Paper 116-2012

Using SAS® to Move Data between Servers

John E. Bentley, Wells Fargo Bank, Charlotte NC, USA

ABSTRACT

In today's data management environments, it is not unusual for UNIX servers to be dedicated to a particular department, purpose, or database. As a result, a data analyst using SAS often works with multiple servers, each with its own data storage environment. Adding complexity, for security reasons our IT partners often do not make it easy for the servers to talk to one another. How do we transfer data between servers? SAS provides several solutions. This paper shows ways to use SAS Data Transfer Services and Remote Library Services combined with Remote Compute Services to move data between servers. Code samples are provided. All levels of SAS programmers and SAS® Enterprise Guide® users working in multiple server environments should find the paper useful.

INTRODUCTION

Let's define a situation that's common in financial institutions and probably most other places. We're running SAS on a UNIX analytical server but our data is on a different UNIX server. This describes a simple *distributed computing environment*. There can be variations on our situation but a common theme is that the analytical software is on one server and the data is on another. Other complications might be present, such as

- SAS may not be on both servers;
- If SAS is on both servers, the version or product stack might be different;
- The servers might have different levels of security; and
- SAS/CONNECT® may not be available on one or either servers.

Now here's the problem we have to solve: We need a programmatic solution (no manual intervention) that will let a SAS program on the analytical server use the data on the data computer. Maybe that sounds familiar? As usual with SAS, there are multiple ways to solve the problem. Some are more efficient than others in terms of resource usage and clock time, some are more robust than others, but all are probably appropriate for one situation or another.

Although our code examples in this paper focus on transferring SAS data sets from one server to another, the same techniques can easily be extended to flat files and external databases. There are a few items to note about the examples.

1. SAS code is about 98% platform-agnostic. The same program runs on Windows, Z/OS, and all the flavors of UNIX. Only operating system-specific items must be changed. The code examples here were developed with SAS 9.1.3 running under Win/XP and AIX and should work anywhere with at most minor tweaking
2. I'm using this configuration: Working on a WinXP desktop and using the SAS Editor. Using SAS/CONNECT to execute code on an AIX SAS analytical server. Data is on another AIX server. Base SAS and SAS/CONNECT are on both servers.
3. Solution 4 addresses situations where SAS/CONNECT isn't on the data server.
4. The code examples don't use macro variables but I strongly encourage you to think about using them for things like file and data set names and directory paths so that your code is flexible.

But first we will begin with some background on how SAS handles distributed computing environments. From there we will explore specific solutions and get into what everyone really wants… code!

WORKING IN A DISTRIBUTED COMPUTING ENVIRONMENT

As you might expect, a distributed computing environment calls for a distributed solution. That, is, our program will have to span servers. Depending on the solution we choose, it might execute code on one or both servers, physically move the data from the remote to the local host, or just read the data from the remote host. SAS provides the capabilities or *services* that allow us to do these things—remote data services and remote compute services.

Cheryl Garner and Stephanie Reinard long ago presented a paper titled "The SAS System: A Complete Client/Server Solution" that is so exactly what you need to know that I'm going to just condense and summarize some of the material they present. It is important to understand at least the basics of client/server because the implications and side effects of choosing one approach over another can be significant especially when working with big data. For

more information about working in multiple server environments, the Garner-Reinard paper is great starting point.

SAS/CONNECT

SAS/CONNECT has been around since Release 6.04 and enables SAS programs to run in a distributed computing environment comprised of two or more computers. Basically, it provides the middleware that allows us to leverage our computing resources by distributing processing to the most appropriate machine. This results in expanding our data access capabilities. For it to work though, SAS/CONNECT must be installed on both machines. A 'local' SAS session runs on the computer that initializes or starts the connection—this is the *local host*. SAS/CONNECT on the local host establishes a connection to a *remote host* and then starts a 'remote' SAS session there. Bingo! We've established a client/server environment.

But which host is the client and which is the server? A server serves a client, so the client is one that originates the connection. SAS documentation though prefers to identify the computers as local and remote because it's possible to build a chain of client/server environments or have a fan configuration of one client with multiple servers and it can get messy using client and server in those situations. In this paper I use the phrases 'analytical server' and 'data server' for local and remote. These are, respectively, the data target and the data source.

SAS/CONNECT allows a wide range of problem-solving approaches but it isn't mandatory for solving our particular problem. It provides the services introduced below that if used together can provide a flexible, robust solution for most situations. For those who are already familiar with using SAS/CONNECT in a desktop-UNIX server configuration, it can easily be extended to work the same way in a server to server client-server configuration.

