Paper 143-2011

Using SAS® to Move Data betweenServers

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

ABSTRACT

In today's complex data management environments, it is not unusual for UNIX servers to be dedicated to a particular department, purpose, or database. As a result, a SAS® data analyst often works with multiple servers, each with its own data storage environment. For security reasons, our IT partners often do not make it easy for the servers to talk to one another. So how do we transfer data between servers? SAS provides a number of solutions. This paper shows ways to use Remote Compute Services and SAS Data Transfer Services along with Procs and Data steps to move data between servers. Code samples are provided. This presentation is designed for all levels of SAS programmers and is relevant to SAS Enterprise Guide® users as well.

INTRODUCTION

Let's define a situation that's common in financial institutions and probably most other places. We're running SAS on a UNIX server but our data file is on a different UNIX server. This describes a very 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 both servers.

Now here's the problem we have to solve: We need a programmatic solution (no manual intervention) that will let our SAS program use the data located on another 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 we focus in this paper on moving data sets between servers, the same techniques work very well with flat files. But 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!

There are a few items to note about the code examples.

1. As we all know 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 were developed with SAS 9.1.3 using Win/XP and AIX and should work anywhere with at most minor tweaking
2. I'm using the following 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 use them for things like file and data set names and directory paths.

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 all of these things—remote compute services and remote data 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 we need that I'm going to just condense and summarize some of the material they present. It is important to understand at least the basics because the implications and side effects of choosing one approach over another can be significant especially when working with massive amounts of data. For a 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 environment over two or more computers. Basically, it provides the middleware that allows us to access data on different systems and leverage our computing resources by distributing processing to the most appropriate machine. For it to work though, SAS/CONNECT must be installed on both machines. A 'local' SAS session runs on the computer that initializes the connection--this is the *local host*. SAS/CONNECT establishes a connection and starts a 'remote' session on a *remote host*. Viola', we have a client/server environment. But which host is the client and which is the server? The client is the one that originates the processing—the server serves the client. But SAS prefers to identify the computers as local and remote. In this paper I use analytical server and data server, target and source.

SAS/CONNECT allows a wide range of problem-solving approaches but it isn't mandatory for solving our particular problem. But it provides the services introduced below that if used together can provide a flexible, robust solution. . For those who are familiar with using SAS/CONNECT is a desktop-server configuration, it works the same way in a server-server configuration.

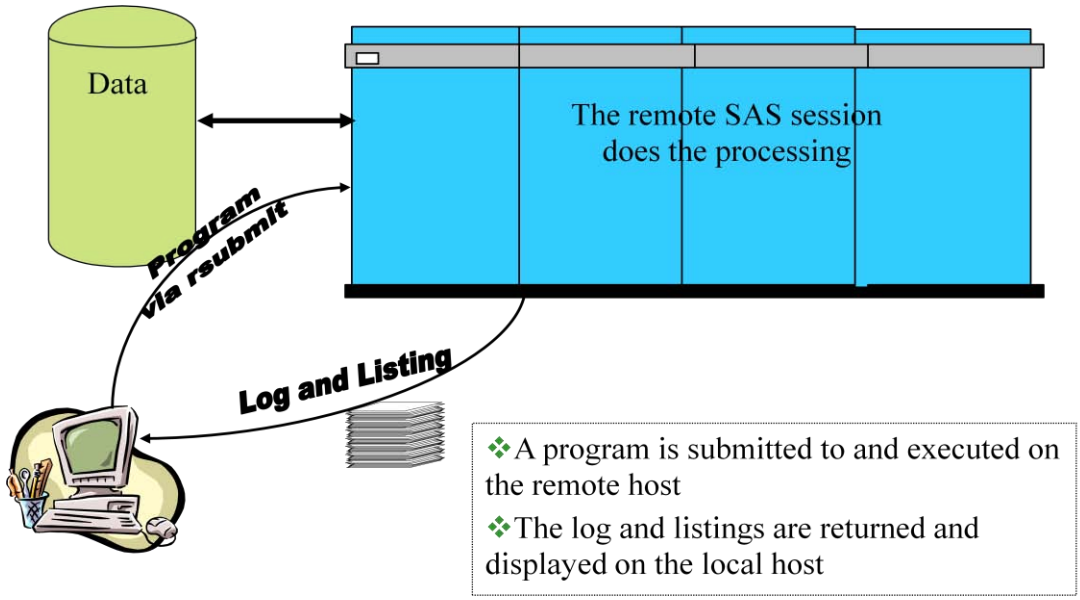
The primary advantages or SAS/CONNECT is that it provides remote processing and remote data access capabilities. With remote processing, a SAS user on the local host can easily submit code for execution on the remote host by using the RSUBMIT and ENDRSUBMIT command. All code between RSUBMIT and ENDRSUBMIT executes on the remote host. Unless the log and listing are redirected, they will be returned to the local host.

