Paper 224-30

# An Animated Guide to Basic Parallel Processing

Russ Lavery, Contractor

## ABSTRACT

SAS® has added many new features to V9.1, features that can dramatically reduce the clock-time-to-solution for jobs that are both CPU bound and/or I/O bound (clock-time-to-solution is how long a programmer has to wait to get his/her answer). Simply put, V9.1 employs a strategy of Divide and Conquer. SAS takes the code that a programmer submits and attempts to split that task into small tasks, tasks that can execute independently. SAS writes mini-programs called threads to execute the small-tasks. Many SAS Procs will implement threading, but to gain any benefit, the hardware must be able to execute multiple tasks independently. A rough recommendation for hardware capable of taking advantage of SAS 9.1 features is: a machine with 4 independent CPUs, a Gig of RAM (or more), an I/O channel for each CPU and no more than two disk drives per I/O channel.

This paper explores several of the features of parallel processing available in SAS. 9.1. It concentrates on recently introduced features, especially features that can be employed on a multi-CPU (in this case only 2 CPUs) home networked computer. It will illustrate these features by working through two small multi-processing projects. Due to time limitations, this paper will not be able to explore the very exciting V9.1 capabilities for simultaneous reads of data. In V9.1 SAS has taken the important features of the SPD Server and coded them into a libname engine (the SPD Engine) that is shipped with every version of Base SAS. Importantly, the SPD engine will support partitioning of a data set, a new index structure and faster I/O.

## INTRODUCTION

SAS has been making progress, on several fronts, in its constant efforts to improve time-to-solution. Major efforts are underway to insure the most up-to-date algorithms are used and this has resulted in powerful new features that can dramatically reduce time to solution – at the expense of increased hardware utilization. SAS now allows users to use multiple CPUs if they are available and provides tools for monitoring and managing multiple CPU machines. SAS has implemented “threading” (having SAS Procs spawn sub-programs that can run small parts of the Proc independently, and which seek out underutilized resources in single CPU single I/O channel machines). Finally, new versions of SAS allow programmers to partition data sets over multiple drives and simultaneously read through multiple data channels.

It should be noted that SAS has developed these features to be used on a multi-CPU server and started development on the product long before Dual CPU chips were on the market. Upgrading servers by replacing Single CPU chips with Dual CPU chips can affect site licenses and fees. Little testing has been done on P.C.s and it is not certain if a Dual CPU desktop will allow SAS to reduce time to solution, or if the housekeeping associated with threading will make a dual CPU machine perform at the same speed as a single CPU machine.

There are some very exciting capabilities in SAS V9.1 that come at the expense of added complexity in SAS code and hardware. Along with the new features in SAS come new words, concepts and commands. The beginning of the paper is a series of short paragraphs about the new concepts, or information that will help link the new material to what typical SAS programmers already know.

The major deliverables of this paper are two heavily annotated programs displaying some of the features, and some of the complexities, of parallel processing. It is hoped that the paragraphs at the start of the article will act as reference materials for understanding the annotations. Unfortunately, there are too many new multiprocessing features for this paper to address.

## THE STRUCTURE OF THE PAPER WILL BE:

1) Different Scenarios for running code
2) Trade offs that you must make to run parallel sessions
3) Machine/Hardware considerations
4) Paragraphs about new words and concepts
5) A list of parallel ready procedures – or procedures that can generate sub-programs (threads)
    CPU threading procs and I/O threading Procs

6)An example of scaling up –multiple sessions executing simultaneously on a multiple CPU machine and also illustrating

Piping
A child session of SAS inheriting a work directory from its parent
A child session writing to a libref and the parent reading from a libref- both librefs pointing to the same directory.

7)An example of scaling out – sending jobs to a remote machine and also illustrating
Passing macro variables to the remote symbol table
Redirecting the output to the local computer with proc printto
Using SYMDEL for its "macro clearing" function on the remote machine
Data transfer Services (DTS) capabilities using Proc Download (no Proc Upload shown)
Data transfer using RLS - making a remote library look local
The session, and work directories persisting after the code stops executing remotely
Sending the log.list output to the a drive on the remote machine
The option sysrputsync=yes
The option cmacvar the monitors the status of the rsubmit
The notify option on signon
Loading of the path to the remote work directory into a macro on the local machine and using it later to access data
The option persist=yes used to keep the work directory in existence after the remote submit stops
The routing of log/list output to drive on the local machine
It illustrates saving a perm file on the remote machine,
Techniques that can be used to access remote data
Some housekeeping techniques to remove tables and librefs

Conclusions, References, Acknowledgments, Contact Information

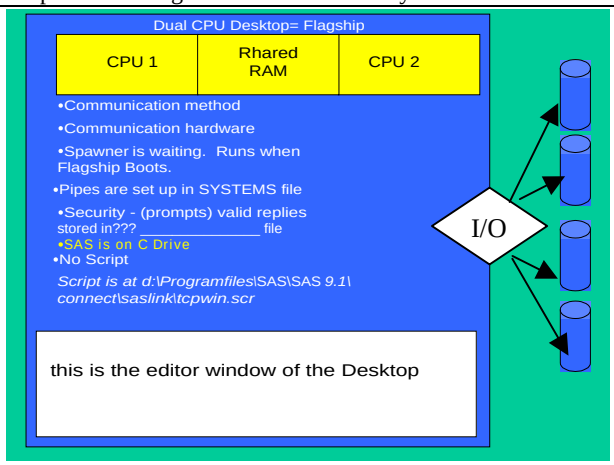
## 1) DIFFERENT SCENARIOS FOR RUNNING CODE

A small network was available for testing these new SAS features. Two machines were on the network: a dual CPU desktop with four disk drives and a single CPU laptop. Pictured below are two ways to run programs on this system.

Example one illustrates a programmer sitting at the dual CPU desktop and submitting SAS commands to that machine. We have long been able to invoke multiple sessions of SAS on a one CPU machine, but this has not helped performance. With two CPUs in a machine it is possible to have two different sessions of SAS running, with fewer conflicts for CPU resources. It is also possible to utilize the two CPUs by having one SAS session create threads that use both CPUs. The issue of making data sets created in one session available to other sessions is a new challenge for SAS programmers and is considered in the paper.

The second example illustrates a more complicated method of running SAS jobs. In example two, instructions are sent from the laptop to a remote machine. There is the potential for controlling job assignment to the CPUs on the remote machine as well as simultaneously continuing to work on the local machine. The issue of making data sets, from one session and machine, available to different sessions on different machines is considered in the paper.

Example 1: Running Dual CPU from its keyboard




Example 2: Logging on to Dual CPU from Laptop

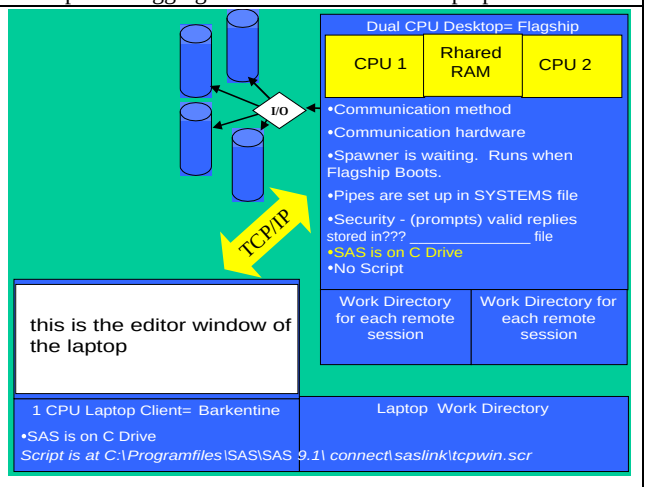



Figure 1

## 2) TRADE OFFS

Taking advantage of these new features, especially if the programmer wants to set up a system at home, involves learning material that is not normally required by people working in a corporate environment. Running parallel processes involves more hardware than is normally required for a home PC. Multiple CPUs computers are required (2 CPUs minimum -four suggested) as are systems with multiple I/O channels. The systems pictured above are not really powerful enough to take advantage of SAS V9.1. The two CPUs will quickly become overburdened with multiple sessions of SAS and the one I/O channel will quickly become a bottleneck.

Setting up a home system requires skills with tools, like SAS Connect, and how they work with the OS for the networked machines. Those skills are usually present in an IT group when in a corporate environment. Nevertheless, programmers can configure home networks so they can learn and practice parallel processing in the comfort of their home.

In a work or home environment, programmers need to learn some new tools for parallel processing. There are commands for starting/stopping/monitoring the new SAS sessions. There are commands to send SAS code to a remote session and to wait for the remote tasks to be done. Finally, there are new commands and concepts for managing libraries. Each SAS session has its own set of librefs and its own work directory. The passing/accessing of files is more complex when multiple sessions and machines are involved. Finally, when the programmer has the system bought, set up, networked and has learned the new commands, s/he must still deal with the issue of network traffic inherent with sending jobs and data out over a network.

