PharmaSUG 2026 - Paper AP-251

# Modernizing Clinical Research Analytics with Cloud-Optimized SAS Procedures

Jim Box and Mary Dolegowski, SAS Institute

## ABSTRACT

As clinical research organizations continue migrating analytical workloads to the cloud, they gain access to scalable, multi-threaded computing environments capable of accelerating processing across datasets of any size. This session explores a new generation of cloud-based SAS® procedures designed to take advantage of this modern architecture. We will examine how these procedures differ from their traditional SAS 9 counterparts, highlight performance and usability improvements, and discuss practical considerations for adopting them in real-world clinical analytics workflows.

## INTRODUCTION – CLOUD ARCHITECTURE

Traditional SAS environments that utilize SAS 9 coding are generally designed to run on a single server, which has a fixed amount of processing power and memory, which limits analytical performance to those resources. SAS 9 procs were designed to maximize the performance based on the idea of a single-threaded compute context, be it a PC, a large core server, or even a grid. HP PROCS like **hpreg** were an early step towards parallel processing by running across multiple cores or nodes in a grid environment, but their reach was not very far. SAS Viya was designed for cloud-based distributed computing environments and dynamically distributes work across all available resources and can scale up and involve more resources when needed. Because of this new architecture, there are a whole bunch of new procedures available to run jobs cloud natively. This architecture consists of something called **CAS actions**, which are the cloud-native building blocks for analytics methods. They are smaller and more focused than a PROC, and can be accessed by multiple languages, including SAS, R, Python, and even REST APIs. For the purposes of this paper, we will focus on the CAS-enabled PROCS which string together multiple CAS actions in the familiar language of SAS PROCS.

## EXAMPLE – MIXED MODELS

The easiest way to think about this is with an example. Mixed modeling is a statistical methodology that looks at data with both fixed and random effects. Fixed effects are factors of main interest that don't change over time – think treatment group, dose, demographic groups. Random effects account for variability between subjects or groups, like site. They are often used to look at data over time where data is collected for a subject at multiple visits. In clinical trials, they might be used to see how a clinical score changes over time. These models are computationally intensive and can take a while to execute. Consider a dataset like in Figure 1. In this case, we have created a dataset with repeated measures of students' Math scores over time. There is a variable for what Neighborhood these students live in and what school they attend, as well as some fixed values for those neighborhoods. This particular dataset has 1000 schools, 5 neighborhoods in each, and looks at two students in each neighborhood over four quarters of school, for a total of 40000 records.

Figure 2 shows the SAS 9 code that would be used to run this model. The specifics of the results are not really important, what we are interested in is the compute time. As you can see in Figure 3, this model took **7:28.71s** to converge and produce results. There is a lot of computing going on in a model like this, with many parameter estimates and a complex covariance structure must be computed. In a single-threaded architecture, this will take a while.

## Sample Data

| Obs | SchoolID | nID | Neighborhood | bInt | bTime | bTime2 | sID | Time | Math |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 2.77444 | 14.5513 | 0.39982 | 1 | 1 | 19.1731 |
| 2 | 1 | 1 | 1 | 2.77444 | 14.5513 | 0.39982 | 1 | 2 | 33.2356 |
| 3 | 1 | 1 | 1 | 2.77444 | 14.5513 | 0.39982 | 1 | 3 | 51.0963 |
| 4 | 1 | 1 | 1 | 2.77444 | 14.5513 | 0.39982 | 1 | 4 | 68.3591 |
| 5 | 1 | 1 | 1 | 2.77444 | 14.5513 | 0.39982 | 2 | 1 | 18.4134 |
| 6 | 1 | 1 | 1 | 2.77444 | 14.5513 | 0.39982 | 2 | 2 | 35.7161 |
| 7 | 1 | 1 | 1 | 2.77444 | 14.5513 | 0.39982 | 2 | 3 | 49.8063 |
| 8 | 1 | 1 | 1 | 2.77444 | 14.5513 | 0.39982 | 2 | 4 | 68.2294 |

