Technical Paper

Scalability Solution for SAS® Dynamic Cluster Tables

SAS® Scalable Performance Data Server® 4.3 and Later

Table of Contents

Introduction

SAS Scalable Performance Data Server is designed to meet the storage and performance needs for processing large amounts of SAS data. As the size of data grows, the demands for processing the data quickly increase, and the storage architecture must change to keep pace with business needs.

Hundreds of sites worldwide use the SAS Scalable Performance Data Server, which hosts data warehouses in excess of 30 terabytes. SAS Scalable Performance Data Server provides high-performance data storage for customers from industries such as banking, credit card, insurance, telecommunications, health care, pharmaceutical, transportation, and brokerage and government agencies.

This paper provides an overview of dynamic cluster tables in SAS Scalable Performance Data Server 4.3 as well as enhancements that have been included in later releases. **Note:** Where the enhancements are discussed, this paper also documents their respective releases. Earlier releases of SAS Scalable Performance Data Server supported two types of clustered data tables: time-based partitioning and partition by value (in an experimental form). In SAS Scalable Performance Data Server 4.3 and later, dynamic cluster tables enable both the partitioning of data based on criteria in the data and parallel loading of the cluster tables. Dynamic clusters also enhance the manageability of tables. New table options, as well as SQL planner enhancements, have been added to take advantage of these new capabilities and to improve query performance.

Readers are assumed to have a basic understanding of SAS Scalable Performance Data Server tables and the concepts of data partitioning, symmetric multiprocessing, and disk configuration for I/O scalability. SAS Scalable Performance Data Server performs best with multiple CPUs and parallel I/O, and the hardware on which it runs must be set up correctly to obtain the maximum benefit. That is, you must maximize the server's input/output (I/O) in order to enable the parallel I/O that will feed the CPUs.

Cluster tables

SAS Scalable Performance Data Server provides a virtual table structure, which is called a *clustered data table* (Figure 1). A *cluster* contains a number of members, where each member is a SAS Scalable Performance Data Server table. The clustered data table uses a layer of metadata to manage the members.

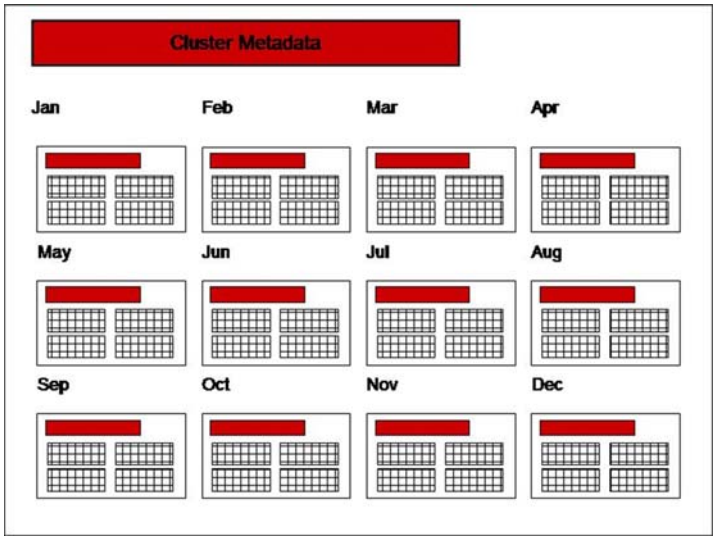

Figure 1: Clustered data table

The virtual table structure in SAS Scalable Performance Data Server offers flexible storage that enables users to organize tables based on values (including SAS date, time, or datetime values) that are contained in numeric columns. Introduced in SAS Scalable Performance Data Server 4.3, this new type of organization is called a *dynamic cluster table*.

Loading benefits of dynamic cluster tables

The power of dynamic cluster tables derives from their ability to load and process data in parallel, making management of large data warehouses easier. Dynamic cluster tables also provide the flexibility to add new data to or to remove historical data from the clustered table by working with the metadata for the cluster without affecting underlying tables. This capability reduces the time needed to complete the job. Additionally, a complete refreshing of a dynamic cluster table requires a fraction of the disk space that would be necessary otherwise, and the refresh process can be divided into parallel jobs that complete sooner. The features of dynamic cluster tables reduce the downtime of the table for maintenance and improve the availability of the warehouse. The benefits of dynamic cluster tables are achieved using simple commands in the SAS Scalable Performance Data Server operator interface procedure (PROC SPDO) to create and alter a cluster. Such operations run in seconds.

Commands for creating and undoing a cluster

Clusters are simple structures, and creating or undoing a cluster takes only seconds. The two most basic commands are CLUSTER CREATE and CLUSTER UNDO. A third command, ADD, is discussed later, and the syntax for a fourth command, LIST, is also presented in a later section. You execute all of these commands within PROC SPDO.

The CLUSTER CREATE command requires three options:

1. the name of the cluster table (*cluster-table-name*) that will be created
2. a list of SAS Scalable Performance Data Server tables that will be included in the cluster (using the MEM= option)
3. the number of slots (using the MAXSLOT= option), for member tables, that the cluster will have.

The following example shows the syntax for PROC SPDO with a CLUSTER CREATE command.

```
PROC SPDO LIBRARY=domain-name;
   SET ACLUSER user-name;
   CLUSTER CREATE cluster-table-name
                  MEM = SPD-Server-table1
                  MEM = SPD-Server-table2
                  MEM = SPD-Server-table3
                  MEM = SPD-Server-table4
                  MEM = SPD-Server-table5
                  MEM = SPD-Server-table6
                  MEM = SPD-Server-table7
                  MEM = SPD-Server-table8
                  MEM = SPD-Server-table9
                  MEM = SPD-Server-table10
                  MEM = SPD-Server-table11
                  MEM = SPD-Server-table12
                  MAXSLOT=24;
QUIT;
```

When you run the CLUSTER CREATE command, it initiates the creation of a new layer of metadata, which contains information about the SAS Scalable Performance Data Server tables that are included in the cluster. The SAS Scalable Performance Data Server tables listed in the command are hidden from direct access by users, making the data accessible through the cluster only. Underneath the top layer of metadata, nothing is changed. Each table still contains the same columns, rows, and indexes. You can see the tables in a cluster in output from the CONTENTS procedure.