## 3) MACHINE/HARDWARE CONSIDERATIONS

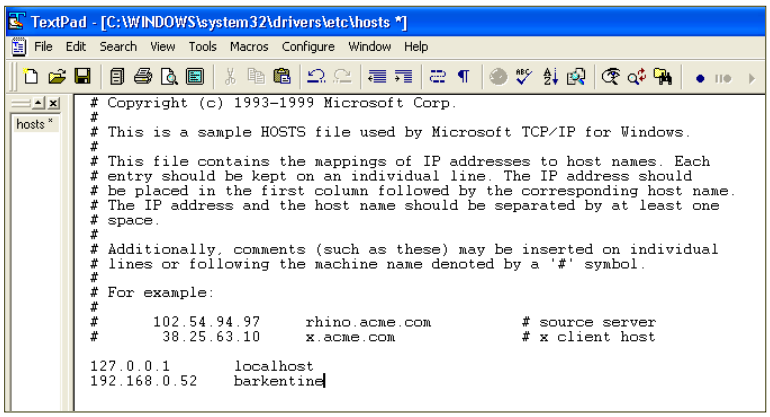
These new SAS tools need new hardware and, sharing of purchasing mistakes, made by the author, might help others. The author was able to get a dual CPU server (almost-obsolete and cheap) board with a pair of 1.1.Gig CPUs. Four 7200 RPM IDE drives were installed in the box. With that setup, the author is disappointed, not with performance, but with the idea that SAS has more capabilities than can be exploited by this box. SAS handles the multiple CPU issue well but seems to want more than two. There are hints that the overhead of threading makes two CPU machines little faster than a one CPU machines. There has been little parallel processing performance testing done on PCs.

If no specific instructions are given, SAS will automatically create threads that use all CPUs. However, SAS has commands to give programmers some level of programming control over CPU use, a tempting feature to any programmer who wants to be master of the machine. However, with only two CPUs it seems best to run one session of SAS and let SAS manage the multiprocessing. The author is currently looking for a semi-obsolete (cheap) 4 CPU motherboard. SAS now has I/O capabilities that can not be exploited by most Business PCs. To take advantage of the new SPD Engine requires multiple I/O channels and this is not common. The author is planning to upgrade I/O hardware but has not made any final decisions.

## 4) NEW DEFINITIONS

Along with the new features in SAS come new complexity, words, commands and environments. Setting up a home network and running SAS over the network requires information about many different computer characteristics and processes. The important characteristics are explored below. SAS connect, a common connection technology while not cutting edge, was selected.

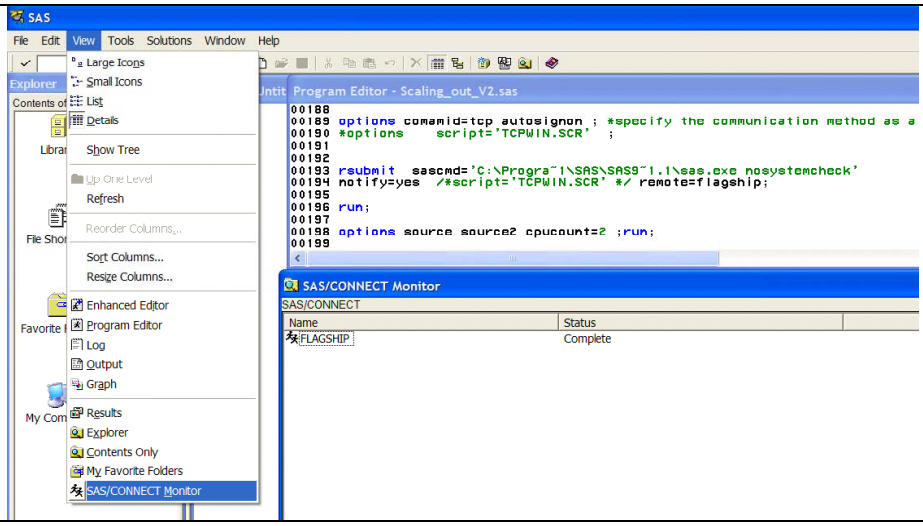
| | |
|---|---|
| %sysRput<br>%sysLput | These commands are used to load macro variables into, and get macros from, the remote session macro symbol table. |
| Amdahl's law | This rule is about decreasing returns to scale. It simply says that jobs on an eight CPUs on a machine will not run eight times as fast as on a one CPU machine. Some CPU resources must be spent managing the job and not doing "useful" work. |
| Asynchrionous processing | Another name for parallel processing or multi-processing. |
| Block Data read | SAS has, in the past, read data observation by observation. This has allowed the implementation of useful tools like point= and the use of indexes. The V9 I/O sub-system reads blocks of data (multiple observations) and delivers more data – faster. |
| Comamid | Communication access method is the method your local session uses to communicate with the remote host. Values are specified with the comamid system option. A typical windows command is : tcp |
| Data Transfer Services | This, like remote library services, is a way to get data from/send data to a remote machine. Data Transfer services has only two commands, Proc Upload and Proc Download, both of which must be inside a rsubmit-endrsubmit block. Upload and download can transfer SAS data sets, catalogs, indexes and have several options to increase their usefulness. |
| Dependent parallelism | This describes when two process can execute simultaneously but are related. The classic example of dependent parallelism (implemented in V9 by piping) is a data read |

| | |
|---|---|
| | followed by a sort. The data step reads the first observation, executes statements on that first observation and sends the observation, not to an output data set, but through a "pipe" to the already running Proc Sort.<br><br>As each observation "clears" the data step, it is piped to Proc Sort. Proc Sort Sorts observations as they come out of the pipe. In Dependent Parallelism, the processes are not truly simultaneous, but the second process is able to start before the first one is finished. If the job logic permits, several steps can be linked by pipes, not just two. |
| Distributed processing | Taking a job, dividing it into smaller logical parts, and running parts on one resource (maybe a P.C.) and parts on another resource. |
| Hybrid Index | The SPDE has a new index that is a combination of a B-tree and a bitmap. If a value in the index is unique, the location of that observation is in the B-tree. If the value of the indexed valuables occurs in multiple observations in the data set, there is one entry in the B-tree and that entry contains a bitmap that can be decoded to the locations of the observations. The base SAS engine does not support hybrid indexes. |
| Independent parallelism | Independent Parallelism is when the tasks to be run in parallel do not interact with each other. The classic example of this is to read to two text files and merge them. The two independent tasks would be: task1= read text file A and sort it and Task2= read text file B and sort it. These tasks can be done on separate machines (or on separate CPUs on one machine) and the tasks do not interact until task3 merge the data sets. |
| Inheriting libraries<br><br>Accessing libraries | There is a complexity to be addressed when remote submitting code to "other" SAS sessions. The output of the remote submit is usually needed as input for further processing- by SAS code in another session. Each session has it's own, very convenient to use, work directory and programmers like to use the work directory to avoid managing permanent data sets. The additional complexity in parallel processing is that a programmer usually needs/wants to have access to the work directories of all sessions. The inheritlib options can be placed on the signon or the rsubmit statement as shown below<br>Rsubmit task1 wait=No inheritlib=(work=locwork);<br><br>Accessing results from different sessions can be done with several methods. If the parent SAS session is on a SMP and the children sessions also run on the same SMP, the inheritlib option can be used to have SAS make all sessions use the same work directory. If the parent and children sessions are on different machines, the full path to the work directories of a child session can be loaded into a macro variable and passed back and forth between the children and parent using the %SYSRput and %SYSLput commands. Once the location is known, the data can be accessed. |
| IP naming conventions<br><br>IP naming conventions (contd.) | Creating a human readable name/alias for the Internet Protocol Address used internally by your system is a good idea. The laptop in this network is called barkentine.<br>This alias is put in the host's domain name table as shown below. (Unix= /etc/hosts Win=c:\WINDOWS\system32\drivers\ect\hosts)<br>  <br> |
| Libname | Issuing the command "Libname _all_ \| Libref List;" will list all (or the specified) librefs to |

| | |
|---|---|
| | the log. DANGER!! just "Libname _all_;" clears all user defined librefs. Making this command a keyboard abbreviation might minimize mistakes. |
| Local machine/ remote machine | There are a variety of different names for the machines in a network. Frequently, SAS users find themselves working at a desktop PC and sending jobs to a larger, faster machine. The machine the programmer is using for typing is the local machine or the client machine. The bigger machine to which the programmer rsubmits jobs is the host machine, the server or the remote machine. |
| LOG= OUTPUT= options on the Rsubmit command | These options allow programmers to route the output from multiple rsubmits to the same files. If the programmer invokes this functionality with two commands (a fileref and an option on the rsubmit). This gives the programmer flexibility in controlling logs and listings (not only locations but options like mod new ). |
| Messaging | You can have MPConnect notify you when the remote submit "hits" an endrsubmit line. This is done be the notify=YES option on the FIRST signon for a remote session. |
| MP Connect | Multi-Process Connect (MPConnect) is new in V9 and enables SAS to run and co-ordinate tasks, or separate SAS sessions, allowing work to be done simultaneously. MP connect allows the programmer to start multiple session of SAS. These sessions can be on the client or the server machine. Running multiple sessions takes resources and is, generally, not useful on one CPU machines. |
| Parallel Processing | SAS defines parallel processing as dividing of an application (a SAS program or a SAS Proc) into subunits of work so that the subunits can be executed at the same time. The execution can occur on the same machine (parallel computing if the machine has multiple CPUs. Parallel I/O if the machine has multiple disk controllers) or on separate machines. Piping (where task2 starts before task1 is finished) is also parallel processing. |
| Partitioned data set | SAS V9.12 supports partitioned data sets. A partitioned data set is a logical data set that has been split up into multiple physical blocks, blocks spread out over multiple disks. The SAS programmer partitions the data sets using the SAS SPD Engine, the SPD Server, or one of the Access engines. The SPD engine is shipped free of extra cost with V9 and above. The Base SAS engine will not partition data sets. |
| Persist<br><br>Persist (contd.) | Options Persist=NO, on an rsubmit command, will cause the remote session to shut down when it is receives the endrsubmit command. When the SAS session shuts down, the work directories are deleted and resources are freed for other tasks. This option interacts with AutoSignon to produce the following states:<br><br>Autosignon=NO Persist=No -> two commands are required to start the remote session and the session shuts down when the remote session sees rsubmit.<br>Autosignon=NO Persist=YES -> two commands are required to start the remote session and the session does not shut down when the remote session sees rsubmit. Remote work directory contents are available for later access if programmer signs on again.<br>Autosignon=YES Persist=No-> Only a rsubmit command is required to start a remote session and the session shuts down when the remote session sees rsubmit.<br>Autosignon= YES Persist= YES Only a rsubmit command is required to start a remote session and the session does not shut down when the remote session sees rsubmit. Remote work directory contents are available for later access if programmer signs on again. |
| Piping | Piping has been available through many operating systems and in earlier versions of SAS. Piping is the redirection of output and will likely be very important in V9 +, especially as multiple CPU machines become the business standard.<br><br>A piping example that can be easily done in windows is:<br>Open the DOS window Programs by clicking:<br>Start-programs-Accessories-Command prompt and a black window will appear.<br>The DOS command **Tree c:\ /F** will send, *to the screen,* a tree listing of the c drive (Control C stops the long : - ( run of output )<br>The command **Tree c:\ /F > C:\temp\C_tree.txt** redirects, or pipes, the output that normally is sent to the screen to the file c:\temp\C_tree.txt.<br>As a definition: piping changes, or re-directs, the destination of output. |

