

The SAS® System: A Complete Client/Server Solution

SAS Institute Inc., Cary, NC
Cheryl A. Garner & Stephanie Reinard

INTRODUCTION

Applications of all kinds have four basic elements. First there's the need to access data. Then there's a need to manage data — to get the data in the right format for further investigation. There's also a need to analyze data, applying logic in an effort to translate raw data into useful information. And finally, there's a need to present the information obtained from analysis.

The access to the data and the processing involved in managing and analyzing the data can be distributed across platforms in a variety of ways. The following sections describe the types of services provided by the SAS System to enable optimal distribution of these activities. These services include compute services, data transfer services, and a new capability offered by the SAS System - remote library services. Guidelines for choosing the right service or combination of services are also given along with further usage details.

SERVICES FOR IMPLEMENTING DISTRIBUTED APPLICATIONS

The SAS System provides two sets of services to enable the development of distributed applications. The first is compute services; these services give you access to all of the computing resources on your network. The second is remote data services; these services give you access to all of your data regardless of where they are stored. Most often, the best implementation of a distributed application incorporates a combination of compute and remote data services.

Compute services enable utilization of remote computing resources, including hardware, software, and data, to most efficiently execute an application and to maximize the use of all computing resources. The SAS System enables you to move any or all portions of an application's processing to a remote machine to take advantage of remote hardware resources, to utilize software available in the remote environment, to interface with existing legacy systems, and to execute against the remote copy of the data. The results of the remote processing are then returned to the local machine. This is useful when the remote machine has hardware and/or software resources available to more efficiently perform the task at hand. It can also be preferable if the amount of data to be processed is too large to be moved to the local machine or if it is updated too frequently for a local static copy to be useful.

Remote data services enable access to data stored in a remote environment. This access may be as a single user of the data or as multiple users simultaneously accessing the data. It may be to data stored in SAS data sets, external databases, SAS catalogs, or external files. This capability can be further divided into *data transfer services* and *remote library services* (RLS). Data transfer services provide a method for moving a copy of the data from one machine to another machine where a physical copy is created. Subsequent local processing takes place against the local copy of the data without generating further network traffic until you decide to update the copy of the data with another transfer. RLS provide access to remote data libraries and move the data through the network as the local execution requests it. A copy of the data is not written to the local file system and the data must pass through the network on any subsequent use by the local processing.

These services are by no means mutually exclusive. The degree to which each is applied and the most beneficial combination of

services is completely determined by the needs and goals of each individual application. Each set of services has a fairly defined set of benefits and drawbacks. By matching these characteristics to the characteristics of your data and your application needs, you can determine how to best combine these services in order to write the most efficient distributed application.

IMPLEMENTATION CONSIDERATIONS

COMPUTE SERVICES

The *compute services* available in the SAS System can give you easy access to many of the remote resources on your network from a single local SAS session. This results in maximum utilization of all of the computing resources within your organization. Figure 1 illustrates the concept of compute services.

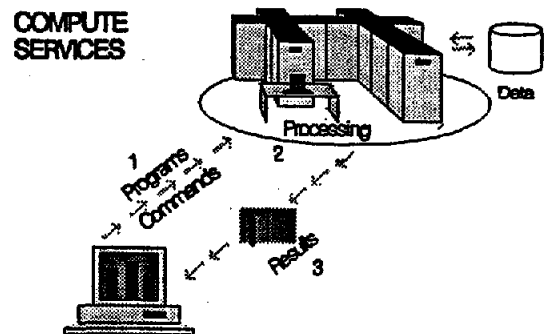


Figure 1 Compute Services

1. Programs and commands are sent from the local machine to the remote machine.
2. The processing takes place on the remote machine.
3. The log, list, and graphical output from the remote execution are returned to the local machine.

Compute Services - Advantages

One or more remote machines on your network may have vector processors or other larger, faster hardware resources that would more efficiently accomplish the CPU intensive portions of your application. Compute services allow you to move any or all segments of an application to a remote machine for execution. The results of the remote execution are returned to the local SAS session to be used with further local processing and/or managed by the local graphical user interface (GUI).

