence in which partitions are used in a fixed, cyclical order when the table is loaded. One partition receives a portion of the data; the next partition in the sequence receives the next portion, and so on. When all partitions have been used, the sequence starts again at the beginning. For example, suppose you have 120 million rows of data that are loaded into a SAS SPD Server table. The table is stored in a domain that has 12 data paths, with each data path on an independent file system. In this example, each partition has 10 million rows in which to evenly distribute the data across all data paths. (Details about how to control the partition size are explained later in “Controlling and Setting Partition Size”.) The data-path names for this example are DATA1 – DATA12. The data paths are registered sequentially in the server. When a table is loaded, the server does not know how many partitions the table will require. No method exists for passing information to the server about the width of the table, the number of rows, and the number of data paths so that the server can set an optimum partition size. The partition size must be determined and set in one of the following ways (which are discussed in more detail later in this paper):

- You can set the partition size using a macro variable or a table option.
- If you do not set the partition size manually, the software will use the server’s MINPARTSIZE setting. The system administrator sets the value for the MINPARTSIZE parameter (discussed later in “Controlling Partition Size with the MINPARTSIZE= Server Parameter”) for the server.
- If the partition size is not set by either of the first two methods, the system will use the SAS SPD Server default minimum setting (16 megabytes).

Starting Point for Round-Robin Partitioning

On the first load of a table, the server randomly chooses a data path and begins a cycle of writing rows into the different data paths. Using a random starting point for the round-robin partitioning is the normal behavior when SAS SPD Server creates a table. The server is shipped with the option RANDOMPLACEDPF, which is set in the SPDSSERV.PARM file that turns on the feature.

Let’s suppose that the server starts with the DATA7 path on the initial load and writes rows of data into that partition first. The server only loads one partition at a time. When a partition reaches the predetermined number of rows for the partition size, the server loads subsequent rows into the next partition in the sequence. This sequential process continues until the server has written a full partition into the last data path (in this example, DATA12) of the sequence. At this point, the process cycles to the DATA1 path, where the next partition is written. The process continues this way until all of the rows have been loaded.

By choosing a random starting point, the server will better distribute the data across the domain data paths, especially in cases where you have tables of differing sizes. For example, suppose that you have 12 tables in which every data partition is 512 megabytes. Four of the tables are *small tables* because they will not distribute across all data paths with a partition size of 512 megabytes; each small table only contains three data partitions. The next four tables are *medium tables* because each table will have a data partition in each data path; each medium table contains 16 partitions. The last four tables are *large tables* that have at least four partitions for every data path; in this example, each large table contains 49 partitions.

If all of these tables start in the same data path, the distribution process will place more data in the first data paths. Without random placement of the starting points, tables that do not partition evenly will be skewed into the first few data paths. Overloading just a few paths in the file system will create performance problems. The data will be more evenly distributed when the server is allowed to select random starting points. For example, the server might use

- DATA2, DATA7, DATA9, and DATA12 as the starting points for the small tables
- DATA1, DATA5, DATA6, and DATA10 for the medium tables
- DATA4, DATA8, DATA10, and DATA11 for the large tables

A more even distribution will help alleviate the I/O constraints on the file system and improve performance.

Table 1 illustrates the distribution of partitions for all of the table sizes. In addition, the table shows the total number of partitions across all data paths, with and without random starting points.

Data Paths	Small Table with Random Starting Points	Small Table without Random Starting Points	Medium Table with Random Starting Points	Medium Table without Random Starting Points	Large Table with Random Starting Points	Large Table without Random Starting Points	Total Partitions with Random Starting Points	Total Partitions without Random Starting Points
DATA1	1	4	6	8	16	20	23	32
DATA2	2	4	5	8	16	16	23	28
DATA3	1	4	5	8	16	16	22	28
DATA4	1		5	8	17	16	23	24
DATA5			5	4	16	16	21	20
DATA6			6	4	16	16	22	20
DATA7	1		6	4	16	16	23	20
DATA8	1		6	4	17	16	24	20
DATA9	2		5	4	16	16	23	20
DATA10	1		5	4	17	16	23	20
DATA11	1		5	4	17	16	23	20
DATA12	1		5	4	16	16	22	20
Total	12	12	64	64	196	196	272	272

Table 1. Data Distribution with and without Random Starting Points

Controlling and Setting Partition Size

Even with file-size limits and a higher number of files that the process can open, it is important to set a good partition size for SAS SPD Server tables. One of the main reasons to control the partition size is to avoid creating a table that has too many partitions, which can cause problems.