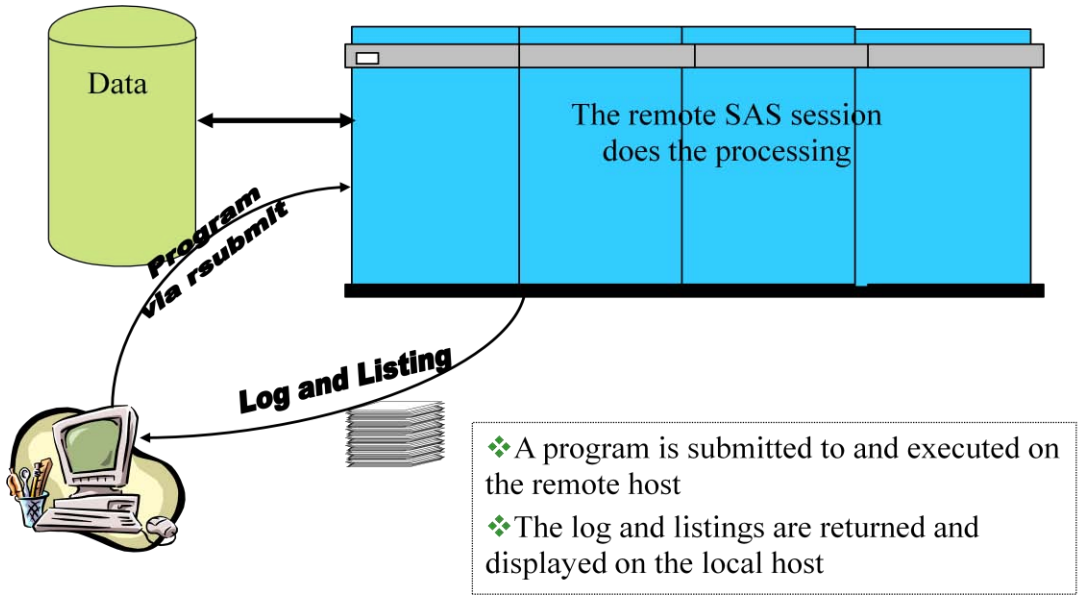
The primary advantages of SAS/CONNECT is that it provides remote data access and remote processing capabilities. With remote processing, a SAS session on the local host can execute code on the remote host via the RSUBMIT and ENDRSUBMIT commands. All code between RSUBMIT and ENDRSUBMIT executes on the remote host. Unless the remote log and listing are explicitly redirected to stay on the remote host, they return to the local host.

This paper will not provide details on SAS/CONNECT itself because there are quite a few good SUGI papers on the topic. Michael Sadof's SUGI 30 paper is a recent one that's very good.

REMOTE COMPUTE SERVICES

Remote Compute Services (RCS) allow us to effectively utilize all the computing resources available, not just those on the local desktop or server we're working from. RCS allows us to move any part of our processing to a different computer with the goal of running the code where it makes the most sense. When we use SAS/CONNECT to start a remote session we gain access to the remote CPUs as well as the directories on the remote platform.

In Display 1 below, our environment consists of a desktop, a server, and the data storage (DASD, storage area network, or network attached storage) attached to the server. We write our program on the desktop using the SAS Editor and save it to our C: drive. But we execute that program on the server because that's where the data is and maybe we need the server's computing power. The program's log and listing are returned to the desktop editor's Log and Output windows from where they can be reviewed and saved to our C: drive.



Display 1. Remote Compute Services Between a Desktop and a Server

The advantages are clear—

- Access to data not otherwise accessible to the analytical server unless we move it there.
- Ability to use more powerful server resources for faster execution.
- Compliance with corporate IT data security standards and policies.
- The SAS Editor remains available for coding as does the local printer, hard drives, and LAN.

But there are also a few possible downsides to be aware of—

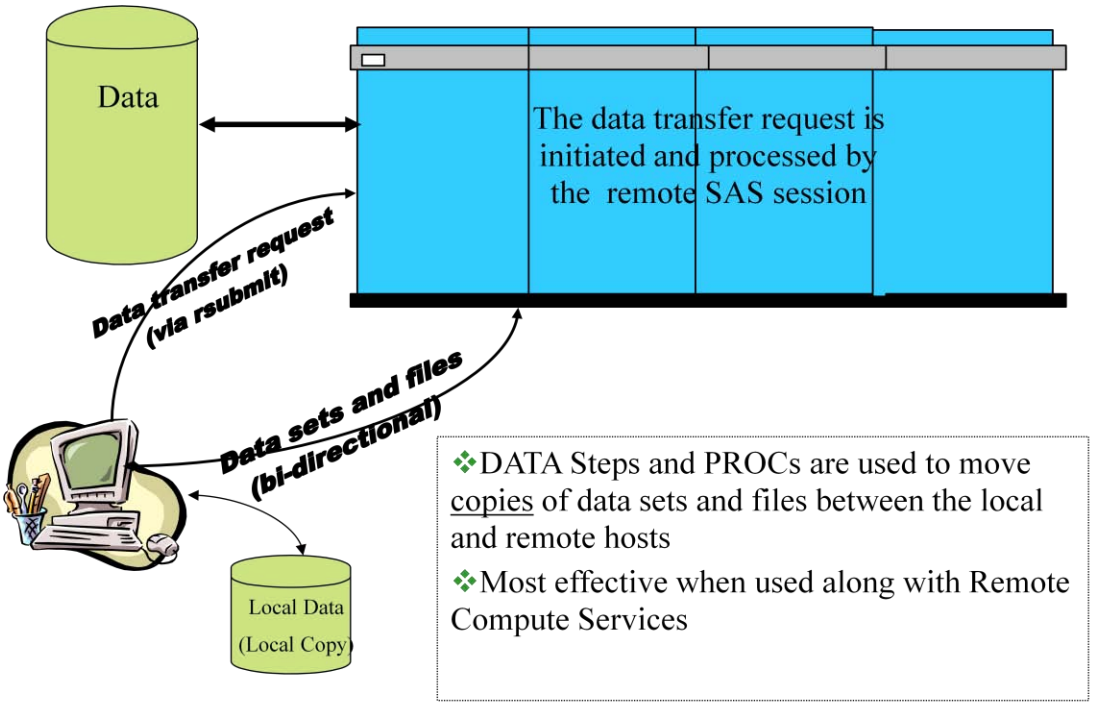
- Code development, testing, and debugging might be an issue if you work on a heavily-used server or are charged for the CPU cycles you use.
- If you work on a production server then production jobs could be impacted by your development and testing.
- Network traffic is increased a bit, but not much.

