

SAS® as a Compute Server for Java

Andrew A. Norton, Trilogy Consulting Corporation, Kalamazoo, Michigan

JAVA FOR SOFTWARE DISTRIBUTION

The Internet provides a revolution in connectivity: any machine can connect to any other, without special setup.

Java extends this revolution to software distribution, an area that received little previous attention. It is now possible to provide applications to anyone anywhere, without prior setup and on any platform. These applications will execute on their machine.

This is the true significance of Java. It is a quite nice object-oriented programming language, with strong typing and garbage collection. But more importantly, it has been designed to be distributed across the Internet, efficiently and safely.

A look back: Client-server architecture

A few years back, Client-server was the hot new thing. Client-server split processing between two machines, the "client" (typically a PC running a GUI application) and the "server" (typically a mainframe running a DBMS). The interconnection was SQL-based, such as ODBC.

There could be many client machines attached to a server machine, because the client GUI interacted with humans and only occasionally needed to query the database server. Client-server architecture let the GUI run on a dedicated PC, giving quick response time and attractive PC graphics and user interfaces. At the same time, large databases could be shared between multiple users, letting them all see up-to-date information and avoiding the cost of replicating this data resource.

A further advantage was scalability. With a centralized system (such as on a mainframe), when the capacity was reached, the entire system would need to be upgraded. With a client-server system, if you needed more clients, you simply plugged them in. If you needed to upgrade the server, you could keep your existing client investment.

Remote Compute Services

As I mentioned before, the most common use of client-server architecture was SQL-based data servers, as in ODBC or SAS/CONNECT® Remote Library Services. But SAS Institute and others also provide remote compute services.

The general premise here is to reduce the size of the data transfer by bringing across the network only the answer you need, rather than the raw data. A secondary advantage is that you can use the resources of the server machine (such as fast I/O of large volumes of data or special software).

Of course SQL provides the ability to do some computations on the host machine, such as summarizations and subsetting. But compute services such as the RSUBMIT feature of SAS/CONNECT allow developers to specify entire complex programs to be run on the server machine, and to return results other than data sets (such as graphs or reports).

There is also another advantage, which we will hear more about later. It is easier to develop and maintain a complex application if it is split into several smaller parts. There is a natural separation

between the presentation of results and how those results are computed. Compute servers (and similar mechanisms such as SQL views or stored procedures) allow business logic to be defined and maintained separately from the application presentation. In object-oriented terminology, this is called "encapsulation." The presentation side does not need to know how the data is computed, and the business logic side does not need to know what will be done with the results. In fact, the same compute server may be used to support different related applications.

Problems with Client-Server

What were the disadvantages of the traditional client-server architecture and what does Java do to address them?

In the case of SAS and other popular client-server tools, special software is sold and installed on the client side. In addition, the application of interest is installed on the client, often by sharing the code by means of a local area network (LAN).

Another disadvantage was that the client-side software often is restricted to particular platforms (such as Microsoft Windows only). This problem occurs even with SAS, because although SAS/AF applications could operate on virtually any platform, they required porting (i.e., a recompilation) when moved.

Here comes the Internet

The Internet lets you connect up to sites anywhere in the world, simply by typing a URL address. You can make applications available to whomever you wish, without advance setup, simply by authorizing access.

The original applications were HTML pages, simple static displays. But the technology quickly evolved towards the capabilities that had become familiar through client-server architecture.

First was the dynamic generation of HTML pages based upon parameters selected by the user, either by running programs through the Common Gateway Interface (CGI) or by using a HTML generator such as Cold Fusion in conjunction with an ODBC-compliant database (such as SAS). These approaches still kept virtually all of the processing on the server side.