In addition, a remote machine may have peripherals attached that would otherwise be inaccessible to you on a stand-alone local system. A PC in another part of your organization may have a plotter or printer attached that you need to use. By simply directing your local SAS session to move the graphics processing to the remote PC and pointing to its output device, you can get a hard copy of your graph from the remote plotter.

Data center rules or data characteristics may mandate a single centralized copy of the data needed by your application. By moving the processing to the remote system where the data reside, there is no need to transfer or create additional copies of

the data. This can address security needs as well as enable access to data sources that are too large or dynamic to be transferred.

Compute Services - Side Effects

Compute services, by definition, require remote CPU cycles. Because the processing takes place on the remote machine, it does require CPU cycles from that machine. This could be a drawback if the goal is to offload work from the remote system. The remote CPU impact could be minimized by utilizing data transfer services to transfer a copy of the data to the local machine and maintaining the processing on the local CPU. Impact could also be minimized by combining compute services and data transfer services. Refer to the section below, "Combining Compute Services and Data Transfer Services," for more information.

Compute services also create network traffic in order to return the results of the processing back to the local machine. If the processing needs to be repeated frequently, the best solution would be to utilize data transfer services to transfer the necessary data to the local system and maintain the processing in the local environment.

Another side effect of compute services is that remote production data systems will still be impacted as the data is subsetted or pre-processed. This can be minimized by careful implementation of your application to eliminate or reduce multiple passes of the data.

REMOTE DATA SERVICES

There are many applications, or parts of applications, that require processing to be done in the local SAS session, but the data required by that processing reside on a remote file system. In this case, it is necessary to move the data to the local processor for execution. There are two types of services available with the SAS System that can remove the boundaries of your organization's heterogeneous data sources and enable remote data access: data transfer services and RLS.

Data Transfer Services

For many applications, *data transfer services* offer the maximum benefit. Figure 2 illustrates the concept of data transfer services.

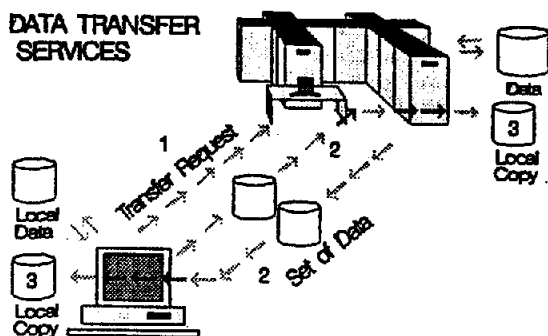


Figure 2 Data Transfer Services

1. The local session sends a transfer request to the remote session.
2. The request may be either to copy the data from the local session to the remote session or to copy the data from the remote session to the local session.

3. The copy of the data is then available for processing on the target system without further network traffic.

Data Transfer Services - Advantages

A major benefit of data transfer services is to offload work from a remote system to one or more local machines and boost response time for production systems running in the remote environment. Once the data have been downloaded to the local machine, all subsequent data access and processing is done by the local processor. By moving a copy or subset of the original data to the local machine, applications can be run on the local machine without further remote CPU consumption or impact on remote data systems.

Moving a copy of the data to your local system adds robustness to your decision support environment. In the case of a network failure that would temporarily eliminate access to the remote data, you can continue working with your local copy of the data.

You can transfer only the data that you need by using WHERE processing to dynamically subset the data as it is being transferred to the local machine. This reduces network traffic and gives you only the data you need on the local system.

Data transfer services not only facilitate moving data from a larger source to a requesting local workstation, but also support the model of a centralized control point, such as a mainframe, initiating communication to a network of workstations. This enables centralized distribution of data and/or applications. Automated jobs can be set up to distribute data and/or applications during non-peak hours to multiple machines that have need of the data and/or applications for the next day's work. Likewise, jobs can be set up to query a network of workstations for the purpose of gathering data and storing them in a centralized repository.

Yet another reason to utilize data transfer services is for backup purposes. Data and/or applications can be copied from one system to another system with more memory resources to provide a backup in case of loss on the local system.

In addition, by moving a copy or subset of the data to the local system, applications can be developed and tested without further dependence or impact on remote system response time.