REMOTE DATA SERVICES

Remote Data Services let us access data stored on a machine other than the one we're executing our program on. The data can be SAS data sets, external files, or, if used in conjunction with RCS, an RDBMS such as Oracle. There are two types of remote data services: Data Transfer Services and Remote Library Services.

DATA TRANSFER SERVICES

Data Transfer Services (DTS) provide the capability of physically transferring a copy (usually) of the data from where it resides to another machine. The graphic below shows that we submit a data transfer request to the remote host and a copy of the data is moved to the local host. We can either send or retrieve data using appropriate commands such as Proc Upload and Proc Download. DTS is most effective when RCS first tailors the data set to contain only the elements you need. As much as possible, make the data set as small as possible.



Display 2. Data Transfer Services Between a Desktop and a Server

The advantages of DTS include—

- The ability to offload data storage from a heavily used system to a one with available capacity. DTS uses minimal CPU cycles on the data source machine.
- After copying the data, application development can continue without impacting the remote system.
- Automated jobs can be set up to perform backups or archiving onto different machines, gather data from multiple sources, or pre-position data for tasks like data scrubbing or data quality control.

But because there are no free lunches there can be a few disadvantages—

- DTS creates multiple copies of the data, so if we're performing updates then the data can get out of sync.
- Data security policies may not allow multiple copies of the same data file.
- Network performance will be degraded when the volume of data being copied is large. 'Large' of course is site-specific.
- The time it takes to transfer the data may be unacceptable.

Here are a couple tips and clues.

- Move only data that is really needed. This is the #1 rule.
  - The Data step WHERE= option dynamically subsets the data as it is being transferred.
  - The Data step KEEP= or DROP= options retains only variables that are needed.
  - Use Remote Compute Services to create a data set containing only what you need.
- Don't move large amounts of raw data unless the processing time on the target is reduced by more than the transfer time.

REMOTE LIBRARY SERVICES

Remote Library Services (RLS) is the second capability provided by Remote Data Services. DTS allows us to copy entire data files from one machine to another but RLS provides read, write, and update access to the remote data just as if it were in a local library, even if the remote host's architecture is different (AIX vs. HP-UX vs. WinXP). RLS supports SAS data sets and views, SAS MDDB databases, and external databases such as Teradata and Oracle.

RLS starts a *remote engine* in the local SAS session that works with librefs defined in the remote SAS session. This allows local SAS procedures and Data steps to access data sets in the remote session just as if the data were local. But what if our data is an external file? In that case we'd first use Remote Compute Services to read it into a data set so we can use a libref and RLS to access it.

A LIBNAME statement gives the local host access to remote data via RLS. After connecting via SAS/CONNECT to the remote host, simply issue a standard LIBNAME statement pointing to a directory on the remote host but include the SERVER= option so that the libref points to the remote host.

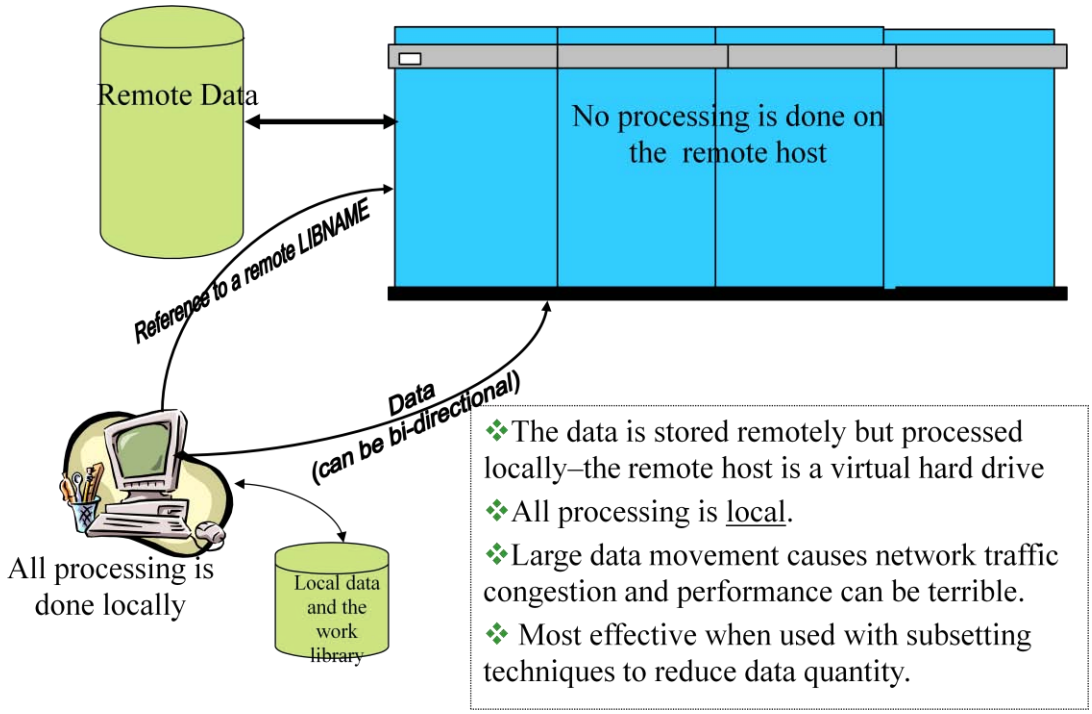
```
SIGNON <session-id>;
LIBNAME <libref> <'remote-directory-path'> SERVER=<session-id>;
```