Then came ways of executing code on the client side. An early approach was "plug-ins," downloading programs to be run by a client-side processor such as SAS. These worked well, but there were certain disadvantages. First, the user would need to buy and install the plug-in. In the case of SAS, the cost was significant and the installation effort also deterred casual usage. Second, there were potential security holes. SAS (as one example of many) did not distinguish between programs downloaded from a stranger and those written by the end-user. So a hostile or erroneous program could be run in a plug-in and take actions such as reformatting your hard disk.

Universal access requires tools that are universally available. By the same token, if you are going to be downloading programs constantly and are unfamiliar with their contents, security is a major concern. And there should be no difficulties if you want to use a platform different from that for which the application was originally written.

Java as a universal and safe platform

Sun originally developed Java for television set-top controllers, but then realized it was just what the Internet needed. In many ways, it resembles SAS, but has some strategic advantages for Internet use.

The structure of Java resembles the familiar "pyramid" structure of the SAS system. SAS has a large applications area built upon a smaller core; the core is in turn implemented using the host kernel, which is only a small percentage of the total and the only piece that needs to be reimplemented for a different platform. Similarly, Java is made up of classes, the language itself, and the Virtual Machine. Only the Virtual Machine and a few platform-specific classes need to be implemented for a different platform. When you compile a Java program, it does not compile to machine language but rather to "bytecodes" which are run on the Virtual Machine. So when you download a Java class, it is ready to go regardless of the current platform.

Here we see the first differences: the Java runtime is distributed for free and is small enough to be easily downloaded. And it is widely available, incorporated into Microsoft and Netscape browsers and the forthcoming releases of many operating systems. So we can count on it being available.

The other big advantage is not so visible. Java is designed to run safely, to let you connect and run stranger's programs without fear of viruses, and run your own programs without fear of damaging bugs.

To do this, it first makes a distinction between programs that you explicitly install on your machine ("applications") versus those that are downloaded automatically ("applets"). Built into the Virtual Machine is the ability to check the actions of all applets against a security manager, so that applets cannot read or write to local resources (disks or printers), start new processes, and can only connect back to the machine they came from.

Java has received a great deal of attention in the past two years, and its weaknesses are being remedied rapidly.

1. You might think that Java is slow because the bytecodes are interpreted by the virtual machine. There are virtual machine implementations available from several vendors, and the newer ones use "just-in-time" compilation to machine language.
2. Suppose you use an applet over and over again. Surely you don't want to download the code repeatedly. Netscape caches your applet code in the same way it caches your HTML and images. In addition, "push" products such as Marimba's Castanet cache Java code on your machine and update it automatically.
3. Downloading applets over the Internet takes extra time now because each class is downloaded separately (in the same way that each image is downloaded separately on a Web page.) In the next release of Java (1.1) it will be possible to bundle everything you need into a single JAR file (similar to compressed ZIP files.)

Client-Server, Again

The first popular Java applets were small stand-alone programs, typically animations. But Java applets can connect back to the host, and also interact with the user. So the forthcoming generation of Java applets brings back the client server architecture, but with a dynamically downloadable client.

Suppose you want to provide your application to the world (or at least your world, you can restrict access using firewalls). The Internet can be quite slow, so you don't want to download more than necessary. All the client really needs is the user interface and the ability to connect back to the server. The database can stay on the server side, and respond to requests from each of many clients.

There is a design issue here: a tradeoff between download time and network traffic. There is not a single answer what is worthwhile to transfer to the client side and what to keep on the server side. It depends on the network speed, how often the application is used, how focused the target of interest is. Marimba's Castanet product can also be used to maintain distributed data; for example you could send people a CD to get started and then use automated periodic Castanet updates to keep it current.

WHAT DOES SAS OFFER?

If we use Java on the client side, what should we use on the server side? Well, we could always use Java. But our options are broader than that, since the server is not subject to the security restrictions of client applets. We could use any ODBC-compliant database. Or we could use SAS.

So why use SAS as a server for Java? For one thing, in our shop and probably many others, we have more skilled SAS programmers than Java programmers. Java is a difficult language with a substantial learning curve (although the task is eased by Java code generators such as Symantec's Visual Cafe).

