Does Your Application Have a Performance Problem? Put it in a SAS® Pipe and Smoke It! (An Introduction to SAS Pipes)
Rob Russell, The Hartford, Hartford, CT

ABSTRACT

Traditional application design uses SAS WORK datasets to pass data between steps. There is a newer technique that uses a technology called piping. In piping, 2 steps are processing at the same time and data is passed from one step to the other through memory. This technique eliminates IO and can significantly compress the wall clock time that a series of steps takes to run. This paper will provide an introduction to this piping technique.

THE PROBLEM, POTENTIAL SOLUTIONS

Time is a finite resource. If we have 2 applications which, all other things being equal, perform differently – we will consider the application that runs in less time to be better. Our users are generally sensitive to performance of our applications. Our users want answers to questions or data upon which to base decisions sooner, not later. Faster is generally better. This paper will focus on reducing wall clock time. In your SAS log, “Wall Clock Time” is the “Real Time” statistic. “CPU Time” presents the amount of processing time spent. On a multi-processor machine, it is common to see CPU Time greater than Real Time. This simply means that the step was probably one of the SAS procs that can internally multi-thread or parallel process, using more than one of your machine’s CPUs at the same time. For the rest of this paper, I will use the term “Real Time” instead of Wall Clock Time because that is what matches the SAS log.

There are many ways that we can reduce the amount of Real Time required to run an application. We could add hardware resources, reduce the amount of data being processed or introduce other application changes to lower Real Time.

• Adding hardware resources tends to be an expensive option; additionally it frequently is very expensive to add enough hardware resources to effect a significant change in an application’s Real Time. Adding hardware can also affect machine lease agreements which adds significant complexity to this option.
• Reducing the amount of data processed will logically reduce Real Time, but that reduction in data may simply not be feasible. If your users need a particular set of data to answer their question(s), telling them to use less data could garner you a “what are you thinking?” look.
• We are left with looking at our application to find ways to reduce the amount of Real Time required to process data through our application.

The use of SAS Pipes combined with Parallel or Overlapped Processing are some application solutions that may be available to reduce the amount of your application’s Real Time.

TRADITIONAL, SEQUENTIAL APPLICATION DESIGN

Unless your application already takes advantage of parallel processing, your multi-step SAS programs are currently sequential in nature. Sequential means that your program has a series of steps which run, in order, one at a time, passing data from one to another.
An example of such a program might be:

- Step 1 - read data in (from 1 or more files), process it and write out results to a temporary dataset in the SAS workspace as temp_dataset1.
- Step 2 - sort temp_dataset1 and write it out to the SAS workspace as temp_dataset2.
- Step 3 - read temp_dataset2 and process it further, writing it out to the SAS workspace as temp_dataset3.
- Step 4 - sort temp_dataset3 and write it out to the SAS workspace as temp_dataset4.
- Step 5 - read temp_dataset5 and create a permanent output, report, output CSV file, HTML, etc.

Graphically, a sequentially organized program might look like the following, as displayed in Figure 1.

Figure 1
Traditional Sequential Processing Model

A key point to understand is that in sequential processing, there is one instance of SAS running. One could think of that as being a single SAS slave that's available to do work for you. This single instance of SAS processes the data and programs as we traditionally expect - top to bottom, a single step at a time. In this processing model, a SAS step must complete before the next step can even begin. The data is typically stored in temporary (SAS WORK) datasets that are passed between steps.
PARALLEL AND OVERLAPPED PROCESSING

Let's look at 2 ways that we can reduce application Real Time - by introducing parallel and/or overlapped processing into our application. Both parallel and overlapped processing use the SAS MP CONNECT facility to have multiple SAS tasks doing work for us at the same time. I would differentiate between parallel and overlapped processing by looking at the flow of data to and from the child tasks. If the control of the data flow remains with the parent task, I would call that a "Parallel Processing" application. An "Overlapped Processing" application is one where the parent task releases control of the data flow to the child tasks and data flows directly between the child tasks. See Figure 5 for a representation of an overlapped application.

This paper will describe setting up a test of a parallel processing application. Figure 2 graphically lays out the difference between a sequentially organized application and a parallel processing application. Note that there is no mention of SAS Pipes in either diagram.