Data Transfer Services - Side Effects

Transferring a copy of the data to another file system creates multiple copies of the data. The remote data may be updated so frequently that it is not feasible to keep a local copy of the data current enough to be useful. Also, security restrictions at your site may prohibit multiple copies of the data. In this case, consider compute services if the amount of data involved is large. If the amount of data is small to medium, RLS allow the processing to take place on the local system and the data to come from a remote source as the execution requests it.

If the volume of the data that would need to be moved is too large, then data transfer services alone are not sufficient to meet your needs. The increased network traffic as well as the time it would take to transfer the data may be unacceptable for your application. This scenario needs the functionality of compute services to either move all of the processing to the remote system to execute against the remote data source or a combination of compute services and data transfer services. Refer to the section below, "Combining Compute Services and Data Transfer Services," for more information.

Data transfer services are less transparent. Transferring the data

is an explicit step that must be done prior to initiating any local processing. If the volume of data to be accessed is not large and your application requires more transparent access, RLS are a better solution.

Remote Library Services

With the addition of RLS, the SAS System now provides a comprehensive set of tools for implementing a distributed application with the greatest flexibility and efficiency. RLS provide access to remote data as if they were in a local library. Figure 3 illustrates the concept of RLS.

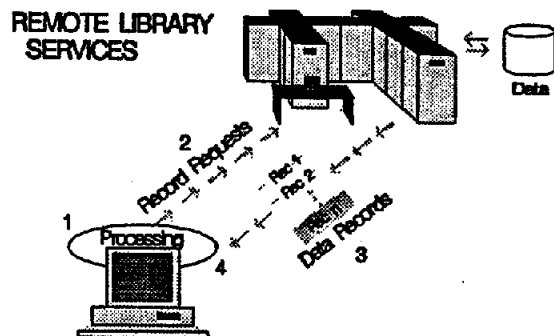


Figure 3 Remote Library Services

1. The processing takes place in the local session, which is unaware that it is accessing remote data versus local data.
2. The local session passes record requests to the remote session.
3. Records are returned from the remote session to the processing program of the local session.
4. This cycle continues until the processing is complete.

Remote Library Services - Advantages

RLS enable you to access remote data in a multi-user environment through the use of a multi-user server. This gives a big boost to productivity by allowing multiple users to simultaneously access and update the same data source while guaranteeing the integrity of the data.

Now it is also possible to perform updates to data in external databases. RLS enable a single user accessing data stored in an external database to perform updates to that data through the use of the single-user server. This permits updates of remote database data as a result of local processing.

If you need to maintain a single copy of the data on the remote machine and keep the processing on the local system, then you need RLS. This is the superior choice if the amount of data needed by the local processing is small, the remote data are very frequently updated, or your data center rules prohibit multiple copies of data.

RLS also provide more transparent data access. This eliminates an explicit step of coding an upload or download of the data before processing. It also permits the GUI of an application to reside on the local system while the data remain in the remote environment (for example, a local FSEDIT of a remote data set). And, it allows applications to be built that provide seemingly identical access to local and remote data without requiring any knowledge on the part of the enduser as to where the data reside.

Remote Library Services - Side Effects

Accessing data through RLS is not efficient for large amounts of data. The data must still move from the remote machine to the local machine, but the data are not written to the local disk. If the local processing is against large amounts of data, it is usually better to use compute services to move the processing to the remote system rather than transfer all of the data.

Similarly, RLS are not efficient for multiple passes of the data. If the procedures being run make multiple passes of the data or the entire procedure must be run more than once against the data, it would be better to transfer a copy of the data to the local machine. Then you would incur the network traffic cost once rather than paying the cost on each pass of the data.

Response time may be degraded by accessing remote data that are being simultaneously updated by other users. If this impact cannot be tolerated, you may need to transfer a copy of the data to the local system and remove the live data from your application.

Careful thought must be given to the following: the amount of data that will be accessed by your application, whether you need multi-user or single-user data access, whether your application will be making multiple passes of the data, and the resulting effect on your network load. Analyzing these criteria will help you determine when to utilize RLS and when to utilize data transfer or compute services.

COMBINING COMPUTE SERVICES AND DATA TRANSFER SERVICES

Several of the scenarios described above suggested a combination of compute services and data transfer services as the optimal choice for portions of a distributed application. The combination of these services provides tremendous flexibility and efficiency to any distributed application.