Yes, we could implement everything in Java, but why? SAS has built-in capabilities that would be difficult and time-consuming to duplicate: data management, statistics, graphics, report writing. Let's let Java do what it does best (portable graphical interfaces) and SAS do what it does best.

Why not just use a database management system? SAS can process ODBC queries, but so can many other systems. There are three answers to this:

1. SAS is specifically designed primarily for read-only data warehouse use, and thus may be faster, take less storage space, or be cheaper than alternatives emphasizing transaction processing.
2. We need to not only respond to ODBC queries, but also prepare and manage the data. SAS gives us alternatives in addition to SQL such as the DATA step potentially easing this work.
3. There is more to life than SQL. SAS can generate graphs for us, compute statistics, and so forth.

Of course, SAS is not always the answer. Sometimes you need a full DBMS for extensive multi-user transaction processing.

TYPES OF SERVERS

SAS can be used to connect up to the web in three ways:

1. Publishing: HTML pages and graphics images are produced in advance, then delivered to the user as requested. This limits the amount of information that can be provided, as everything has to be precomputed whether it is used or not.

2. **Data Server:** SAS provides data to Web applications such as Cold Fusion through ODBC or its Java variant, JDBC. This is nice so far as it goes, but SAS is limited by the ODBC interface to the same features as any other SQL database.
3. **Compute Server:** The user specifies programs to be executed and parameters for those programs (for example from an HTML form or a Java applet). SAS executes the specified programs and dynamically produces results which are then delivered to the user. SAS as a Compute Server provides access to the full power of the SAS system.

KINDS OF COMMUNICATION

We can only use SAS in ways that SAS allows us to communicate with it. So the choice of communications method dictates what we can do. Furthermore, we need to consider both the capabilities for submitting programs to SAS and for retrieving the results.

Sockets

Both Java and SAS can read and write streams of characters to "sockets". So if you want to pass some information to SAS, you can write text from Java, and return text from SAS. On the SAS side, sockets appear as special kinds of files, manipulated with the PUT and INPUT statements.

Sockets can be read and written by DATA steps or by SCL. A loop continues to read the socket, pausing at each INPUT statement until a record is available. There must be some convention for signaling that a connection should be closed.

Only one user can be handled at a time. Multiple users could be handled with multiple SAS sessions.

An added advantage of sockets is that they are available in base SAS with no additional licensing fee required.

Output could be written to external files and then picked up by Java.

JDBC (SQL)

For several years SAS has provided ODBC server support. ODBC is a standardized interface based upon SQL for querying and manipulating relational databases. It includes provisions for retrieving the resulting data sets.

JDBC is a variant of ODBC specifically designed for Java. SQL statements can be submitted to the JDBC server (i.e., SAS). The results are returned one observation at a time, in sequence. Within each observation, variable values can be retrieved by random access according to variable position or variable name. There is also support for metadata information such as variable type and length.

Sun provides a JDBC-ODBC bridge that makes use of existing ODBC drivers. Presumably it is somewhat slower than direct JDBC support because it entails an additional layer of processing.

SAS Institute similarly provides a JDBC driver which receives JDBC requests and relays them to a SAS/SHARE server.

JDBC lets Java access SAS data server capabilities. You can submit a SQL query and retrieve the results. You can also add, delete, or modify rows of the table using SQL.

SAS jConnect

Suppose you want to do more than SQL. Perhaps you want to run a statistical procedure such as PROC GLM, or a DATA step. Or perhaps you want to generate a graph using SAS/GRAPH and return that image to Java.

SAS jConnect lets you open a remote SAS session on the host and submit SAS code. On the host side it looks like a SAS/CONNECT session.

SAS jConnect does a great job of submitting SAS code, but does not do much to address how information is returned. Information is returned in the form of SAS Log and Output window lines. If the result is a SAS data set, it may be possible to retrieve this using JDBC. There are two possible complications here: First, that you must be sure the job has completed before retrieving the result; Second, that the JDBC server will be a separate SAS session that must be able to access the data set (so you will need SAS/Share if the JDBC server cannot obtain exclusive access to the data set).