The second command, CLUSTER UNDO, removes the cluster metadata. When the metadata layer is removed, you can view the tables in the cluster as normal SAS Scalable Performance Data Server tables. The following example shows the syntax for the CLUSTER UNDO command.

```
PROC SPDO library=domain-name;
   CLUSTER UNDO cluster-table-name;
QUIT;
```

Rules for dynamic cluster tables

Dynamic cluster tables are crucial structures in large warehouses and data marts; therefore, certain rules must be followed.

Users who create dynamic cluster tables must have control privileges for the access control list (ACL) so they can alter ACLs for the tables or in a domain.

Dynamic cluster tables are read-only structures. Therefore, you cannot perform updates, insertions, deletions, or modifications of rows for data that is stored in a dynamic cluster. In addition, you cannot create indexes, append rows, or change column labels and formats. To work with individual tables or rows in the cluster, you must use the UNDO command to remove the structure.

You can only perform four operations on a dynamic cluster table: creating a cluster, undoing a cluster, adding tables, and listing the member tables. Examples of these operations are illustrated in “Cluster table syntax”. To perform any operation other than these four, you must first undo the cluster with the CLUSTER UNDO command. Then, you must make any changes to individual tables in the cluster. Once you make the changes, you can re-create the cluster using the CLUSTER CREATE command.

The following rules have differences between release 4.3 and later releases:

- In the original release (SAS Scalable Performance Data Server 4.3), all member SAS Scalable Performance Data Server tables in a cluster must be identical; that is, they must have the same definitions for columns names, column widths, table indexes, formats on the columns, partition size, compression (ON /OFF), and so forth. You cannot change table names, column names, column formats, indexes, or anything else about any member table, including the cluster name. In release 4.4, the partition size of member tables can be different. The partition size is important for controlling the number of files and distribution of data across file systems. Data volume often follows business cycles. So, allowing a dynamic cluster to contain member tables with different partition sizes enables administrators to account for changes in data volume during different times within the business cycle or as a result of business growth over time.
- When you create a dynamic cluster in the original release, all member tables must have the same owner, and the owner of the cluster must be the same as the owner of the member tables. For example, suppose USER1 creates a table, and USER2 also creates a table. These tables cannot be members of the same cluster, regardless of what ACL permissions are granted on the tables. In release 4.4, member tables in a dynamic cluster can have different owners. However, the owners of the member tables that will be part of the dynamic cluster must grant control privileges to the cluster owner.

Unique indexes

Dynamic clusters support the use of unique indexes across the cluster table. To implement a unique index across the cluster table, each member-table index must be unique. In addition, the values in each unique index for a member table must contain values that are unique across all member tables.

When member tables are combined into a cluster or when a member table is added to a cluster, the application checks values for uniqueness across the entire cluster. If the values are not unique, the process of adding or combining member tables halts. The process also halts if you add a member or members to an existing cluster and the software determines that the new index or indexes contain values that already exist in one of the cluster indexes.

SORTEDBY table option

The SORTEDBY table option has been an important feature of SAS tables for many years. This option sets table metadata that SAS Scalable Performance Data Server uses to determine if a sort is required. If the data is already sorted (ordered), the software can bypass sorting it. The SORTEDBY option stores three pieces of information about a table's sort order in the table metadata:

- whether the table is sorted or not
- a list of columns by which the table is sorted
- the order type (ascending or descending) for each sorted column

The best practice for managing member tables that are in sorted order is to use the SORTEDBY table option. SAS Scalable Performance Data Server 4.4 TSM2 (Technical Support Maintenance release 2) has been optimized to take advantage of the SORTEDBY information. In certain situations, when the order of the data requested matches the SORTEDBY column, the software streamlines the processing, which saves time and resources.

In SAS Scalable Performance Server Data 4.3, you cannot set the SORTEDBY option at the dynamic cluster level, and the information at the member-table level is not used. In release 4.4, you can set the SORTEDBY option at the dynamic cluster level, and there are cases where the information will be used for optimization at the member-table level.

In SAS Scalable Performance Data Server 4.4, when member tables are virtually concatenated through the metadata into a single logical table, you must carefully determine whether to use the SORTEDBY table option for each dynamic cluster. For example, suppose you have a dynamic cluster that contains two tables, each of which are sorted by a column called CARDHOLDER. Each table contains the values 1 through 5. When the two tables are virtually concatenated into a dynamic cluster, the global order of the CARDHOLDER column is the order in which the tables are concatenated together. That is, in this example, the new order in the dynamic cluster will be: 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, which is not in sorted order. So in this example, you should not apply the SORTEDBY option to the dynamic cluster table.

The following examples illustrate other cases where you should and should not use the SORTEBY option. In these examples, each table has a column called CARDHOLDER, which contains numeric values that are in a sorted order. TABLE2 is always concatenated to the end of TABLE1, creating a dynamic cluster called DYNAMICCLUSTER.