Your application will benefit from combining these services if you need information from data that are stored on a remote system and you do not want to move a copy of the data to the local system. Reasons for not moving a copy of the data could include the following: the amount of data is too large, the data are frequently updated, or you want to avoid data duplication. Regardless of the motivation for reducing the amount of data that is transferred, incorporating compute services will achieve your goal. Compute services enable you to format and preprocess data into a subsetted or summarized form on the remote system prior to transferring the subsequent smaller amount of data to the local platform. This balances the use of CPU cycles between the local and remote systems and minimizes the amount of data contributing to network traffic.

REMOTE LIBRARY SERVICES IN DETAIL

RLS is made up of a *remote engine* that executes in the local or client session and a *server* that executes in the remote or server session. This allows SAS procedures and data steps that execute in the local session to access SAS data sets in a remote data library just like they would in a local library. Figures 4 and 5 show the difference between the implementation of a LIBNAME statement that accesses a local SAS data library and a LIBNAME statement that accesses a remote SAS data library.

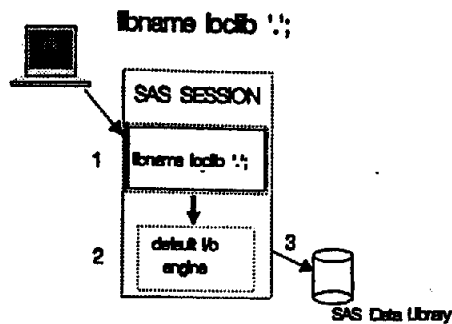


Figure 4 *LIBNAME Statement for Local Data Library*

1. A **LIBNAME** statement is submitted to the local SAS session specifying a libref and a local SAS data library.
2. Data requests are passed to the default (or specified) I/O engine.
3. The I/O engine obtains the results of the data requests from a local SAS data library

Now compare this to a **LIBNAME** statement that accesses a remote SAS data library.

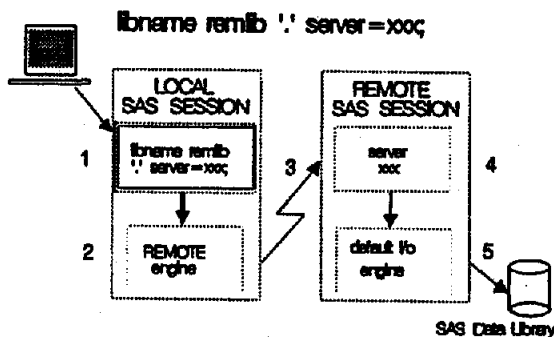


Figure 5 *LIBNAME Statement for Remote Data Library*

1. A **LIBNAME** statement is submitted to the local SAS session specifying a libref, a remote SAS data library, and a **SERVER=** option.
2. The **SERVER=** option tells SAS to pass data requests to the remote engine.
3. The remote engine sends data requests to the server. The server is identified by the value of the **SERVER=** option on the **LIBNAME** statement. The server is probably running in a SAS session on a remote machine, but this is not a requirement.
4. The server then administers the requests to SAS files on behalf of the local/client SAS session. These requests are serviced by the default (or specified) I/O engine.
5. The I/O engine obtains the results of the data requests from the remote SAS data library and the results are returned through the remote engine.

The SAS System provides two types of servers: a single-user server and a multi-user server.

The Single-User Server

The single-user server executes in a remote SAS/CONNECT session. This server is controlled by and dedicated to the local user of a SAS/CONNECT conversation and provides that user with transparent access to remote SAS data libraries. Because this type of server is dedicated to a single user, it can be used to update data stored in external DBMS's.

In order to use a single-user server, you must be signed on to the platform where the remote data reside. You then locally submit a **LIBNAME** statement to specify a remote SAS data library, giving the remote session id as the value of the **SERVER=** option, as shown in the following example:

```
SIGNON <rssid>;

LIBNAME <libref> <REMOTE> <'SAS-data-library'>
      SERVER=<rssid> <engine/host-options>;
```

(Note: If you have a multi-user server running on the remote machine, you do not need to sign on in order to access a library through this server. In fact, you would not want to incur the overhead of signing on if you only need to access the multi-user server in the remote environment.)

