SAS® GLOBAL FORUM 2020 #SASGF

Paper SAS4579-2020

Accelerate DATA Step BY-Group Processing in SAS® Viya®

Jason Secosky and David Bultman, SAS Institute Inc.

ABSTRACT

BY-group processing is a method of processing observations from one or more data sets so that the observations are grouped by common variable values. SAS® Cloud Analytic Services (CAS) on SAS® Viya® enables you to code your DATA steps such that BY groups are operated on in parallel. Parallel processing can improve performance. In this presentation, learn how to combine tables in CAS with the DATA step SET and MERGE statements. Also, learn how to use FIRST. and LAST. variables in the context of CAS to perform operations at the start or end of a BY group. We also cover issues you might run into as you convert your SAS®9 DATA steps to run in CAS. Come see how to accelerate your DATA steps in CAS.

INTRODUCTION

DATA step is a programming language to prepare tables for analysis. It excels at modifying existing values, computing new values in a row, and combining tables.

In SAS, a DATA step runs in a single thread or core on your system. This is where our problem lies. With big data, running in a single thread is slow. Some DATA steps can take hours to complete. With SAS® Cloud Analytic Services (CAS) on SAS® Viya®, we have an environment where tens or hundreds of threads are available, across several machines. When you program your DATA steps to run in CAS, all those threads become available for you to use to improve performance with massive parallel processing.

Our earlier papers, (Secosky 2017) and (Bultman and Secosky 2018), introduce how to take advantage of parallel processing in CAS with DATA step. This paper describes how to perform BY-group processing in DATA step and how processing BY groups in CAS is faster than SAS while using familiar, straightforward coding.

The first section of this paper introduces BY-group processing in DATA step. The following sections show how to process groups of rows in a single table with the SET statement, and then how to combine groups from two or more tables with the MERGE statement. In these sections, a simple performance comparison between SAS®9 and CAS is presented along with notable differences between SAS and CAS.

SET STATEMENT

The SET statement is one of the most frequently used statements in DATA step. Its function is to process existing data sets in SAS and existing tables in CAS. With no options specified, SAS sequentially reads each observation in the named data sets, one observation at a time until there are no further observations to process. In CAS, multiple rows are read simultaneously because the data is divided among available threads. Each thread processes the rows assigned to it sequentially until there are no more rows to process. Each thread operating on part of the overall data is one strategy that makes CAS faster than SAS.

Because SAS reads rows sequentially, there is the notion of preserving original data set order. However, in CAS there is a tradeoff. Speed improvements from parallel processing are gained at the expense of not keeping track of row or observation order. Using a BY statement can make SAS and CAS more similar, but there are some differences, which we point out in this section.

SET STATEMENT WITH BY

In SAS you need to sort your data on the BY variable before using the SET statement with BY. In CAS you do not need to sort. To illustrate, let's use the following data set to find the minimum animal weight for each animal.

Animals

animal	weight
Ant	0.01
Dog	10.89
Cat	9.85
Ant	0.01
Bird	1.30
Ant	0.02
Cat	8.75
Bird	1.50
Dog	20.67
Cat	7.46
Bird	2.01
Cat	8.51
Ant	0.01
Bird	1.47
Dog	32.50
Dog	30.54

When you run the following code in SAS to find the minimum animal weight, you must sort on the BY variable, ANIMAL:

```
proc sort data=mysas.animals; by animal; run;

data mysas.min_weights;
  retain min_weight;

  set mysas.animals;
  by animal;

  if first.animal then
    min_weight = .;

  min_weight = min(weight, min_weight);

  if last.animal then
    output;
run;

proc print data=mysas.min_weights; run;
```

The PRINT procedure produces the following report of minimum weights:

Animals

animal	weight
Ant	0.01
Bird	1.30
Cat	7.46
Dog	10.89

With data stored in CAS, you can only run the DATA step code without sorting. After loading the data into CAS with the CASUTIL procedure, you change the libref in the DATA step code to a CAS engine libref for the DATA step to automatically run in CAS.

```
proc casutil;
  load data=mysas.animals;
quit;

libname mycas cas caslib=casuser;

data mycas.min_weights;
  retain min_weight;

  set mycas.animals;
  by animal;

  if first.animal then
    min_weight = .;

  min_weight = min(weight, min_weight);

  if last.animal then
    output;
run;

proc print data=mycas.min_weights; run;
```

PROC PRINT produces the following report:

Animals

animal	weight
Ant	0.01
Dog	10.89
Bird	1.30
Cat	7.46

ROW RETRIEVAL ORDER FROM CAS

Although the numeric results are the same, the order of output in the report is different. In CAS, performance improves by splitting data across machines and assigning parts to different threads on each machine. When pulling data back to SAS for reporting, each machine reports its data at the same time and the order they arrive in is non-deterministic. This is one reason the row order is not maintained.

If row order in reporting is important, sorting in SAS is required. This code uses the SORT procedure to pull rows from CAS, sort them, and create a SAS data set used for reporting. In this case, the original sort order is maintained.

```
proc sort data=mycas.min_weights out=mysas.min_weights_cas;
  by animal;
run;

proc print data=mysas.min_weights_cas; run;
```

In the section on merging tables, we see how the row order is not maintained within BY groups and how to resolve the issue by creating a ROWID variable.

IMPLICIT SORT IN CAS

The reason you do not need to sort is CAS implicitly constructs each BY group and assigns groups to threads. If you are processing a table multiple times with the same BY statement, implicitly constructing each BY group for each DATA step takes additional time each time the DATA step runs. To avoid redundant grouping, you can use the partition action to create a table with a permanent grouping of the data.

PARALLEL PROCESSING

When a DATA step runs in CAS, BY groups are assigned to threads. Many BY groups can be assigned to a single thread. However, a BY group cannot span threads.

Let's modify the preceding code slightly to demonstrate where each row resides by adding a PUT statement:

```
data mycas.min_weights;
  retain min_weight;

  set mycas.animals;
  by animal;

  if first.animal then
    min_weight = .;

  min_weight = min(weight, min_weight);

  if last.animal then do;
    put _hostname_= _threadid_= animal= min_weight=;
    output;
  end;
run;
```

The code produces the following output:

```
_HOSTNAME_=worker-03 _THREADID_=22 animal=Bird min_weight=1.3
_HOSTNAME_=worker-04 _THREADID_=33 animal=Dog min_weight=10.89
_HOSTNAME_=worker-06 _THREADID_=102 animal=Ant min_weight=0.01
_HOSTNAME_=worker-06 _THREADID_=105 animal=Cat min_weight=7.46
```

Four threads on three workers were assigned BY groups. Worker-03 and Worker-04 were assigned one BY group and Worker-06 was assigned two BY groups. On Worker-06, each BY group was assigned to a different thread.

When running in SAS, a single thread is reading the rows; when running in CAS, four threads are reading in parallel. This parallelism gives CAS faster run times than SAS when operating on large data.

PERFORMANCE COMPARISON OF SAS AND CAS

When running in SAS, a single thread is reading and writing rows. In CAS, many threads are processing in parallel. To demonstrate the performance improvement from parallel processing, let's create two tables with the same schema as the preceding example but larger. To generate a large amount of data to process, we generate animal names based on random permutations of the first 18 English letters in this code.

Here is the LIBNAME for SAS:

```
libname mysas '/your/path/here/sgf';
```

The LIBNAME for CAS connects the SAS client to the current caslib, which allows SAS to load the data into a CAS table.

```
libname mycas cas caslib=casuser;
```

First, we generate the large animals data set into the mysas library, and then we use the CASUTIL procedure to load it into a CAS table.

```
data mysas.animals;
  length animal $ 18;
  array letters[18] $ 1 ('A' 'B' 'C' 'D' 'E' 'F' 'G' 'H' 'I' 'J' 'K' 'L'
                         'M' 'N' 'O' 'P' 'Q' 'R');
  drop letters: i j seed;

  seed = 4;
  do i = 1 to 1E6;
    call ranperm(seed, of letters[*]);
    animal = cat(of letters[*]);
    do j = 1 to 100;
      weight = rand('uniform');
      output;
    end;
  end;
run;

proc casutil;
  load data=mysas.animals outcaslib=casuser;
quit;
```