Example 1

```
      /* The two tables created and concatenated in this example are not valid
         candidates for the global SORTEDBY table option for dynamic clusters
         because the values in the CARDHOLDER column repeat after five rows.
                                                                              */

   data spdslib.table1 (sortedby=cardholder)
        spdslib.table2 (sortedby=cardholder);
      do cardholder=1 to 5;
         output;
      end;
   run;


   proc SPDO lib=spdslib;
      cluster create combined_tables
                    mem=table1
                    mem=table2
                    maxslot=2;
   quit;
```

Example 2

```
      /* The tables in this example are valid candidates for the SORTEDBY table
         option because the values for the tables do not repeat.              */

   data spdslib.table1 (sortedby=cardholder)
        spdslib.table2 (sortedby=cardholder);
      do cardholder=1 to 5;
         output spdslib.table1;
      end;
```

```
   do cardholder=6 to 10;
      output spdslib.table2;
   end;
run;

proc SPDO lib=spdslib;
   cluster create combined_tables
                  mem=table1
                  mem=table2
                  maxslot=2;
quit;

proc datasets library=spdslib;
   modify combined_tables (sortedby=cardholder);
quit;
```

Example 3

```
 /* The tables in this example are not valid candidates for the global
    SORTEDBY table option because values overlap in the CARDHOLDER column for
    each table.                                                          */

data spdslib.table1 (sortedby=cardholder)
     spdslib.table2 (sortedby=cardholder);
   do cardholder=1 to 5;
      output spdslib.table1;
   end;
   do cardholder=4 to 8;
      output spdslib.table2;
   end;
run;

proc SPDO lib=spdslib;
   cluster create combined_tables
                  mem=table1
                  mem=table2
                  maxslot=2;
quit;
```

Example 4

```
   /* The tables in this example are valid candidates for the global SORTEDBY
      option because, even though there is overlap of values in the CARDHOLDER
      column, the overlap is only for a single value (5) in both tables. */

data spdslib.table1 (sortedby=cardholder)
     spdslib.table2 (sortedby=cardholder);
   do cardholder=1 to 5;
      output spdslib.table1;
   end;
```

```
   do cardholder=5 to 9;
       output spdslib.table2;
   end;
run;

proc SPDO lib=spdslib;
   cluster create combined_tables
                 mem=table1
                 mem=table2
                 maxslot=2;
quit;
```

When a dynamic cluster table is a valid candidate for the SORTEDBY option, the administrator of the table can apply the option to the table using the DATASETS procedure, as shown previously in Example 2. The following example turns on the SORTEDBY flag for the dynamic cluster:

```
proc datasets library=spdslib nolist;
   modify dynamiccluster sortedby=(CARDHOLDER)
quit;
```

When you use the SORTEDBY flag with a dynamic cluster, the software does not perform any validation that the concatenated tables are in sorted order. The administrator of the dynamic cluster must ensure that the data values in the column specified in the SORTEDBY table option are in sorted order.

Cluster table syntax

The following examples illustrate the syntax for the four cluster functions that were discussed previously in “Rules for dynamic cluster tables.”

Example 1: Syntax for Creating a Cluster

PROC SPDO LIBRARY=&domain;
SET ACLUSER *user-name*;
CLUSTER CREATE *sales_hist*
MEM = *sales_table1*
MEM = *sales_table2*
MEM = *sales_table3*
MEM = *sales_table4*
MEM = *sales_table5*
MEM = *sales_table6*
MEM = *sales_table7*
MEM = *sales_table8*
MEM = *sales_table9*
MEM = *sales_table10*
MEM = *sales_table11*
MEM = *sales_table12*
MAXSLOT=*number_of_slots*;
QUIT;

Example 2: Syntax for Undoing a Cluster

```
PROC SPDO LIBRARY=domain-name;
     SET ACLUSER user-name;
     CLUSTER UNDO sales_hist;
QUIT;
```

Example 3: Syntax for Adding Tables

```
PROC SPDO LIBRARY=domain-name;
     SET ACLUSER user-name;
     CLUSTER ADD sales_hist
                    MEM = 2005sales_table1
                    MEM = 2005sales_table2
                    MEM = 2005sales_table3
                    MEM = 2005sales_table4
                    MEM = 2005sales_table5
                    MEM = 2005sales_table6;
QUIT;
```

Example 4: Syntax for Listing Sales History Data

```
PROC SPDO LIBRARY=domain-name;
     SET ACLUSER user-name;
     CLUSTER LIST sales_hist;
QUIT;
```

Managing cluster tables

Often, either the size of a data table or the window in which the table must be loaded creates a problem for managing the table. So the main purpose of dynamic clusters is to make management of data tables easier. This section provides examples that illustrate how dynamic clusters can benefit the loading and management of tables in a data warehouse.

Loading in parallel

A dynamic cluster table comprises SAS Scalable Performance Data Server tables that are called *member tables*. To load a clustered table in parallel, the software uses separate, concurrent jobs to load each member table. You can either use SAS/CONNECT® software's MP CONNECT functionality or SAS® Data Integration Studio to create and manage the overall process. When all of the individual load jobs are complete, the software issues a simple command to create the cluster. Then, the dynamic cluster table can be accessed as a single virtual table.

The scalability of a parallel load depends on the scalability of the hardware. The first area that must scale is the server I/O. Parallel loading requires multiple, concurrent instances of writing to the disk because each member table is being loaded simultaneously. If the hardware I/O does not scale appropriately, the loading process becomes bound and degrades performance.

Scalability is also required with regard to the number of CPUs on the server. Consider the example of an index creation, which is a CPU-intensive process. SAS Scalable Performance Data Server is capable of creating multiple indexes on the same table in parallel. As long as the system has enough CPU processing power, parallel-index creation in SAS Scalable Performance Data Server is highly scalable, and it requires only a single pass through the table to read the data. Even a

single-index creation can benefit from CPU scalability. The creation process for each index is multithreaded, and distributing the threads across multiple CPUs improves performance greatly. If you use multiple job runs to create indexes in parallel, the jobs will become bound unless the server has enough cycles to handle the load.

