ABSTRACT

The objective of Phase IV (post-marketing) trial is to test the safety/efficacy of the treatment on larger population. Such trials generally yield massive data. Even some Phase II/III trials which have large number of enrolled subjects or have larger study duration contain huge databases. It has been observed that SAS® programmers experience problems while working with large SAS datasets. Even basic SAS programming steps become difficult to run on such huge datasets. The problems include high compilation and execution time, SAS I/O errors, memory/disk full, unexpected errors, session time out and so on. Also as the complexity level of study derivations increases, working with huge data becomes more difficult and time consuming and inefficient programming only adds more to it. It becomes more difficult when one needs to validate such a huge data. In such scenarios, it is challenging for the programmer to work around with ever-increasing volumes of data without having to spend hours of pre-processing and preparing those data, and to be able to embed the results of the analytics. This paper provides basic details of SAS High-Performance Analytics (SAS HPA) which can help enormously in reducing the compilation and execution time. The paper also discusses some useful tips and techniques that the Programmer and Validator can use in SAS to improve efficiency while working with huge datasets.

INTRODUCTION

Clinical trials are conducted in various phases to test the safety and efficacy of drug. In early clinical development stage, small numbers of subjects are enrolled in the trial to assess the safety/tolerability of the drug. If the objective of the trial is met, it further moves to next phases. The overall sample size is then increased to collect more information on a diverse population to assess the safety/efficacy aspects of the drug. In some clinical trial designs, the volume of the data plays vital role. As the population size increases, the time taken to handle and analyze the data becomes more complex.

A traditional SAS/STAT procedure provides various procedures to analyze the data; however sometimes using the basic procedures is not enough. In this paper, we will discuss

A) The basics of SAS HPA procedures (Single and Distributed mode) and its benefits in a single machine mode. These fundamentals are discussed through the basic syntax of PROC HPSUMMARY and its comparison with the traditional SAS PROC SUMMARY procedure.

B) Tips for programmers while using BASE SAS when dealing with massive data.

C) Other useful techniques.

SAMPLE DATASET USED IN EXAMPLES:

Consider a clinical trial dataset, DM, created by appending demographic information of a single compound from several studies. This appended DM dataset has around 10 million observations and 16 variables.
A. BASICS OF HPA PROCEDURES

In order to deal with the challenges of massive data, SAS has introduced HPA procedures that leverage huge data to make faster decision making. Beginning SAS 9.4, the HPA procedures are integrated along with their associated MVA products. These procedures run in 2 different modes:

1) Single-Machine mode – In this mode the HPA procedures work on a PC, a Server and a Grid but will only uses the cores of the machine assigned to execute it.

2) Distributed mode – A cluster of nodes with SAS High-Perf ormance Analytics Infrastructure which makes the HPA procedures very powerful. In this mode the data will be loaded into the memory distributed across multiple nodes. The execution involves the cores of all the multiple nodes.

SAS HPA procedures are integrated and work on 3 main components:

1) SAS Grid Computing – shared environment for large jobs and support a growing number of users efficiently.

2) SAS In-Database – executes logic in the database itself. It uses a MPP (Massive Parallel Processing) for faster execution of tasks.

3) SAS In-Memory Analytics – eliminates need for disk based processing for faster analysis. Divides processes into pieces with computation distributed across nodes. SAS In-Memory Analytics had a new addition announced in Dec 2011 in the form SAS High Performance Analytics. The HPA procedure enormously reduces the processing time of the model.

SAS HPA procedure can execute in symmetric multiprocessing (SMP) or massively parallel processing (MPP) mode. In SMP mode the procedure supports multithreading on client machine. The procedure uses the number of CPUs on the machine to determine the number of concurrent threads. In MPP mode the analysis occurs on several nodes of distributed computing environment. In distributed mode the HPA performs the analytics on an appliance that consists of cluster of nodes. This appliance can be:

1) a database management system (DBMS) appliance on which the SAS High-Performance Analytics infrastructure is also installed.

2) a cluster of nodes that have the SAS High-Performance Analytics infrastructure installed but no DBMS software installed.
Distributed (or MPP) mode has several variations

1) **Client-data (or local-data) mode:** The input data for the analytic task are not stored on the appliance or cluster but are distributed to the distributed computing environment by the SAS High-Performance Analytics infrastructure when the procedure runs.