It is important to recognize that this libref does not exist on the remote host. Only the local host can use it. If we want a libref that exists on the remote host to be available to the local host then use this next approach.

```
SIGNON <session-id>;
RSUBMIT <session-id>;
   LIBNAME <libref> <access-engine> <'directory-path'>;
ENDRSUBMIT;
LIBNAME <remote-libref> REMOTE SERVER=<session-id>;
```

Diagram 3 illustrates using Remote Library Services. A local libref assigned to a directory on the remote host so that the remote host's data is available to the local host. A Data step for example then passes record requests to the "remote" libref and returns the data—the physical location of the data is transparent to the user.

.



Display 3. Using Remote Library Services with a Desktop and a Server

The advantages of Remote Library Services can be substantial—

- RLS is transparent to GUI applications. To the application, all data appears to be local.
- Cross-Environment Data Access (CEDA) activates automatically when reading data sets and views across different operating systems.
- Only individual records are transferred from the remote data source to the local processor so network traffic can be minimized.
- A single copy of the data can be shared without violating data security policies or being concerned that the data will get out of sync. With SAS/SHARE®, users can simultaneously update the data while maintaining data integrity.

The disadvantages can also be substantial. Careful thought must be given to the amount of data being transferred because network traffic can be easily impacted.

- All processing is local so RLS can be problematic if used to move big data. An improperly constructed Data step using a subsetting IF instead of a SET statement WHERE= option can impact the network if the data set is large.
- Multiple passes through a large data set should be avoided. If a large data set is used more than once then the data set should be copied to the local host.

COMBINING REMOTE COMPUTE AND DATA TRANSFER SERVICES

The disadvantages of DTS hinge on the amount of data being transferred or accessed. The solution to this is to combine Remote Compute and Data Transfer Services for an optimal solution that, as Garner and Reinard put it, "provides tremendous flexibility and efficiency to any distributed application".

Reasons for not moving an entire data set include (a) the volume of data are too large, (b) the data is frequently updated, and (c) data duplication is not allowed. Regardless of the reason, you should *always think about using Remote Compute Services for preprocessing* on the remote host before invoking Data Transfer Services. Often simply using SET options to reduce the amount of data being moved by DTS is an excellent solution. The rule is to always minimize the amount of data being moved. Your Network Administrators will appreciate it.

AND NOW THE SOLUTIONS!

Each of our examples assumes that we are working on a desktop and using SAS/CONNECT to execute code on a SAS analytical server via RSUBMIT-ENDRSUBMIT. The data we need is not on the analytical server so we'll call it the target server because that's where we move the data to. The server where the data resides is the source server. Base SAS and SAS/CONNECT are on the both servers. For the first examples we'll assume that we need the entire data set on the target server and then increase complexity of the solutions.

SOLUTION 1: USING THE DESKTOP AS A MIDDLEMAN FOR DATA TRANSFER SERVICES

For this approach we must signon from the desktop to start SAS sessions on both the source and target servers. We assign a libref on each server and then use Procs Download and Upload to move the data set. This is the least efficient solution and is probably unworkable for large data sets because we're moving the data twice through the network and landing it on a comparatively slow desktop hard drive. And then we have three copies of the data—the original on the source server, the desktop copy, and the copy on the target server.

```
** We're already connected to the source data server with session-id=source.  ;
rsubmit source;
    ** Assign a libref. ;
    libname source '/marketing/phone_calling/southeast';

** Now download a copy of the data set to the desktop work library. ;
** Procs Download and Upload must always execute on the remote host. ;
proc download inlib=source outlib=work;
    select new_client_leads;
run;
endrsubmit;

** We're already connected to target, the SAS analytical server.     ;
** Now upload a copy the data set to the target server. ;
rsubmit target;
libname target '/marketing/temp_data';

proc upload inlib=work outlib=target;
    select new_client_leads;
run;
endrsubmit;

** The desktop is still connected to the data source so disconnect. ;
** We continue to direct code to execute on the target analytical server. ;
signoff source;
```

SOLUTION 2: TWO UNIX SERVERS AND DATA TRANSFER SERVICES

In this example we're already signed on to the target server. First we assign a libref there and then signon from the target server to the source server to start a SAS session there. With the two servers connected we can use Data Transfer Services to "pull" our data set from the source to the target.