The parallel processing diagram on the right shows a potential parallel processing implementation that does not use SAS pipes for the transfer of data between the application steps. This type of implementation could be utilized if you were running in an environment where you were not allowed to use TCP/IP ports to establish SAS pipes. Note that this implementation does not use SAS pipes – this is just using MP CONNECT to spawn child sessions to sort the 4 chunks of data. All that is needed to implement this technique is MP CONNECT and enough permanent disk space to create the datasets that are passed from the Main Task to the Child Tasks.

Figure 2.
ADD SAS PIPES

Let us now introduce the point of this paper – the SAS pipe using TCP/IP as a means of transferring data between 2 SAS tasks. Piping is a feature of MP CONNECT that SAS added to MP CONNECT in SAS9. Piping eliminates the use of temporary datasets and can significantly reduce the Real Time requirements of an application.

A key point of pipes is that they move data from one application step to another by record, not by dataset. In other words, when a DATA step or PROC writes a record out through a pipe, that record is immediately available to the application which is reading data from that pipe. Compare and contrast this to a sequentially organized application or the “permanent dataset style of parallel processing”, where a DATA step or PROC must close and release the entire output dataset before the next application step can start reading it.

The easiest way to think of a pipe using TCP/IP ports is as follows. Imagine a stream of data records flowing through a program. One of the fields on these data records is the port number and that port number corresponds to the application that record is destined for. When a record comes in through the TCP/IP ports and has a port number corresponding with one of your spawned SAS tasks, the record is routed to that SAS task. An analogy to this would be a traffic cop directing traffic at an intersection. At this intersection, there are many roads coming in and roads leading out. Data records flow in through the input roads and – based on their port number – flow out the appropriate road to their destination application. The roads are our pipes.

The diagram below shows the next step in our evolving application. Our earlier example used parallel processing to allow the multiple sorts to run concurrently. Now, instead of creating permanent datasets to hold each of those 4 chunks of data to be sorted, we will transfer the data through the SAS pipes to the Sort tasks, and from the Sort tasks directly back to the final Merge.

The only real difference here is that, instead of using datasets to transfer the data – we’re using the SAS pipe technology.

Figure 3.
CODING and TESTING

The actual code used for the tests is attached in the Appendix. The programming example is essentially what SAS has on their website as an example for piping. One important note is that, when using implicit port assignment (where you let the system give you a port vs. hardcoding a port in your program), you must authenticate to the metadata server for every SAS task — including the child/spawned tasks. That is not shown on the SAS website.

I ran my tests on an HP Superdome, which is configured as a 24 processor machine. I ran the tests early in the morning (between 6:00 AM and 6:30 AM) so that there would be minimal other load on the machine. These tests were also ran at the end of the month, when all monthly processing would have been long completed. I did not setup and run a test of the intermediate scenario, where Child tasks are used to sort the data but the data is contained in permanent datasets.

The bottom line of my testing was that the Real Time required to process this test was reduced by 52%. The CPU overhead was only an additional 12%. Figure 4 shows the specifics behind the testing results. In my opinion, cutting our Real Time in half and only paying a 10%-15% CPU penalty seems like a bargain to me.

Figure 4.

<table>
<thead>
<tr>
<th></th>
<th>Traditional Sequential Processing</th>
<th>Parent Task</th>
<th>Child Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Real Start Time</td>
<td>Real End Time</td>
<td>CPU Time</td>
</tr>
<tr>
<td>Split</td>
<td>01:58.57</td>
<td>01:56.61</td>
<td>---</td>
</tr>
<tr>
<td>Sort 1</td>
<td>02:13.72</td>
<td>03:19.85</td>
<td>---</td>
</tr>
<tr>
<td>Sort 2</td>
<td>02:26.08</td>
<td>03:27.87</td>
<td>---</td>
</tr>
<tr>
<td>Sort 3</td>
<td>02:32.60</td>
<td>03:26.64</td>
<td>---</td>
</tr>
<tr>
<td>Sort 4</td>
<td>02:22.58</td>
<td>03:24.42</td>
<td>---</td>
</tr>
<tr>
<td>Merge</td>
<td>04:17.78</td>
<td>03:59.12</td>
<td>---</td>
</tr>
<tr>
<td>Total Job</td>
<td>15:51.33</td>
<td>19:34.51</td>
<td>---</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Parallel Processing with Implicit Pipes</th>
<th>Parent Task</th>
<th>Child Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Real Start Time</td>
<td>Real End Time</td>
<td>CPU Time</td>
</tr>
<tr>
<td>Split</td>
<td>02:42.48</td>
<td>02:07.92</td>
<td>---</td>
</tr>
<tr>
<td>Sort 1</td>
<td>---</td>
<td>---</td>
<td>07:15.76</td>
</tr>
<tr>
<td>Sort 2</td>
<td>---</td>
<td>---</td>
<td>07:15.75</td>
</tr>
<tr>
<td>Sort 3</td>
<td>---</td>
<td>---</td>
<td>07:17.47</td>
</tr>
<tr>
<td>Sort 4</td>
<td>---</td>
<td>---</td>
<td>07:15.70</td>
</tr>
<tr>
<td>Merge</td>
<td>04:33.15</td>
<td>04:09.14</td>
<td>---</td>
</tr>
<tr>
<td>Parent Task</td>
<td>6:25:08.01</td>
<td>6:32:46.86</td>
<td>07:38.84</td>
</tr>
<tr>
<td>Child Tasks</td>
<td>---</td>
<td>---</td>
<td>29:04.68</td>
</tr>
<tr>
<td>Total Job</td>
<td>07:38.84</td>
<td>21:50.82</td>
<td>---</td>
</tr>
</tbody>
</table>