2) **Alongside-the-database mode:** The data are stored in the distributed database and are read from the DBMS in parallel to a high-performance analytical procedure that runs on the database appliance. In this mode, the data is loaded in a distributed environment using PROC HPDS2, DATASTEP, PROC APPEND, or SQL pass-through.

3) **Alongside-HDFS mode:** The data are stored in the Hadoop Distributed File System (HDFS) and are read in parallel from the HDFS.

4) **Alongside-LASR mode:** The data are loaded from a SAS LASR Analytic Server that runs on the appliance.

**DETERMINING SINGLE-MACHINE MODE OR DISTRIBUTED MODE:**
HPA procedures use the following rules to determine whether they run in single or distributed mode:

1) If a grid host (DNS) is not specified, the analysis is conducted in single-machine mode on the client machine that runs the SAS session.

2) If a grid host is specified, the behavior depends on whether the execution is alongside the database or alongside HDFS. If the data are local to the client, you need to use the NODES= option in the PERFORMANCE statement to specify the number of nodes on the appliance or cluster that you want to engage in the analysis. If the procedure executes alongside the database or alongside HDFS, you do not need to specify the NODES= option.

**LIST OF SAS HP PROCEDURES INCLUDED IN THE SAS HPA SERVER:**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>HPLOGISTIC</td>
<td>HPCOUNTREG</td>
<td>HPSLO</td>
<td>HPREDUCE</td>
<td>HPTMINE</td>
<td>HPFORECAST</td>
</tr>
<tr>
<td>HPREG</td>
<td>HPSEVERITY</td>
<td>Select features in OPTMLP</td>
<td>HPNEURAL</td>
<td>HPTMSCORE</td>
<td></td>
</tr>
<tr>
<td>HPMLIN</td>
<td>HPQLM</td>
<td>OPTLP</td>
<td>HPFOREST</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HPNLMOD</td>
<td>OPTMODEL</td>
<td></td>
<td>HP4SCORE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HPSPLIT</td>
<td></td>
<td></td>
<td>HPSCORE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HPGENSELECT</td>
<td></td>
<td></td>
<td>HPSVM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HPCORR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Common set includes:** HPSUMMARY, HPSAMPLE, HPDS2, HPDMDB, HPIMPUTE, HPBIN, HPCORR