| | |
|---|---|
| | In V9.12 output from one Proc/Data Step can be piped directly to another Proc/Data Step, without the need to write the data to a work file on disk. Thus piping saves disk space and often time. There are limitations to this process that will keep it from being used in all situations. The pipe must connect Procs/Data Steps running in different sessions and so using a pipe requires that the programmer be willing to "pay" for starting separate sessions of SAS. (The piping commands will work on a one CPU machine, but will likely slow the process down.)<br><br>Additionally, the receiving Procs/Data Step must be able to handle data in a "one pass manner". "One pass" means that the observation is processed as it arrives. The basic Proc Print is a one pass Proc. Proc Print, with the Uniform option, is not a one pass Proc and can not be on the receiving end of a pipe. Making all the pages in a Proc Print uniform requires a second pass through the data. |
| Reducing time to solution<br><br>Reducing time to solution (contd.) | Reducing time to solution has been a major thrust of V9 and means reducing the "clock time" that a SAS user waits for her/his solution. The simplistic description of this strategy has been to divide and conquer- to break the big job down into smaller jobs that can, *given appropriate hardware*, run simultaneously. Reducing time to solution using new SAS techniques requires a programmer understand, and balance, use of three new tools:<br><br>1) parallel multi-threaded computation (multiple CPUs)<br>2) multi-threaded I/O (usually requiring multiple I/O channels)<br>3) partitioned I/O (step 2 and using the SPD Engine to partition data sets)<br><br>Fully using the new tools in V9 for multi-processing will require machines more powerful than the typical business PC of 2004. Major reduction in time to solution will require multiple CPUs (2 minimum and 4 are suggested) and multiple I/O (one I/O channel per CPU and not more than two Drives per CPU).<br><br>V9.1 SAS can split both computing tasks and I/O tasks into "smaller independent programs" called threads and have these tasks (threads) execute simultaneously. SAS V9.1 requires that the PC hardware be able to accept, and execute, the small independent programs (threads). In reading a large data set, SAS can create several mini-programs, each of which can read parts of the data set -simultaneously.<br><br>However, if there is only one I/O channel and one disk on the PC, all the threads must compete for access to the one channel to the data. In this case the process will be I/O bound and there will be no time reduction from using SAS V9. I */O* features. Each thread needs to be have "free" access to the resources it needs in order to reduce time-to-solution. Many SAS Procs can change from being I/O bound to CPU bound as the programming task changes. This complicates the process of optimizing code.<br><br>Generally, a programmer will seek to reduce time to solution by:<br>1) using additional CPUs and letting SAS automatically multi-thread the Proc<br>2) using a multi-threaded single partitioned I/O strategy.<br>3) partitioning the data set to improve performance.<br><br>These new speed tools add a new level of complexity to programming SAS jobs and using these tools makes business sense for longer running jobs- but maybe not for the typical ad-hoc query. Employing the new tools takes additional programming time and system resources and the payback might be small for quick running jobs.<br><br>As an illustration of the complexity involved in analyzing jobs for performance improvement, Proc Print reads the data from top-to-bottom and prints in that order. |

| | |
|---|---|
| | Multi-threading the I/O to deliver several observations at the same time will not help Proc Print run faster.<br><br>In contrast, Proc Summary and Proc Report build internal files from the input file and do not need data in any set order. These Procs might benefit from multi-threading I/O, especially if the tasks being performed are simple (like summation).<br><br>If the two Procs are being asked to do complex calculations in their internal files, they might become CPU bound and not benefit from faster I/O. Proc Sort is generally I/O bound but the chance of being I/O bound is a function of the length of the sort key divided by the record length.<br><br>While parallel computing and parallel reading of data are speed tools in SAS V9.12, there is no parallel writing in SAS V9.12. |
| Remote library services | Remote Library Services (RLS) is a general term SAS uses to describe the techniques a programmer can use to access a SAS data library not on the machine into which s/he is typing. RLS enables a programmer to define a libref in the local session that connects to data on a remote machine.<br><br>Using options on the lebref, and a two-part data set name, makes the remote data appear to be part of the local machine. RLS can allow read/write access to remote files, be they SAS, Oracle or DB2 files. RLS is implemented through SAS Connect, or SAS Share, and impacts network traffic. |
| Rsubmit & Endrsubmit | Rusbmit and endrsubmit are commands to start/stop sending SAS code to be run on a SAS session. Since a machine can have multiple sessions of SAS running, and the sessions are usually named, a common rsubmit/endrsubmit option is the name of the session to which the code should be directed. There are very many options for Rsubmit. SAS can be configured to allow a rsubmit to start a session without an explicit Signon command. |
| SAS Connect Monitor | The SAS connect monitor is a window that allows you to monitor SAS sessions. The window, shown below, Illustrates the way to view the session and the look of the window. Status has just a few values, like complete or running. |

| | |
|---|---|
| |  |
| SAS Session | A SAS session is a running “instance” of SAS. Typically, a SAS session is started by clicking through START-Programs-SAS-SASV#. Multiple sessions of SAS can be run simultaneously, (by repeatedly clicking START-Programs-SAS-SASV# ) .Each SAS session takes resources and has it’s own work directory. |
| Scalable and Not Scalable Program Components | While SAS has done great work implementing scaling to make for faster time-to-solution, not all jobs are scalable.<br><br>Sorting was one of the first tasks to be re-programmed and generally scales well. If the data read takes a small percentage of the total time to sort, scaling up by using multiple CPUs will speed up the job. However, if the file is very wide (many variables) and the read time is a high (60%) of the total clock time, sending the actual sort to two CPUs will only affect the 40% of the clock time that is not associated with the data read. If the read time is 60% of the total sort time, adding an I/O channels is in order.<br><br>Calculating a mean with Proc Means is not likely to be CPU bound, because calculating a mean is an easy, and fast, process. Calculating an average with Proc Means will get faster if we scale up by adding I/, not CPUs. Some statistical procedures are likely to be CPU bound. In some statistical procedures processing an observation takes more time to process than to read an observation. |
| Scaling out | Scaling out is sending part of the job to another machine – to make life more complicated you can also scale up on the machine to which you scaled out. |
| Scaling up | Scaling up is adding hardware resources to a machine to allow SAS to perform tasks in parallel. This can be having two or more CPUs on one machine to allow parallel computing or having multiple disk drive controllers to allow simultaneous reading from different disks. The basic SMP machine should have 4 CPUs and lots of I/O. |
| Security in the script | A parameter on the spawner program can allow userid and password to be part of the process of establishing a connection. If you call the spawner with the –protection option you must supply a Userid and password to make a connection. The passwords are stored in the ACI.dat file. A common current idea is that security is provided by the users ability to log onto the local machine and not in the connection to the server. |
| ServerV= | ServV=”connect-server-definition”| __all__ displays the server definitions that are stored in the SAS Metadata Repository. This tells you the options that are set for the connected session.<br>The following example displays the values that can be specified for the CONNECT server |

definition that is named hrmach1.

```
Server=              hrmach1 -- SAS/CONNECT Server
Remote Session ID=   sashost
ServerComponentID=   A5Z3NRQF.AR00005L
Remote Host=         hrmach1.dorg.com
Communication Protocol= TCP
Port=                2267
Scriptpath=          tcpunix.scr
Tbufsize=            4096
Macvar=              hrmach_macvar
Inheritlib=          (work=cwork)
Wait=                No
SignonWait=          No
Status=              No
Notify=              Yes
Netencrypt=          No
```

Service file and pipes

There are two types of pipes implicit and explicit. For an explicit pipe you supply a specific port number or service name, as defined in the service file. For an implicit file, the programmer supplies a pseudo pipe name and the piping engine will select the first available TCP/IP port.

```
*specifies an implicit pipe;
libname out1 sasesock ":pipe1";
```

```
*Explicit pipes with aliases often involve the service file;
*find examples of explicit pipes in the exmples;
```

**Partial contents of a service file**

```
# Copyright (c) 1993-1999 Microsoft Corp.
# This file contains port numbers for well-known services defined by IANA
# Format:
# <service name>  <port number>/<protocol>  [aliases...]   [#<comment>]
echo                7/tcp
echo                7/udp
discard             9/tcp    sink null
discard             9/udp    sink null
……..……..many lines not shown ……..
sasesock          3000/tcp    pipe3000               #this is the port for SAS
piping
sasesock          3001/tcp    pipe3001               #this is the port for SAS
piping
sasesock          3002/tcp    pipe3002               #this is the port for SAS
piping
sasesock          3003/tcp    pipe3003               #this is the port for SAS
piping
sasesock          3004/tcp    pipe3004               #this is the port for SAS
piping
sasesock          3005/tcp    pipe3005               #this is the port for SAS
piping
sasesock          3006/tcp    pipe3006               #this is the port for SAS
piping
sasesock          3007/tcp    pipe3007               #this is the port for SAS
piping
sasesock          3008/tcp    pipe3008               #this is the port for SAS
piping
sasesock          3009/tcp    pipe3009               #this is the port for SAS
piping
sasesock          3010/tcp    pipe3010               #this is the port for SAS
piping
man               9535/tcp                           #Remote Man Server
```