Remote data services include file transfer services. File transfer can be accomplished using Procs Copy, Upload, and Download, or a Data step. There are many good examples in the SAS/CONNECT user's manual. This paper will not review SAS/CONNECT capabilities in any further detail 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 allow us to effectively utilize all the computing resources available, not just those on the server or desktop that we're working from. SAS 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 directories on the remote platform to as well as the remote CPUs.

In Display 1 below, our environment consists of a desktop, a server, and the data storage for the server (DASD, SAN, NAS). We write our program on the desktop using the SAS Editor and save it to our C: drive. But we execute the program on the server because that's where the data is and 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 saved locally.



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

The advantages are clear—

- Access to data not directly accessible to the analytical server without moving the data to the analytical server.
- 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—

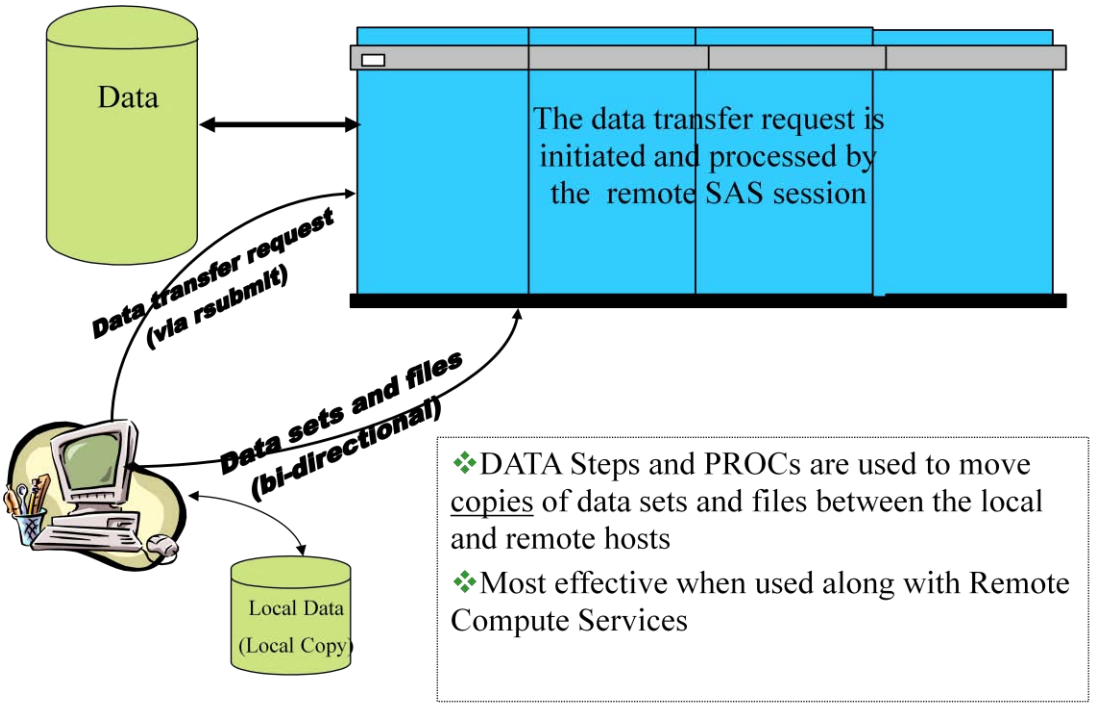
- Code development, testing, and debugging might be a drawback if you work on a heavily-used server or are charged for the CPU cycles you use.
- If you work on a production server, production jobs could be impacted by your development and testing.
- Network traffic is increased a bit, but not much.