Adding new member tables

SAS Scalable Performance Data Server enables you to add a new member table to an existing cluster in a matter of seconds. When a cluster is created, the required option MAXSLOT= sets the maximum number of slots (member tables) for the cluster table. For example, if a cluster is created with MAXSLOT=36, then up to 36 tables can be part of the cluster. If the existing cluster has 24 slots in use, then 12 more slots are available for tables to be added to the cluster.

You can increase the maximum number of slots for a cluster, however. If all 36 slots in the cluster are filled, for example, but you need to add additional data:

- Undo the cluster using the CLUSTER UNDO command.
- Create a new cluster with the CLUSTER CREATE command.
- Set a higher value in the MAXSLOT= option for the new cluster.

The latest release of SAS Scalable Performance Data Server makes it easier to add new data by extending a clustered data table. You can independently create the table you want to add to the cluster, then test and verify the table before you add it.

Regardless of the size of the new table, snapping it into a dynamic cluster takes only seconds. Data that already exists in the cluster is not affected by this process.

Removing existing member tables

You can only remove existing member tables when the cluster structure is removed (using the CLUSTER UNDO command). The tables then become standard SAS Scalable Performance Data Server tables that you can archive, copy to other domains, or delete. Once you remove any unwanted member tables, you simply re-create the cluster using the CREATE CLUSTER command.

Performing maintenance in parallel

Once the member tables are unclustered, multiple processes can run in parallel on individual tables. For example, while one process loads data into a new table, another process can refresh existing tables individually to remove deleted rows and to coalesce space within the tables. Simultaneously, the application can delete some tables to free space while leaving other tables unchanged. When all processes are finished on the individual tables, you can quickly re-create the dynamic cluster table.

Refreshing a table

Complete refreshing of a table is often done in order to do the following:

- recapture space from rows that have been deleted in the table
- update existing rows
- reorder data to optimize performance.

However, performing a complete refresh on a table can temporarily consume twice the disk space of the table itself. With such tables, this space issue can limit the update process of a warehouse or a data mart. When server space is scarce, the amount of data that can be refreshed at any given time may be limited. This behavior can cause an increase in the load or in the Refresh window, or it can make refreshing of tables impossible.

However, you can avoid space issues on the disk by undoing the cluster into the original tables, and then refreshing the much smaller, individual tables. Using this method, the maximum space requirement is only twice the size of the largest slot in the cluster.

For example, if a dynamic cluster table has 12 equally sized slots, a complete refresh process will only use temporary disk space equal to 1/12th of the entire cluster size. To perform this type of refresh process:

1. Uncluster the tables.
2. Use the free space to refresh one of the individual tables.
3. Back up the refreshed section of the cluster table.
4. Delete the old table to create more free space.
5. Repeat steps 1– 4 until all individual tables have been refreshed.
6. Re-create the cluster table.

If the server has enough free space available to refresh more then one table at a time, then the refresh process can be performed in parallel on multiple tables.

Refreshing multiple tables with limited space

Often a data mart or warehouse has several tables that need to be refreshed concurrently on a regular schedule (monthly, weekly, and so on). When space limits how many tables can be refreshed concurrently, you can use dynamic cluster tables as a method to enable concurrent refreshing.

For example, suppose you have 12 equally sized tables to refresh, but the server only has enough space to refresh three tables at a time using a standard refresh process. The amount of time it will take to complete a standard refresh process on all of these tables will be equal to the amount of time it takes to refresh three tables concurrently by four groups of three. If each group requires two hours, the total run time will take eight hours. On robust and scalable hardware, much of the server’s resources can remain idle while refreshing only three tables, and much of the free space allocated for the refresh process would not be in immediate use.

In this situation, dynamic clusters can help speed the refresh process. Instead of running the process on the 12 standard tables, you can create 12 clustered data tables with six equally sized slots each. By creating dynamic cluster tables, you end up with 72 slots across 12 cluster tables. The free space that previously could only handle a refresh process on three tables can now hold data for refreshing 18 slots concurrently. By breaking the job into 72 smaller jobs, all of the CPU and memory resources can now be used on the refresh process. Running these smaller jobs in parallel enables the system to continue processing one job while it waits for I/O on another job, which enables a much faster refresh process.

Creating restart points

Managing and updating complex data stores often requires a divide-and-conquer approach. The method that SAS Scalable Performance Data Server uses to divide the load process is to create *restart points.* These are designated points along the load process where the process can be restarted from scratch if a problem occurs. These points should not be dependent upon any earlier step in the process.

For example, suppose you have a single job that takes two hours that you can split into two parts, and each part takes an hour. The first part runs to completion, but the second part fails. So only the second part needs to run again. Restart points enable you to run the individual part. By design, dynamic clusters build in these restart points that are not dependent on each other.

Conceptual diagrams

Figure 2 illustrates an example in which you can use snap-in loading to add new tables to an existing cluster.

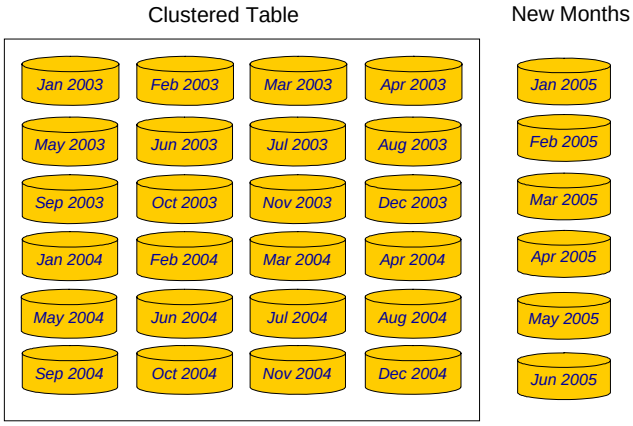

Figure 2: Snap-in loading

Figure 3 illustrates how tables are added in seconds to a cluster using the ADD command.