Now, let's compute the minimum weights in SAS and CAS.

```
18         proc sort data=mysas.animals; by animal; run;

NOTE: There were 100000000 observations read from the data set
      MYSAS.ANIMALS.
NOTE: The data set MYSAS.ANIMALS has 100000000 observations and
      2 variables.
NOTE: PROCEDURE SORT used (Total process time):
      real time           31.91 seconds
      cpu time            52.82 seconds
```

```
19
20         data mysas.min_weights;
21           retain min_weight;
22
23           set mysas.animals;
24           by animal;
25
26           if first.animal then
27             min_weight = .;
28
29           min_weight = min(weight, min_weight);
30
31           if last.animal then
32             output;
33         run;

NOTE: There were 100000000 observations read from the data set
      MYSAS.ANIMALS.
NOTE: The data set MYSAS.MIN_WEIGHTS has 1000000 observations and 3
      variables.
NOTE: DATA statement used (Total process time):
      real time           9.03 seconds
      cpu time            8.62 seconds
```

Output 1: BY-Group Processing in SAS

```
669        proc cas;
670          partition / table={name="animals"
                                groupby="animal"
                                orderby="animal"}
                         casout="animals_part"; run;
671        quit;

NOTE: PROCEDURE CAS used (Total process time):
      real time           19.26 seconds
      cpu time            0.02 seconds

674        data mycas.min_weights;
675          retain min_weight;
676
677          set mycas.animals_part;
678          by animal;
679
680          if first.animal then
681            min_weight = .;
682
683          min_weight = min(weight, min_weight);
684
685          if last.animal then
686            output;
687        run;

NOTE: Running DATA step in Cloud Analytic Services.
NOTE: The DATA step will run in multiple threads.
NOTE: There were 100000000 observations read from the table ANIMALS_PART
      in caslib CASUSER(jaseco).
NOTE: The table min_weights in caslib CASUSER(jaseco) has 1000000
```

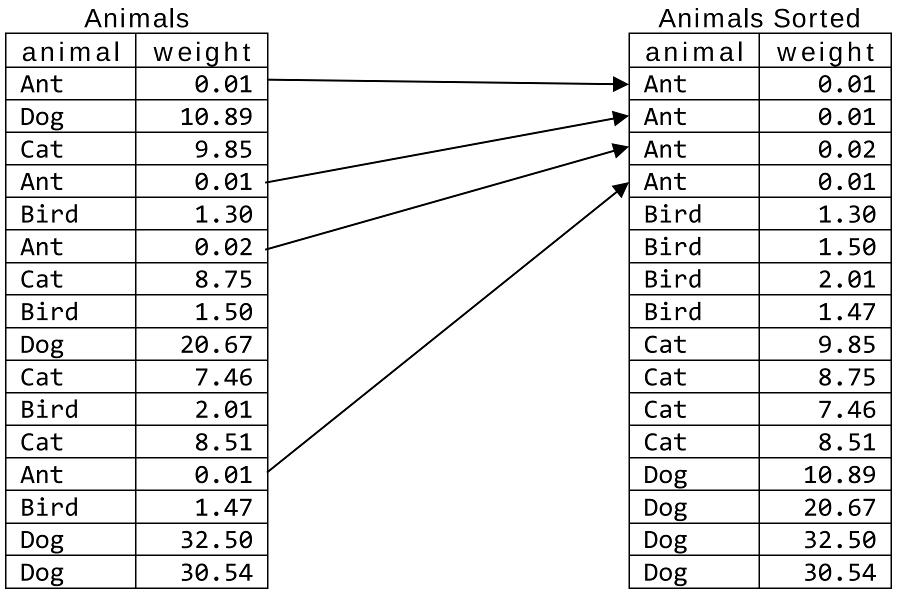
```
      observations and 3 variables.
NOTE: DATA statement used (Total process time):
      real time           2.72 seconds
      cpu time            0.00 seconds
```

Output 2: BY-Group Processing in CAS

The operation in SAS takes a total of 40.94 seconds, whereas in CAS it takes 21.98 seconds. In this case, four machines were used by CAS. Of that time, 19 seconds were spent grouping the data and 3 seconds were spent processing with the DATA step.