By controlling the partition size, you reduce the

- risk of reaching any possible operating-system limit for the system file descriptors
- overhead that occurs with opening and closing files
- frequency of thread switching as threads finish processing one partition and begin processing another partition.

The following sections present basic guidelines for setting partition size. These sections provide

- information about how to use the MINPARTSIZE= server parameter to set partition size
- examples that illustrate different practices for setting the partition sizes for tables
- details on what nuances to avoid.

Controlling Partition Size with the MINPARTSIZE= Server Parameter

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 in the SPDSSERV.PARM file. This parameter sets a minimum partition size before a new partition is created. You cannot override this minimum size without making a change in the SPDSSERV.PARM file. This 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.

The MINPARTSIZE= parameter sets a default minimum partition size for any table that is stored by the SAS SPD Server, which reduces the need 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 sizes 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 5 gigabytes. If the MINPARTSIZE= parameter is set to 256 megabytes, the table will have approximately 960 data partitions. If the partition size for this table is changed to 5 gigabytes, the table will contain only 48 partitions. The MINPARTSIZE= parameter also helps control the number of partitions for tables that are created via GUI interfaces. (Interfaces that submit generated structured query language (SQL) might not have the ability to set specific partition sizes for a table.)

Setting a higher minimum partition size reduces the distribution of smaller tables. To model the effects of limiting the distribution of smaller tables, this paper uses an example with 12 data paths. The domain stores data on 12 independent file systems, each of which can read 100 megabytes per second. Using a 3-gigabyte table, the example uses different MINPARTSIZE= settings to illustrate the types of effects each size has on reading the table. In Table 2, you can see that a higher MINPARTSIZE= setting affects the amount of time that it takes to read a 3-gigabyte table.

However, notice that the amount of time that is required to read a table decreases when the MINPARTSIZE= setting is small enough to distribute the data evenly across all data paths in the domain.

MINPARTSIZE= Setting	Total Partitions	Time (in Seconds) Needed to Read a Partition	Time (in Seconds) to Read a Table in Parallel
1024 megabytes	3	10	10
512 megabytes	6	5	5
256 megabytes	12	2.5	2.5
128 megabytes	24	1.25	2.5

Table 2. Partition Size Affects the Time It Takes to Read a Table

Instead of using the minimum partition size for very large tables (those that have more than 500 partitions), you should set a more specific partition size to reduce the number of files that are used by the table. The section “Estimating Row Counts” discusses the nuances of not having an even distribution of partitions across all data paths. You can use any of several simple approaches to set an appropriate partition size.

Setting a Specific Partition Size for a Table

The most important way to control partitioning in a domain is to set the table partition size when the production tables are loaded. The size and record layout of production tables are the most predictable table layouts in a data-warehouse environment. Production tables are often loaded by using batch jobs that are written to optimize the load process. These jobs can incorporate specific partition sizes that optimize the number of partitions that will be created for each table that is loaded.

You can override the server MINPARTSIZE= value for a table in a SAS job in either of two ways. The first method is to use a macro that will globally set the partition size for all tables that are created that follow the macro setting. The syntax for the macro is

```
%LET SPDSSIZE =n;
```

In this syntax, n specifies the partition size (in megabytes) that you want to use. The size of the partition that is set by the macro, with one exception, will be used as the partition size for all tables that are created in that job after the macro is set. The following program illustrates tables that are created with a partition size that is set by a %LET macro:

```
%let spdssize=512;

/* All tables will have a partition size of 512M. */

data spdslib.table1;
  . . .more SAS statements. . .
run;
```

```
data spdslib.table1;
  . . .more SAS statements. . .
run;
```

```
data spdslib.table1;
  . . .more SAS statements. . .
run;
```

The benefit of using the macro is that it applies globally to the job that follows after you set the macro. However, the macro setting might not be ideal for all of the tables that are created.

