SAS® Client-Server Development: Through Thick and Thin and Version 8

Eric Brinsfield, Meridian Software, Inc.®

ABSTRACT

SAS Institute has been a leader in client-server technology since the release of SAS/CONNECT software and the SAS/ACCESS Pass-Through Facility. With the release of Version 8 and new developments in the database and telecommunications industries, client-server applications have moved into the next generation. This paper will explain the principles of good client-server design with a specific focus on SAS application development, thin-client architectures, and three-tier optimization. In addition, the paper will cover how client-server design options differ between Version 6 and Version 8 of SAS software.

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. At last year's SESUG conference, I suggested several general ways to improve the performance of your client-server designs (Brinsfield[1]). I also shared 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. In this discussion, I will review basic client-server issues and then focus on thin clients and SAS Version 8.

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 database technology may be evolving to the point where developers do not need to worry about it. 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, 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 PROCC 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.

In a fat client application, moving the work to the workstation is quite simple. In a thin client application, you may not have that option. In the future, however, thin clients may evolve into virtual fat clients, because as data transfer rates increase, so can the size of the program that is downloaded to your thin client dynamically. Even though the application does not permanently reside on your machine, the code may run there.

In either case, 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 scream continually 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 and falls under the fat client architecture.

§ 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. Pushing your joins and queries to the database rather than performing them on the SAS fat client workstation, reduces data transfers and improves performance.

In SAS Version 7 and 8, SAS now optimizes access to databases automatically. Under performance optimizations, SAS Version 8 documentation claims that SAS/ACCESS now "passes joins directly to the relational DBMS for processing". You may still want to benchmark your queries during development to determine exactly

where that applies.

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. I will discuss three-tier architecture more later.

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

At SESUG 98, I offered ten tips for client-server development. In this paper, many of the tip names are the same, but the tips themselves have been upgraded or changed to address thin client architecture or Version 8 features.

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. Last year, I described how to setup SAS/CONNECT on a standalone workstation. The same tricks can be used for thin client SAS/IntrNet applications or SAS/ACCESS applications.

For example on Windows NT, you can install Microsoft® Peer Web Server and SAS/IntrNet software on the same workstation. Once you have defined all the connections, you can develop and test your thin client applications without connecting to anyone else.

Also, most database vendors now provide a "workstation" version or "developer" version of their database software. If you need to develop a client server application that accesses database tables, you can run the database software on your own workstation. The SAS code that you develop will be identical, no matter where the database resides. Only your connection parameters will change and those can be conveniently stored in a central piece of code.

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.

As I suggested in the last tip, you can centralize your database connection strings also. For example, if you were using ODBC to access a Microsoft SQL Server™ database, you might connect to the database like this:

```
Proc SQL;

connect to ODBC (

"DRIVER={SQL Server}; SERVER=mersrv;UID=userid;

PWD=XX; DATABASE=dbname );
```

If you create a System DSN named MERSRV in the ODBC Data Source Administrator, you can simplify the connection string to:

```
Connect to ODBC("dsn=mersrv; uid=userid; pwd=xx");
```

By using the ODBC Data Source Administrator, you can isolate the connection information, so it can be changed easily without modifying any SAS code.

To centralize your final connection string, you can store it in a file that is retrieved by your application or you can simply set it in your startup macro or startup code. Once that is done, you can create methods in SCL or macros that automatically retrieve the connection string at the appropriate time. For example, a PROC SQL startup might look like this:

```
Proc Sql;  %connect(odbc);
```

Where the %connect macro retrieves the complete connection string from a text file that can be easily modified and builds the correct syntax for the connect statement.

Tip #3: Check for connections

If you were using SAS/CONNECT in SAS releases prior to Version 7, you may have encountered problems with lost connections to the SAS/CONNECT server. To minimize disruption to our users when that happens, we developed standard FRAME entries, and now classes, that perform all of the signon functionality. They can be called from anywhere within the application. We develop these methods for any activity that establishes or requires 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.

In Version 8, SAS/CONNECT will automatically issue a SIGNON statement if a connection does not exist when you issue an RSUBMIT. The SIGNON statement will use the current global settings for SAS/CONNECT. By default, RSUBMIT will automatically sign off at the end of the RSUBMIT, but you can set option PERSIST=YES to disable that default.

Unfortunately, "autosignon" has not been implemented for use with remote library services. So, if you are issuing LIBNAME statements with the REMOTE engine, you will still need to develop code to check for the connection and you will need to set PERSIST=YES to make sure that RSUBMIT does not disconnect your users automatically.