The single-user server is initialized in the specified remote SAS session upon execution of a **LIBNAME** statement with a valid remote session id as the value of the **SERVER=** option. The access method specified to establish the SAS/CONNECT conversation through **SIGNON** is also used for communicating with the single-user server.

In the following example, a **SIGNON** is executed to a remote UNIX node named remnode. Then, a **LIBNAME** statement pointing to a directory on remnode is submitted with the remote session id of remnode specified as the **SERVER=** value. This results in the initiation of a single-user server on remnode and the libref 'a' being defined to this server running on remnode.

```
options comamid=tcp;
filename rlink 'tcpunix.scr';
signon remnode;

libname a '/usr/local/prod' server=remnode;
proc datasets lib=a;
quit;
```

The libref 'a' can then be used in subsequent local execution to access data in the remote library without first moving a physical copy of the data to the local session. Note that the data must still move through the network to the local session in order to be processed. However, the data are not written to disk to produce a local copy of the data.

The Multi-User Server

A multi-user server is initiated by a server administrator and left running to service data requests from multiple users. This type of server provides concurrent update access to remote SAS data libraries and manages that access to ensure the integrity of the data.

In order to access a remote library through a multi-user server, you submit a LIBNAME statement that specifies the remote server name for the SERVER= option as in the following example:

```
options comamid=tcp;
libname remlib 'sasdemo.edu.screens'
server=server1;
```

Using WHERE Statements

When using RLS, one way to reduce the amount of data that needs to move through the network to local processing is to use WHERE statement processing on the local procedures and/or data steps. Whether you are accessing a single-user server or a multi-user server, the WHERE statement is passed to the server's execution and evaluated on the remote side so that only those data meeting the selection criteria are moved across the network to the local session. If the data that you are accessing are stored in an external database, the WHERE statement is passed to the database and evaluated if possible. If the data base cannot complete the evaluation, the server completes the evaluation before returning any of the data to the remote engine in the local session.

Accessing External DBMS Data

RLS via the single-user server provide update access to data stored in external databases. The SAS/ACCESS® engines as well as the SQL engine all recognize the single-user server as one user and therefore enable update mode for external database sources. However, these same engines prohibit update access to external databases when using RLS with a multi-user server. Updating is prohibited because it would be impossible for either a multi-user server or a data base product to detect conflicting requests from multiple users and therefore impossible to guarantee data integrity and security or to provide audit trails.

Using SAS Data Views

You can also use RLS with SAS data views. SAS data views include SAS/ACCESS views, DATA step views, and PROC SQL views. A view is a SAS data set that describes other data. It is processed as a data set by an engine that reads the underlying data and uses the description to return the data in the requested form. This process is called view interpretation.

When the library that contains the view is accessed through a server, the view is interpreted in the server's session by default. This means that the engine is loaded and called by the server to read and transform the underlying data and a minimum of data is moved through the network. The local processing is unaware that a view is involved.

If the local and remote machine architectures are the same, you can cause the view to be interpreted in the local session. This is done by specifying RMTVIEW=NO on the LIBNAME statement used to define the remote library. If the architectures are not the same, views can only be interpreted in the remote session.

Interpreting a view as data can produce significant processing demands. When a view is interpreted locally, that frequently means that a lot of data has to flow to the local session. This removes processing demands from the server session and may be desirable in order to increase server response time. In the case of a single-user server, the server is dedicated to you so tuning server response time is not an issue. In the case of a

multi-user server, however, you often must balance network load with server response.

Security Considerations

When using RLS to access a single-user server you do not face any additional security exposure. You must sign on to establish a connection to the remote environment before issuing a LIBNAME to define a remote library. Because the SIGNON command requires a valid userid and password to gain access to the remote environment and establish the connection, you are provided the same level of security that you would have by locally logging on to the same system. You also have the ability to restrict access to members of SAS data libraries, including SAS data files, SQL views, SAS/ACCESS views, and data step views, by assigning passwords to them. You can assign passwords to all member types except catalogs. Once the password is assigned for a certain level of protection, you must specify the password to perform any operation that level of protection prevents. You can specify three levels of protection: read, write, and alter. Also, RACF or any other security features inherent in the target system still apply to RLS. Therefore, whether you are accessing a remote database, remote SAS data sets, or remote views, you have the same level of security with a single-user server that you would with local access to these data sources.