ROW ORDER WITHIN BY GROUPS

When using SAS data sets, the original order in which rows were added to the data set is maintained. When using PROC SORT to group data, the rows within a group retain the same order as the original data set. For example, if we were to sort our original animals data set on the variable, ANIMALS, the result would look like this:



Animals

animal	weight
Ant	0.01
Dog	10.89
Cat	9.85
Ant	0.01
Bird	1.30
Ant	0.02
Cat	8.75
Bird	1.50
Dog	20.67
Cat	7.46
Bird	2.01
Cat	8.51
Ant	0.01
Bird	1.47
Dog	32.50
Dog	30.54

Animals Sorted

animal	weight
Ant	0.01
Ant	0.01
Ant	0.02
Ant	0.01
Bird	1.30
Bird	1.50
Bird	2.01
Bird	1.47
Cat	9.85
Cat	8.75
Cat	7.46
Cat	8.51
Dog	10.89
Dog	20.67
Dog	32.50
Dog	30.54

Notice how the rows are grouped by animal and the order of weights is the same as in the original data set. In some applications, the original order of rows can be significant. The original order might indicate an earlier row was observed earlier in time than a later row. For example, in the animals data set, the order might indicate the weight on successive days and the latest weight is the last row of the group. If so, a program to extract the latest weight would look like this:

```
data mysas.latest_weight;
  set mysas.animals;
  by animal;
  if last.weight;
run;
```

Based on our earlier examples, merely changing the libref to our CAS libref causes the DATA step to run in CAS.

```
data mycas.latest_weight;
  set mycas.animals;
  by animal;
  if last.weight;
  run;
```

Let's run both programs and compare the output:

SAS Output

animal	weight
Ant	0.01
Bird	1.47
Cat	8.51
Dog	30.54

CAS Output

animal	weight
Ant	0.01
Bird	1.50
Cat	8.51
Dog	20.67

The results from SAS and CAS do not match. The weights for "Bird" and "Dog" are different. The reason the results do not match is because CAS does not maintain the original row order. When a table is split across multiple machines, rows from a group might be split across multiple machines. To form a group, rows are sent to a single machine. The order in which the rows are added to the group is the order in which they are received, not the original row order. The order in which the rows are received is non-deterministic and can be different each time the program runs.

When we computed the minimum animal weight in the last section, the order of rows was not important. If we want the latest weight, the original row order becomes important. To preserve the original row order in CAS, we must add an additional column that indicates the original row order. This program adds a variable named ROWID that contains an integer to indicate the original order.

```
  data mycas.animals;
    set mysas.animals;
    rowid = _N_;
  run;
```

Note that many data sets already have a column to indicate original row order. This column might be a date observed column. If this is the case, there is no need to add another column.

When using a BY statement in CAS, the rows are grouped on the first BY variable and ordered within the group on all BY variables. Adding ROWID as the last BY variable ensures rows within a group have the original row order and produce the same results in SAS as in CAS. Here is a new CAS program that produces the same results as SAS:

```
  data mycas.latest_weight;
    set mycas.animals;
    by animal rowid;
    if last.weight;
  run;
```

The columns in this table show how rows might be grouped on different machines in CAS. The first column shows the table after loading it into CAS. The rows are arbitrarily divided among machines. The second column indicates how rows are grouped on machines by the variable, ANIMAL. Although the rows are grouped, the order within a group is arbitrary and does not necessarily represent the original row ordering. The third column groups rows on the variable, ANIMAL, and orders within groups on ANIMAL and ROWID. The grouping and ordering maintain the original row ordering from the SAS data set.

