

SAS Client-Server Development: Tricks of the Trade

Eric Brinsfield, Meridian Software, Inc.

ABSTRACT

Client-server computing is not just a matter of trading data between host and client. Whether you are developing in a client-server environment or using a client-server application, the well-designed client-server system should optimize the location of processing and minimize the transfer of data. This paper offers tips to the SAS client-server developer by discussing traditional client-server techniques using SAS/CONNECT as well as clarifying the impact of thin-client Web development.

Where appropriate during this discussion, I will evaluate the pros and cons of performance strategies. Where possible during the presentation, I will demonstrate the results of interesting design choices.

INTRODUCTION

Over the past several years, I have taken part in the development of a variety of client-server applications. For most of that time, we developed in a distributed configuration or virtual office environment. Each developer could connect to the home network when necessary, but would typically operate independently. As a result, we needed to devise techniques for developing client-server applications without the server.

The virtual office architecture also served to amplify the effects of poorly designed client-server software. As a result, we learned how to optimize our applications by placing the intensive processing on the right machine and keeping the large files in the appropriate place. In this discussion, I will suggest several general ways to improve the performance of your client-server designs. Then, I will share some of our techniques for developing client-server applications in a virtual office environment and for developing efficient client-server systems.

With the growing popularity of Internet applications, which quite often communicate over phone lines, implementation of good client-server design is more important than ever. The rules have changed, but the issues are the same. I will address thin-client design issues in more detail in the *Inet Solutions* section in a paper entitled: *SAS Client-Server Development: Taking it to the Web*.

DEFINITIONS

Client-server computing

Generally, a client-server application provides a graphical user interface that runs on a PC or workstation that accesses and processes data on a separate server running the same or different operating system. Using this architecture, the user gets a nice user interface, but shares a centralized and sharable image of the data and benefits from extra power on the server or servers.

The user is not usually aware of where the data resides or where the processing occurs. When designed correctly, this client-server attribute can be referred to as *transparency of location*.

Most people think client-server means simply accessing data on a remote server from a local workstation. Unfortunately, client-server systems are not that simple, but we are getting closer. The client-server designer must optimize where specific processing is performed rather than just linking to remote data and performing

all manipulations on the local workstation. The designer must consider where intensive computing will occur.

Now, vendors of database management systems have simplified this process over the last several years by designing smarter database systems. The DBMS internally and automatically performs much of the intensive processing involved with searching, sorting, joining, and extracting. As a result, client-server developers may not need to worry about where the processing occurs as long as they utilize the capabilities of the DBMS appropriately and the database can provide all they need.

Client-server development

The term "client-server development" can have two interpretations:

1. Development of a client-server application
2. Development of an application or program in a client-server environment.

I use both definitions, but I need only clarify the second. When working in a distributed environment, we often find it useful to develop code locally on a stand-alone workstation using local editors, small sets of data, and faster processors. In this case, a graphical user interface is not required for a client-server environment. As a SAS programmer, you can develop code locally and easily submit code remotely using SAS/CONNECT. I will address these techniques in more detail below.

Remote vs local

Who is remote and who is local? Depending on the communications software vendor and on the user's perspective, the remote may turn out to be the local and the local the remote. To avoid the confusion, I will try to use client and server rather than local and remote. In most discussions of SAS/CONNECT software, "remote" refers, however, to the server side of the equation. For example, when you specify the SAS system option "REMOTE=", you are pointing to the SAS/CONNECT server system. In other words, if you are using SAS/CONNECT to initiate a "remote" session, that makes you the local.

PERFORMANCE CONSIDERATIONS

Most software applications can be tuned by optimizing data access, data processing, and data display. With client-server systems, you must also try to optimize:

- machine-to-machine data transfer rates
- location of processing
- location of data.

These new optimization factors can be addressed in terms of:

- Bandwidth

If your bandwidth is a limiting factor, try to reduce the quantity of data transferred across the network or modem. For any analysis or query, try to determine when the resulting data sets are the smallest. Transfer at that point and continue processing on the client, if possible.

For example, if you intend to subset the data that you extract from a database or data set on the server, perform the

subset operation on the server before you download it. Use proc download to transfer the data from the server to the workstation, feeding the output data set directly to your next step, such as a PROC REPORT program.

- **Server CPU vs workstation CPU**

In general, most client-server applications are written to concentrate most of the CPU-intensive processing on the server. In a multiuser environment this is the most robust and scalable, but if the server is weak or overloaded, you could justify diverting more of the work to the workstation, if the power is there.