**LIST OF HP PROCEDURES BY THE DATA ANALYSIS PURPOSE:**

<table>
<thead>
<tr>
<th>Data Preparation</th>
<th>Data Exploration/Transform</th>
<th>Analytics and Modeling</th>
</tr>
</thead>
<tbody>
<tr>
<td>HPDS2</td>
<td>HPSUMMARY</td>
<td>HPLOGISTIC</td>
</tr>
<tr>
<td>HPDMDB</td>
<td>HPDMDB</td>
<td>HPLOGISTIC</td>
</tr>
<tr>
<td>HPSAMPLE</td>
<td>HPSAMPLE</td>
<td>HPLOGISTIC</td>
</tr>
<tr>
<td></td>
<td>HPREDUCE</td>
<td>HPLOGISTIC</td>
</tr>
<tr>
<td></td>
<td>HPIMPUTE</td>
<td>HPLOGISTIC</td>
</tr>
<tr>
<td></td>
<td>HPBIN</td>
<td>HPLOGISTIC</td>
</tr>
<tr>
<td></td>
<td>HPSPLIT</td>
<td>HPLOGISTIC</td>
</tr>
<tr>
<td></td>
<td>HPSUMMARY</td>
<td>HPREDUCE</td>
</tr>
<tr>
<td></td>
<td>HPDMDB</td>
<td>HPREDUCE</td>
</tr>
<tr>
<td></td>
<td>HPSAMPLE</td>
<td>HPREDUCE</td>
</tr>
<tr>
<td></td>
<td>HPREDUCE</td>
<td>HPREDUCE</td>
</tr>
<tr>
<td></td>
<td>HPIMPUTE</td>
<td>HPREDUCE</td>
</tr>
<tr>
<td></td>
<td>HPBIN</td>
<td>HPREDUCE</td>
</tr>
<tr>
<td></td>
<td>HPSPLIT</td>
<td>HPREDUCE</td>
</tr>
<tr>
<td></td>
<td>HPSUMMARY</td>
<td>HPREDUCE</td>
</tr>
<tr>
<td></td>
<td>HPDMDB</td>
<td>HPREDUCE</td>
</tr>
<tr>
<td></td>
<td>HPSAMPLE</td>
<td>HPREDUCE</td>
</tr>
<tr>
<td></td>
<td>HPREDUCE</td>
<td>HPREDUCE</td>
</tr>
<tr>
<td></td>
<td>HPIMPUTE</td>
<td>HPREDUCE</td>
</tr>
<tr>
<td></td>
<td>HPBIN</td>
<td>HPREDUCE</td>
</tr>
<tr>
<td></td>
<td>HPSPLIT</td>
<td>HPREDUCE</td>
</tr>
<tr>
<td></td>
<td>HPSUMMARY</td>
<td>HPCOUNTREG</td>
</tr>
<tr>
<td></td>
<td>HPDMDB</td>
<td>HPCOUNTREG</td>
</tr>
<tr>
<td></td>
<td>HPSAMPLE</td>
<td>HPCOUNTREG</td>
</tr>
<tr>
<td></td>
<td>HPREDUCE</td>
<td>HPCOUNTREG</td>
</tr>
<tr>
<td></td>
<td>HPIMPUTE</td>
<td>HPCOUNTREG</td>
</tr>
<tr>
<td></td>
<td>HPBIN</td>
<td>HPCOUNTREG</td>
</tr>
<tr>
<td></td>
<td>HPSPLIT</td>
<td>HPCOUNTREG</td>
</tr>
<tr>
<td></td>
<td>HPSUMMARY</td>
<td>HPSEVERITY</td>
</tr>
<tr>
<td></td>
<td>HPDMDB</td>
<td>HPSEVERITY</td>
</tr>
<tr>
<td></td>
<td>HPSAMPLE</td>
<td>HPSEVERITY</td>
</tr>
<tr>
<td></td>
<td>HPREDUCE</td>
<td>HPSEVERITY</td>
</tr>
<tr>
<td></td>
<td>HPIMPUTE</td>
<td>HPSEVERITY</td>
</tr>
<tr>
<td></td>
<td>HPBIN</td>
<td>HPSEVERITY</td>
</tr>
<tr>
<td></td>
<td>HPSPLIT</td>
<td>HPSEVERITY</td>
</tr>
<tr>
<td></td>
<td>HPSUMMARY</td>
<td>HPFOREST</td>
</tr>
<tr>
<td></td>
<td>HPDMDB</td>
<td>HPFOREST</td>
</tr>
<tr>
<td></td>
<td>HPSAMPLE</td>
<td>HPFOREST</td>
</tr>
<tr>
<td></td>
<td>HPREDUCE</td>
<td>HPFOREST</td>
</tr>
<tr>
<td></td>
<td>HPIMPUTE</td>
<td>HPFOREST</td>
</tr>
<tr>
<td></td>
<td>HPBIN</td>
<td>HPFOREST</td>
</tr>
<tr>
<td></td>
<td>HPSPLIT</td>
<td>HPFOREST</td>
</tr>
</tbody>
</table>

**HPSUMMARY PROCEDURE**

HPSUMMARY procedure computes the basic descriptive statistics of the variables. It can run on either single or distributed machine mode. The HPSUMMARY procedure provides the similar functionality and syntax as traditional SAS SUMMARY procedure. In general, PROC HPSUMMARY does the following:

- Calculates descriptive statistics based on moments
- Calculates and estimates quantiles, which includes the median
- Calculates CI for the mean
- Identifies extreme values
- Performs t test