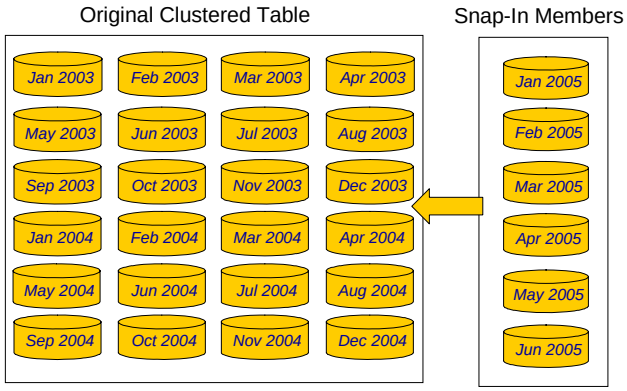

Figure 3: Using the ADD command to snap tables into a cluster

Figure 4 illustrates loading a parallel table with a dynamic cluster.

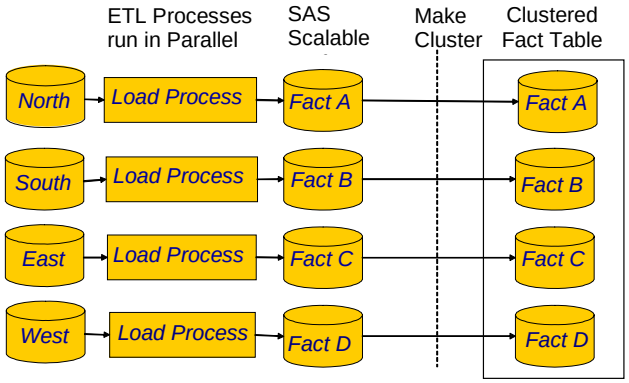

Figure 4: Partition loading

Figure 5 illustrates how unclustered SAS Scalable Performance Data Server tables are refreshed individually using available space.

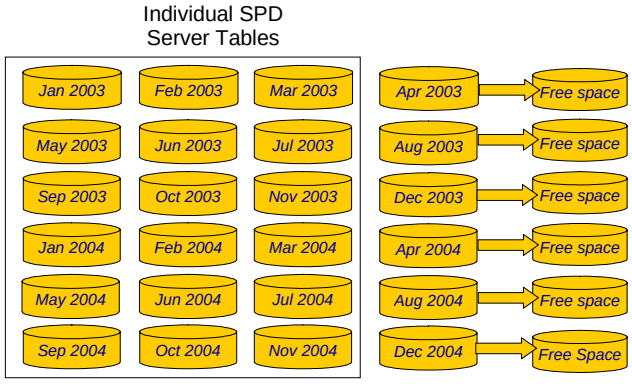

Figure 5: Limited-space table refresh

MINMAXVARLIST= option

The MINMAXVARLIST= table option was a new option in SAS Scalable Performance Data Server 4.3. This option enables you to specify a list of numeric columns during table creation. In SAS Scalable Performance Data Server 4.4, the MINMAXVARLIST= option also supports character columns. When you use this option, SAS Scalable Performance Data Server stores the minimum and maximum values from the specified columns in the table metadata. These values are used by a new layer of WHERE-planner optimization, which can greatly enhance performance.

In addition to the WHERE planner, the application has another layer called MIN/MAX planning. This layer determines whether the entire set of rows in the table are all true or all false based on the range of values that are stored in the table metadata and the values from the WHERE-clause predicate that match the MIN/MAX columns. When the values are either all true or all false, this planning layer bypasses the normal WHERE-clause processing, which requires more resources. When the MIN/MAX planner cannot determine whether the table rows are either all true or all false, the planner reverts to the second layer of WHERE-clause processing, which selects or filters rows by using available indexes or a full-table scan. The MIN/MAX planning layer is the highest level of WHERE-clause processing that is performed by the server.

Consider the example of a column in a SAS Scalable Performance Data Server table with a minimum value of 1 and a maximum value of 10. Based on the predicate from the WHERE clause, MIN/MAX planning returns different results, as shown in Table 1.

Selection Criteria	**Pre-Evaluation Result**
Between 1 and 10	True
Less then or Equal 10	True
Greater then or equal 1	True
NOT GT 11	True
NOT LE 0	True
1 <= column <= 10	True

Selection Criteria	**Pre-Evaluation Result**
Greater then 11	False
Less then 1	False

Selection Criteria	**Pre-Evaluation Result**
= 5	Undetermined
Greater then 1	Undetermined
Less then 10	Undetermined
Greater then equal 5	Undetermined
1 < column <=10	Undetermined
Column in (value list)	Undetermined

Table 1: MIN/MAX planning with minimum value of 1/maximum value of 10

In this example, for the selection criteria **`=5`** and **`IN (value list)`**, the planner cannot determine a value of **`True`** for ALL rows in the table because the column has a range of values of 1 through 10. The MIN/MAX processing has no information on what other values exist in the column. In this case, where the MIN/MAX planning results are undetermined, the application passes the processing to a lower-level (WHERE-clause) processing that can handle the case.

A case does exist in which the MIN/MAX planning is able to determine that all rows have values of **True** for an equality expression that selects or filters the rows in a table. This case occurs when the minimum value and maximum value are equal; that is, there is one unique value in the table column, as shown in Table 2.