The Output Delivery System (ODS) planned for version 7 fits in nicely here, because output from statistical procedures will be available in the form of data sets.

Suppose the output is a graph or some other kind of output than a data set. You can write the output to an external file, and retrieve that from the host using Java. One possible complication is multiple users (see below).

One disadvantage of SAS jConnect is that the initial connection must wait for a new SAS session to start up. (This is also an advantage, of course, because each user gets a clean session dedicated to them.) An alternative is the SAS/IntrNet Application Dispatcher.

SAS/IntrNet Application Dispatcher

In the olden days (last year), SAS was executed by means of the Common Gateway Interface (CGI). Parameters (such as the name of a macro) would be passed through an HTTP request. The Web server would invoke a program (often written in Perl) that would strip out the parameters and invoke a SAS program. The SAS program would generate HTML that would then be passed back to the client.

Java can submit HTTP requests and receive the returned HTML. So this method would allow SAS programs to be run on request. And if a Java program is doing the receiving rather than a browser, the returned text need not be HTML at all, but could be any kind of data.

The disadvantage of the CGI method is the time delay required to start a new SAS session every time there is a request. So the new SAS/IntrNet product provides an improved way of handling this called the Application Dispatcher. A CGI program still receives the HTML request, but instead of launching a new SAS session, it connects to an existing SAS session via sockets. It can launch programs stored as external files, macros, or SCL methods. Unlike jConnect, the Application Dispatcher cannot launch ad hoc programs.

Application Dispatcher has several advantages beyond jConnect. It can direct traffic to several different SAS compute servers, depending upon the request. And it automatically keeps track of multiple users for you, so the outputs don't get misdirected.

CORBA

With jConnect and the ApplicationDispatcher we now have ways of submitting SAS code for execution. But we still do not have ways of communicating directly with SCL objects. Java is an object-oriented language, yet we are converting out of object-oriented form when we enter SAS. It would be preferable to send messages directly to SCL objects and receive responses just as if they were Java objects. This is what CORBA does.

CORBA connects up objects across different machines as if they were on a single machine, and connects objects in different languages. You can even pass objects as parameters from one language to another.

In a sense, sockets, JDBC, jConnect, and ApplicationDispatcher are all subsets of CORBA, for they can all be expressed in the form of messages sent to a SAS session. But CORBA allows any message to be sent, and results to be returned in the structured form of objects rather than text streams.

SAS does not yet support CORBA, but SAS Institute has hinted that this will be a path for the future. It is possible to use CORBA to connect clients and servers, then connect up to SAS locally.

SECURITY

I previously discussed how Java protects against hostile action against clients by downloaded applets. Now let's talk about attempts to disrupt servers.

There are attempts to perform unauthorized activities on other's computers, ranging from inspection of private files to damage of those files to execution of programs. The safest route would be not to have any connections to the Internet at all. But what we would like to do is keep the bad guys out while letting the good guys in, and this is much trickier.

Publication of static web pages could be safely accomplished by simply installing those pages on an isolated machine. But nowadays we want to provide access to databases and to enable running programs on the server.

Firewalls

Firewall programs inspect the requests made from the Internet and determine which to allow. They check the source of the requests and the nature of the requests. For example, they may allow HTTP requests but disallow Sockets requests. Unfortunately, we may have chosen to implement our client-server system using a forbidden access method, or selected a product that itself was implemented in such a way.

It is not always easy to determine exactly what the nature of the request is. Suppose we develop a Sales Brochure Request System for the public using sockets. When we get a socket request, how can we be sure that it is a Sales Brochure Request and not something more damaging? After all, sockets is a very general and powerful mechanism.