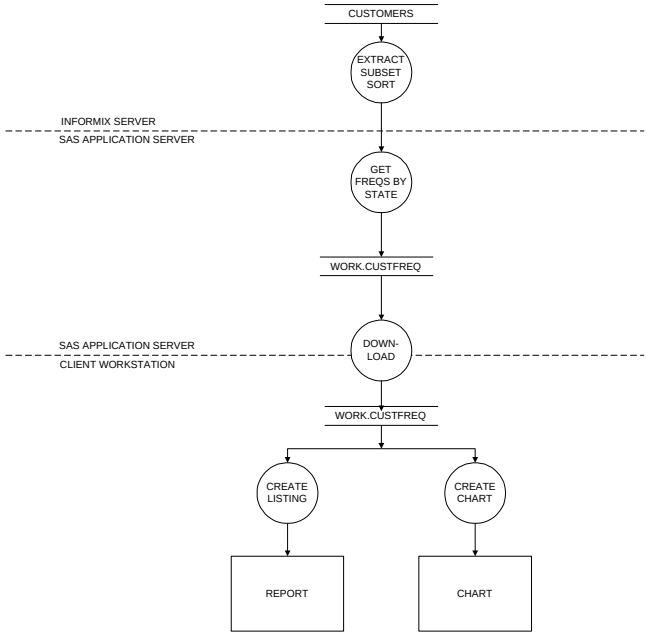
Tip #4: Diagram your application

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.

Note that with a three-tier application, you can think of the relationship between the database server and the application server as a fat-client architecture. In other words, all of your fat-client experience can be applied to the back end of a three-tier design.

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.



1

2 Figure 1 Data Flow Diagram of a three-tier system

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

If your data is only a vision at the time you are designing the application, you can still 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*" (Brinsfield[2]) for more detailed examples of this type of analysis.

If you already have actual data, 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 with SAS/CONNECT

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=grd out=grd 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: Minimize transfers across the network by using DBMS features

Study your DBMS documentation with the goal of finding tools that you can use to push more of the processing to the machine where the data resides. In other words, try to process and reduce the data on the machine where it is stored, so you do can minimize the volume you transfer.

Stored Procedures

For complicated processing, you can often find existing stored procedures that are provided with the DBMS software or you can create your own. With stored procedures, you can push frequently executed processes to the database server. As a simple illustration of calling a stored procedure from SAS, the following code renames a table in SQL Server:

```
Proc sql;  %connect(odbc);

Execute  (sp_rename table1,table2, OBJECT) by odbc;
```

Where sp_rename is a documented stored procedure available within SQL Server. Everything inside the Execute ( ) is submitted to the DBMS for interpretation and processing.

Temporary Tables

Another technique for reducing data transfer, involves the user of temporary tables. Many DBMS packages permit users to create temporary tables for personal use that are automatically deleted when the user disconnects.

You can take advantage of these temporary tables by using them as temporary lookup tables or tables that must be accessed multiple times. Rather than bringing the tables back to the SAS work directory on your workstation, use them on the database server. For example, in SQL Server you can use the following code to create a temporary table called #nccusts, that will contain a list of social security numbers.

```
execute

  (select ssn

         into #nccusts

         from customerlist

         where (state = 'NC' )
```

```
  ) by odbc;
```

You can then use #nccusts as a selection list for multiple queries to the database. For example, one query that is used to create a SAS data set call NCPURCH might look like this:

```
Create table ncpurch as

     Select * from connection to odbc

     (select *

         from purchases

         where ssn in (select ssn from #nccusts)

         order by purchasedatetime desc

       );
```

The point of this example is not to show you how to perform this specific query, but to demonstrate how to create a temporary table and use it without bringing the data (in this case nccusts) back to your SAS session first. This technique also comes in handy for avoiding unnecessary data conversions between SAS and the DBMS, such as date conversions between SAS and SQL Server. If you do all of your date comparisons within the DBMS, you do not have to convert.

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. In most cases, however, upgrading your servers is a better long-term solution.

In the multi-tier architecture, you may just need to move the hard work to the server that has the muscle. This will always be true for the thin client application.

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 on 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.

With new thin client alternatives, the best strategy will be the opposite of the last recommendation. Depending on the workload on your servers, you could license all of the SAS products on one application server, using SAS/Intr-

Net to pass requests to the SAS server. Those users that do not use SAS for any other purpose than to access your application would not be required to license SAS on their workstation.

In Version 8, you can also utilize the new SAS Integration Technologies product. Similar to SAS/IntrNet, this new product enables you to develop user interfaces using other development tools, such as Microsoft Visual Basic, that can communicate with SAS on the application server.

Tip #10: Use three or more 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(odbc);
      create table custlist as
      select * from connection to odbc
         (select *
          from customers
          where state = "&state"
          order by city
          );
   run;
   quit;

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

The use of PROCC DOWNLOAD is not always necessary. You can also use remote library services to read only the table directly from the server machine, if the data set resides in an allocated remote library.

The three-tier architecture will play a very important role in Web-based client-server applications as I have suggested in all of my earlier examples. All of the same client-server rules apply to thin client applications. They just get applied to a different part of the architecture with the goal of minimizing how much data is transferred to the Web browser.

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, SAS/ACCESS, SAS/IntrNet, and the new SAS Integration Technologies products all provide you with the tools to control exactly where and when any pro-

cess 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

[1]Brinsfield, Eric 1998. SAS Client-Server Development: Tricks of the Trade. Proceedings of the SESUG 98 Conference, Norfolk, VA.

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

ACKNOWLEDGEMENTS

Cathy Brinsfield of Meridian Software, editing and formatting.

TRADEMARKS