Selection Criteria	Pre-Evaluation Result
= 5	False
= 1	True
In (1)	True
In (1,2)	True
Between 0 and 2	True

Table 2: MIN/MAX planning with minimum and maximum values of 1

MINMAXVARLIST= option and cluster tables

MIN/MAX processing might not seem particularly useful when it is applied to a single table because the planner is making a True/False determination for all the values in the table. However, when you combine MIN/MAX planning with the flexibility of dynamic cluster tables, powerful performance benefits emerge.

Suppose you have a dynamic cluster table named SALES_HIST_2004 that contains 12 equally sized slots, where each slot holds millions of rows. Each slot contains all of the records for a specific month based on the values in the column called PROCESS_DTE RANGE. For example, all rows where the range for PROCESS_DTE is between 01JAN2004 and 31JAN2004 are in the slot SALES_HIST_012004 exclusively. Now, assume that there are an equal number of rows present for each day of the month.

You create each slot by using the MINMAXVARLIST= option, which saves the minimum and maximum values for the column PROCESS_DTE. For example, suppose you have PROCESS_DTE records that range from 01JAN2004 through 31JAN2004. The MINMAXVARLIST= option saves the values 01JAN2004 and 31JAN2004, respectively. This slot of the dynamic cluster table has one value for each day of the month, and it contains 31 distinct values in the PROCESS_DTE column. Table 3 shows the relationship between PROCESS_DTE and the slot tables.

PROCESS_DTE RANGE	SLOT NAME
01JAN2004 – 31JAN2004	SALES_HIST_012004
01FEB2004 – 31FEB2004	SALES_HIST_022004
01MAR2004– 30MAR2004	SALES_HIST_032004
01APR2004 – 31APR2004	SALES_HIST_042004
01MAY2004- 31MAY2004	SALES_HIST_052004
01JUN2004 – 30JUN2004	SALES_HIST_062004
01JUL2004 – 31JUL2004	SALES_HIST_072004
01AUG2004– 31AUG2004	SALES_HIST_082004
01SEP2004 – 30JSEP2004	SALES_HIST_092004
01OCT2004 – 31OCT2004	SALES_HIST_102004
01NOV2004– 31NOV2004	SALES_HIST_112004
01DEC2004 – 31DEC2004	SALES_HIST_122004

Table 3: Relationship between PROCESS_DTE and slot tables

Consider the following query run against SALES_HIST_2004:

```
select *
   from SALES_HIST_2004
   where PROCESS_DTE between
         '01JUL2004'd and '30SEP2004'd;
```

The planner recognizes that the MIN/MAX information is stored in the metadata for the column. It then performs a pre-evaluation on each individual slot of SALES_HIST_2004, identifying slots in the cluster that have values of **`True`**, **`False`**, or **`Undetermined`**. Table 4 shows the results for each slot of the dynamic cluster table.

Slot Name	Pre-Evaluation Result
SALES_HIST_012004	False
SALES_HIST_022004	False
SALES_HIST_032004	False
SALES_HIST_042004	False
SALES_HIST_052004	False
SALES_HIST_062004	False
SALES_HIST_072004	True
SALES_HIST_082004	True
SALES_HIST_092004	True
SALES_HIST_102004	False
SALES_HIST_112004	False
SALES_HIST_122004	False

Table 4: Results for dynamic cluster table slots

Any table for which the MIN/MAX processing returns a value of **`False`** will be bypassed for further processing, because a result with a value of **`False`** means no row within that slot meets the selection criteria. The planner will also select all rows from the slots where the MIN/MAX value is **`True`**.

Similarly, consider the following SQL query where only a single PROCESS_DTE is selected instead of a range:

```
select *
   from SALES_HIST_2004
   where PROCESS_DTE='15SEP2004'd;
```

Table 5 shows the results of the MIN/MAX processing. Eleven out of twelve slots are eliminated using the MIN/MAX pre-evaluation. With eleven slots eliminated, the planner performs further processing on the one remaining slot, where it is undetermined how many, if any, rows meet the selection criteria. If the MINMAXVARLIST option has been used and an index exists on the PROCESS_DTE column, the WHERE-clause evaluation processing might use the index to evaluate that specific portion of the WHERE predicate.

Slot Name	Pre-Evaluation Result
SALES_HIST_012004	False
SALES_HIST_022004	False
SALES_HIST_032004	False
SALES_HIST_042004	False
SALES_HIST_052004	False
SALES_HIST_062004	False
SALES_HIST_072004	False
SALES_HIST_082004	False
SALES_HIST_092004	Undetermined
SALES_HIST_102004	False
SALES_HIST_112004	False
SALES_HIST_122004	False

Table 5: Results of MIN/MAX processing

This example illustrates how you can use MIN/MAX planning and SAS Scalable Performance Data Server indexing capabilities together. If you expect many user queries that select single values for the column PROCESS_DTE, then it is appropriate to create an index. In the previous example, even though MIN/MAX planning eliminated 11 of the 12 slots, the planner still had a lot of work to do to further subset the millions of rows in the one slot that remains. However, with an index for the PROCESS_DTE column, the planner can efficiently select rows that match the 15SEP2004 date. Because September has 30 calendar days, you can estimate that one value of PROCESS_DTE will contain 1/30th of the rows in the slot. Therefore, you can see that performance can be greatly enhanced by creating and using an index for queries where a single value is selected.

MIN/MAX planning also works when other columns have indexes and the WHERE clause has multiple predicates. For these cases, the planner first performs the MIN/MAX processing and eliminates any slots with a value of **`False`**. Then the WHERE planner uses the index or indexes to select or filter the rows of that table.

For example, suppose all slots have an index for the column titled STATE. The STATE column for each slot contains 50 values. The SAS Scalable Performance Data Server evaluates the following WHERE clause:

```
where PROCESS_DTE
      between '01JAN2004'd AND '31JAN2004'd
      and STATE='CT'
```

SAS Scalable Performance Data Server first eliminates slot tables by using MIN/MAX processing on the PROCESS_DTE column. Then, it uses the STATE index to select rows that match the second predicate in the WHERE clause. Similar to the previous example, the use of a secondary index greatly improves query performance because the SAS Scalable Performance Data Server can select a fraction of the rows from the member table.