You will need to analyze the processes you are designing in order to determine where the processing should occur. Clearly, the choice is really an interaction between the capacities of bandwidth and processor abilities. You should also consider maintenance before you decide to split processing too far, but a good object-oriented design could solve that aspect of the problem.

- **Server disk storage vs workstation disk storage**

The option to redistribute some of the work to the client machine may be thwarted by the lack of disk space on the client. Many companies reduce workstation disk drive sizes and require users to store most data on network drives. In a situation where you have a Unix application server and your personal data is stored on Novell, your application may encounter multiple transfers of data: once from Unix to the workstation and then from the workstation to the Netware server. If bandwidth is high, this is not a problem.

On the other hand, if the systems administrators continually scream about disk usage on the servers, you may be able to take advantage of larger hard drives on the workstations. Utilizing local hard drives forces you to consider back-up requirements and storage capacity of each individual workstation that will use the application. Remember, however, that not every workstation using your client-server application has to behave the same way. You can design user modifiable configuration options that permit one user to take greater advantage of the workstation, while another relies on the server. This optimization technique applies primarily to work data sets rather than permanent.

- **Database software**

As I mentioned earlier, database management software now performs much of the data retrieval processing for you. SAS/ACCESS has been improving over the years to make more use of the capabilities of the DBMS software. For the best performance in SAS Version 6, use the SQL Pass-Through Facility, which allows you to send SQL directly to the database, which utilizes its own optimizer.

If you decide to create views that will be accessed on the server side, try to include where clauses in the view if possible. This is more efficient than applying the where clause when you use the view.

If your database server is heavily loaded, you should also consider a three-tier client-server architecture. With this design, the client software and processing resides on one machine, your SAS server application resides on another, and database system resides on a third. This design offers many advantages if you can afford the extra SAS application server.

In summary, you must strive to reduce the transfer of data across the network or modem and attempt to give the work to the machine with the muscle. The basic concept is quite simple, but applying the idea often requires some extra time and analysis at the beginning of the project.

DEVELOPER TIPS

Tip #1: Develop in a vacuum

Just because you are developing a client-server application does not mean you need to use a real server to develop your application. Although it works better with a multi-tasking operating system, such as Windows NT, you can simulate a client-server environment using a single workstation, running software such as Windows 95 or Windows NT.

In simple terms, you can run the client application and the server application on the same machine. For example, to run both the SAS/CONNECT client and the SAS/CONNECT server on the same Windows NT Workstation using TCP/IP, do the following:

1. Install SAS/CONNECT on the workstation
2. During installation, be sure to install the "SAS job spawner" service.
3. If you are not on a network, you will need to activate your systems HOST file to make sure that your host name, specified in the REMOTE= option, can be resolved to an IP address. On Windows NT, you can edit your host file in `C:\winnt\system32\drivers\etc\host`. Be sure the host file is active and your host name has been assigned to an IP address.
4. Edit the script file provided by SAS in `c:\sas612\connect\saslink\tcpwin.scr`. You should verify that the command to start SAS is correct for your system.
5. Verify that the SAS job spawner service is running. To view the services, open the Control Panel and double click on Services.
6. Start SAS.
7. Submit the options statement:
`OPTIONS REMOTE=hostname COMAMID=TCP;`

Where hostname is the name you assigned to your IP address in the HOST file.
8. Submit the SIGNON command using the appropriate TCPWIN script. Note that it is more convenient to allocate the script to a filename of RLINK, so you can simply issue the SIGNON; statement with no other options.

Tip #2: Centralize your start-up options

Whether you're writing a client-server application or not, isolating host specific options, such as libref and fileref assignments, in one location for easy modification makes sense and should be a standard practice. We typically write our applications so they can run "real" client-server or "simulated" client-server by just setting an option in the autoexec or other startup logic.

In this situation, you must also identify libraries on the server side, so your startup logic should also provide for changing these values depending upon your client-server setting. If you designed both sides of the application (client and server), the paths may be the same for simulated and real, but you should plan for them to be different.

For example,

```
%global remstat remname accmeth signscr;
%let remstat=YES;
%let signscr=&approot.meridian.scr;
%let remname=meridian;
%let accmeth=tcp;
```

where remstat indicates whether remote connection should be available, remname is used in the REMOTE= option and accmeth is used in the COMAMID= option.