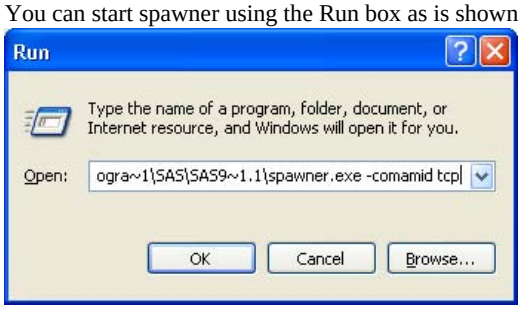
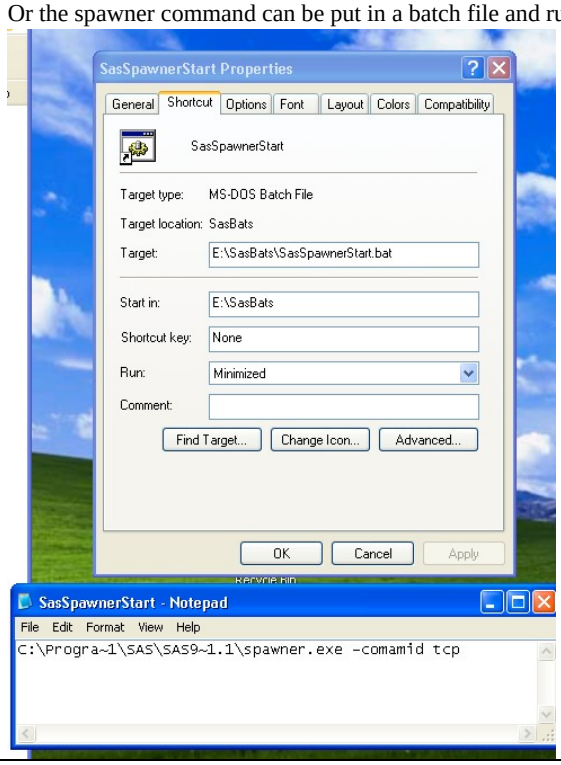
| | |
|---|---|
| Signing on and Signing off the machine | Issuing a Signon command starts a new session of SAS -similar to clicking the SAS icon on your desktop. By issuing appropriate commands, this session can be started on the client machine or the server. Additional sessions of SAS can be assigned unique names and function will independently of each other. Each session can do independent tasks and (by default) has it's own work directory.<br><br>A session is started by the Signon command (accepting many options) and closed with a signoff command. Many rsubmit/endrusbmit commands can be issued to the same named session, with eachrsubmit/endrsubmit having access to files in the session's work directory. The signoff command closes the SAS session and clears the work directories. Running multiple sessions of SAS has overhead costs. Sessions take time to start and resources as they run.<br><br>Issuing the signoff command in SAS shuts down the SAS session on the remote machine. The TCP ability is not affected by this command. The spawner program remains alert, listening for communication, and ready to spawn a new session of SAS when it receives the next signon. |
| Signing on to a remote session | Options AutoSignon=Yes; when run in the local SAS session enables the local SAS session to invoke a new SAS session when a rsubmit is made. If autosignon=No is used, the programmer must issue two commands Signon and Rsubmit. |
| Signon/Signoff tasks/script | You can instruct SAS to perform tasks at signon/signoff. These tasks can be coded into the signon script or into a separate signoff script (that can be specified as an option in the signoff command). If a signoff script is not specified, SAS will check the signon script to see if it contains code to be executed on signoff. A common signoff task is to load a macro variable with a value that indicating the success of the signoff and that can be checked by later program logic. |
| Signon vs autosignon | You can explicitly execute the SIGNON statement to establish a connection between the client session and the server session. A signon entails accessing the machine that the session will run on and then invoking a SAS/CONNECT server session.<br><br>An autosignon is an implicit signon to the server when the client issues a remote submit request for server processing. When the AUTOSIGNON global system option is set, the RSUBMIT command, or statement, automatically executes a SIGNON and uses any SAS/CONNECT system global options in addition to any connection options that are specified with the RSUBMIT. For example, if you specify either the NOCONNECTWAIT global system option or the CONNECTWAIT=NO option in the RSUBMIT command or statement, asynchronous RSUBMITs will be the default for the entire connection. |
| SMP (Symmetric Multi Processor) | This is the term applied to a computer with multiple CPUs. Typically multiple CPUs on one machine are of the same clock speed and share memory (Symmetric multi-processing). When sending jobs to different machines over a network, it is possible that not all machines have the same processing speed (clock, I/O memory access) and the programmer can encounter Non-Symmetric multiprocessing. |
| Spawner | A spawner program (on a host/server) listens for incoming requests to connect the requesting program (SAS) to a program (SAS) on the host/server machine. In our examples, when the spawner receives a signal, it starts a SAS session that will accept rsubmit commands. The SAS spawner program is written by SAS and, in Windows XP, is often in C:\Program Files\SAS\sas9.1\ Often a command to start the spawner is put into a batch file that runs at startup, as is seen below. |
| SPDE | The SPD ENGINE is new in V9.1 and was developed from the SPDS. SAS suggests that users consider the SPD Engine an entry-level scalable product, or "SPDS lite". The SPDE is a libname engine and incorporates the most important features of SPDS. The function of the SPDE is to speed the access of large, partitioned (different physical files) data sets. The SPDE generates separate threads, where each thread has direct access to a data set partition. Reading of the data (subject to I/O hardware limitations) is done in parallel. |

| | |
|---|---|
| SPDE (contd.) | Using SPDE requires more hardware than is on the typical 2004 business PC. SPDE requires multiple I/O channels (multiple disk controllers) and multiple CPUs. The basic machine for Version 9.1 might be a single PC desktop connected to a four CPU, machine with many drives.<br><br>SPDE has several very interesting features. It can do an implicit sort of data as it passes data to an application or procedure. The source data file is not sorted, just the data that is passed to the application/proc. If the SPDE is specified in conjunction with a where clause, the SPDE engine will create multiple threads and apply the where clause to each of the threads/data streams as the data is read. Additionally, while the base SAS engine will only support one index in a where clause (and no indexes using an OR), the SPDE engine will use multiple indexes indices and index use with with an OR in the where clause. An exciting features in the SPDE is a new hybrid index that combines hashing and bitmapping. This hybrid index can be very important to people processing big files.<br><br>Partitioning of the dataset must be specified in SAS, via code, as the data set is created. SPDE requires a hierarchical file system and support for long file names.<br><br>If you care, SPDE supports more than 32,767 variables and more than 2,147,483,647 rows on 32 bit platforms. To take full advantage of this software the hardware should have: Multiple CPUs, Multiple I/O channels, Multiple disks and a large amount of data that is stored in partitions. |
| SPDE data set components | A SPDE data set has more components than a V8 SAS data set. Components are:<br>MDF: This contains a description of the data. This includes variable names and labels, location of the index (which is an external file), and pointers to other partitions.<br>DPF: This is the data section and contains the “rows and columns”. Each partition has its own DTF.<br>HBX: This contains all the unique values for indexed variables and a pointer to where on the disk observations with that value can be found.<br>IDX contains values for the index that repeat and is a bitmapped field with information on how to get the data. |
| SPDS | SAS Scalable Performance Data Server is a multi-user, client-server data server that optimizes storage and speeds processing of large SAS data sets. It has been a stand-alone Windows/Unix product for years. SPDS supports multiple CPUs and multiple I/O channels and runs these resources in parallel. The SPDS has all the capability of SPDE and more. It will apply parallelism to SQL pass through and group by clauses. It supports security and backup features.<br><br>To take full advantage of this software, the hardware should have: multiple CPUs, multiple I/O channels, multiple disks and a large amount of data that is stored in partitions. Unlike the SPDE, SPDS runs in it’s own session. |
| Speeding up I/O | This is done by: first, optimizing the read rate for each partition in the data set and, secondly, allowing multiple partitions to be read simultaneously using multiple I/O controllers and SAS SPDE. The multiple controllers are managed [controllers are controlled ; - ) ] by independent threads. If there is only one I/O controller you will not see the benefits that V9.1 can provide. It does not matter how many I/O threads are launched, if they all queue up and go through one I/O controller. |
| SSA | SSA stands for SAS Scalable Architecture, a set of sub-systems that make parallel processing and partitioned I/O available to the entire SAS system. They do this through accessing multiple CPUs simultaneously, multiple I/O channels simultaneously and allowing the overlap of I/O and data processing via piping. SSA works in tandem with SAS MP Connect, which enables programmers to start/manage multiple sessions of SAS. |