Figure 6 shows a dynamic cluster with MIN/MAX processing on the 2004 member tables, where all of the tables are filtered except the January 2004 member.

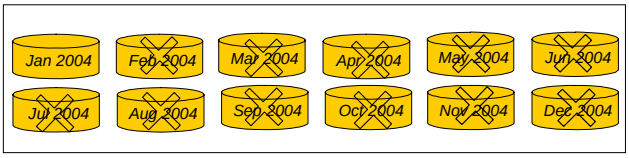

Figure 6: Dynamic cluster with MIN/MAX processing on 2004 member tables

Dynamic cluster BY-clause optimization

SAS Scalable Performance Data Server often uses dynamic clusters to manage huge data sets. The processing of such data sets often needs further manipulation by a SAS job, which requires the data to be ordered. Traditionally, the process of ordering huge data sets is difficult because this type of processing can take significant amounts of time, and it consumes large amounts of system resources. SAS Scalable Performance Data Server 4.4 TSM2 has a new feature called BY-clause optimization that helps reduce the consumption of system resources.

To use dynamic cluster BY-clause optimization, the order of the dynamic cluster member tables must match the order of the BY clause. Every member table in the dynamic cluster must be ordered, using the SORTEDBY option to specify the column or columns that are ordered. When you build your dynamic cluster table from members that are presorted by the BY-clause columns, the dynamic cluster table can then use the BY-clause optimization.

When the BY clause from a SAS job matches the SORTEDBY columns for the member tables, SAS Scalable Performance Data Server orders the rows from the member tables by interleaving them. As a result, the BY clause can process the table rows without using any sort work space that would normally be needed to physically order all the rows. Ordering the rows by interleaving them eliminates any delay in the return of the first record. As a result, BY-clause optimization enables the processing of huge data sets that would be impossible to handle otherwise.

For example, suppose a system has sufficient CPU, memory, and work-space resources to sort a 50-gigabyte data set in a reasonable amount of time. This system accumulates 50 gigabytes of new data every month, it will grow to 600 gigabytes by the end of 12 months. If the system is not robust enough, it might not be able to sort a 600-gigabyte table. However, you can SAS Scalable Performance Data Server to create a dynamic cluster table from the twelve, 50-GB member tables and store each month of data in a member table that is ordered with the SORTEDBY option.

At this point, the BY-clause optimization can easily handle the ordering of the rows. Then, SAS can process steps that require BY clauses for the ordered member column of the 600-gigabyte cluster. For example, suppose you run a MERGE statement in a DATA step, and the dynamic cluster table is used as the master source for the MERGE statement. The BY clause in that MERGE statement will then trigger the dynamic cluster BY-clause optimization. As a result, the operation completes in the time that it takes to interleave the individual member tables, without using any additional work space and without any implicit BY-sort delays.

Dynamic cluster By-clause optimization can be combined with a limited set of WHERE clauses on dynamic cluster tables. For the WHERE-clause optimization to work in conjunction with BY-clause optimization, SAS Scalable Performance Data Server must be able to determine whether the WHERE clause is trivially true or trivially false for each member table in the dynamic cluster. A *trivially true* WHERE clause is one in which the condition being considered is true for every row in the member table. A *trivially false* WHERE clause is one in which the condition being considered is false for every row in the member table. After all of the member tables are determined to be trivially true or trivially false, the BY-clause optimization will operate only on the set of member tables that are trivially true.

SAS Scalable Performance Data Server keeps metadata about indexed values in dynamic-cluster member tables. If the WHERE clause criteria can be determined to be true or false based on this metadata, WHERE-clause optimization is possible on a member-by-member basis for the entire dynamic cluster. Consider, as an example, a dynamic cluster that has 12 member tables, where each member table contains a specific month of data. For this example, each member table either has an index or has had the MINMAXVARLIST= option executed on the column QUARTER (1=Jan-Mar, 2=Apr-Jun, 3=Jul-Sep, 4=Oct-Dec). Suppose you need to run a MERGE statement in a DATA step, and the MERGE statement contains WHERE QUARTER=2. Because the QUARTER column has been indexed or had the MINMAXVARLIST= option run on that column, SAS Scalable Performance Data Server will determine whether each member table is trivially true or

trivially false. The software then processes only the member tables that are trivially true for the BY-clause expression. This means that only the member tables for April, May, and June are processed using the BY-clause optimization.

The BY-clause optimization is triggered when member tables all have an applicable sort order for the BY clause that is asserted. When the sort order is strong (validated), SAS Scalable Performance Data Server does not perform checks to verify the order of BY variables that are returned from the member table. When the sort order is weak (such as from a sort assertion that was a data set option), the software performs additional checking to verify the order of BY variables that are returned from the member table. If an invalid BY-variable order is detected, SAS Scalable Performance Data Server terminates the BY clause and displays the following error message:

```
ERROR: Clustered BY member violates weak sort order during merge.
```