CAS Table

Worker-01

animal	weight	rowid
Ant	0.01	1
Bird	1.30	5
Dog	20.67	9
Ant	0.01	13

Worker-02

animal	weight	rowid
Dog	10.89	2
Ant	0.02	6
Cat	7.46	10
Bird	1.47	14

Worker-03

animal	weight	rowid
Cat	9.85	3
Cat	8.75	7
Bird	2.01	11
Dog	32.50	15

Worker-04

animal	weight	rowid
Ant	0.01	4
Bird	1.50	8
Cat	8.51	12
Dog	30.54	16

BY ANIMAL

Worker-01

animal	weight	rowid
Ant	0.01	1
Ant	0.01	13
Ant	0.02	6
Ant	0.01	4

Worker-02

animal	weight	rowid
Bird	1.47	14
Bird	2.01	11
Bird	1.30	5
Bird	1.50	8

Worker-03

animal	weight	rowid
Cat	9.85	3
Cat	8.75	7
Cat	7.46	10
Cat	8.51	12

Worker-04

animal	weight	rowid
Dog	10.89	2
Dog	30.54	16
Dog	32.50	15
Dog	20.67	9

BY ANIMAL ROWID

Worker-01

animal	weight	rowid
Ant	0.01	1
Ant	0.01	4
Ant	0.02	6
Ant	0.01	13

Worker-02

animal	weight	rowid
Bird	1.30	5
Bird	1.50	8
Bird	2.01	11
Bird	1.47	14

Worker-03

animal	weight	rowid
Cat	9.85	3
Cat	8.75	7
Cat	7.46	10
Cat	8.51	12

Worker-04

animal	weight	rowid
Dog	10.89	2
Dog	20.67	9
Dog	32.50	15
Dog	30.54	16

MERGE STATEMENT

The MERGE statement combines rows from two or more data sets in SAS and tables in CAS. When used with a BY statement, rows with the same BY variable values are combined. SAS sequentially reads each table, combining rows, one observation at a time until there are no further observations to process. In CAS, multiple rows are read simultaneously because entire BY groups are assigned to available threads. Each thread processes the BY groups assigned to it sequentially until there are no further BY groups to process. Each thread operating on its BY groups at the same time is one strategy that makes CAS faster than SAS.

Because SAS reads rows sequentially, there is the notion of preserving the original data set order. However, in CAS there is a tradeoff. Speed improvements from parallel processing are gained at the expense of not keeping track of row or observation order. Using a BY statement can make SAS and CAS more similar, but the same differences when using a SET statement with BY exist with the MERGE statement.

MERGE STATEMENT WITH BY

As with the SET statement with BY, you need to sort your data in SAS on the BY variable before using the MERGE statement with BY. In CAS you do not need to sort. To illustrate, let's use a plants data set and an animals data set.

Plants

common	plant
a	Apple
b	Banana
c	Coconut
d	Dewberry
e	Eggplant
f	Fig

Animals

common	animal
a	Ant
b	Bird
c	Cat
d	Dog
e	Eagle
f	Frog

When you run the following code in SAS to merge the two data sets, you must sort on the BY variable, COMMON:

```
proc sort data=mysas.plants; by common; run;
proc sort data=mysas.animals; by common; run;
data mysas.merged;
  merge mysas.plants mysas.animals;
  by common;
run;
```

With data that is stored in CAS, you can only run the DATA step code without sorting.

```
data mycas.merged;
  merge mycas.plants mycas.animals;
  by common;
run;
```

The reason you do not need to sort is CAS implicitly constructs each BY group and assigns groups to threads. Many BY groups can be assigned to a single thread. However, a BY group cannot span threads.