**Figure 1**: Sample data suitable for a Mixed Model

```
title "Standard 9.4 approach to Mixed Models";
proc mixed data=SchoolSample;
   class Neighborhood SchoolID;
   model Math=Time Time*Time / solution;
   random int Time Time*Time / sub=Neighborhood(SchoolID) type=un;
   ODS EXCLUDE ClassLevels;
run;
title;
```

**Figure 2**: PROC MIXED, a SAS 9 procedure.

```
NOTE: Convergence criteria met.
NOTE: PROCEDURE MIXED used (Total process time):
      real time           7:27.89
      cpu time            7:28.71
```

**Figure 3**: PROC MIXED compute time

Now let's see what happens when we run a model leveraging the cloud architecture available to us. Figure 4 shows how the new **PROC LMIXED**. Note that in this case, the model structure and options are exactly the same as the non-CAS PROC. This is not always the case, as the underlying algorithm is different, but for this specific example, the code structure was the same, and we also got the exact same results for the model. Again, this does not always happen, differences in the algorithms and differences in the options may lead to slight differences in decimal values of estimates.

**Figure 5** shows the impact of utilizing a cloud native method: we dropped the compute time from roughly seven and a half minutes to less than one second.

```
TITLE "Using L-Mixed, Multi-Threaded on a dataset in Work Library";
proc lmixed data=WORK.SchoolSample;
   class Neighborhood SchoolID;
   model Math = Time Time*Time / solution;
   random   int Time Time*Time / sub=Neighborhood(SchoolID) type=un;
run;
title;
```

**Figure 4:** PROC LMIXED

```
NOTE: Convergence criterion (ABSGCONV=0.00001) satisfied.
NOTE: SAS Viya processed the request in 0.119994 seconds.
NOTE: PROCEDURE LMIXED used (Total process time):
      real time             0.16 seconds
      cpu time              0.41 seconds
```

**Figure 5**: Multi-threaded compute time

Note that in this instance, the dataset was in a SAS 9 work library, stored on file as a sas7bdat file. For large data, performance can potentially be improved by having the dataset in memory (in a CAS library, for example), but the algorithms can run on data in either type of library. The Viya compute engine will utilize the cloud no matter where the data is stored. When data is in a CAS library, it will be spread across multiple nodes at random. If the PROC has by-processing, the data will be spread across nodes according to the by-groups.

## OTHER PROCEDURES

The SAS Viya Platform Programming Documentation page (https://helpcenter.unx.sas.com/test/doc/en/pgmsascdc/**default**/procs2actions/p0275qj00ns5pen16ijvuz8f8j5k.htm) lists all of the SAS Procedures in SAS 9 that have an analogous CAS-enabled multi-threaded PROC, and is an immensely useful page (Figure 6). As you can see, there are even CAS versions of the HP PROCS that will be even more optimized for the cloud.

As an aside, a neat trick about a link to SAS documentation – if you change the version number in the URL to the word "default" as seen in green above, you will be taken to the most current version of the SAS document – this is very handy.

| SAS Procedure | CAS-Enabled Procedure | CAS Action | CAS Action Example | Category |
|---|---|---|---|---|
| FACTOR | EFA | faNFactors | Extracting and Interpreting Solutions in Exploratory Factor Analysis | Statistics |
| | | faExtract | | |
| FASTCLUS | KCLUS | clustering.kClus | Clustering with the k-Means Algorithm | Statistics |
| FREQ | FREQTAB | freqTab.freqTab | Produce Frequency and Crosstabulation Tables | Statistics |