| Difference (Parallel vs. Sequential) | -52% | 12% |
CAVEATS

I have no test data to support the following statements (yet), but I would offer the following as my beliefs about the use of MP CONNECT and SAS Pipes:

- It doesn’t make sense to use these technologies unless you have a lot of data. In other words, the gains by using the technology has to be large enough to outweigh the additional system overhead you will incur.
- You need a machine with multi-processors and adequate memory. If you either of those hardware resources is constrained, I suspect that your overall implementation will suffer badly.
- The use of these technologies definitely adds an additional layer of complexity to the application. Your application should be important enough to warrant the time you will spend implementing these technologies and the increase in long-term programming support time.

FUTURE WORK

Figure 5 shows the “Overlapped” style of application that I expect to test in the future.

Figure 5.

CONCLUSION

SAS Pipes definitely offer a potential solution to reduce Real Time in your application. This paper is really just an introduction to the topic and does not even cover the way I really want to use pipes, which is in an “overlapped” application. SAS Technical Support said that the piping technology does work in the overlapped style that I want to use it. That will be the topic of another, longer paper for a future conference.

REFERENCES

Start with the SAS website. [http://support.sas.com/rnd/scalability/connect/piping.html](http://support.sas.com/rnd/scalability/connect/piping.html). This link will take you to one of the base web pages that will introduce you to Piping technology. From there, you can search and find a number of related web pages which will help as you delve deeper into using MP CONNECT and Piping.

ACKNOWLEDGEMENTS

Thanks to John Mathieu, our Systems Administrator for his help in getting the TCP/IP port assignment to work and chasing down other technical details related to the use of SAS pipes.
DISCLAIMER

The contents of this paper express the work and opinions of the author and do not necessarily represent recommendations or practices of The Hartford. The code is supplied to illustrate possible solutions with SAS and no warranty is implied that its use will fit any particular situation.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. You may contact the author at:

Rob Russell  
The Hartford  
Corporate Actuarial, HO-GL-140  
Hartford Plaza  
Hartford, CT 06115  
Work Phone: (860) 547-4777  
Work Email: Rob.Russell@thehartford.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 registered trademarks or trademarks of their respective companies.
APPENDIX

Pipetest_nopipes.sas

/* This program is a standard, sequentially processed  */
/* set of steps. This is the basis for comparison in our*/
/* testing.                                          */

data test1 test2 test3 test4;
  drop i j;
  length x $ 80;
  x = repeat(" ", 79);
  j = 0;
  do i=1 to 100000000;
    k=ranuni(1);
    if ( j = 0 ) then
      output test1;
    else if ( j = 1 ) then
      output test2;
    else if ( j = 2 ) then
      output test3;
    else output test4;
    j = j + 1;
    if ( j = 4 ) then j = 0;
  end;
run;

proc sort data=test1 out=test1;
  by k;
run;

proc sort data=test2 out=test2;
  by k;
run;

proc sort data=test3 out=test3;
  by k;
run;

proc sort data=test4 out=test4;
  by k;
run;

data bar;
  set test1; by k; output;
  set test2; by k; output;
  set test3; by k; output;
  set test4; by k; output;
run;
Pipetest_implicit_pipes.sas