Having Base SAS and SAS/CONNECT on both the analytical and data servers provides a wide range of approaches for moving data. It lets us establish a client-server relationship between the target server (the local client) and the data server (the remote server) in the same way we establish connectivity between a desktop and the analytical and data servers we worked with in Solution 1. With this direct connectivity between the servers, we eliminate the desktop middleman so all things being equal our clock time will be half of what it was in Solution 1.

Keep in mind that in a client-server relationship the biggest machine is not necessarily 'the server'. Whichever machine initiates the connection is the client of the other(s). In Solution 2 the target server receiving the data could be more powerful than the data source server but it is still the client in the relationship.

```
rsubmit target;
   ** Assign a local libref. ;
   libname target '/marketing/temp_data';

** Connect the target server to the data server. ;
%include '/home/bentleyj/connect_target_to_source_box.sas';

** Assign a libref on the remote host and then move the data set by;
** downloading it from the remote host to the local host.          ;
** Remember: Procs Upload and Download execute on the remote host. ;
rsubmit source;
   libname source '/marketing/phone_calling/southeast';

   proc download inlib=source outlib=target;
      select cust_w_prod1_2012_02
             cust_w_prod2_2012_02;
   run;
endrsubmit; ** source;

** Disconnect from the data server. ;
signoff source;

** Make sure everyone can read the data sets. ;
x 'chmod 755 /marketing/temp_data/cust_w_prod1_2012_02.sas7bdat';
x 'chmod 755 /marketing/temp_data/cust_w_prod2_2012_02.sas7bdat';

endrsubmit;  ** target ;
```

As seen above, once we have a client-server relationship between the two servers, it's not difficult to copy data from one to the other but the key is to use SAS/Connect to establish a connection between the two servers. And that's not tricky at all. If you're familiar with SAS/CONNECT then you know that it starts by locally executing a very short program that runs a script that starts a SAS session on the remote server. The RSUBMIT and ENDRSUBMIT commands direct SAS code to the remote session for execution. The log and listing by default return to the local host.

In the Solution 2, on the target server we %included `/home/bentleyj/connect_target_to_source_box.sas` and that's what connected it to the source server. This file is on the target server and here's what that it looks like. Only the first four lines are really needed. The lines inside the RSUBMIT block execute automatically on the remote server after the connection is established and after the remote autoexec.sas programs execute.

```
%let source=thick_as_a_brick.jethrotull.rock.net;

options remote=source;  ** use comamid= option if default TCP is not right. ;

filename rlink "/home/bentleyj/automatic_sas9_signon.scr";
signon source;

rsubmit source;
   %put The _sasHomeDir on the source server= &_sasHomeDir;

   options replace nocenter dsoptions=nomissopt
          sastrace=",,t," sastraceloc=saslog nostsuffix;

   libname infxDB informix server="cust_mktg " dbsliceparm=ALL
                           read_isolation_level=dirty_read;

   %let infxUser= d507201;
   %let infxUserPwd= &_infxServerPwd;
endrsubmit;
```

The program calls the script `/home/bentleyj/automatic_sas9_signon.scr` to start a SAS session on the data source server, session-id= *source*. The script is the standard SAS-provided TCP/IP connect script that has been

modified for unattended login. This is modification is important.

Below are is a snippet from the script. Notice that two INPUT commands are commented out and replaced with TYPE commands. In a manual startup the INPUT commands cause the system to open input windows asking for your user-id and password and wait while you enter the values and hit <enter>. When we're starting a SAS session on one server from another there's no way to have pop-up windows on the client, so we have to make SAS do it for us. Replacing an INPUT command with a TYPE command cause SAS to send the value specified to the system at the appropriate time. The LF commands are important—they send the <enter> command.

```
<snip>

log "SIGNON NOTE: Connecting to a SAS Server.";

waitfor 'login:', 10 seconds: nouid;
/* input 'Userid?'; */
type "d507201" LF;

waitfor 'Password', 10 seconds: noupass;
/* input nodisplay 'Password?'; */
type "red_Hat#Size9" LF;

<snip>
```

In our example here we've hardcoded the password. Hardcoded passwords are generally frowned upon by IT security. I've had trouble using a SAS encrypted password I think because the password must be sent to the operating system and the OS can't decrypt SAS encryptions but I've spoken with others for whom they've worked. Using SAS macro variables is a work-around but still requires hard coding the values somewhere, like in the user's autoexec.sas. A reasonable solution (to me anyway) is to assign passwords stored as variables a compiled Data step to macro variables via CALL SYMPUT. The compiled Data step is stored on the local machine and the user's autoexec.sas runs the compiled Data step. That way the password is available for use when the local host connects to the remote host and it's just complicated enough to confuse everyone except a SAS user familiar with the autoexec.sas and compiled Data steps.