**Syntax:**

```
PROC HPSUMMARY <options> <statistics-keywords>;
    CLASS variables </options>;
```

```
FREQ variable;
OUTPUT <OUT = SAS-dataset> <output-statistic-specifications> </AUTONAME>;
PERFORMANCE performance-options;
TYPES requests;
VAR variables <WEIGHT=weight-variable>;
WAYS list;
WEIGHT variable;
RUN;

Note: We can also use ATTRIB, FORMAT, LABEL and WHERE statements and any global statements.

COMPARISON BETWEEN HPSUMMARY AND SUMMARY PROCEDURE

In most of the cases the analysis performed with HPA procedure is similar to that performed with traditional SAS/STAT procedures; however with HPA procedures the performance gain can be seen with increase in the number of CPUs.

In the below example we have PROC SUMMARY and PROC HPSUMMARY code run on a single machine mode with multithreading enabled with number of threads = 4.

<table>
<thead>
<tr>
<th>PROC HPSUMMARY</th>
<th>PROC SUMMARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code:</td>
<td>Code:</td>
</tr>
<tr>
<td>PROC HPSUMMARY data = dm;</td>
<td>PROC SUMMARY data = dm;</td>
</tr>
<tr>
<td>CLASS trt age race sexc;</td>
<td>CLASS trt age race sexc;</td>
</tr>
<tr>
<td>VAR height weight;</td>
<td>VAR height weight;</td>
</tr>
<tr>
<td>TYPES () trt<em>age trt</em>race trt*sexc;</td>
<td>TYPES () trt<em>age trt</em>race trt*sexc;</td>
</tr>
</tbody>
</table>
| OUTPUT OUT = demog_stat2 n=n mean=mean
| std=std median=median min=min max=max; | OUTPUT OUT = demog_stat2 n=n mean=mean
| RUN;                                | RUN;                              |

In Single Machine Mode

Log:

From the above result we can see the slight increase in performance of the HPA procedure. This is because most of the SAS modeling procedures support multi-threading in single machine mode. Both SUMMARY and HPSUMMARY use the multithreading functionality however HPA procedures show better performance. In HP procedures in order to modify the number of threads we can use PERFORMANCE statement in HPSUMMARY e.g., PERFORMANCE NTHREAD=10.

In the above single machine mode example although the performance increase is not significant, the actual benefits of HPA are visible when you work on distributed machine with multiple nodes (MPP mode). In certain scenarios depending on the environment in which we are running HPA procedures can run 15 to 20 times faster than the traditional SAS procedures.

ADVANTAGES OF HPA PROCEDURES
- Enables the users to run on a single machine mode where SAS is installed.
- Enables the users to run on distributed mode to distribute data and computation on different cluster of machines
- HPA procedures are multithreaded and can exploit all the cores available, whereas not all MVA procedures are multithreaded.
- In order to leverage the benefits of HPA procedures, in SAS 9.4 the procedures in the SAS HPA products are included with the associated MVA product, and we can run these procedures in single-machine mode without licensing the high-performance product.

DISADVANTAGES OF HPA PROCEDURES
- In order to enable the full functionality of HPA procedures organization need to spend extra cost to setup distributed grid environment.
- As of today we have only few HPA procedures available.
B. PROGRAMMING TIPS WHILE DEALING WITH MASSIVE DATA

While working with massive data along with the HPA benefits the SAS programmer must also follow the best practices. Below are the few tips for a programmer to follow while working with massive data.

PROBLEM 1: SIZE OF DATA
SOLUTION: It is imperative to reduce the size of the data for faster processing. In case of massive data, it is extremely important to create a dataset with reduced size to overcome few of the memory constraints. A simplest way to do this is to delete the unwanted observations and variables. It is also recommended to look at the length and attribute of the desired variables which may have major impact on the size of the datasets.

a. EFFICIENT PROGRAMMING
Various base SAS statements are available to delete the unwanted observation. The major used options are Sub-setting IF, WHERE and CASE condition. However it is preferable to use WHERE statement instead of IF statement as WHERE condition is applied on data before they enter Program Data Vector (PDV) and in case of "IF" it is applied after data comes out from PDV.

EXAMPLE
In Demographic tables, there is requirement to analyze data for male subjects. The first step would create a reduced size data which can be achieved through PROC SQL and efficient data steps as described below.

<table>
<thead>
<tr>
<th>DATASET STATEMENT</th>
<th>PROC SQL STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA DM1;</td>
<td>PROC SQL;</td>
</tr>
<tr>
<td>SET DM;</td>
<td>CREATE TABLE DM1 AS</td>
</tr>
<tr>
<td>WHERE SEXN = 1;</td>
<td>SELECT * FROM DM</td>
</tr>
<tr>
<td>RUN;</td>
<td>WHERE SEXN = 1;</td>
</tr>
<tr>
<td></td>
<td>QUIT;</td>
</tr>
</tbody>
</table>

b. COMPRESS DATASETS:
Using the COMPRESS=YES dataset option reduces the size of data by around 10%-15% of its original data size. It is recommended that you do not compress the intermediate datasets.

c. DROP UNWANTED VARIABLES
Size of the data is directly proportional to the numbers of variables present in the data. This results in increased processing time. If the user does not wish to retain all the variables from the dataset, the unwanted variables can be dropped using the available dataset options like DROP= and KEEP= OR using DROP or KEEP statement. A suitable choice between KEEP and DROP should be made. In the above example, variable SEX is not required in the final dataset and hence DROP statement can be used rather than using KEEP statement.

EXAMPLE
DATA DM2;
SET DM;
** Drop unwanted variable I from DM dataset;
DROP I;
RUN;

d. VARIABLE LENGTHS
Reducing the length of the variables also plays an important role in reducing the overall size of the data. Suppose we have to create TRTC and DOSE variables from TRT variable as shown below.

In the example, the newly created variable TRTC and DOSE in the DM3 dataset has default length of 200 and 8 respectively. Basic programming will help to reduce the length of observation as shown below.

EXAMPLE
DATA DM3;
SET DM;