| | |
|---|---|
| Synchronous and Asynchronous SIGNONs | Difference between Synchronous and Asynchronous SIGNONs<br>*A signon is processed either synchronously or asynchronously and the SAS definitions are:*<br><br>**synchronous**<br>Client session control is not regained until after the signon has completed. Synchronous processing is the default processing mode.<br>**asynchronous**<br>Client session control is regained immediately after the client issues the SIGNON statement. Subsequent programs can execute in the client session and in the server sessions while a signon is in progress. |
| TCPWIN.scr is often found in c:\program_files\sas\sas9.1\connect\saslink\ | The script file establishes the remote end of the communication link to the host. It contains parameters for the type of link to be made, and the code is written by SAS,<br>In SAS 9.1 the SAS spawner is often in C:\Program Files\SAS\sas9.1\<br><br>You can define a default fileref for a script in a FILENAME statement. The default script fileref is RLINK. If you specify RLINK as the fileref for your script, you do not need to specify a fileref or a filespec in SIGNON and SIGNOFF commands or statements. When SAS executes a SIGNON or a SIGNOFF command without a specified fileref or a filespec, SAS automatically searches for a file that is defined with RLINK as the fileref. If RLINK has been defined, SAS executes the corresponding script.<br><br>You can insert a FILENAME statement in the SAS autoexec file to automatically start and end a SAS/CONNECT server session. An autoexec file contains SAS statements and commands that you want to execute automatically each time you invoke SAS. Its purpose is to automate the execution of statements, commands, and entire programs that you use routinely in SAS processing. If you use an autoexec file that contains a FILENAME statement to define your script's fileref, you do not have to type and execute the FILENAME statement each time you want to establish a connection. ***Below is a tcpwin.scr (script) file as it was shipped from SAS.***<br>***The use of a script is a great help in reducing typing on sign in and in forcing security practices (ID & Password) . The script below is provided by SAS and, as installedd, does little. The script issues the command that is in blue. It can also be modified to increase security (see green).***<br><br>`/* trace on; */`<br>`/* echo on; */`<br>`/*--------------------------------------------------------------------*/`<br>`/*-- Copyright (C) 1993 by SAS Institute Inc., Cary NC --*/`<br>`/*-- --*/`<br>`/*-- name: tcpwin.scr --*/`<br>`/*-- --*/`<br>`/*-- purpose: SAS/CONNECT SIGNON/SIGNOFF script for connecting --*/`<br>`/*-- to a Windows (either 95 or NT) host via the TCP --*/`<br>`/*-- access method. --*/`<br>`/*-- --*/`<br>`/*-- notes: 1. You must have the spawner program executing on --*/`<br>`/*-- the remote 95 or NT workstation in order for the --*/`<br>`/*-- local session to be able to establish the --*/`<br>`/*-- connection. If the spawner is not running on the --*/`<br>`/*-- remote node, you will receive a message telling --*/`<br>`/*-- you the connection has been refused. --*/`<br>`/*-- --*/`<br>`/*-- 2. You must have specified OPTIONS COMAMID=TCP --*/`<br>`/*-- in the local SAS session before using the signon --*/`<br>`/*-- command. --*/`<br>`/*-- --*/`<br>`/*-- assumes: 1. The command to execute SAS in your remote (W95 or --*/`<br>`/*-- WNT) environment is "sas". If this is incorrect --*/`<br>`/*-- for your site, change the contents of the line --*/`<br>`/*-- that contains: --*/`<br>`/*-- type 'sas ... --*/` |

TCPWIN.scr is often found in c:\program_files\sas\sas9.1\connect\saslink\ (contd.)

```
/*-- support:   SAS Institute staff                                    --*/
/*--------------------------------------------------------------------*/
   log "NOTE: Script file 'tcpwin.scr' entered.";
   if not tcp then goto notcp;
   if signoff then goto signoff;
   /* --------------- TCP SIGNON ----------------------------------*/
   waitfor 'Username:'
         , 'Hello>'                :  ready
         ,  120 seconds            :  noprompt
         ;
   input 'Userid?';
   type LF;
   waitfor 'Password:' , 120 seconds: nolog;
   input nodisplay 'Password?';
   type LF;
   waitfor 'Hello>'
         , 'access denied'         :  nouser
         , 120 seconds             :  timeout
         ;
ready:
   log 'NOTE: Logged onto Windows... Starting remote SAS now.';
   /* noterminal suppressses prompts from remote SAS session.    */
   /* nosyntaxcheck prevents remote side from going into syntax  */
   /* checking mode when a syntax error is encountered.          */
   /* The Win spawner supplies the following options by default: */
   /*    -DMR -COMAMID TCP $SASDMR MSGQUEUE -NOLOGO -NOTERMINAL  */
   type 'sas -device grlink -nosyntaxcheck' LF;
   waitfor 'SESSION ESTABLISHED', 120 seconds : nosas;

   log 'NOTE: SAS/CONNECT conversation established.';
   stop;
/*---------------- TCP SIGNOFF ---------------------------------------*/
signoff:
   log 'NOTE: SAS/CONNECT conversation terminated.';
   stop;
/*--------------- SUBROUTINES ----------------------------------*/

/*--------------- ERROR ROUTINES -------------------------------*/
notcp:
   log 'ERROR: Incorrect communications access method.';
   log 'NOTE: You must set "OPTIONS COMAMID=TCP;" before using this';
   log '      script file.';
   abort;

noprompt:
   log 'ERROR: Did not receive userid prompt.';
   log 'NOTE:  Ensure spawner process is running on remote node.';
   abort;

nolog:
   log 'ERROR: Did not receive password prompt.';
   abort;

nouser:
   log 'ERROR: Unrecognized userid or password.';
   abort;

nosas:
   log 'ERROR: Did not get SAS software startup messages.';
   abort;

timeout:
   log 'ERROR: Timeout waiting for remote session response.';
   abort;
```

<table>
<tr><td></td><td></td></tr>
<tr><td>The spawner can be started manually, if desired using the run box.<br>or the spawner command can be put in a batch file and run at startup.<br><br>For windows to windows communication you should select TPC as your communication method. This means that one of the parameters you pass to the spawner should be comamid tcp.</td><td>First, find the spawner.exe program. On the current install, the spawner is C:\Program Files\SAS\SAS 9.1\spawner.exe. the Windows search facility can be used search to find it. You can start spawner using the Run box as is shown below.<br> 
<br>Or the spawner command can be put in a batch file and run at startup.<br> 
</td></tr>
<tr><td>Thread</td><td>SAS defines threads as:<br>A single path of execution of a process in a single central processing CPU<br>A basic unit of program execution in a thread enabled operating environment<br>An independent flow of control that is scheduled by the operating environment</td></tr>
</table>

**5) THREADED AND PARALLEL READING PROCEDURES IN SAS 9 ARE:**

It will take some time for the whole SAS system to be programmed to automatically multithread. Multithreading is being applied in the areas that have the most benefit to SAS customers. Currently multithreading Procs are shown below.

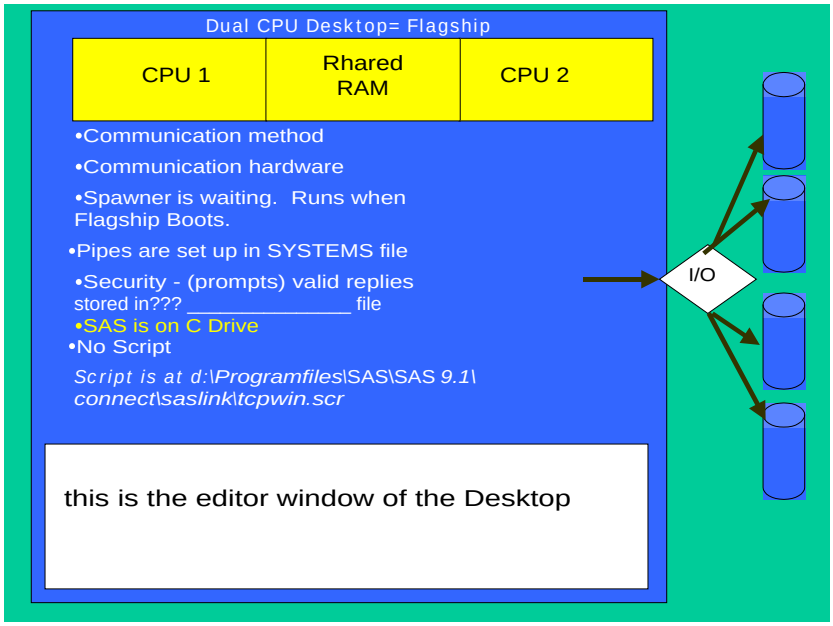
| Threaded Procedures | | | | Procedures support parallel reading with SPD Engine |
|---|---|---|---|---|
| Means/summary | Report | Sort | SQL | Sort Reg DMReg DMMine |
| Tabulate | Reg | GLM | Loess | In V9.1, gains from parallel reading are can also come from NON-SPD engines. Hardware with sufficient I/O bandwidth (eg. striped drives) can perform parallel partition reads. |
| DMreg | DMine | | | |

## 6)EXAMPLE ONE – SCALING_UP (SCALING UP IS USING MORE THAN ONE CPU ON A MACHINE)

**********
**********

Program name =
scaling_up_pipe_Ex_2.sas
Author Russ Lavery

This runs on a Dual CPU Desktop
The desktop has 4 drives
C: SAS is on C -
D:
E:
F:
G: Storage for output




```
/*************************************************************
*************************************************************
*************************************************************
Program name = scaling_up_pipe_Ex_2.sas
Author Russ Lavery



******Overview*************************************************
```