When using RLS to access a multi-user server that is running on any operating system other than MVS or CMS, you are more vulnerable accessing your data because you do not go through any login process to validate yourself to the remote system. However, there are methods for restricting access not only to a server but also to the data that are available to a server.

You have the ability to run secured multi-user servers on both MVS and CMS because both the APPC and TCP/IP access methods have the capability to run secured on these operating systems. Multi-user servers in these environments run secured by default. This requires any user trying to access the server through a LIBNAME statement to specify a valid userid/password combination for the remote system. If the specified userid/password is valid for the remote MVS or CMS system, you are effectively "logged in" and can now access the server.

If your multi-user server is running on an operating system other than MVS or CMS, there are several security measures that can be implemented. (The following measures can also be used with servers on MVS and CMS to provide security in addition to the secured access methods.) You can limit access to a server by specifying a user access password (UAPW) during server initialization. This requires the user to specify the password on the LIBNAME statement used to establish communication to the server. You can also limit the data that are available to the server with the NOALLOC option during server initialization. The NOALLOC option prevents users from defining libraries to the server and restricts users to accessing only those libraries defined by the server administrator. Finally, of the data available to the server, you can limit the type of access to that data by specifying passwords for the individual files, as described above.

Tuning Guidelines

The first rule of thumb is to run your application with all of the default settings and determine if there are performance problems. This will also provide a baseline for comparison if you determine that you need to tune the environment.

It is possible to tune the number of observations that are transmitted in each exchange with a server; this is called multi-observation buffering (MOBbing). By default, the server does MOBbing when a file has been opened with an access

pattern and mode of output or sequential input. The following paragraphs use the term "open mode" to indicate the way that a particular file is opened: input, output, or update. The term "access pattern" is used to indicate the way that a particular file is being accessed: sequential, random, BY-group rewind, or two-pass sequential.

The TOBSNO= data set option specifies the number of observations to be transmitted in each multi-observation exchange with a SAS server. However, the ability and/or desire for the server to do MOBBing is dependent on the open mode and access pattern for the file being requested.

If the open mode is anything other than UPDATE and TOBSNO is set, then the specified number of observations will be transmitted in each exchange with the server. It should be pointed out that if the server is a multi-user server and the access pattern is non-sequential, for example with FSBROWSE, it is not advised to use MOBBing because the data source may have been updated by the time you scroll to the subsequent observations from the buffer. Also, things like FSBROWSE are generally driven by human interaction and do not suffer the performance exposures that a programmatically driven procedure would suffer. However, if your server is a single-user server, all data access will be limited to one user and probably more sequential so it may be more beneficial to use the TOBSNO option.

If this option is not specified, its value is calculated based on the observation length and the size of the server's transmission buffers. A server's default transmission buffer size is 32K. If the server is a multi-user server, the transmission buffer size is set with the TBUFSIZE= option on the PROC SERVER statement. Be sure to remember that modifying the TBUFSIZE= option will affect all users of the server. If the server is a single-user server, the default buffer size is used and cannot be altered. Once the value has been calculated the lesser of the following three numbers will be used to determine how many observations to send in one exchange with the server: 100 observations, 2/3 of the data set, or the calculated value.

If a file is opened in UPDATE mode, only one observation will be returned per exchange with the server. This is because there may be multiple people updating the data source (if the server is a multi-user server) and since only one observation can be locked at a time, there would be no way to guarantee data integrity if an observation were sent unlocked in a MOB to more than one user. So with something like FSEDIT, there is no way to force MOBBing. This also applies to a single-user server.

The TOBSNO= option is valid only for data sets opened for input or output and accessed through a server via the REMOTE engine. If this option is specified for a data set opened for update or accessed via another engine, it is ignored. PROC COPY and the DATA step SET statement (without POINT= or KEY=) typically use sequential access. PROC FSEDIT and PROC FSBROWSE typically use non-sequential access.