The second method that you can use is to set a partition size with the PARTSIZE= table option. The setting for the PARTSIZE= table option takes precedence over the macro setting. The number that you use in the table option is always specified in megabytes and is enclosed in parentheses following the table definition, as shown in the following example:

```
data spdslib.partsize_example (partsize=512);
  /* other statements */
run;
```

Using the PARTSIZE= table option is beneficial because it enables you to set a specific partition size on a table-by-table basis. However, there is a drawback to using the table option. When you use the PARTSIZE= table option, the partition size that you define only applies to that specific table.

The following program creates three tables. In this program, the CONTENTS procedure contains the partition size. Notice that the partition size for two of the tables is set by the macro %LET SPDSSIZE, and the partition size for the third table is set by the PARTSIZE= table option.

```
libname path1 sasspds 'path1' server=sunflare.5140 user='anonymous';

%let spdssize=512M;

data path1.table1_512m_partsize
  path1.table2_512m_partsize
  path1.table3_1024m_partsize (partsize=1024);
  do i=1 to 1000;
    output;
  end;
run;

proc contents data=path1.table1_512m_partsize (verbose=YES);
run;
```

The option VERBOSE=YES that is used in the CONTENTS procedure displays table attributes that are specific to SAS SPD Server. The following output, which is generated by PROC CONTENTS, shows the partition size (in bytes) for PATH1.TABLE1_512M_PARTSIZE. The partition size is shown at the bottom of the output.

The CONTENTS Procedure

Data Set Name	PATH1.TABLE1_512M_PARTSIZE	Observations	1000
Member Type	DATA	Variables	1
Engine	SASSPDS	Indexes	0
Created	Thu, Sep 21, 2006 02:16:02 PM	Observation Length	8
Last Modified	Thu, Sep 21, 2006 02:16:02 PM	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	Default		
Encoding	wlatin1 Western (Windows)		

Engine/Host Dependent Information

Blocking Factor (obs/block)	4094
ACL Entry	NO
ACL User Access (R,W,A,C)	(Y,Y,Y,Y)
ACL UserName	ANONYMOU
ACL OwnerName	ANONYMOU
Data set is Ranged	NO
Data set is a Cluster	NO
Data Partsize	536870912

Basic Approaches for Controlling Partition Size

Because SAS SPD Server's parameter file sets a minimum partition size for all domains, you can control partitioning sufficiently for some tables by using the MINPARTSIZE= parameter. However, keep in mind that one rule does not fit every situation and that the determination of an appropriate table size for controlling partition size (using methods discussed later) is subjective.

One basic approach is to determine and use a partition size that creates between 100 and 200 partitions for each table. The section "Creating Multiple Partitions per Data Path" discusses approaches to creating tables based on the number of partitions that you want per data path. Using an approach that has either 100 or 200 partitions requires that you have some knowledge of the table, but this approach is simple enough to provide a good distribution of data across all data paths. You can achieve good distribution of data with table sizes that are estimated in bytes. To use this method to determine the partition size, estimate the size of the table in megabytes and divide that number by the number of partitions that you want.

Another basic approach is to set a specific number of partitions for each data path. You should start this approach by determining the partition size that is required in order to have one partition per data path. Such a determination requires that you have specific knowledge (or a good estimate) of the table's row width and the number of rows. After you determine the specific partition size, you can easily determine the size of partition that is necessary to create 2, 3, 4, 5, or more partitions per data path.

Note: These calculations require that you round up the number of rows. The CEILING function in SAS (or similar functions in spreadsheets) rounds up a number to a significant digit. The syntax for the CEILING (alias CEIL) function is as follows:

CEILING (number, significance)

To perform the base calculation, you need to know the

- observation length
- number of rows in the table or a close estimate
- number of data paths in the domain.

The calculation consists of two parts. The first part determines the number of rows per data path.

CEILING (*row-in-table/number-of-data-paths,1*)

The value from the first calculation becomes input for the second calculation, and the result of the second calculation is the partition size.

CEILING (*((rows-per-partition * observation-length) / (1024*1024), 1)*)

The following examples illustrate how to calculate the partition size for a table that has one partition per each of 12 data paths. The table in Example 1 has a row width of 1043 bytes and 93,256,104 rows.