/* This is basically just the piping example shown on the SAS */
/* website.  Note that the Metadata logon is done in EACH of */
/* the spawned/child sessions.  This is required for the child */
/* sessions to be able to use the Metadata Server, which is a */
/* requirement for using implicit (vs. explicit) port assignment. */

/* set up the global option to cause the spawned sessions */
/* to be started with the same command that was used to */
/* invoke the client/parent SAS session. */

options metaserver=erapprod.thehartford.com /* network name/address of the metadata server. */
  metaport=8561 /* Port Metadata Server is listening on. */
  metauser="sasdemo" /* Domain Qualified Userid */
  metapass="overlaid_in_the_SAS_paper" /* Password for userid above. */
  metaprotocol=bridge /* Protocol for Metadata Server. */
  metarepository=Foundation;

options sascmd="!sascmd";
libname out1 sasesock "pipe1";
libname out2 sasesock "pipe2";
libname out3 sasesock "pipe3";
libname out4 sasesock "pipe4";
libname in1 sasesock "pipe5";
libname in2 sasesock "pipe6";
libname in3 sasesock "pipe7";
libname in4 sasesock "pipe8";
signon p1;
signon p2;
signon p3;
signon p4;
rsubmit p1 wait=no;

options metaserver=erapprod.thehartford.com /* network name/address of the metadata server. */
  metaport=8561 /* Port Metadata Server is listening on. */
  metauser="sasdemo" /* Domain Qualified Userid */
  metapass="overlaid_in_the_SAS_paper" /* Password for userid above. */
  metaprotocol=bridge /* Protocol for Metadata Server. */
  metarepository=Foundation;
libname in1 sasesock "pipe1";
libname out1 sasesock "pipe5";
proc sort data=in1.test out=out1.test;
  by k;
  run;
endrsubmit;
rsubmit p2 wait=no;

options metaserver=erapprod.thehartford.com /* network name/address of the metadata server. */
  metaport=8561 /* Port Metadata Server is listening on. */
  metauser="sasdemo" /* Domain Qualified Userid */
  metapass="overlaid_in_the_SAS_paper" /* Password for userid above. */
  metaprotocol=bridge /* Protocol for Metadata Server. */
  metarepository=Foundation;
metapass="overlaid_in_the_SAS_paper"        /* Password for userid above. */
metaprotocol=bridge                           /* Protocol for Metadata Server. */
metarepository=Foundation;
libname in2 sasesock "pipe2";
libname out2 sasesock "pipe6";
proc sort data=in2.test out=out2.test;
by k;
run;
endrsubmit;
rsubmit p3 wait=no;

options metaserver=erapprod.thehartford.com /* network name/address of the metadata server. */
metaport=8561                                 /* Port Metadata Server is listening on. */
metauser="sasdemo"                            /* Domain Qualified Userid */
metapass="overlaid_in_the_SAS_paper"         /* Password for userid above. */
metaprotocol=bridge                           /* Protocol for Metadata Server. */
metarepository=Foundation;

libname in3 sasesock "pipe3";
libname out3 sasesock "pipe7";
proc sort data=in3.test out=out3.test;
by k;
run;
endrsubmit;
rsubmit p4 wait=no;

options metaserver=erapprod.thehartford.com /* network name/address of the metadata server. */
metaport=8561                                 /* Port Metadata Server is listening on. */
metauser="sasdemo"                            /* Domain Qualified Userid */
metapass="overlaid_in_the_SAS_paper"         /* Password for userid above. */
metaprotocol=bridge                           /* Protocol for Metadata Server. */
metarepository=Foundation;

libname in4 sasesock "pipe4";
libname out4 sasesock "pipe8";
proc sort data=in4.test out=out4.test;
by k;
run;
endrsubmit;

data out1.test out2.test out3.test out4.test;
drop i j;
length x $ 80;
x = repeat(" ", 79);
j = 0;
do i=1 to 100000000;
k=ranuni(1);
if ( j = 0 ) then
        output out1.test;
else if ( j = 1 ) then
        output out2.test;
else if ( j = 2 ) then
        output out3.test;
else if ( j = 3 ) then
        output out4.test;
end;
output out3.test;
else output out4.test;
j = j + 1;
if ( j = 4 ) then j = 0;
end;
run;
data bar;
    set in1.test; by k; output;
    set in2.test; by k; output;
    set in3.test; by k; output;
    set in4.test; by k; output;
run;
waitfor _ALL_ p1 p2 p3 p4;
    signoff p1;
    signoff p2;
    signoff p3;
    signoff p4;