SOLUTION 3: SAS/CONNECT AND REMOTE LIBRARY SERVICES

After seeing the previous two, this solution is a bit different in that we move specific records instead of the entire data set. After connecting the analytical server to the source data server we assign a libref there to the directory that holds the data we want and then we use RLS to make it appear to be local libref. Then we use Proc Copy to first transfer two entire data sets from source to target and then we do it with a subset of the records.

We said earlier that RLS lets use reference a remote libref just like it was on the local host. That's great, but when moving an entire data set, you might not want to use a Data step to do it. A Data step reads each record one-at-a-time into the program data vector and then writes them one-at-a-time to the output data set. It does it very fast, but Procs Upload, Download, and Copy move data in blocks so they're much faster, especially when dealing with big data.

```
rsubmit target;
   libname target '/marketing/temp_data';

   ** Connect to the data server. ;
   %include '/home/bentleyj/connect_target_to_source_box.sas';

   ** Assign and map the remote libref for local access. ;
   ** Use the method that doesn't RSUBMIT to the remote server. ;
   libname source '/marketing/phone_calling/southeast' server=source;

   ** Copy the file.  ;
   proc copy inlib=source outlib=target;
      select cust_w_prod1_2012_02
             cust_w_prod2_2012_02;
   run;
endrsubmit;
```

Now let's do a couple different things to make the examples a bit more realistic.

- Use a Data step to move a subset of one of the data sets by using a couple SET options to dynamically select only the records and fields we want without creating a data set on the target server.
- Have Remote Compute Services to create a Work data set on the source server, map that Work library so that the local server can use it, and use Proc data sets to move, not copy, that data set.

So here we copy a subset of one of the remote files to the local Work library, moving only the records and fields we need. Then we use Remote Compute Services to subset the data into a Work data set on the source server and Remote Library Services and Proc Datasets transfers the data to the local Work library ;

```
rsubmit target;
   ** Still connected to the data server from previous example. ;
   data cust_w_prod1a_2012_02;
      set source.cust_w_prod1_2012_02
        where=(prodSubCode in (2812 3872 4887 4889)
              keep=cust_id prodSubCode hm_: lastDate lastTotAmt);
   run;

   ** On the data server, create a data set in the Work library. ;
   ** First assign a libref—the one created earlier doesn't exit ;
   ** only on local host. ;
   rsubmit source;
      libname source '/marketing/phone_calling/southeast';

      data in_both_lists;
         merge source.cust_w_prod1_2012_02(in=inList1)
               source.cust_w_prod2_2012_02(in=inList2);
         by cust_id;
         if inList1 and inList2;
         <transformations and derivations>
      run;
   endrsubmit;

   ** Map the remote Work library for local access. ;
   libname remWork slibref=work server=source;

   ** Move the data to the analytic server Work library. ;
   proc datasets nolist;
      copy out=work in=remWork move;  ** ←Move option deletes the ;
      select in_both_lists;           ** data set on the remote host. ;
      run;
   quit;

signoff source;
endrsubmit;
```

SOLUTION 4: USING FILE TRANSFER PROTOCOL, FTP

A 30-second Intro to FTP and SFTP

There are two good things about using SAS's File Transfer Protocol (FTP) and Secure File Transfer Protocol (SFTP) access methods. (1) They come with Base SAS so SAS/CONNECT is not needed and (2) they're easy to use. But there is a potential major problem though. If you need the Secure File Transfer protocol (SFTP) then you need SAS 9.2 because 9.1.3 supports only FTP. They mostly share the same arguments and options and the code to use them is very similar for tasks like moving a file or two.

What's the difference between FTP and SFTP? In a nutshell, FTP provides an open unsecure connection for file transfer between two hosts over a network. In many places this is a no-no because clever people with bad intentions can intercept the network packets and read the data, including passwords. SFTP is an enhanced version of FTP in which both the commands and the data are encrypted during transmission. That's the secure aspect. It might seem like overkill to use encrypted file transfers inside a corporate network, but that's the world we live in nowadays. Your

IT security folks probably either already have SFTP in place or are moving to it. If you don't know, find out.

FTP and SFTP can co-exist on the same server just as different versions of SAS can co-exist. The capabilities provided by SFTP are quite similar to FTP and the syntax is immediately familiar to FTP users. But because they use different file transfer protocols you can't use an FTP client to talk to an SFTP server or vice versa without middleware to establish a bridge. Be careful if you see a reference to "Secure FTP" because it might not be SFTP. There are techniques for using FTP over a secure connection via 'tunneling' and that's variously called Secure FTP, FTPS or FTP over SSH.

Some SFTP Details

SFTP requires that the network connecting between the two secure servers be running an Open SSH SSHD server. Implementation and configuration details are handled by your Network Administration team, thank goodness. Be aware that the SAS documentation for *Filename Statement, SFTP Access Method* says "The SFTP access method relies on default send and reply messages to OpenSSH commands. Custom installs of OpenSSH that modify these messages will disable the SFTP access method." For example, on UNIX the SFTP prompt must be 'sftp>' or 'sftp >'.