This program is submitted in interactive PC SAS. A "parent" session of SAS is running and giving us an editor window. It is accessing both CPUs on flagship, but with low utilization.

```
****SECTION Preliminary ************************************
This section sets several system options that are then available for all rsubmits.
It allows SAS to use 2 CPUs, sets a path to SAS and reduces error checking with the
nosystemcheck.
It specifies autosignon so that any rsubmit can automatically start a session.
It also uses a data _null_ to identify the path to the work directory for flagship (just for
background).

****SECTION TASK1 ************************************
This section takes advantage of the autosignon in the parent session flagship and sets wait=NO
This section reads the dataset, sashelp.class, and sends it into two pipes/sockets: 3001 and 3002
It creates it's own session of SAS called task1
The output of the pipes will be accesses in two different ways:
    1)The child session inheriting librefs from the parent
    2)through hardcoding of a path to a directory to be used for data storage

Issues:
```

```
 piping and sending output though two pipes/sockets
 the child sesion inhertits the work directory from the parent

****SECTION TASK2 ************************************
This session specifies wait=NO
This section reads from pipe 3001
It inherits the local work directory from it's parent session Flagship
        and writes output to the parent work directory

****SECTION TASK3 ***********************************
This session specifies wait=NO
This section reads from pipe 3002
It writes to a permanent file in G:\gil_out
It shows the listtask command.
****SECTION FOUR  ***********************************
ends the sas sessions and prints files from two directories.
For one file, sorted, we print from the parent work dir- after all the sessions were closed
        - the parent work dir was used by the children as well - It was shared.
For the other file, Gil_out.Name_srt we wrote the output of the children session
        to a perm file to illustrate another method of passing files between sessions
************************************************************
************************************************************
***SECTION PRELIMINARY***********************************/
options cpucount=2 sascmd='C:\Progra~1\SAS\SAS9~1.1\sas.exe
         nosystemcheck' autosignon;
options comamid=tcp remote=flagship ;
RUN;
```

Many connect parameters can be set either as system options or as parameters on an rsubmit statement. Here some system options are set and become available for any rsubmit. Generally, if the same parameter is also specified as a rsubmit parameter, it overrides the system option FOR THAT SESSION. Autosignon is set as a system option.

```
*A session is running.  What is the parent work directory?;
data _null_;
PrntWrk=pathname("work");
put PrntWrk=;
RUN;
```

Sharing files between sessions is a skill to be learned. For learning, Show the parent work directory in the log.

```
****SECTION TASK1 ***********************************;
```

This rsubmit, because of autosignon above, creates another session of SAS (like clicking start-programs-SAS-SAS 9.1) and then sends code to that session. This child session of SAS competes for resources with the parent session.

```
rsubmit task1 wait=no inheritlib=(work=PARwork) sysrputsync=yes ;
```

Send all code between Rsubmit & Endrsubmit to TASK1 session. Wait=NO is an instruction to the parent to send code (from the Rsubmit to the Endrsubmit) off to run remotely - and then immediately start processing the rest of the program (which just happens to be another Rsubmit). Inheritlib lets the Task1 session inherit the work directory location of the parent. Both sessions send work files to the same spot and so work files are easily accessible by both sessions. This session illustrates pipes/sockets, not inheritance and nothing is

```
    libname OutPipe1 sasesock ":pipe3001";
    libname OutPipe2 sasesock ":pipe3002";

    data OutPipe1.flowPARwork
          OutPipe2.flowHC;
          set sashelp.class;
          age_Months=age*12;
     run;

     %sysrput Tsk1Wrk=%sysfunc(pathname(work));
```

Send output, via a standard libref, through two pipes/sockets (pipe3001 and pipe3002) that have been defined in the service file (see the service file description above). This task does not create files in the work directory, saving I/O, space and if you have a multiple CPU machine

Generally, SAS decides where the work directory for Task1 will be. It is a place on the remote system. As illustration, we send the path to Task1's work dir to a macro variable called Tsk1Wrk in the parent session. Later the parent session could use this macro variable in a libref and gain access to the work directory of Task1 – if Task1 has not been "closed".

```
     %put &Tsk1Wrk; *see below;
     RUN;
endrsubmit ;
****SECTION TASK2 ************************************;
```

rsubmit task2 wait=no sysrputsync=yes inheritlib=(work=PARwork);

This rsubmit creates another session of SAS (like clicking start-programs-SAS- SASV9) that takes resources. The session will inherit the work library from the parent. The parent work directory can be accessed in this session by use of the Libref PARwork. This session of SAS competes for resources with the parent session and Task1. It might be advisable to have one CPU for every session of SAS.

```
libname InPipe sasesock ":pipe3001";
```

Information/observations were sent into a Pipe/socket (called pipe3001) in the session Task1. Here we configure a libname to "point to" that pipe so we can take observations out of the pipe - by simply specifying the libname. Pipes/sockets pass observations, but ONLY BETWEEN SESSIONS!

```
Proc Sort data=InPipe.flowPARwork out=PARwork.sorted threads;
by age_Months sex;
run;
```

Sort the data as it comes in from the pipe/socket and send output –through the libref PARwork - to a file called sorted in the parent work directory. Allow threads to speed up processing.

```
listtask _all_;
```

List current tasks. Useful if you do not have the SAS Connect Manager window open

```
%sysrput Tsk2Wrk=%sysfunc(pathname(work));
RUN;
%put &Tsk2Wrk;
run;
endrsubmit ;
```

As an exercise, load the path of the task2 work directory into a macro variable (Tsk2Wrk) on the parent. The %put that follows will fail, because it is looking in the Macro Symbol Table of Task2 and the macro variable Tsk2Wrk is in the parent Symbol Table.

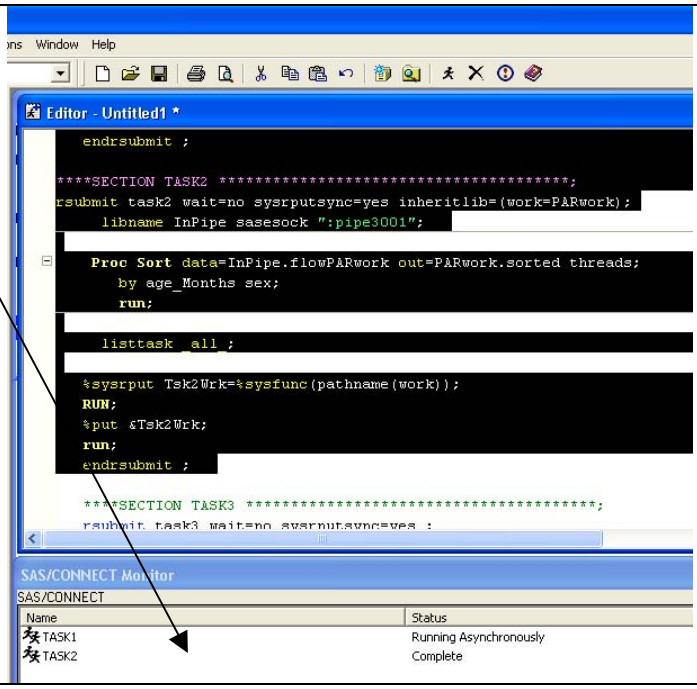
Here is a screen print of the SAS connect manager after highlighting and running the program as far as task2.

An rsubmit just means stop sending code to that session. While the persistence option is set to yes, sessions persist- stay active - until they are manually terminated.

Here you can see that task1 and task2 are in existence and taking up resources.

Since these sessions are still running, their work directories are still in existence and, with good technique, be accessed.

Note that the parent session is not shown. There is a third session of SAS running. That is the session that is providing the editor window in the screenprint to right. It is not shown in the connect monitor.



```
****SECTION TASK3 ************************************;
   rsubmit task3 wait=no sysrputsync=yes ;
```

Because of the system option autosignon, this rsubmit will start a session called Task3 and send code to it. The Wait=NO options means that the parent should send the code between Rsubmit and Endrsubmit to Task3 and immediately start processing additional code from the parent (whatever is after the endrsubmit).

```
     libname InPipe sasesock ":pipe3002";
     libname Gil_out "G:\Gil_out";
```

Information/observations were sent into the pipe/socket (called pipe3002) in Task1. Here we take observations out of the pipe by simply specifying the libname. Pipes/sockets pass obs., but ONLY BETWEEN SESSIONS. Below, the sort sends output to a perm file in a hard coded location. All sessions can, if you hard code the path, access this information.

```
               Proc Sort data=InPipe.flowHC out=Gil_out.Name_srt threads;
               by name sex;
               run;
```

```
     listtask _all_;
     run;
   %sysrput Tsk3Wrk=%sysfunc(pathname(work));
   RUN;
  endrsubmit ;
```

List all running tasks. Other commands like this are: waitfor, rget , rdisplay, killtask

and,
put the path of the Task3 Work directory into a macro variable on the parent

Here is a screen print of the SAS connect manager after highlighting and running the program as far as task3.

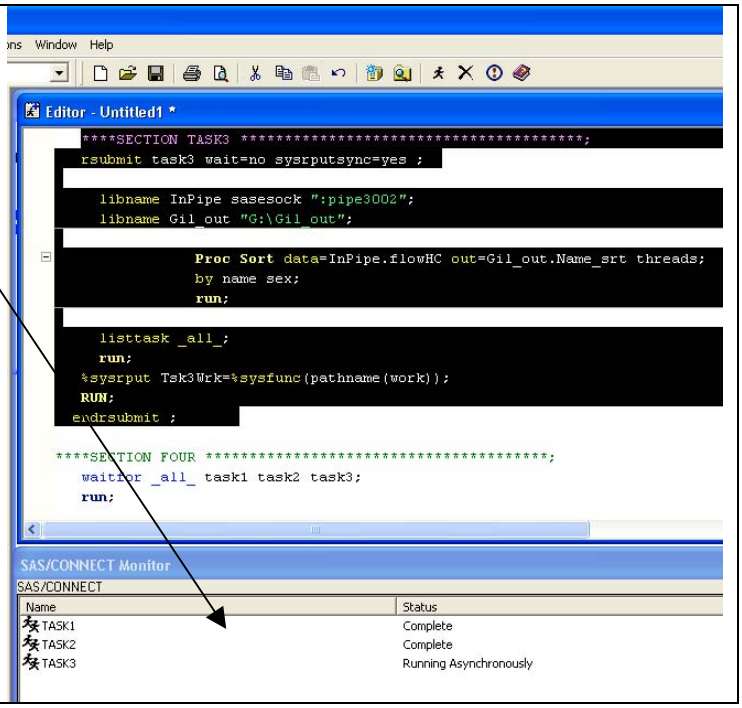
A rsubmit just means stop sending code to that session. While the persistence option is set to yes, sessions tend to persist- stay active - until they are manually terminated.