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 datasets or external files, 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 of the data from where it resides to another machine. The graphic below shows that we submit a data transfer request to the remote session and a copy of the data is moved. We can either send or retrieve data using appropriate commands such as Proc Upload and Proc Download.



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 system 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; and
- Automated jobs can be set up to perform backups onto different machines, collect data from multiple systems, or pre-position data for tasks like data scrubbing or data quality control.

But because there are no free lunches there are a few disadvantages.

- DTS creates multiple copies of the data, so if we're performing updates then the data can get our of sync. Also it could be that data security policies do not allow multiple copies.
- Network performance will be degraded when the volume of data being copied is large. What constitutes 'large' is site-specific.
- Depending on what you're doing, 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.
  - 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.
- Don't move large amounts of raw data unless the target machine is faster than the source machine and the processing time saved more than makes up for the transfer time. Otherwise use Remote Compute Services to pre-process the data and then use Data Transfer Services to move a tailored dataset.

REMOTE LIBRARY SERVICES

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

RLS starts a *remote engine* executing in the local session works with a server session executing in the remote session. This allows SAS procedures and Data steps running locally 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 could then use RLS to access it.

A properly specified LIBNAME statement is what gives the local host access to the remote data under 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 REMOTE engine specification and the SERVER= remote session id.

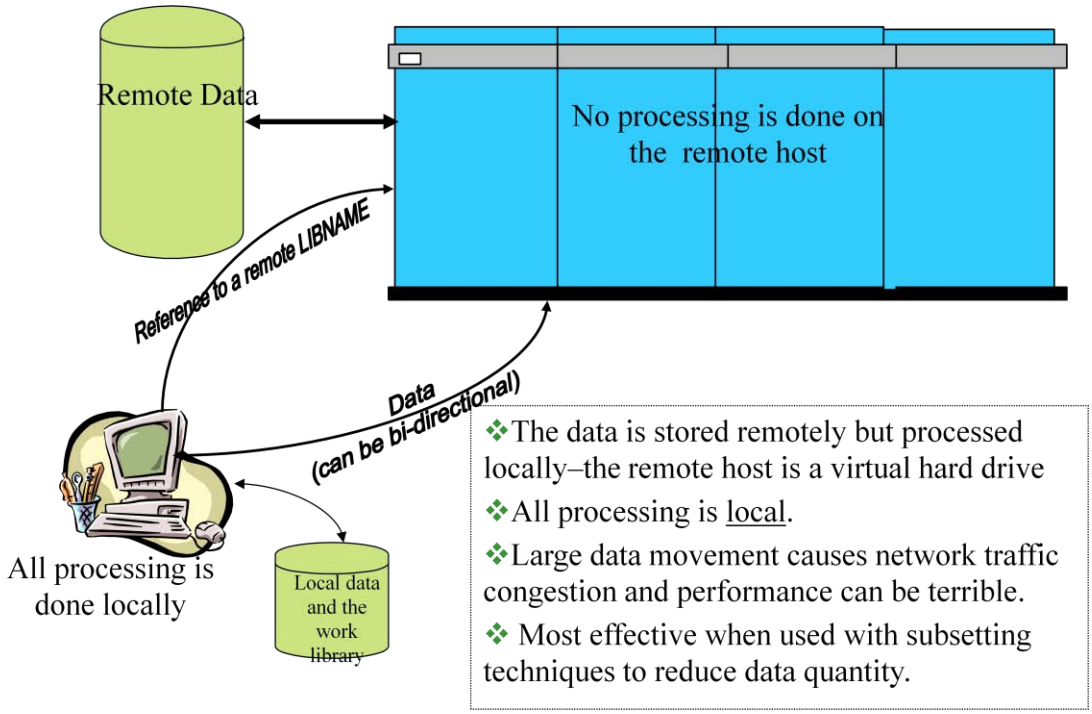
```
RSUBMIT;
   SIGNON <session-id>;
   LIBNAME <libref> REMOTE <'directory-path'> SERVER=<session-id>;
ENDRSUBMIT;
```