Tip #3: Check for connections

In addition, we have developed standard FRAME entries, and now classes, that perform all of the signon functionality, so they can be called anywhere within the application. We develop these methods for any activity that relates to establishing or ensuring a connection to the server.

Using these methods, it becomes a simple matter to test for a connection at any point when the application is running. If the connection is lost, you can easily attempt to reconnect by calling the reusable methods.

Tip #4: Diagram your application

We all know we should diagram our applications, but we often rush to start coding and forget to diagram the system before we start. With client-server applications, diagramming is even more important and rewarding.

The classic data flow diagram has survived all of the paradigm shifts we have experienced over the last 20 years. Even object oriented programmers still use some form of data flow diagram. By diagramming a specific program or query without regard to client-server design, you easily see where it makes sense to split the processing between server and client.

Figure 1 shows an example data flow diagram for a three-tier system. The dotted line indicates the boundary between the client, application server, and database server.

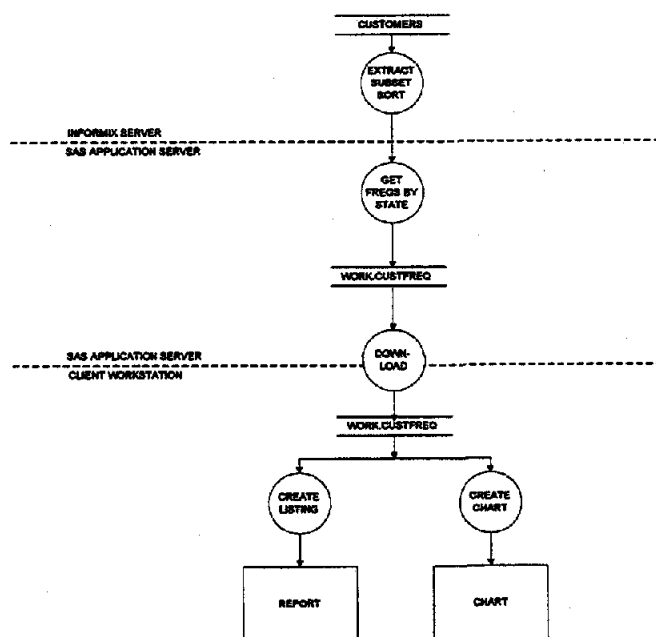


Figure 1 Data Flow Diagram

Tip #5: Use SAS to analyze the impact of your database on performance

If your data is only a vision by the time you are designing the application, you can forecast the size of the data tables you will be working with. Using basic math, you can estimate the range of a typical query result and apply it to your data flow diagrams. Refer to "Optimizing Client-Server Performance through Data Warehouse Design" by Eric C. Brinsfield for more detailed examples of this type of analysis.

If you have actual data already, you can use the FREQ, MEANS, and UNIVARIATE procedures to analyze the number of rows you will be dealing with at almost any summarization level. Again, by applying these numbers to your data flow diagrams, you can predict the sizes of most of your intermediate data sets created by your program.

Tip #6: Minimize transfer of data across network

As I mentioned before, if network transfer rates are low, try to minimize the amount of data you transfer. Using the results of your analysis in Tip #5, your choices should be clear.

For example, SAS/GRAPH programs often create very large GSEG catalogs, which would transfer slowly across the network. As an alternative, you can generate the necessary data for the graph on the server side, but generate the graph on the client side. This offers the added advantage of placing the graph generation on the machine with the target device.

Besides reducing the amount of data transferred across the network, splitting graph programs makes it easier to produce graphs that match the devices you will be viewing or printing on.

For example, the following code could be submitted from the SAS Display Manager program editor or from a submit block in SCL.

```

Rsubmit:
  %inc remsrc(graph1);
  proc download data=grdata out=grdata
    status=no;
  run;

  %sysrput gstatus=&gstatus;
endrsubmit;

%inc appsrc(graph2);
  
```

All code between RSUBMIT and ENDRSUBMIT executes on the server. The SAS program graph1.sas resides and executes on the server machine, while graph2.sas resides and executes on the client machine. There are ways to improve the maintainability of this type of processing, but this code demonstrates the concept.

%SYSRPUT passes the macro variable gstatus back to the client SAS session. The program graph1.sas sets a value for gstatus to indicate the success of processing. Your client-side application can react accordingly after checking &gstatus.

Tip #7: Understand the difference between Remote Compute Services and Remote Library Services in SAS/CONNECT

SAS/CONNECT software provides two primary client server tools: Remote Compute Services and Remote Library Services. Remote Compute provides the programmer with the capability to submit code to the server as well as the client processor. To send processing to the server issue the RSUBMIT command or RSUBMIT statement. Using remote submits and local submits, you have complete control.