Example 1

Number of Rows per Partition

```
ceiling (93256104/12, 1)
```

The result for this calculation is 7,771,342 rows per partition.

Partition Size

```
ceiling ((7771342 * 1043) / (1024*1024), 1)
```

Per the second calculation, each partition size should be 7,731 megabytes.

Example 2

The table in Example 2 contains 253,562,411 rows and has a width of 954 bytes.

Number of Rows per Partition

```
ceiling (253562411/12, 1)
```

In Example 2, the number of rows per partition is 21,130,201.

Partition Size

```
ceiling (21130201 * 954) / (1024*1024), 1)
```

The resulting partition size is 19,225 megabytes.

To set a specific partition size, you must have specific knowledge of the table as well as control of the job that creates the table. However, sometimes you might not know the exact number of rows, so you must use an estimate. The section “Estimates for Row Counts” explains ways to estimate the size of a partition.

The examples in this section are based on a single partition per data path. However, in certain situations, a single-partition model is an inefficient approach for partitioning a table. The next section discusses situations in which it is better to create multiple, smaller partitions.

Creating Multiple Partitions per Data Path

Certain situations call for multiple partitions per data path; for example, a situation in which data is appended to a table on a daily, weekly, or monthly basis might be better suited for multiple partitions for each data path. From an I/O standpoint, using the single-partition model in this type of situation can be inefficient. Suppose you have a table with 120 million rows and one partition with 10,000,000 rows of data that is loaded on a monthly basis. Such a table will not be distributed across all data paths and file systems until the table is fully loaded with 12 months of data. The single-partition model would create access-pattern problems because new data that is loaded into a table is often accessed more frequently than old data. The new month of data that is added will reside on a single data path, which limits the I/O and defeats SAS SPD Server’s ability to enable a scalable environment.

Using the single-partition model on a hypothetical server, one month of data with 10 million rows in one data path can be read in 5 minutes. However, you can decrease the processing time and increase efficiency by distributing the 10 million rows across 12 partitions (one for each month) per each data path. With 12 partitions per path, you will have 144 data partitions in the final table. Each month, 1/12th of the new data will be loaded into a partition in one data path, 1/12th will be loaded into the next data path, and so on, until the entire month is loaded across all data paths. This arrangement enables better threading for each month of data. While 10 million rows in a single data path can be read in 5 minutes, the processing time decreases by a factor of 12 when a month’s data is distributed across all data paths. The hypothetical processing time for a single month becomes 25 seconds when you use a model of 12 partitions per data path (5 minutes divided by 12 partitions).

Loading a month of data across all 12 data paths is important for enabling scalable access to the data. Often, data is loaded by using a specific table column and an index that is applied to that column. Generally, the column is a monthly date column (such as a transaction date or the date that the transaction was processed) in the table. Indexing a column provides quick access to rows in the table for a specific date or for a range of dates in the column. In this type of situation, the single-partition model results in each month being stored on a single data path. Even when the table is fully loaded, the data distribution will limit the reading of that month to 10 million rows per 5 minutes if a user or group of users concurrently select data for a specific month and the index is used. This limitation occurs because that month of data resides on a single data path. As with the previously discussed example, a model that has 12 partitions per data path is more appropriate for this example. The following table shows the total number of data partitions that are required when the number of data paths and months of data vary.

Data Paths in Domain	Final Number of Months in Table	Partitions per Data Path	Total Data Partitions
6	12	6	72
	24	6	144
	48	6	288
	60	6	360
	72	6	432
12	12	12	144
	24	12	288
	48	12	576
	60	12	720
	72	12	864
24	12	24	288
	24	24	576
	48	24	1152
	60	24	1440
	72	24	1728

Table 3. Total Number of Data Partitions Needed for Varying Numbers of Data Paths and Sets of Data

These examples are cases where each month has an equal number of rows. However, that is rarely the case in real-world situations. Businesses have busy months and slow months, which means that the number of rows of data vary over time. So, often, the calculations for partition size are estimated numbers. Regardless, the model that is presented here provides a good starting point for understanding how to partition a SAS SPD Server table.