The SFTP access method supports only the OpenSSH client on UNIX and PUTTY on Windows and the appropriate executable must be in the $PATH or PATH search path. SAS recommends public key authentication when connecting to a remote SSHD server. For troubleshooting SFTP problems, try to manually accomplish a similar task without involving SAS to insure that the configuration and key authentication are correct.

Command Line FTP via the X Statement

If you're comfortable with command line FTP or SFTP, then just use the X statement to submit the appropriate command strings to the operating system. Alternatively, the X statement can run an FTP script that resides somewhere in the server's file system. The SYSTEM function and CALL SYSTEM routine accomplish the same thing as the X statement but take more overhead. If you're interested, the References and Resources section at the end has a link to a web site that provides examples and tutorials for FTP scripts. Be sure to keep in mind the limitations and implications of running system commands via the X statement.

If you like scripting, SFTP has a batchfile option. Important to note is that the first argument is empty quote marks. This is where the path to the external go but because we're using a batch file we must set it to blank because it is required.

```
filename weekly sftp ' ' host="unixhost1" user="userid"
                      batchfile="/marketing/sftp_scripts/weekly_move.bat";

data _null_;
   infile weekly;
run;
```

The FTP Access Method

The FTP and SFTP access methods are quite similar so in this paper our code samples will reference FTP because most people are probably more familiar with that one.

The FILENAME statement's FTP Access Method greatly simplifies transferring files via FTP in SAS code. No scripts are needed but there are options that in certain situations will be useful or even necessary. For example, for text files the default logical record length is 256 so you may need to use the LRECL= option. I strongly recommend that you flip thru the on-line documentation for FTP or SFTP before putting fingers on the keyboard. There are some very good examples available there that will jump start writing your own code.

The first code example we want to move a SAS data set from the data server to the work library of the analytical server. We start by assigning the directory location of the SAS Work libref on the analytical server to a macro variable. We have to query SAS to get the directory because we don't know it—by default it's dynamically assigned when the SAS session initializes so it changes each time. Then we assign two filerefs that use the FTP Access Method—one for the source directory and the other for the target directory. After that, a Data step handles the file transfer.

```
** This runs on the local host (target server);
rsubmit target;
   %let _workDir = %sysfunc(getoption(work));

   filename source ftp '/regional_data/central/partners/10_PNTR'
                  host='consrisk.prod.wachovia.net'
                  binary dir
                  user="&_user" pass="&_userPwd";

   filename target ftp "&_workDir"
                  host='ecommprd.usa.wachovia.net'
                  binary dir
                  user="&_user" pass="&_userPwd";

   data _null_;
      infile source('p10_pntr_scrub_08.sas7bdat')  truncover;
      input;
      file target('partners_removed_201008.sas7bdat';
      put _infile_;
   run;
endrsubmit;
```

The FILENAME statements will look familiar but they use arguments and options specific to the FTP Access Method. The very first item after the access method (FTP or SFTP) is the argument that contains either the physical name a file or a directory location. One fileref tells where the file is coming from and the other tells where it's going. This is the only required argument. The remaining specifications are options that can be in any order.

- HOST= provides the IP address or alias of the server.
- BINARY tells SAS to move the file in 'image mode' with no line delimiters—this is required for SAS data sets.
- I like using the DIR option because it makes all the files in the specified directory available.
- The USER= and PASS= options give the user-id and password needed to access the server.
- The DEBUG option is not shown but is useful for troubleshooting. If you get an error message of "Source file doesn't exist", it might mean you don't have read permission.

The FTP Access Method requires that a Data step be used and that the INFILE and FILE statements use the filerefs as functions. The FILENAME's DIR option allows the filerefs to act as functions to which we pass the data set name as the argument. This lets us use one fileref to move multiple files. If we didn't need to do this, we'd place the file name in the quotes instead of the directory path and use the CD= option to position us in the desired directory. Then we could use a standard Data step to read the data set and create a data set in the Work or a previously assigned permanent library.

Our first FTP example uses the INFILE statement's TRUNCOVER option. It's required for SAS data sets because they're variable length—TRUNCOVER prevents error messages when a line is shorter than the INPUT statement expects.

Here is an example that reads a flat file from the data source server and creates a SAS data set in the Work directory. Keep in mind that because the subsetting IF executes locally, every record must be brought over the network to the analytical server. But if SAS/CONNECT isn't available you can always fall back on FTP/SFTP.

```
rsubmit target;
   filename raw ftp 'customer_activity.dat' cd='/marketing/raw_data'
            user='guest' host='rubadubdub.3men.tub.net'
            recfm=v user="&_user" pass="&_userPwd";

   data mydata;  infile raw;
      input <record layout>;
      if <condition>;
   run;
endrsubmit;
```

One last item about the FTP and SFTP Access Methods—when moving a SAS data set, the number of records reported in the log will not be the number of observations in the data set. The log from moving a data set with 88,476 observations shows reading and writing only 42,576 records. SAS data sets are moved as binary images, so this is actually the number of blocks moved. So it's a good idea to run Proc Contents before and after using moving a data set via FTP or SFTP. If your program needs built-in validation you can use the dictionary tables to capture record counts before and after the file transfer. See Frank Dilorio's papers on dictionary tables for details and ideas.