Examples

Consider a database of medical patients' insurance claims, with quarterly-claims data sets that are named ClaimsQ1, ClaimsQ2, ClaimsQ3, and ClaimsQ4. Each quarterly-claims table is sorted by columns that are named PatID (for Patient ID) and ClaimID (for Claim ID). The member tables are combined into a dynamic cluster that is named ClaimsAll. The following code illustrates how the dynamic cluster is created:

```
data SPDS.ClaimsQ1;
    . . . more statements . . .
run;

data SPDS.ClaimsQ2;
    . . . more statements . . .
run;

data SPDS.ClaimsQ3;
    . . . more statements . . .
run;

data SPDS.ClaimsQ4;
    . . . more statements . . .
run;

proc sort data=SPDS.ClaimsQ1;
   by PatID ClaimID;
run;

proc sort data=SPDS.ClaimsQ2;
   by PatID ClaimID;
run;

proc sort data=SPDS.ClaimsQ3;
   by PatID ClaimID;
run;

proc sort data=SPDS.ClaimsQ4;
   by PatID ClaimID;
run;
```

```
proc spdo lib=SPDS;
  create cluster ClaimsAll
                 mem=ClaimsQ1
                 mem=ClaimsQ2
                 mem=ClaimsQ3
                 mem=ClaimsQ4
  ;
quit;
```

Consider the DATA step MERGE statement to be submitted to the ClaimsAll dynamic cluster table:

```
data SPDS.ToAdd SPDS.ToUpdate;
   merge SPDS.NewOnes(IN=NEW1)
         SPDS.ClaimsAll(IN=OLD1);
      by PatID ClaimID;
      select;
         when(NEW1 and OLD1)
            do;
               output SPDS.ToUpdate;
            end;
         when(NEW1 and not OLD1)
            do;
               output SPDS.ToAdd;
            end;
run;
```

If ClaimsAll was not a dynamic cluster table and the table was not sorted, the DATA step containing the MERGE statement would perform an implicit sort to satisfy the BY clause. However, ClaimsAll is a dynamic cluster table, and the individual member tables are sorted. As a result, the dynamic cluster BY-clause optimization knows (from the metadata) that it can bypass the step of physically ordering the rows and use BY-clause optimization to interleave the sorted member tables instantaneously (again without additional work space or delays). The previous example merges the transaction data named NewOnes into new rows that are appended to the data for the next quarter.

Consider that the member data sets ClaimsQ1 and ClaimsQ2 either have indexes or they have had the MINMAXVARLIST= option run on the column Claim_Date. In the following example, the WHERE clause determines whether each member table is true or false for each quarter. The WHERE-clause evaluation process will determine that the rows in the member ClaimsQ1 are trivially true, while the rows in the members ClaimsQ2, ClaimsQ3, and ClaimsQ4 are trivially false. As a result, the BY-clause optimization only processes member table ClaimsQ1.

```
data SPDS.RepClaims;
   set SPDS.ClaimsAll;
   where Claim_Date BETWEEN '01JAN2007' and '31MAR2007';
   by PatID ClaimID;
run;
```

Suppose that the range for Claim_Date is changed in the WHERE clause:

```
data SPDS.RepClaims;
   set SPDS.ClaimsAll;
   where Claim_Date BETWEEN '05JAN2007' and '28JUN2007';
   by PatID ClaimID;
run;
```

When the WHERE-clause optimization evaluates the members of the table to be either trivially true or trivially false, the BY-clause optimization will be used in combination with the WHERE clauses. When one member is not trivially true or trivially false, the BY-clause optimization is bypassed, and all of the rows that are selected by the WHERE clause are physically sorted. In this example, the WHERE clause is not trivially true or trivially false for the member tables ClaimsQ1 and Claims Q2. When the WHERE clause evaluates ClaimsQ3 and Claims Q4, they are determined to be trivially false. As a result, the BY-clause optimization is passed, and the server's implicit sorting facility is used (with the WHERE clause applied to the entire table).

Cluster security

This section covers two aspects of security in the dynamic cluster model. The first aspect is the safety and integrity of underlying data that is contained in the cluster members. The safety and integrity of data are crucial because a cluster table often contains the entire historical view for a table.

To maintain safety and data integrity, the SAS Scalable Performance Data Server only enables users to perform the four functions mentioned earlier in "Cluster Table Rules." As discussed previously, you can only create a cluster, undo a cluster, add tables, and list member tables. You CANNOT delete clusters. The main purpose of a cluster is to provide you with a safe tool for managing tables. Often, the underlying tables in a dynamic cluster are a continuous set of historical data. If users were allowed to delete clusters, they could delete all of the historical data in a table. So while you can remove individual members, you cannot delete the cluster and all of the historical data tied to it.

The second method of security in the SAS Scalable Performance Data Server is the implementation of access control lists (ACLs). ACLs that are applied to an individual SAS Scalable Performance Data Server table do not apply when that table becomes a member in a cluster table. Any ACL that existed before the table became part of a cluster will still apply when the cluster is undone. When a dynamic cluster table is created, the same SAS Scalable Performance Data Server security model that applies to a standard SAS Scalable Performance Data Server table also applies to a dynamic cluster table. To allow or restrict users from reading a cluster table, the creator of the cluster can apply restrictive ACLs to the cluster just as they could do for any other SAS Scalable Performance Data Server table.

Conclusion

Dynamic cluster tables are designed to enhance and optimize the performance of operations on warehouses and data marts while consuming fewer disk resources. Clusters enhance performance in several ways:

- By creating virtual tables (clusters) from multiple SAS Scalable Performance Data Server tables, the SAS Scalable Performance Data Server can perform parallel loading that results in faster extract, transform, and load processes.
- Clusters enable you to refresh tables while using less space and administration.
- You can quickly and easily add incremental data or remove old data from a table, thereby reducing downtime for applications that are dependent on the warehouse or data mart.
- The inclusion of MIN/MAX metadata for columns of a table optimizes query planning. When teamed with scalable hardware, the SAS Scalable Performance Data Server's dynamic cluster tables and MIN/MAX optimization improve the manageability, scalability, and performance of enterprise data stores.

SAS Institute Inc. World Headquarters +1 919 677 8000 To contact your local SAS office, please visit: **www.sas.com/offices**