Estimating Rows Counts

Often, the number of rows that are loaded into a table is an estimate rather than a specific count to set the partition size. Estimation can occur in situations where business cycles are seasonal and where the amount of data changes with those seasons. In a seasonal cycle, a final table might have 120 million rows, but the number of rows varies throughout the months of the business cycle. When that estimate is not equal to the actual number of rows that are loaded, the distribution of data will not be optimal, which can affect performance. The following examples (with estimates higher and lower than the actual number of rows) show how estimating row counts can affect performance.

Example 1: Using an Estimate That Is Higher than the Actual Number of Rows

Suppose you have a table that contains exactly 110 million rows, but you estimate the number of rows to be 120 million. In the single-partition model, when you use too high an estimate for the number of rows, the server creates a table in which the data is not distributed across all of the data paths; that is, 1 data path (file system) will not have a partition. You can measure the limited distribution of the table across only 11 of the 12 data paths by calculating how many rows are read in 5 minutes. In this example, each of the 11 partitions contains 10 million rows, which the server can read in 5 minutes. However, a table that has evenly distributed rows across the 12 data paths will contain only 9,166,667 rows per partition. The server reads partitions of that size

about 8% faster (or 4 minutes, 35 seconds) than it can read 10 million rows in 11 partitions. Regardless of how many partitions are created, the best run time that you can achieve is 4 minutes and 35 seconds for a table of this size. Optimal performance is achieved by evenly distributing a table across all data paths.

If you use too high an estimate for the number of rows, one or more data paths will not be used in the partitioning process. While using 2 partitions per data path changes the number of rows per partition and improves the total throughput, it might not improve the processing time. The best run time will be the longest amount of time that is required by a thread to process all of the partitions that are assigned to that thread. When you use an estimate, it is best to use multiple partitions per data path. If you use 2 partitions for each of the 11 data paths in this example, you will have a model that contains 22 partitions. Ten of the data paths will have 10 million rows, and the other 2 paths will have 5 million rows each. In this example, run time does not improve because the server still requires 5 minutes to read 10 million rows from the 10 data paths. This observation leads to an important conclusion. Performance is determined by the data paths that have the largest number of rows. You can create small enough partitions to affect run time. However, the best case is always an even distribution because the work load is evenly distributed.

Example 2: Using an Estimate That Is Lower than the Actual Number of Rows

If you use too low an estimate for the number of rows, performance will be affected in a different way. Suppose you have a table that has 130 million rows, but you set the partition size based on an estimate of 120 million rows. In this case, the server will create more partitions than are expected. This case creates a situation where the extra partitions can also negatively affect run time. In a single-partition model, when you use an estimate that is lower than the actual number of rows, the table will have unexpected partitions. For this example, 11 of the data paths contain a single partition with 10 million rows, and 1 data path contains 2 partitions (for a total of 20 million rows in that data path).

An example with 12 threads (as was used in the earlier example) illustrates how the extra partition slows the processing of the table. Each thread works on its own partition. If you have 12 threads and 13 partitions, the remaining partition will not be processed until one of the threads completes the processing of its own partition. The processing time is approximately 5 minutes for each of the first 12 threads (each of the threads can process their respective partitions in 5 minutes). However, one thread will perform double the work (because it must also process the 13th partition), which requires an additional 5 minutes of processing time. Increasing the thread count to overcome this situation will not achieve optimal performance because multiple threads will be reading from the same data path, which, potentially, can introduce I/O constraints.

As you can see, underestimating the number of rows in a single-partition model can increase processing time because a set of rows in the table will not be distributed well across all of the data paths. In such a case, changing the model from 1 to 2 partitions per data path can positively influence performance. Also in a case of underestimation of the number of rows, the use of multiple, smaller partitions can also positively affect performance. In a model that has 2 partitions per data path, for example, the example table has 26 partitions. In this model, the distribution of rows across the data paths results in 5 million rows per partition. So, 10 of the data paths have 10 million rows, and the remaining 2 data paths have 15 million rows (7.5 million per path). Because the server can process 10 million rows (in one data path) in 5 minutes, it will process the data in the 2 paths (with 15 million rows) in 7 minutes and 30 seconds.