In above example the libref does not exist on the remote host. Only the local host can use it. If we want the libref created on the remote host and the local host mapped to it then use this approach.

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

Diagram 3 illustrates using Remote Library Services. A local libname is assigned to a directory on a remote host so that processing can take place on the local host. A Data step for example passes normal record requests to the "remote" libref and returns the data—the physical location of the data is transparent.

.



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

The advantages of Remote Library Services are substantial.

- RLS is transparent to GUI applications. To the application, all data appears to be local.
- Cross-Environment Data Access (CEDA) kicks in automatically to read 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 though can also be substantial. Careful thought must be given to the amount of data being accessed because network traffic can be easily impacted.

- All processing is local so RLS can be problematic large amounts of 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

Most of 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 the volume of data is too large; the data is frequently updated; and 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. Alternatively, simply using SET options to dynamically filter and minimize the amount of data being moved by DTS. Always minimize network traffic. The 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. The data we need is not on this server so we'll call it the target server. The server where the data resides is the source server. Base SAS and SAS/CONNECT are on the source server. For the first examples we'll assume that we need the entire data set on the target server and then increase complexity of the examples .

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

For this approach we must signon to start a SAS session 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. Now we have three copies of the data.

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

   ** Now download the data set we want.                                ;
   ** 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, our target SAS analytical server.     ;
** Now we do use the same process to move the data set to the target server ;
** except we use Proc Upload.                                               ;
rsubmit target;
   libname target '/groups/marketing/projects/temp_data';

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

** The desktop is still connected to both source and target so disconnect. ;
signoff source;
signoff target;
```

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 we 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 copy our data set directly 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 true client-server relationship between the local target server (the client) and the remote data server (the server) in the same way we set up the relationship between a desktop (the client) and our SAS analytical server (the server). It's a bit counter-intuitive, but in a client-server environment the biggest machine is not always the 'server'. Whichever machine initiates the connection is the client. In Solution 2 below the analytical server receiving the data is the client of the data server that, well, serves the data. With this direct relationship, we eliminate the desktop middleman so all things being equal our clock time will be half of what it was in Solution 1.

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

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

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

      proc download inlib=source outlib=target;
         select    cust_w_prodList1_2010_08
                   cust_w_prodList2_2010_08;
      run;
   endrsubmit;      ** source;

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

   ** We might need to make sure others can read the data sets. ;
   x 'chmod 755
   /groups/marketing/projects/temp_data/cust_w_prod1_200908.sas7bdat';
   x 'chmod 755
   /groups/marketing/projects/temp_data/cust_w_prodList2200908.sas7bdat';

endrsubmit;   ** target ;
```

Using SAS/Connect to create a client-server relationship between two servers is 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 remote SAS session on the analytical 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 above solution, we included `/home/bentleyj/connect_target_to_source_box.sas` on the target server to connect to the source server. Here's what that program looks like. Only the first four lines are needed. The lines inside the RSUBMIT block execute automatically on the remote server after the connection is established, after the remote autoexec.sas executes.