Similar problems arise with other access methods. HTTP is often accepted by firewalls when other access methods are not. So there are a variety of products that perform "tunneling", i.e. they convert a request to HTTP, pass it across the firewall, and convert it back. Of course there is a performance price to pay.

Host connectivity

As I mentioned before, Java distinguishes between applets (which can be automatically downloaded and run) and applications (which must be explicitly installed by the user). Applets run in a "sandbox" that limits their activities. One crucial limitation is that applets can only connect to the host from which they were downloaded.

But suppose the SAS server is not the same as the Web server. This is actually the usual configuration: Web servers need to be able to respond to a large volume of requests quickly, so they usually pass off other requests to other servers. This provides better scalability because you can add additional servers as needed.

How does the applet connect to the compute server? It needs to connect indirectly, through a routing program on the host. Every message goes from the applet to host router to computer server or follows the same path back.

Remember the SAS Application Dispatcher: The CGI program resides on the host and relays messages to the SAS server. The same configuration applies to Symantec's DB Anywhere: The DB Anywhere server resides on the host and connects to data sources elsewhere. Similarly, Visigenic's Visibroker (a CORBA implementation) resides on the host and relays messages between the applet (a CORBA client) and the CORBA server.

But sockets and jConnect do not have this architecture and thus applets are limited to connecting to SAS sessions on the host. You could build a repeater to route socket traffic to another server, however.

MULTIUSER ISSUES

It is not safe to assume that a solution that works for a single connection will work for multiple users. Suppose you want to execute a SAS job that will create a graph to be returned to the Java applet client. Since Java can read a file on the host machine, you would think that a SAS job could simply create a GIF file, and then Java read that file.

But if there are multiple users this could potentially get confused. How do you ensure that the file you are downloading was in fact created by the SAS job you submitted? One safe approach is to write the output into a directory dedicated to that client. This is the approach that the SAS/IntrNet Application Dispatcher uses. If you write your own solution (for example, using sockets) you will need to address the same issues.

Because the Application Dispatcher is HTML-based, another issue arises. HTML has no sense of history. How do you know whether a user is still connected? To handle this properly, old directories are deleted after a certain period of time (perhaps thirty minutes).

LICENSING ISSUES

OK, anybody can type in a URL and start using SAS. So you only need to buy one copy of SAS for the entire company, right?

Not so fast. Obviously SAS Institute would have a few problems with this scheme. For the official word you will have to contact your SAS Institute sales representative, but two fundamental concepts are user versus server based licensing and internal versus external use.

On what basis do you determine software usage fees? At BOF workshops at SUGI, three approaches have been discussed: charge by the user, charge by the capacity of the machine, or charge by the actual usage. The first two approaches have been implemented by SAS Institute.

On platforms designed for multiple users, server-based licensing is used. On UNIX or MVS, for example, you are charged according to the power of the machine. You are welcome to use the machine for as much of a SAS load as it will carry, and you pay the same price even if you only use SAS rarely. SAS Institute has no problem with Intranet connections to such a machine, because you've paid for it. But notice that server-based licensing discourages casual or experimental use, and discourages using multiple servers.

PC platforms such as Windows 95 are licensed with the expectation that these are indeed single-user machines. This is called user-based licensing.

SAS Institute is also concerned with who the users are. SAS is licensed to particular companies. If you are going to open up usage to the public, SAS Institute will review your plans and determine a price.

FUTURE DEVELOPMENTS

Trilogy's research and development into connections between Java and SAS continues. Next year at the SAS Users Group International Conference I will be presenting a tutorial titled "SAS and Java for Interactive Graphics."

ACKNOWLEDGEMENTS

SAS, SAS/AF, SAS/CONNECT, and SAS/IntrNet are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.

AUTHOR CONTACT

If you would like further information please contact:

Andrew A. Norton
Trilogy Consulting Corporation
5278 Lovers Lane
Kalamazoo MI 49002
(616) 344-4100
aanorton@trilogyusa.com