Reducing the partition size further to 4 partitions per data path will provide additional performance improvements. The improvements occur because the work that is needed to process the extra 10 million rows is further divided among the system's resources. Using 4 partitions per data path creates a table that has 52 partitions, which benefits the processing time even more. The row distribution in this model results in 2.5 million rows per partition. Eight of the data paths contain 10 million rows each. The remaining 4 data paths contain 12.5 million rows. The data paths that have 12.5 million rows will each require approximately 6 minutes and 15 seconds to be read by a thread. This amount of time is the run time that is required to read the entire table.

As you can see, it is better to balance the distribution more evenly. The closer you get to a balanced distribution, the more you approach maximum performance.

Partitioning a Dynamic Cluster Table

There are advantages to storing a year's worth of data in standard tables; however, you can accomplish much of the same results with a dynamic cluster table by using member tables that comprise smaller monthly tables. Because dynamic clusters contain multiple SAS SPD Server tables, partitioning a dynamic cluster table presents a different situation. However, you can add and remove data more freely in dynamic clusters. The ability to add and remove data enables the administrator to manage very large tables that contain more data than is manageable in standard SAS SPD Server tables.

When you partition a dynamic cluster table, you should first determine the size of the member tables. Then, you need to determine the partition size for the members in the dynamic cluster table. To determine the partition size, you follow the same process as that used for standard SAS SPD Server tables. However, you need to be more aware of the number of partitions in the final dynamic cluster table and the kernel setting that limits the number of files that a process can open. For example, a kernel setting of 16,386 creates a greater risk of reaching the limit for how many files can be opened.

The dynamic cluster enables tables that have different partition sizes to coexist in the cluster. For example, you can mix member tables that have data partition sizes of 5 gigabytes with member tables that have partition sizes of 1 gigabyte. Therefore, you can mix sets of data for which the partitions are likely to be different; for example, you can mix annual, semi-annual, quarterly, or monthly data in a dynamic cluster. This capability of handling tables that have different partition sizes also permits the optimization of partition size for each member of the dynamic cluster table.

Consider a hypothetical case that uses a 12-month table with 120 million rows in which the rows vary by month. In this example, we will consider just two of the member tables:

- December (which has 15 million rows)
- May (which has 5 million rows)

Using a single partition size for a 10-million row table does not provide optimal distribution for either member table, as was explained previously for examples where the estimates for the number of rows were too high or too low. Because dynamic clusters can contain member tables that have different partition sizes, you can set different partition sizes for the December and May

tables, but each table will still be distributed across all 12 data paths. So, you can have a partition size for the December table that will hold 1.25 million rows and a partition size for the May table that holds 416, 667 rows. By individually optimizing the partition size for each member table, the data will be properly distributed across all data paths.

The next example presents a cluster table that has 120 million rows of data. The original example table (containing 12 months of data) is divided into 12 individual member tables for the cluster. Each cluster member contains 10 million rows. Because the member tables are monthly tables, this example uses the single-partition model. Table 4 illustrates how the number of data partitions increases with the addition of more members (for each year of data that is stored) in this example. Notice that each table has 6 indexes. For each index, the SAS SPD Server requires 2 files, so each member has 12 index files. In addition, each table has its own metadata file and one additional metadata file, which are required for the cluster.

The 12-month dynamic cluster table contains 144 data partitions. However, the total number of index files in the dynamic cluster table increases from 12 to 144 because each member has 12 index files. Each of the 12 members also has one metadata file and the additional cluster metadata file (for a total of 13 more files per member). So the 12-month table has a total of 301 files (144 + 144 + 13).

Data Paths in Domain	Months in Final Dynamic Cluster	Member Partitions per Data Path	Total Data Partitions	Metadata and Index Files for 6 Indexes	Total Files in Dynamic Cluster Table
12	12	1	144	157	301
	24	1	288	313	601
	48	1	576	625	1201
	60	1	720	781	1501
	72	1	864	937	1801

Table 4. Number of Data Partitions for a Dynamic Cluster Table with 120 Million Rows of Data

Partitioning Dynamic Cluster Tables Further with Table Column Values

In addition to the most common type of partitioning (by a date column), you can further partition a cluster table by other column values, for example, a geographic region. So, the table for yearly data can be divided into a set of tables that are loaded in parallel based on values in the DATE column and values in the REGION column. If a table has four REGION values that are distributed across all the data paths, then the number of partitions will increase by a factor of 4. For a very large table that is highly partitioned, the dynamic cluster can contain over 10,000 data partitions. If the kernel setting is low, a job that has more than 1 table might reach the server’s limit for the number of files it can open. You must consider this possibility when you create a dynamic cluster table. With data partitioned into smaller sections, it is possible for each member table to be distributed across 6 or 8 data paths, thereby reducing the number of partitions in the cluster table. You can see the effects of secondary partitioning (by REGION) on the table in Table 5.