** Set length of variables before creating;
LENGTH trtc $15 dose 3;
trtc = substr(trt, 1, 12);
dose = input(substr(trt, 15, 2), best.);
RUN;

e. DELETE UNWANTED DATASETS
In SAS, every new data is saved in WORK library. These datasets will remain in memory until SAS session ends. The basic rule is to delete the datasets if not required for further processing. These temporary datasets can be deleted using either PROC DATASETS or PROC SQL. Deleting temporary datasets will free up lot of space which can be used by SAS for processing program or storing other datasets.
In the above sample example DM1, DM2 and DM3 is created. If we do not want these dataset for further processing it is advisable to delete the dataset as shown below.

**EXAMPLE**

<table>
<thead>
<tr>
<th>PROC DATASETS</th>
<th>PROC SQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROC DATASETS; DELETE DM1 DM2 DM3;</td>
<td>PROC SQL; DROP TABLE DM1, DM2, DM3;</td>
</tr>
<tr>
<td>RUN;</td>
<td>QUIT;</td>
</tr>
</tbody>
</table>

**PROBLEM 2: SORTING LARGE DATASETS**

**SOLUTION:** Sorting the huge data is always a challenge. While sorting the massive data, the major issue arises in the amount of disk space required. Sorting always takes 3-4 times more space than the actual data size, so sort the data, ONLY if required.

The most common method is to divide the data into small subset and then sort and append the data after sorting. This will help in case you have limited disk space. Another method is to adjust the SORTSIZE and MEMSIZE options to increase the amount of memory that SAS use for sorting.

You can also use the TAGSORT option to sort a large data set. The TAGSORT option stores only the BY variables and the observation numbers in temporary files. The BY variables and the observation numbers are called tags. At the completion of the sorting process, PROC SORT uses the tags to retrieve records from the input data set in sorted order. When the total length of the BY variables is small compared to the record length, TAGSORT reduces temporary disk usage considerably because sorting just the BY variables means sorting much less data. However, processing time is usually higher than a regular sort because TAGSORT increases CPU time and I/O usage in order to save memory and disk space.

**EXAMPLE**

PROC SORT DATA = <dataset-name> TAGSORT;  
BY variable-names;  
RUN;

**PROBLEM 3: MERGING THE DATASET**

**SOLUTION:** SQL vs. Merge and FORMAT

a. **PROC SQL vs. Merge Statement**
   - The efficiency of the 2 methods depends on the type of data and the user requirements.
   - The general consideration is that, if we have to do a simple merge/inner join, data step will be more efficient than SQL code.
   - When data sets are large and unsorted, SQL INNER join will be efficient. In most cases, data step match merge generally efficient than PROC SQL OUTER JOIN.

b. **CREATING FORMAT**
   This will be helpful where we need to merge the dictionary (MEDDRA or WHODRUG) information with the concomitant or medical history datasets. In the scenario, we will merge the dataset by CODE to get corresponding TERM without actually sorting and merging it.

   For e.g., suppose we have a concomitant information dataset CT and the corresponding WHODRUG information from the WHODRUG dataset. Typically, we will sort the CT and WHODRUG datasets by CODE variable and then merge using data-steps merge statement. Another way to merge these 2 datasets is to get the information as a format and then use the format to get the coded TERM.

**EXAMPLE**

** Creating variable FMTNAME, START and LABEL;**
DATA WHOFMT (KEEP = START LABEL FMTNAME);
  SET WHODRUG;
  FMTNAME = "WHOFMT";
  START = CODE;
  LABEL = TERM;
RUN;

** Creating Format WHOFMT from dataset;**
PROC FORMAT CNTLIN = WHOFMT LIBRARY = WORK;
RUN;

** Merging Format to get the final coded information;**
DATA CT1;
  SET CT;
  TERM_NAME = PUT(TERM, WHOFMT.);
RUN;
C. OTHER USEFUL TECHNIQUES

a. SYNTAX CHECK BEFORE RUNNING DATASET:
While running your code with big data make sure your code is syntactically correct. This will help to avoid unnecessary re-runs yielding the waste of time while running the code. In order to avoid this, before running the entire program make sure you use option OBS = N to check if your code is syntactically correct and there are zero ERROR and WARNING. N indicates number of observations SAS should process. Similarly, in PROC SQL we can use INOB= and OUTOB= option to restrict the row for processing. Along with this we can also use FEEDBACK, NOEXEC and VALIDATE options in PROC SQL to check if the code is syntactically correct.

b. AVOID I/O ERROR
In the SAS System, SAS I/O, the most common input/output (I/O) processing, is performed on SAS files such as SAS data sets, SAS catalogs, and SAS index files. Most SAS I/O processing is sequential I/O, which means that the DATA step and SAS procedures read a data set from the beginning of the file to the end of the file. The data is read from or written to pages that are typically 4—64 K in size. With standard SAS options, each time the dataset is read/write to a file it is first stored in a cache file. A large SAS jobs will cause numerous SAS datasets to reside in the cache. However processing large datasets (>=2GB) reduces I/O throughput. Reading several distinct SAS data sets of this size in succession causes the file cache to expand greatly. Even reading a single SAS data set of this size causes the file cache to expand because the cache manager attempts to retain as much of the data as possible. A huge file cache causes a latency each time you read from or write to a large file. The latency occurs because the cache manager has to search through the cached pages of data when it is accessing the file.

The solution is to bypass the file cache. In SAS we can implement the SGIO option in the SAS configuration. The scatter-read feature reads data directly from the disk. The gather-write feature writes data directly to the disk. To activate scatter-read/gather-write file access globally in SAS, specify the SGIO system option either on the command line or in the SAS configuration file. For example, specify the SGIO system option on a command line when you invoke SAS, as follows:

```
c:sas> sas --sgio
```

SAS 9.1.3 SP4 and later also contain the SGIO= data set option to enable and disable SGIO processing for specific datasets.

SGIO processing (activated with the SGIO system option) provides each SAS I/O file with the ability to use scatter read/gather-write access. However, not all SAS I/O files will be opened for SGIO processing even if the SGIO option is specified when you invoke SAS. If a SAS I/O file does not meet the required criteria (https://support.sas.com/resources/papers/IOthruSGIO.pdf), the SGIO processing remains inactive for that file even if you specify the SGIO system option or data set option. If a file is not using SGIO processing, no warning message appears in the SAS log. To obtain maximum throughput, use the SGIO feature apply the BUFSIZE= and the BUFNO= system options.

CONCLUSION
High Performance HPA procedures are modeling procedures are developed for high performance statistical modeling while working with massive data. In single machine mode HPA procedures takes the advantages of the multi-threading capabilities of the system. In distributed mode the HPA procedures exploits all the 3 core components while working with massive data for faster performance. Also as the syntax are similar to traditional SAS procedures its ease to learn and adapt. Furthermore along with the increasing technology advantages, the basic efficient steps help to overcome few of the memory constraints and processing time.

REFERENCE
SAS Support: Base SAS® 9.4 Procedures Guide High-Performance Procedures


https://support.sas.com/resources/papers/IOthruSGIO.pdf

John Cohen, AstraZeneca LP, Wilmington, DE, Table Lookups: Getting Started With Proc Format
CONTACT INFORMATION

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

Dhiraj Dabhi  
Senior Statistical Programmer  
Inventiv International Pharma Services Pvt. Ltd., India  
Marwah Centre, Ground Floor, B Wing, Krishnalal Marwah Marg,  
Andheri (E), Mumbai – 400 072, Maharashtra, India  
T: +91 22 40957363 M: +91 9920466022  
Dhiraj.Dabhi@inventivhealth.com  
http://www.inVentivHealthclinical.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.