For example, let's modify the preceding code slightly to demonstrate where each row resides by adding a PUT statement:

```
data mycas.merged;
  merge mycas.plants mycas.animals;
  by common;
  put _HOSTNAME_= _THREADID_= common= plant= animal=;
run;
```

The code produces the following output:

```
_HOSTNAME_=worker-06 _THREADID_=127 common=b plant=Banana animal=Bird
_HOSTNAME_=worker-03 _THREADID_=32 common=a plant=Apple animal=Ant
_HOSTNAME_=worker-06 _THREADID_=123 common=f plant=Fig animal=Frog
_HOSTNAME_=worker-03 _THREADID_=28 common=e plant=Eggplant animal=Eagle
_HOSTNAME_=worker-04 _THREADID_=61 common=d plant=Dewberry animal=Dog
_HOSTNAME_=worker-05 _THREADID_=94 common=c plant=Coconut animal=Cat
```

Six threads on four workers were assigned BY groups. Worker-03 and Worker-06 each use two threads. Worker-04 and Worker-05 each use a single thread. The BY group for common=b is assigned to Worker-06, thread ID 127, whereas the BY group for common=f is assigned to thread ID 123 on Worker-06.

When running in SAS, a single thread is reading the rows; when running in CAS, six threads are reading in parallel. This parallelism gives CAS faster run times than SAS when operating on large data.

PERFORMANCE COMPARISON OF SAS AND CAS

When running in SAS, a single thread is reading and writing rows. In CAS, many threads are processing in parallel. To simulate working with large data, we generate larger plants and animals tables, and then merge them. The COMMON variable is the uppercase and lowercase English letters. This means there are 52 groups and we are limited to using 52 threads for processing. Here are the DATA steps to generate the larger tables and to load them into CAS:

```
data mysas.animals;
  length common $ 1 animal $ 18;
  array letters[52] $ 1
    ('A' 'B' 'C' 'D' 'E' 'F' 'G' 'H' 'I' 'J' 'K' 'L' 'M'
     'N' 'O' 'P' 'Q' 'R' 'S' 'T' 'U' 'V' 'W' 'X' 'Y' 'Z'
     'a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j' 'k' 'l' 'm'
     'n' 'o' 'p' 'q' 'r' 's' 't' 'u' 'v' 'w' 'x' 'y' 'z'
    );
  drop letters: i seed;

  seed = 4;
  do i = 1 to 1E8;
    call ranperk(seed, 18, of letters[*]);
    common = letters[1];
    animal = cat(of letters1-letters18);
    output;
  end;
run;

data mysas.plants;
  length common $ 1 plant $ 18;
  array letters[52] $ 1
    ('A' 'B' 'C' 'D' 'E' 'F' 'G' 'H' 'I' 'J' 'K' 'L' 'M'
     'N' 'O' 'P' 'Q' 'R' 'S' 'T' 'U' 'V' 'W' 'X' 'Y' 'Z'
     'a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j' 'k' 'l' 'm'
     'n' 'o' 'p' 'q' 'r' 's' 't' 'u' 'v' 'w' 'x' 'y' 'z'
    );
  drop letters: i seed;

  seed = 5;
  do i = 1 to 1E8;
    call ranperk(seed, 18, of letters[*]);
    common = letters[1];
    plant = cat(of letters1-letters18);
    output;
  end;
run;

proc casutil;
  load data=mysas.animals outcaslib=casuser;
```

```
    load data=mysas.plants outcaslib=casuser;
  quit;
```

Now, let's merge the tables and compare the times for SAS and CAS.

```
45         proc sort data=mysas.animals; by common; run;

NOTE: There were 100000000 observations read from the data set
      MYSAS.ANIMALS.
NOTE: The data set MYSAS.ANIMALS has 100000000 observations and
      2 variables.
NOTE: PROCEDURE SORT used (Total process time):
      real time           32.42 seconds
      cpu time            1:03.62

46         proc sort data=mysas.plants; by common; run;

NOTE: There were 100000000 observations read from the data set
      MYSAS.PLANTS.
NOTE: The data set MYSAS.PLANTS has 100000000 observations and 2 variables.
NOTE: PROCEDURE SORT used (Total process time):
      real time           32.85 seconds
      cpu time            1:04.75

48         data mysas.merged;
49           merge mysas.animals mysas.plants;
50           by common;
51         run;

NOTE: MERGE statement has more than one data set with repeats of BY values.
NOTE: There were 100000000 observations read from the data set
      MYSAS.ANIMALS.
NOTE: There were 100000000 observations read from the data set
      MYSAS.PLANTS.
NOTE: The data set MYSAS.MERGED has 100046113 observations and 3 variables.
NOTE: DATA statement used (Total process time):
      real time           29.16 seconds
      cpu time            23.98 seconds
```