Metadata and Index Files in Dynamic Cluster Tables

This section first discusses metadata and index files in relation to standard SAS SPD Server tables, then in relation to dynamic cluster tables. The server will only partition metadata and index files when the file system in which they are stored is full and when additional space is required.

Usually, metadata for a standard SAS SPD Server table is stored in a single file. The only time a table has more than 1 metadata file is when the primary metadata path is full and a current table (that resides in the domain) is modified either by having rows appended, deleted, or changed. Typically, a metadata file is very small compared to the table's data or index components. For each terabyte of disk space that is needed for data (assuming that the tables are compressed), the domain should have 20 gigabytes of disk space for metadata. Metadata for uncompressed tables is much smaller than it is for compressed tables. At a minimum, each table has 2 files for every index.

When the domain has multiple index paths and when a table has an index defined, the starting index path for each index file is randomly chosen. One index file will be placed in the randomly chosen index path, and the second index file will be placed in the next path. This placement helps provide a distribution of index files across all index paths. The distribution might not be even because an index that has a small set of unique values, such as gender (low cardinality), will be smaller than an index that has a large set of values (high cardinality), such as a unique index.

Standard SAS SPD Server tables do not have a limit on the number of metadata files and index files that can be created. However, for dynamic cluster tables, the number of files multiplies according to the number of members in the table. For example, a standard SAS SPD Server table that has 12 months of data has a total of 157 files: 144 data partitions, 1 metadata file, and 12 index files (for 6 indexes). If the table is created as a dynamic cluster table that uses monthly tables and if the number of partitions per data path for the table equals the number of data paths, then the number of files in the dynamic cluster increases to 1885. The cluster comprises the files for the 12 standard SAS SPD Server tables (12x157 files) plus the single metadata file that is required by the dynamic cluster. Table 5 lists the total number of files for the dynamic cluster table based on the number of months of data, the value for REGION (used to further partition the table), and the index and metadata files (based on 6 indexes). Table 5 also shows the total number of files that a dynamic cluster table can have. Compared to a standard SAS SPD Server table, the number of files can increase significantly because of the

- ability to partition a table by using multiple columns
- number of associated metadata and index files.

For a dynamic cluster table that is loaded with monthly member tables, one possible strategy for reducing the number of open files is to change from a monthly load to a quarterly load.

Data Paths in Domain	Months in Final Dynamic Cluster	Number of Regions Loaded	Member Partitions per Data Path	Total Files for Indexes and Metadata	Total Data Partitions	Total Files in Dynamic Cluster Table
12	12	4	1	625	576	1201
	24	4	1	1249	1152	2401
	48	4	1	2497	2304	4801
	60	4	1	3121	2880	6001
	72	4	1	3745	3456	7201

Table 5. Number of Files for Different Dynamic Cluster Tables with 6 Indexes

Conclusion

This paper explains how to set and control partition sizes for tables on the SAS SPD Server. The discussion does not include every type of partitioning, but it

- provides a starting point or model to help you set appropriate partition sizes for tables by using the calculations in "Controlling and Setting Partition Sizes"
- explains single-partition and multiple-partition models
- explains why dynamic cluster tables contain more files than standard SAS SPD Server tables.

This discussion does not explain how to set the kernel, which you need to understand in order to avoid the system limitation on how many files a process can open. For more details about setting the kernel, contact your server administrator.



**THE
POWER
TO KNOW.**

SAS INSTITUTE INC. WORLD HEADQUARTERS 919 677 8000

U.S. & CANADA SALES 800 727 0025 SAS INTERNATIONAL +49 6221 416-0 **WWW.SAS.COM**

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are trademarks of their respective companies. Copyright © 2006, SAS Institute Inc. All rights reserved. 416463.1006