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

options remote=source comamid=TCP;   ** TCP is default access method for ;
                                     ** UNIX and Win, XMS for Z/OS.      ;
filename rlink "/home/bentleyj/automatic_sas9_signon.scr";
signon source;

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

   options msglevel=i replace nocenter obs=max 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= Gua#45T1;
endrsubmit;
```

The program calls the script `/home/bentleyj/automatic_sas9_signon.scr` to start a SAS session on the data server, session-id= source. It's a standard SAS TCP/IP connect script that has been modified for unattended

login. Notice that two INPUT commands are commented out and replaced with TYPE commands. In a manual startup the INPUT commands cause the system to ask 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 manually do the sign on so we have to make SAS do it for us. We've replaced the INPUT commands with TYPE commands that cause SAS to send the values specified to the system at the appropriate times. 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 "d102705" LF;

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

<snip>
```

Hardcoding passwords anywhere is generally frowned upon by IT security. Using SAS encrypted password won't work because the password must be sent to the operating system and the OS can't decrypt SAS encryptions. Using SAS macro variables is a work-around but that still requires hard coding the values somewhere, like in the local autoexec.sas. A reasonable solution (to me anyway) is to assign passwords in a compiled Data step via CALL SYMPUT. The compiled Data step is stored on the local machine and the user's autoexec.sas runs the Data step. That way the password is available for use when the local host connects to the remote host. It's just complicated enough to confuse everyone except a SAS user familiar with compiled Data steps.

SOLUTION 3: SAS/CONNECT AND REMOTE LIBRARY SERVICES

After seeing the previous two, this example is a bit different in that we can 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 transfer two data sets from source to target.

RLS lets use reference a remote libref just like it was on the local host. But even so, when moving an entire data set, avoid using a Data step. 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.

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

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

   ** Assign a library on the data server. ;
   rsubmit source;
      libname source '/home/marketing/phone_calling/regional/southeast';
   endrsubmit;

   ** Map the remote libref for local access. ;
   libname source remote server=source;

   ** Copy the file.  ;
   proc copy inlib=source outlib=target;
      select cust_w_prodList1_2010_08
             cust_w_prodList2_2010_08;
   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 datasets 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;
   ** Havent yet signed off from the data server so libref is still available.
   ;
   data cust_w_prodList1a_2010_08;
      set source.cust_w_prodList1_2010_08
                  (where=(prodCode in (2812 3872 4887 4889)
                   keep=cust_id colatState hm_: lastPurchDate
   lastPurchTotAmt);
   run;

   ** On the data server, create a data set in the Work library. ;
   rsubmit source;
      data in_both_lists;
         merge cust_w_prodList1_2010_08(in=inList1)
               cust_w_prodList2_2010_08(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 source server. ;
      run;
   quit;

signoff source;
endrsubmit;
```

SOLUTION 4: USING FILE TRANSFER PROTOCOL, FTP

A 90-second Intro to FTP and SFTP

There are three good things about using Base SAS's File Transfer Protocol (FTP) access method. SAS/CONNECT is not needed and it's easy to use. But there is a potential major problem though. If you need Secure Shell FTP (SFTP) then you need SAS 9.2 because 9.1.3 supports only FTP. Some options are different, but the FTP and SFTP access methods basically work the same so in this paper we'll look at FTP.

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. 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 are 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 features 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.

Command Line FTP via the X Statement

If you’re comfortable with command line FTP, then you can just use the X statement to submit FTP 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.

The FTP Access Method

The FILENAME statement’s FTP Access Method greatly simplifies transferring files via FTP in SAS code. No scripts are needed but there are lots of 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 there that will jump start writing your own code.

The code example here starts by assigning the SAS Work directory on the local SAS analytical server to a macro variable. We use a macro variable here because we don’t know the Work path as it’s dynamically assigned when the SAS session initializes. 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.