Output 3: Merging in SAS

```
668        proc cas;
669          partition / table={name="animals"
                                groupby="common"
                                orderby="common"}
669      ! casout={name="animals_part" replace=true}; run;
670          partition / table={name="plants"
                                groupby="common"
                                orderby="common"}
                         casout={name="plants_part"
670      ! replace=true}; run;
671        quit;

NOTE: PROCEDURE CAS used (Total process time):
      real time           22.69 seconds
      cpu time            0.00 seconds

673        data mycas.merged;
```

```
674          merge mycas.plants_part mycas.animals_part;
675          by common;
676        run;

NOTE: Running DATA step in Cloud Analytic Services.
NOTE: The DATA step will run in multiple threads.
NOTE: MERGE statement has more than one data set with repeats of BY values.
NOTE: Duplicate messages output by DATA step:
NOTE: MERGE statement has more than one data set with repeats of BY values.
      (occurred 52 times)
NOTE: There were 100000000 observations read from the table PLANTS_PART in
      caslib CASUSER(jaseco).
NOTE: There were 100000000 observations read from the table ANIMALS_PART in
      caslib CASUSER(jaseco).
NOTE: The table merged in caslib CASUSER(jaseco) has 100046113 observations
      and 3 variables.
NOTE: DATA statement used (Total process time):
      real time           4.18 seconds
      cpu time            0.01 seconds
```

Output 4: Merging in CAS

The operation in SAS takes a total of 94.43 seconds, whereas CAS takes 26.87 seconds. The BY variable, COMMON, has 52 values, so 52 threads are used in CAS versus a single thread in SAS. Although there is no PROC SORT needed for CAS, DATA step in CAS has to group rows and assign them to threads.

To show how long the grouping in CAS takes, the partition action is run to permanently group the rows. The grouping takes 22.69 seconds, whereas the sorting in SAS takes 65.27 seconds. The DATA step in CAS takes 4.18 seconds, and in SAS it takes 29.16 seconds.

CONCLUSION

Accelerating your DATA steps in CAS is easy, especially when using BY-group processing. If you have existing DATA step code that runs in SAS, you can simply change the libref from a SAS libref to a CAS libref and take advantage of multiple threads on multiple workers.

There are some cases where you might need to modify your CAS code to help CAS preserve the original data set order. However, this is as straightforward as using the _N_ automatic variable and adding a ROWID column to your original data.

The FIRST. and LAST. variables behave identically in CAS as they do in SAS when using BY-group processing for many types of problems. And when necessary, the use of a rowid coaxes CAS into producing the same results as SAS with accelerated performance.

Whether you are preparing tables for analysis, modifying existing values, computing new values, or combining tables with SET and MERGE, using DATA step in CAS gets you to where you want to be faster.

REFERENCES

Secosky, Jason. 2017. "DATA Step in SAS Viya: Essential New Features." *Proceedings of the SAS Global Forum 2017 Conference.* Cary, NC: SAS Institute Inc. http://support.sas.com/resources/papers/proceedings17/SAS0118-2017.pdf.

Bultman, David and Jason Secosky. 2018. "Parallel Programming with the DATA Step: Next Steps." *Proceedings of the SAS Global Forum 2018 Conference.* Cary, NC: SAS Institute Inc. https://www.sas.com/content/dam/SAS/support/en/sas-global-forum-proceedings/2018/2184-2018.pdf.

ACKNOWLEDGMENTS

Thanks to Melissa Corn for her help in reviewing this paper.

CONTACT INFORMATION

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

Jason Secosky	David Bultman
SAS Institute, Inc.	SAS Institute, Inc.
919-531-3034	919-531-6875
jason.secosky@sas.com	david.bultman@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.