CONCLUSION AND FINAL THOUGHTS

Moving data between servers is only a small part of using SAS in a distributed computing environment but it's an important and powerful capability. With it you can connect servers 1:1 or one: many (the session id tells RSUBMIT which server to execute on) and you can daisy-chain servers together so you can daisy-chain librefs too.

Client-server solutions can quickly get complicated so it's time well spent if you at least roughly flowchart a solution before starting to code. Here's a scenario to think about… Picture yourself living in Hawaii and working there on a desktop connected via SAS/CONNECT to a server in North Carolina. (I like that picture.) A server in Arizona has a data set with fifteen million customer accounts that you need to subset and augment with data in a flat file also on the Arizona server, add a couple fields that reside in a long narrow data set on the Carolina server, then transform and summarize, and finally create a text file that for downloading and exporting into Excel. Each record in the Arizona data set is a thousand bytes long and is updated weekly. The flat file is similar size and update frequency. This has to be a weekly process so it's worth putting some effort into it. What would a client-server solution look like?

A final tip is that in my experience Procs Upload and Download are faster than Proc Copy. A simple test showed that Copy took about 8 minutes but Upload took less than 3 minutes to move the same data set. I haven't asked SAS Support about it, but it could be that Copy uses the operating system copy utility and Upload and Download use an algorithm designed for moving SAS data sets between platforms. The CPU times seems to point to that explanation.

```
rsubmit idsbox;
NOTE: Remote submit to IDSBOX commencing.
44   libname prodWrk slibref=work server=prodBIS;
NOTE: Libref PRODWRK was successfully assigned as follows:
      Engine:        REMOTE
      Physical Name: /sas/work3/SAS_workD9DE0091E0FE_nssd1sasp001
45   libname nat1 '/home/apps/mkt/conversion/national1';
NOTE: Libref NAT1 was successfully assigned as follows:
      Engine:        V9
      Physical Name: /home/apps/ccg_mkt/fdr_conversion/national1
46
47   proc copy inlib=nat1 outlib=prodWrk;
48      select base_from_source;
49   run;
NOTE: Copying NAT1.BASE to PRODWRK.BASE (memtype=DATA).
NOTE: There were 364682 observations read from the data set NAT1.BASE.
NOTE: The data set PRODWRK.BASE has 364682 observations and 52 variables.
NOTE: PROCEDURE COPY used (Total process time):
      real time           8:14.00
      cpu time            0.88 seconds
50
51   <use proc delete to delete work.base_from_source>
54
55      rsubmit prodBIS;
NOTE: Remote submit to PRODBIS commencing.
4          proc upload inlib=nat1 outlib=work connectStatus=no;
5             select base_from_source;
6          run;
NOTE: The data set WORK.BASE has 364682 observations and 52 variables.
NOTE: PROCEDURE UPLOAD used (Total process time):
      real time           2:41.00
      cpu time            2.91 seconds
NOTE: Remote submit to PRODBIS complete.
NOTE: Remote submit to IDSBOX complete.
```

So, where do you go from here? If you use macro variables you'll want to learn how to use %sysrput and %syslput to move macro variables values between servers. If you're looking for something perhaps more challenging, well, spend some time with SAS/CONNECT's MP CONNECT or piping via the SASESOCK engine. I'm envisioning a sort of poor man's grid—I see a grid as just a big intelligent distributed computing environment. This link has code that might give you some ideas. http://support.sas.com/rnd/scalability/tricks/connect.html#pipsrem.

REFERENCES AND RESOURCES

- Barrett, Daniel J., et al. 2005. *SSH, The Secure Shell: A Definitive Guide*.
- Dilorio, Frank and Jeff Abolafia. 2004. "Dictionary Tables and Views: Essential Tools for Serious Applications," *Proceedings of the Twenty-Ninth Annual SAS Users Group International Converence*, Cary, NC: SAS Institute Inc.
- Garner, Cheryl A. and Stephanie Reinard. 1995. "The SAS® System: A Complete Client/Server Solution," *Proceedings of the Twentieth Annual SAS Users Group International Converence*, Cary, NC: SAS Institute Inc.
- Sadof, Michael G. 2005. "Connecting Simply with SAS/CONNECT," *Proceedings of the Thirtieth Annual SAS Users Group International Converence*, Cary, NC: SAS Institute Inc.
- SAS Institute, Inc. *SAS/CONNECT User's Guide.* Cary NC: SAS Institute Inc.
- SAS Institute, Inc. 1995. *Client/Server Computing with the SAS System: Tips and Techniques.* Cary NC: SAS Institute Inc.
- SAS Online Documentation. Search for FTP (version 9.1+) and SFTP (version 9.2+)
- The Kermit Project at Columbia University has good code examples of FTP scripts. http://www.columbia.edu/kermit/ftpscripts.html
- Vaughn, Larry T. 1994. *Client/Server Systems Design and Implementation*. McGraw-Hill: New York.

CONTACT INFORMATION

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

John E. Bentley
Wells Fargo Bank
Charlotte, NC 28226
704-383-2405
john.e.bentley@wellsfargo.com