**Figure 6**: Example of SAS Documentation

## LEVERAGING THE CLOUD FOR OPEN SOURCE

We wrote a paper for PharmaSUG 2025 (AP-141) that shows how to leverage the cloud for Open-Source languages by using PROC GATEWAY. We won't rehash that whole paper here, but the important idea is that you can programmatically tell R code to run in multiple threads when running a model that would have a group-by approach. In the paper we run a GLM where the model for the women's group runs in one thread and the model for the men's group runs in another (Figure 7).

```
/**  Submitting the code to be multi threaded Via Proc Gateway */

PROC GATEWAY NTHREADS=2 lang=R;
submit;

    df <- read_table(list(caslib= 'public', name= 'heart', groupby='Sex', groupbyMode = 'redistribute'))

    print(gw$num_threads)

    if(nrow(df)>0){

    cat("Thread", gw$thread_id)

    model <- glm(as.factor(Status) ~ AgeAtStart + Systolic + Diastolic + Weight + Cholesterol,
        data=df,
        family = binomial)

    assign(paste("model_",df$Sex[1], sep = ""),model)

    mdf = as.data.frame(summary(assign(paste("model_",df$Sex[1], sep = ""),model))$coefficients)
        mdf$Vars = rownames(mdf)
        mdf <- mdf[, c("Vars", names(mdf)[-ncol(mdf)])]

    assign(paste("mdf_",df$Sex[1], sep = ""),mdf)

    print(df$Sex[1])
    head(assign(paste("mdf_",df$Sex[1], sep = ""),mdf), 10)
    }

endsubmit;
RUN;
```

**Figure 7:** Using PROC GATEWAY to multi-thread an R model

We had to manually identify the number of threads to use and explain how to group the activity onto the different threads, but this can be a significant upgrade in how you run R code on big data or with a computationally complex model, which is difficult to do with just R code. Figure 8 shows that the model groups ran on different threads simultaneously.

```
NOTE: Thread 5[1] "Male"
NOTE:                    Vars      Estimate  Std. Error     z value     Pr(>|z|)
NOTE: (Intercept) (Intercept) -1.011163e+01 0.605165457 -16.7088742 1.129478e-62
NOTE: AgeAtStart   AgeAtStart  1.244938e-01 0.006471886  19.2360864 1.846336e-82
NOTE: Systolic       Systolic  2.646623e-02 0.004062030   6.5155193 7.243861e-11
NOTE: Diastolic     Diastolic  1.322024e-03 0.006372544   0.2074562 8.356536e-01
NOTE: Weight           Weight -4.192031e-04 0.002077309  -0.2018010 8.400723e-01
NOTE: Cholesterol Cholesterol  3.565731e-03 0.001175513   3.0333397 2.418632e-03
NOTE: Thread 4[1] "Female"
NOTE:                    Vars      Estimate  Std. Error     z value     Pr(>|z|)
NOTE: (Intercept) (Intercept) -7.5979906517 0.414528775 -18.3292237 4.837596e-75
NOTE: AgeAtStart   AgeAtStart  0.0939128686 0.006724172  13.9664594 2.497369e-44
NOTE: Systolic       Systolic  0.0112491877 0.003205295   3.5095637 4.488425e-04
NOTE: Diastolic     Diastolic  0.0077942557 0.006067017   1.2846932 1.988995e-01
NOTE: Weight           Weight -0.0006085457 0.001855482  -0.3279717 7.429330e-01
NOTE: Cholesterol Cholesterol  0.0015808878 0.001049167   1.5068035 1.318610e-01
NOTE: WARNING: ignoring environment value of R_HOME
```

**Figure 8:** Multi-threaded model execution results

## CONCLUSION

Cloud-enabled architecture allows for new algorithms that can utilize multiple compute threads. In SAS Viya, these analytical methods are called using CAS-Actions, which can be accessed via SAS Code, R or Python Code, or REST APIs. To make things easier for SAS Programmers, we have made CAS-Enabled versions of commonly used PROCS that leverage the CAS-Actions in a familiar PROC Structure, allowing for significant time savings with computationally intensive algorithms and/or large data.

## RECOMMENDED READING

- SAS Procedures and CAS-Enabled counterparts: SAS Help Center: Procedures Listed Alphabetically
- A New Gateway to Open Source in SAS Viya (PharmaSUG 20245 Paper AP-141) https://www.lexjansen.com/pharmasug/2025/AP/PharmaSUG-2025-AP-141.pdf
- Multithreading, Parallelism, Python, and SAS - SAS Support Communities – SAS Communities Blog Post by Ryan King, October 15, 2024.

## CONTACT INFORMATION

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


| | |
|---|---|
| Jim Box | Mary Dolegowski |
| SAS Institute | SAS Institute |
| Jim.Box@sas.com | Mary.Dolegowski@sas.com |