Here you can see that three tasks are still in existence and taking up resources.

Since these sessions are still running, their work directories are still in existence and, with good technique, be accessed.

Note that the parent session is not shown. There is a fourth session of SAS running. It is the session that is providing the editor window shown in the screenprint to right. It is not shown in the connect monitor.

Seeing this window makes the author wish for a 4 CPU machine so that each session could have its own CPU and not compete for resources.



```
****SECTION FOUR ************************************.
   waitfor _all_ task1 task2 task3;
   run;

*What were the work Directories?;
%put &Tsk1Wrk;
%put &Tsk2Wrk;
%put &Tsk3Wrk;
```

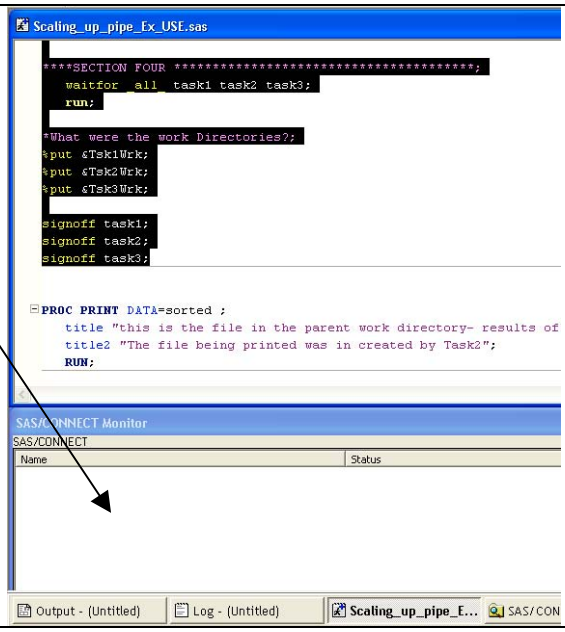
Wait=NO means that SAS sends code to the sessions and immediately starts executing more code from the parent. Here, with the waitfor, we tell the parent to wait till all the sessions (Task1 Task2 Task3) are done.

Show that the parent session knows the work directories of all sessions. These put commands have failed when run on the remote session.

```
signoff task1;
signoff task2;
signoff task3;
```

Signoff the sessions. This shuts down the three SAS sessions and clears their work directories. Librefs and work directories are specific to a session and disappear now.

As can be seen to the right, the signoff commands close the session. Signingoff or "Closing the session" is like issuing File-Exit from the menu, for the remote session. The sessions no longer show in the SAS Connect Monitor Window and their resources are released.




```
PROC PRINT DATA=sorted ;
   title "this is the file in the parent work directory- results of tasks 1 & 2";
   title2 "The file being printed was created by Task2";
   RUN;
```

This prints the file WORK.sorted and we should note that we are printing from the **parent** work directory. When this file was created, we used inheritance to allow the child session have access to a parent work directory.

```
libname Gil_out "G:\Gil_out";
PROC PRINT DATA=Gil_out.Name_srt;
   title "this is from the directory gil_out - results of task 3 ";
   RUN;
```

One session sent files to a hard coded libref – perm files.
That session is "gone", but with hard coding of a libref, the parent (and any session) can get access to that data.

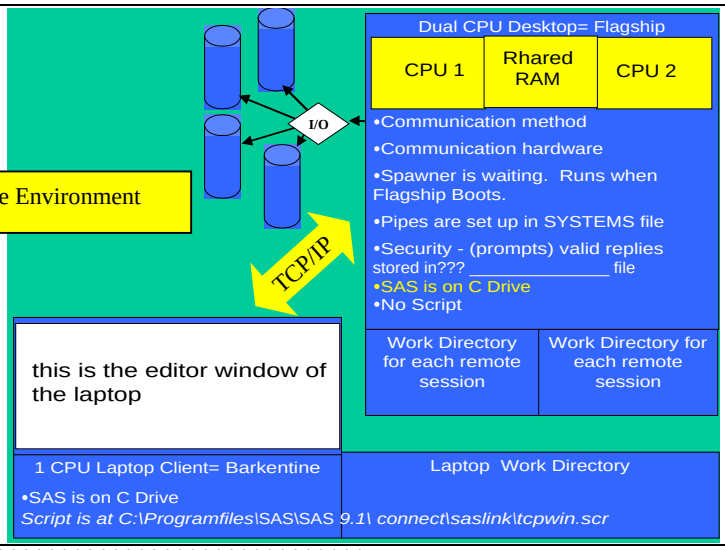
## 7) EXAMPLE TWO - SCALING_OUT (SCALING OUT IS SENDING CODE TO ANOTHER MACHINE)THE PROJECT: FIND ZIP CODES IN PA THAT ARE FARTHER WEST THAN ANY ZIP CODE IN DELWARE

In this example the environment is a single CPU client (a laptop) and a Dual CPU server (a Symmetric MultiProcessor or SMP) connected over a wireless network..

Description of Hardware Environment

This runs from a **Laptop client to a Dual CPU Desktop as a server**

The desktop has 4 drives
- C: SAS is on C
- D:
- E:
- F:
- G: Storage for output -




```
/*********************************************************
*********************************************************
Program name = scaling_out.sas
Author Russ Lavery
****SECTION ONE *************************************
This program illustrates scaling OUT to a remote computer and notify=YES
It does not control the dual CPU features of the remote computer- or Explore Remote Dual CPU
commands.
It does pass macro variables to the remote symbol table
It does show one of the methods of redirecting output to the local computer with proc printto;
It does show the SYMDEL "macro clearing" function on the remote machine
It shows some data transfer Services (DTS) capabilities using proc download (no proc upload)
It shows data transfer using RLS - making a remote library look local;
It shows the session, and work directories persisting after the code stps executing remotely
iT SHOWS THE EXPLICIT SIGNON COMMAND

****SECTION TWO *************************************
This illustrates sending the output to the local drive
        with a proc printo inside the rsubmit-endrsubmit block
It shows the option sysrputsync=yes that instructs SAS to load
        macro variables created with %sysrput to be loaded into the local table as the
        SYSRPUT command executes - and not to wait for an "execution point"
It shows the option cmacvar=task_2_2 loading a 0,1 or 2 into the local macro table
        to indicate the completion status of the rsubmit (0=ok, 1=fail=2=running);
It shows an autosignon from the rsubmit
It shows loading the path to the remote work directory into a macro on the local machine
        and accessing it through remote library services (RLS)
It shows the option persist=yes used to keep the Work directory in existence
        after the rsubmit has finished executing

****SECTION THREE ***********************************

It illustrates saving a perm file on the remote machine to be used later by another session
        and removing it after you are finished, via RLS
It shows the option cmacvar=task_2_3 loading a 0,1 or 2 into the local macro table
        to indicate the completion status of the rsubmit (0=ok, 1=fail=2=running);

****SECTION FOUR  ***********************************
This illustrates techniques that can be used to access remote data
        and some housekeeping techniques to remove tables and librefs;
```

```
*****************PRELIMINARY*****************************/
/*proc sql;
select distinct  state, FIPNAMEL(state) as statename
from sashelp.zipcode;
quit; */
**************SECTION ONE************************************************;
options comamid=tcp remote=flagship;
```

This can be run to see a table of state code vs. state name

Global Options for SAS Connect. Connect with TCP/IP. When you connect, connect to the machine called flagship

```
SIGNON 'TCPWIN.SCR' notify=YES; *notify when ASYNCHH (wait=NO) jobs end;
RUN;
```

Start a new session on flagship- NOW. Notify me, at the client machine, when tasks are done. A Rsubmit command, later in the program, will send code to this remote session. The remote session is running but has no code to execute.

```
%let locthin=10;
%let locwide=42;
%sysLput thin    =&locthin /remote=flagship ;
%sysLput Wide    =&LocWide /remote=flagship ;
%sysLput numofCPU=2        /remote=flagship ;
run;
```

These are state codes that we are going to use later – code executes locally because code is not in a

Create some macro variables locally and put them to the remote session on flagship. This is not new to 9, but is shown in this example because it is useful and not very well known.

```
%put _user_;
```

Put the Client/Parent macro symbol table to the log. Note that we do not see Thin, wide or numofCPU.

```
**printto is outside the Rsubmit-endrsubmit block;
proc printto log="c:\barkentine\laptop.log" new;
run;
```

The position of the Proc Printo, relative to the rsubmit/endrsubmit determines if the files are stored on the remote or local machine. These logs go to the local machine and the laptop must have a dir called c:\barkentine\. If the printo were inside the rsubmit-endrsubmit block (if we were to rsubmit the printo), the output would go to the remote machine.

```
***a silly program.  It uses the zip code file we all have **********;
rsubmit flagship WAIT=YES ; *BECAUSE OF THE YES, YOU DO NOT GET A MESSAGE WHEN JOB
ENDS;
  options threads cpucount=2;
  run;
```