With Remote Library Services, you can establish a link to a remote library (a library on the server) with a LIBNAME that is defined to the client SAS session. This is a very useful feature, but beware.

With Remote Library Services, your programs still run on the client machine, while the data allocated with a remote link resides on the server. You may initiate transfers across the network unexpectedly. For example, if you submit the following code locally (to run on your client workstation):

```

Libname master '\\server\master'
remote=meridian;
  
```

```
Proc sort data=master.accounts;
  By name;
Run;
```

The SORT procedure will read the data from master.accounts, move the data to utility files in your WORK directory, and replace the data set master.accounts. In this case, however, your WORK library is located on your workstation or possibly another server, which means the data is automatically transferred across the network twice, once after the read and again during the write.

The key recommendation is think about where the processing will occur before you use remote library services in a program. Consider using remote LIBNAMEs only for interactive browsing and editing or when you are intentionally moving data from the server to the client.

The following code:

```
Libname master '\\server\master'
remote=meridian;

Proc copy in=master out=work;
  Select accounts;
Run;
```

is essentially the same as using PROC DOWNLOAD to transfer the file to your work library on the client.

Tip #8: Use the muscle where it is available

Evaluate the size and load of your servers as well as the size and power of the client workstations that will be used to run the application. If you work in an environment where everyone has a powerful workstation but the servers are overloaded or under-powered, consider writing your application to distribute the work to the workstations.

Tip #9: Use the software where it is available

SAS Institute offers several pricing plans for their software, but they all depend on the operating system and expected number of users (estimated by the size of the processor). As a result, if you develop a client-server application that uses a large Unix server, the price of SAS on the server will be much higher than the price on the workstation.

For example, if your application will share a server with several other non-SAS applications, which need a very powerful machine, your SAS cost on that server will probably be higher than it should be, given your expected usage. Depending upon the number of users on the client side, you may find it more cost effective to license most of the analytical software for each workstation rather than on the server.

With base SAS, SAS/CONNECT, and possibly SAS/ACCESS on the server, you could distribute all of the other processing to the individual workstations. Note that this is not the ideal arrangement. It is always nice to have all of the SAS products on all machines, but if the cost becomes too high, consider this option. Be sure to analyze it first and call SAS Institute for alternative pricing options.

Tip #10: Consider using three tiers

As mentioned above, using three tiers (client – application server – database server) provides a very appealing design for client-server applications. Not only does it improve performance by splitting the application processing from the database server, but the SAS license will be determined by the size of the application server and not the database server.

As an example, the following code, submitted from the client side, will execute on the SAS application server (middle tier), extract data from the database server (third tier), and download the data to the client (first tier).

```
rsubmit;
  proc sql noprint;
    connect to informix(db=merdb);
    create table custlist as
    select * from connection to informix
    (select *
     from customers
     where state = "&state"
     order by city
    );
  run;
quit;

proc download data=custlist out=custlist
status=no;
run;
endrsubmit;
```

Note that, in this example with Informix, the Informix ESQL/C runtime tools must be installed on the SAS application server.

The three-tier architecture will play a very important role in Web-based client-server applications. I will discuss this further in my follow-up paper referenced earlier.

SUMMARY

Client-server applications come in many flavors. Some are more complicated than others, and therefore require more analysis before proper design can be achieved. SAS/CONNECT provides you with the tools to control exactly where and when any process occurs. As a result, you can use the methods discussed in this paper to simplify your development as well as improve the performance of your application.

REFERENCE INFORMATION

Eric Brinsfield, President
Meridian Software, Inc.
12204 Old Creedmoor Road
Raleigh, NC 27613

Phone: 919.518.1070
FAX: 919.518.1170
email: merecb@meridian-software.com
WWW: www.meridian-software.com

BIBLIOGRAPHY

"Optimizing Client-Server Performance through Data Warehouse Design", Eric C. Brinsfield, Proceedings of the SESUG97 Conference, 1997, Jacksonville, FL

ACKNOWLEDGEMENTS

Cathy Brinsfield of Meridian Software, editing and formatting.
Henrietta Cummings of Meridian Software, technical review.

TRADEMARKS

SAS® and all SAS products are trademarks or registered trademarks of SAS Institute Inc.

Windows 95® and Windows NT® are registered trademarks of Microsoft Corporation.