```
rsubmit;
   %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 some arguments and options specific to the FTP Access Method. We’ve put the path to the working directory in quotes instead of the fully-defined file name normally specified in a FILENAME statement. The HOST= specifies the IP address or alias of the server and BINARY tells SAS to move the file in image (binary) mode with no line delimiters—this is required for SAS data sets. The DIR option makes all the files in the specified directory available to us, and 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. “Source file doesn’t exist” might mean you don’t have read permission.

The FTP Access Method requires a Data step be used and the INFILE and FILE statements use the filerefs as functions. The FILENAME’s DIR option allows the filerefs to act as a function to which we pass the data set name as the argument. This feature lets us use one fileref to move multiple files. If we didn’t need this capability, we’d need only a fileref for the sources. 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.

We must use the TRUNCOVER option with the INFILE statement. 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.

A last tip about the FTP Access Method is that 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. This is actually the number of blocks moved, so if you're unfamiliar with the data it's a good idea to run Proc Contents before and after using FTP to move a data set. If your program needs built-in validation then you can use the dictionary tables to capture record counts before and after the file transfer. See Frank Dilorio's SUGI papers on dictionary tables for details.

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. You can connect one server to many servers—the session id tells RSUBMIT which server to execute on—and you can daisy-chain servers together and you can daisy-chain libref too.

Here's a scenario I like… Picture yourself as a contractor living the US Virgin Islands and working there on a desktop connected via SAS/CONNECT to a SAS analytical server in North Carolina. A server in California has a data set with fifteen million customer accounts that you need to summarize and put into an Excel pivot table. Each record is a couple thousand bytes long and is updated nightly from a DB2 database. That's a lot of data to move with FTP but you could do it with a couple filerefs and a Data step.

A better solution would be to map librefs from the desktop to Carolina to California and then use a Data step with WHERE= and KEEP= options to move only the records and fields needed to do the summary. Even better than that, you can use Remote Compute Services to summarize the data on that server and then transfer only the much, much, much smaller results set to the Carolina server for exporting to Excel.

One tip I have is that Proc Upload and Download seem to be much faster than Proc Copy. A simple test showed that using Copy took about 8 minutes but using Upload took less than 3 minutes. I haven't asked SAS Support about it, but it could be that Copy uses the operating system copy utility but Upload and Download use an algorithm designed expressly for moving SAS data sets between platforms. The CPU times seems to indicate that.

```
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   proc datasets library=prodWrk nolist;
52      delete base_from_source;
53   run;
NOTE: Deleting PRODWRK.BASE (memtype=DATA).
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;
<snip>
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 more challenging, well, I'm thinking of spending some time with SAS/CONNECT's MP CONNECT and piping via the SASESOCK engine. I'm envisioning a sort of poor man's grid—it's just a big intelligent distributed computing environment. This link has code that could be a starting point for development-- http://support.sas.com/rnd/scalability/tricks/connect.html#pipsrem. Good luck!

REFERENCES AND RESOURCES

- 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. 2000. "Multiprocessing with Version 8 of the SAS® System," *Proceedings of the Twenty-Fifth 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.
- LeBouton, Kimberly. 2000. "Smokin' With UNIX Pipes," *Proceedings of the Twenty-fifth 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. 2004. *SAS/CONNECT 9.1 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 CEDA, Cross-Environment Data Access
- SAS Online Documentation. Search for FTP (version 9.1) and SFTP (version 9.2)
- The Scalability Community at http://support.sas.com/rnd/scalability/connect/index.html has some great documentation and code examples of how to use the PIPE engine.
- 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

Disclaimer: The opinions expressed here are the view of the author and do not necessarily reflect the views and opinions of Wells Fargo Bank. Wells Fargo Bank is not, by means of this article, providing technical, business, or other professional advice or services and is not endorsing any of the software, techniques, approaches, or solutions presented herein. This article is not a substitute for professional advice or services and should not be used as a basis for decisions that could impact your business