RSUBMIT means send code to remote session Flagship. Some SAS Connect commands/options interact. In this case, setting Wait=YES means the parent waits for a signal that the remote submit has finished before processing any more code. This option interacts with the notify, and the programmer does not get a message saying the remote submit is done –even though the option is Notify=YES.

```
options cpucount=&numofCPU;
  proc options option=cpucount;
  Proc options option=threads;
  run;
```

Use the macro variable we passed up to the remote session to set the option CPUcount and then use Proc Options to show the current settings of a couple of options.

```
  %put _user_;
  run;
```

Put the user defined macro variables on the remote session FLAGSHIP to the log. Look for the variables we passed up.

```
  *more Negative = more WEST!!!; *sort the file from left to east;
  Proc sort data=sashelp.zipcode out=st_&wide;
     by x;
     where state=&wide;
  run;
```

Some logic follows to compare the east-west positions of postoffices in the file SASHELP.zipcode. I admit it is a silly example.

```
  *more Negative = more WEST!!!;*Get eastern & western points of the thin state;
  proc summary data=sashelp.zipcode;
    class state;
    where state=&thin;
    types ();
    output out=minmx&thin
      idgrp(Min(x)   out[1]     (x)=mostwest_&thin)
      idgrp(MAX(x)   out[1]     (x)=MostEast_&thin);
  run ;
```

Some logic follows to compare the east-west positions of postoffices in the file SASHELP.zipcode.

```
data overlap;
     if _n_=1 then set minmx&thin;
     set st_&wide;
     if  x < mostwest_&thin  or x> mosteast_&thin then delete;
  run;
```

Some logic follows to compare the east-west positions of postoffices in the file SASHELP.zipcode.

```
  proc download data =work.overlap out= work.overlap;
  run;
```

Download the result file from flagship to laptop. Download is just one way of getting files from one session /machine to the other. Accessing files created in other sessions is important and we will see several ways to do this task.

```
  %put rem:remove macro variables from remote session;
  %put rem:show the macro vars are there;
  %put _user_;

  %SYMDEL thin  Wide  numofCPU;
  %put rem:show the macro variables are gone from the remote session symbol table;
  %put _user_;
  run;
```

Symdel is a command that removes macro variables from the macro symbol table. Here Symdel is being executed remotely to clean up the symbol table in the remote session. This code Shows the values of the macro variables, removes them and shows they are gone

ENDRSUBMIT means STOP sending code to remote session Flagship. Let the parent session process what

```
endrsubmit;*************************************************************;
```

Below, show some commands associated with getting to files you want but are not in the client/local work

```
**The session on the remote is still running;
**let's look at the work library of the remote session using RLS;
libname WorkOn2 slibref=work  server=flagship;
run;
```

This lets the laptop, via a libref called WorkOn2 see the work directory on the remote session called flagship.

```
libname _all_ list;
/*!!!DANGER libname _all_; clears all librefs */
```

Look at our defined libnames. Look for the remote

```
proc datasets library=WorkOn2;
RUN;
quit;
```

Look at what s in WorkOn2. This library is on a remote machine.

```
*transfer data from one machine to another with RLS - not upload/download;
*the remote work will be deleted when we execute a signoff;
proc copy in=WorkOn2 out=work memtype=data;
   select minmx&locthin;
run;
```

The laptop session can use the libref WorkOn2 to COPY data from the remote session. There are several ways to get/move/access data and this is just one method.

```
*see what we know about the remote library;
*not too interesting- SAS is on the c drive on the remote machine;
proc sql;
select * from sashelp.vlibnam
where upcase(libname) NE "SASHELP";
quit;
```

This shows that Remote library stuff shows up in the SQL dictionaries. Very Useful.

```
signoff FLAGSHIP; **********END SESSION on remote machine;

run;
proc printto; run;
**printto is outside the -signoff block -stop sending log/list to local file send to
local window;
```

Shut down SAS on the remote machine. Clear out work directories and librefs.

The position of the Proc Printo, relative to the rsubmit/endrsubmit determines if the files are stored on the remote or local machine. These logs were going to a local session drive.

```
**************SECTION TWO*****************************************************;
options comamid=tcp autosignon source source2 cpucount=2;
*specify the communication method as a system option;
```

Specify the autosignon option. If system options are set correctly, a rsubmit, and a few options, will start a session.

below is the rsubmit, with the options it needs to start a session and send it code.

```
rsubmit task2 remote=flagship cscript='TCPWIN.SCR' persist=yes wait=no
         sysrputsync=yes cmacvar=task_2_2 ;*****************;
```

persist=yes keeps the session alive after the rsubmit, wait=no tells the parent to NOT wait for a signal from this rsubmit and to continue processing code. sysrputsync=yes is an instruction to execute the sysrput as soon as it is "seen". cmacvar=task_2_2 creates a macro variable in parent session that can be tested to check status of the Rsubmit.

```
proc printto log ="G:\Gil_out\hope.log"
             print="G:\Gil_out\hope.lst" ; *send to file in remote work dir;
   RUN;
```

The position of the Proc Printo, relative to the rsubmit/endrsubmit determines if the files are stored on the remote or local machine. We are rsubmitting this Proc Printto and files will be stored on remote machine.

```
   proc sort data=sashelp.zipcode out = zip_2;*remote work directory;
     by state county poname;
     where state=10;
   run;
```

Zip_2 is in on the server, in the work directory for the session task2

```
   data _null_;
     delay=sleep(5);
   run;

   PROC PRINT DATA=zip_2 (OBS=10);
    title "send this to the remote drive with the proc printto";
   RUN;
```

The Proc Printo, is inside the rsubmit/endrsubmit block.

```
   proc printto;
   run;

   /*retrieve the work library path*/
   %sysrput path2=%sysfunc(pathname(work));
   RUN;
endrsubmit;
```

Put the path to the work directory into a macro variable on the parent session on the laptop. This will be used in section 4

Stop sending code to Session Task2. Task2 still exists. Work files still exist.

```
run;                              ************************************;
%put ;
%put Look at the macro variable task_2_2(it has a value of &task_2_2) to check
status;
%put 0= rsubmit complete 1=rsubmit failed to execute 2=rsubmit in progress;
%PUT PATH2=&PATH2;
%put ;
   *signoff  FLAGSHIP;
*note-  if you highlight the signoff line and submit it with the rest of SECTION 2
                - SAS runs synchronously and you do not get error messages
        if you change wait=no to wait = yes and submit section2,
                you get synchronous running and no error messages;
run;
```

Check the macro variables we sent to the parent. Check completion status rsubmit.

```
**************SECTION THREE  ********************************************;
rsubmit task3 remote=flagship   wait=no   persist=yes cmacvar=task_2_3 ;
```

persist=yes keeps the session alive after the rsubmit, wait=no tells the parent to NOT wait for a signal from this rsubmit and to continue processing code. sysrputsync=yes is an instruction to execute the sysrput as soon as it is "seen". cmacvar=task_2_3 creates a macro variable in parent session that can be tested to check status of the Rsubmit.

The directory g:\Gil_out only exists on the server/remote machine

```
    libname Gil_out "G:\Gil_out";
    run;

    proc sort data=sashelp.zipcode out =Gil_out.zip_3;
      by state county poname;
    where state=2;
    run;
    data _null_;
      delay=sleep(5);
    run;

    Proc Print Data=Gil_out.zip_3  (obs=10);
    title "send this to the local drive in the rsubmit";
    run;
```

Send output to a perm data set on the server. This does not go away when session ends.

```
endrsubmit;
run;
```

Stop sending code to remote

```
waitfor _all_ Flagship;
```

This tells the parent session to wait, at this point in the program, for Flagship to finish running all sessions and send the parent a "completion signal".

```
%put Rem:--;
%put Rem:Look at the macro variable task_2_3(it has a value of &task_2_3) to check
status;
%put Rem:0= rsubmit complete 1=rsubmit failed to execute 2=rsubmit in progress;
%put Rem:--;
run;

%put __user__;
```

Check the macro variables we sent to the parent. Check completion status of

```
**************SECTION FOUR*********************************************;
%put paths on the server are:

%put REmPath2 is set to &path2;
%put The other path is hard coded;
run;
```

The rest of the program runs in locally. We want access to the work we did remotely.
Libraries are tricky with multiple sessions. Look at some SAS features for managing libraries and accessing data.

```
*these paths are on the server and we can not use a regular libname statement;

libname RemPath2 "&path2"     server=flagship ;
libname RGIL     "G:\Gil_out" server=flagship ;
run;
```

Set remote librefs. Librefs that the parent/laptop can use to see files on the remote/server
&path2 contains the path to the work library of Task2. remote/server.

```
proc datasets lib=rempath2;
proc datasets lib=RGIL;
quit;
```

Use the librefs in a proc datasets – Cool stuff.

```
data both;
set RemPath2.zip_1 RGIL.zip_2;
run;
```

Use the librefs in a datastep – More Cool stuff.

```
proc sql;
 drop table RGIL.zip_2;
 quit;
```

Use the librefs in a Proc SQL to delete files on the server.

```
libname RemPath2; /*clears the libref */
run;
```

Clear the libref

```
signoff  flagship;
run;
```

Signoff- shut down the remote session of SAS and free up resources

**CONCLUSION**

Parallel processing is an exciting new feature is SAS that can crate huge reductions in time-to-solution. Getting a reduction in time to solution takes additional hardware resources and makes the code more complex.

**REFERENCES**

Olsen and Ray (2001), "V9: Scaling the Future " Proceedings of the Twenty-Sixth Annual SAS Users Group International Conference", 26 , paper 137

SAS course notes: Distributed and Parallel Processing with the SAS System Book Code 59388

## CONTACT INFORMATION

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

Russell Lavery
9 Station Ave. Apt 1,
Ardmore, PA 19003,
610-645-0735 # 3
Email: russ.lavery@verizon.net