Paper CT02

How Does a Computer *Really* Process Your SAS Code?

Ben Barnaby-Pass, PHASTAR, UK

ABSTRACT

One of the most fascinating aspects of programming is how the machine you are working with takes in and runs your code. SAS is a complex language and is just like learning any other different language, be it Spanish, English or German. But do you really understand any language without looking at the logic behind it?

Learning the logic of a language is the main reason why I believe learning SAS is wonderful! Understanding how a computer can take a very robust language and turn it into SDTMs/ADaMs/TFLs is eye-opening and really opens your mind to how you can manipulate SAS code into being the most efficient code you can produce. Be it drop/keep or a simple IF statement, you can make SAS create the most wonderful things!

INTRODUCTION

SAS has many different statements and many different outputs that the user may desire. However, there is one thing that is a constant throughout the process of creating a program. That is the need for the code to be as optimized as possible. We all know the issue of running an ADLB table that takes minutes to run. You might even have time to make a cup of tea or coffee in the time that you are waiting. But is there a reason why the code is so slow? Well, it's likely down to the fact that the code isn't quite as efficient as hoped. You may have used an IF statement when a WHERE statement could be used instead. Or you may have used a KEEP statement when a DROP statement is much more cost effective. These little things may add up and increase the running time, or you may be tricked into thinking that this is the case.

I want to see the outcome of using these different statements and see truly whether there is a need for different statements at different times. Or we can see if these statements have 0 effect and you can go about your day using statements as you see fit.

TIMING CODE

First, I will need to outline the code that I am using to be able to get the timing of running code. This code is quite simple:

```
options FULLSTIMER;
```

This code essentially helps us see the processing time of the code. As you can see here:

```
NOTE: DATA statement used (Total process time):
      real time           18.42 seconds
      user cpu time       15.73 seconds
      system cpu time     2.67 seconds
```

I have cropped up the screenshot so that we see only the part of the output that we are interested in. This helps us see how long it takes to run a certain amount of code. So we will be able to see if there are any differences in our KEEP/DROP statements and IF/WHERE statements.

To provide context for the given timing later, I would like to break down what each time is doing:

Real time - This is the time spent executing a job or step. This is the time the user experiences waiting for the job/step to complete.

User CPU time - The time spent by the processor to execute user-written code.

System CPU time - The time spent by the processor to execute operating system tasks that support user-written code (all CPU tasks that were not executing user-written code).

These 3 timing conventions give a broad idea of how your code is being read into SAS and this will help us later on when talking about each process.

KEEP/DROP STATEMENTS

Keep and drop statements are quite simple in nature. You can state whether you want to keep or drop certain variables in different ways. I want to see if it truly matters if you use a keep or a drop on a dataset. Overall, there are 4 ways of making a KEEP and a DROP statement. We have the statement on the data line or the set line.

I have created a dummy ADLB table with ~9 million observations to test these statements on. Here are the 4 statements and how they are written in the code:

```
data adlb2(keep = STUDYID -- ACTARMN);
  set adlb;
run;
```

```
data adlb2;
  set adlb(keep = STUDYID -- ACTARMN);
run;
```

```
data adlb2(drop = TRT01P -- GLOBALFL);
  set adlb;
run;
```

```
data adlb2;
  set adlb(drop = TRT01P -- GLOBALFL);
run;
```

Overall, the plan here is to keep the first 20 variables in all of the tables created. The 2 keep statements on the top are essentially the same, just one has the KEEP on the data line and the other has it on the set line. This is the same for the DROP statements as well. If we run these codes seperately we see these times:

Keep (Data)

```
NOTE: DATA statement used (Total process time):
      real time           18.34 seconds
      user cpu time       15.90 seconds
      system cpu time     2.43 seconds
```

Keep (Set)

```
NOTE: DATA statement used (Total process time):
      real time           17.94 seconds
      user cpu time       15.23 seconds
      system cpu time     2.70 seconds
```

Drop (Keep)

```
NOTE: DATA statement used (Total process time):
      real time           18.25 seconds
      user cpu time       15.56 seconds
      system cpu time     2.68 seconds
```

Drop (Set)

```
NOTE: DATA statement used (Total process time):
      real time           18.27 seconds
      user cpu time       15.46 seconds
      system cpu time     2.78 seconds
```

As you can see, overall, there's no real time difference between each KEEP and DROP statement, as well as the fact that all 4 of them run around the same time. After running this multiple times to see the average of the tests, the tests all ran at similar speeds. This means that there's no real processing difference between any statement. You should be able to run any variation of the KEEP/DROP statement without any real increase in processing time.

IF/WHERE STATEMENTS

IF and WHERE statements are a bit more complicated than KEEP/DROP statements as they change the number of observations rather than taking out variables. This means that they have different uses and needs. However, for the sake of this experiment I am going to use the IF statement in the same context as a WHERE statement, so that we have a fair test when running the dummy tables. For instance, here's the code for both:

```
data adlb2;
 set adlb;
 if trtan = 1 then output;
run;
```

```
data adlb2;
 set adlb;
 where trtan = 1;
run;
```

Both codes do the same thing in keeping the values of 1 within the TRTAN variable. Of course, there are more complex uses for both IF/WHERE statements, however, to be able to run through and take observations out this is the code we will use. When we run these through on the dummy ~9 million ADLB that I have created we get these speeds back:

```
NOTE: DATA statement used (Total process time):
      real time           12.80 seconds
      user cpu time       10.43 seconds
      system cpu time     2.39 seconds
```

If statement

```
NOTE: DATA statement used (Total process time):
      real time           11.72 seconds
      user cpu time       10.15 seconds
      system cpu time     1.56 seconds
```

Where statement

As you can see, there's a big 1 second difference in the real time between the IF and WHERE statements. The WHERE statement is more efficient than the IF statement as it tells SAS not to read all observations from the data set. This saves processing time, so that's where the 1 second saving comes from, as there's no need for the where statements to read through all the ~9 million observations to find the variable and then the value within TRTAN to be able to output the observation.

It's better, in this context, to use a WHERE statement. However, the IF statement is more flexible in the processes that it can do. You can manipulate data easily with an IF statement and that's why, I believe as a rule, you should use a WHERE statement for this context only (pulling observations from one table to form another) and use the more flexible IF statement to create new variables and manipulate data.

INDEX STATEMENTS

An INDEX statement is rather on the nose. It searches through a cell to see if the given variable has that value that the user has determined and then gives you a value back on where the string value starts within the cell. This can be used to deduce whether a given string has that value within itself. If the variable doesn't have that string within the cell it will return a 0 value. Overall, this process must run heavily for quite a long time in searching for the given string. As you can see here:

```
NOTE: DATA statement used (Total process time):
      real time           38.80 seconds
      user cpu time       33.23 seconds
      system cpu time     5.54 seconds
```

The string of code must go through all ~9 million observations and ring through each cell to see if a certain value is going to show up. This is a very strenuous search for a computer and takes up a lot of memory as it holds quite a lot of prerequisite data and current data. If you use multiple INDEX statements throughout your code it will largely increase the time of your code running!

CONCLUSION

The conclusions for these processes are quite concise and easy to deduce.

For KEEP/DROP statements there is no overwhelming difference between each to be able to point in a direction to use. Overall, it comes down to taste and context. There's no need to write 60+ variables in a keep statement when you may only be dropping 4 variables. This is more common sense to the user to be able to write the code as needed and how it would be effective for the user's efforts.

However, it couldn't be more different for the IF/WHERE statement. It is clear to me that the WHERE statement is the most time efficient when used within this context. Of course, there are different uses for an IF and WHERE statement where it makes little sense to use the other. This will come down to how you want to manipulate the data and how the user may feel the IF/WHERE statement will be used.

Finally, in my opinion, the INDEX statement. It is not a good process to use this statement as it requires a heavy amount of data to be held by the computer. This leaves quite a big-time gap for the code to run and overall will lead to you sitting and waiting (which is never good). Overall, I would personally use a SCAN statement. However, I think if you are stuck with using an INDEX statement, then I would suggest trying to manipulate the data before by including different keep/drop statements to help make sure there's quite a small amount of data to read in and read out when running the statements.

CONTACT INFORMATION

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

Author Name: Ben Barnaby-Pass
Company: Phastar
Address: WFH, UK
Email: ben.pass@phastar.com