Restrictions of Remote Library Services

There are some restrictions on what RLS can provide and how these services are provided. Most of these restrictions exist when the local/user session and remote/server session are running on machines that have differing internal representations of data, and the remote library access crosses machine architectures. The following paragraphs detail the current limits and give alternate solutions when possible.

The peer-to-peer access methods are the only access methods with the ability to support RLS. They include:

- APPC
- DECnet
- NETBIOS
- TCP

RLS support for the terminal-based access methods such as ASYNC, HLLAPI, and TELNET is not planned. If you must use an access method that is not supported, you should use compute services or data transfer services to implement your application.

Currently, with respect to RLS, access to data in remote SAS data libraries is limited to SAS data sets when the local system and the remote system have different data representations. These data sets include views that are interpreted in the server session. Therefore, if your local application requires remote SAS catalogs or utility files, you must use compute services or data transfer services to implement your application. We hope to enable remote library access for SAS catalogs in a future version of the system.

Character variables are assumed to contain printable character data and numeric variables are assumed to contain numeric data in floating point format. Storing binary data in character variables or any other unusual use of variables must be avoided when accessing data sets across machine architectures. Character conversion is performed on character variables and numeric conversion is performed on numeric variables. Any unusual data will produce unpredictable results from the translation process.

Especially with numeric variables, translation from one representation to another may alter the value of the variable. A common type of alteration is loss of precision. This occurs when the source representation has more bits of mantissa than the target representation. No warning is produced due to loss of precision during translation. Another type of value distortion is loss of magnitude. This occurs when the source representation has a greater exponent range than the target representation, and a value with a magnitude lying in the excess range of the source representation is converted. A warning is produced when this type of alteration occurs.

SAS data sets, typically on IBM mainframe machines, that may be accessed cross-architecture should not include two-byte numeric variables. Due to their underlying floating point representation scheme, most hosts that SAS supports have a minimum numeric variable length of three. A data set containing a length two variable cannot be accessed with RLS from such a host. You should use compute services or data transfer services in order to access a data set with two-byte numeric variables.

If you want your local SAS session to access both a multi-user server and a single-user server, choose a name for your remote SAS/CONNECT session that is different from the name of the multi-user server you intend to access. Each time a LIBNAME is submitted, the SERVER= value is first checked to see if it matches any currently signed on remote session ids. If there are no remote sessions that match the value of the SERVER= option, it is assumed to be a LIBNAME to a multi-user server. Therefore, single-user and multi-user servers must have unique names. If the names are the same, the single-user server will always override the use of a multi-user server.

A CASE STUDY - STAR AUTOS DEMO

A prototype has been developed of the systems that would be needed to conduct business in a car rental agency. The hypothetical car rental agency, Star Autos, is a franchised

business. The Headquarters for the franchisor, Star Autos, Inc., are in Chicago, IL. Star Autos, Inc. provides vehicle purchasing, marketing, and computer services to franchisees throughout the country. The franchisee owns and maintains cars and local offices, purchasing the franchise name and services from Star Autos, Inc. Both the Headquarters and each of the franchise offices have data and distributed applications that must be implemented and integrated in order to meet their business goals.

Data and Application Requirements of Headquarters Office

Among the application systems provided by the Headquarters are a central reservation database, a corporate reporting system, and a distribution system for distributing reservations and reports to each of the franchise offices on a nightly basis.

The reservation database contains reservations for all of the franchise offices. It is simultaneously updated by multiple travel agents as they add new reservations. It can also be asynchronously accessed by each of the franchise offices for transaction type requests for a specific reservation as well as for adding a new reservation being made at the franchise counter. The Headquarters' reservation database is also utilized by the franchise offices for *ad hoc* report generation based on total reservations and also rental type.

The corporate reporting system generates production reports in order to compare the performance of the franchise offices. These reports give information on predicted revenue based on the number of reservations, total reservations, and on rental types that are currently reserved. The reports generated by this system are stored in external text files.

The distribution system implements an automated method for sending pertinent reservations as well as selected reports to each franchise office on a nightly basis when system load is light and rates are lower. The reservations for each franchise location are copied to the local office each night so that each office will have a current copy of its reservations at the beginning of the next day's business. Each franchise maintains a data set that specifies a set of reports that the franchise is interested in receiving. Part of the distribution system is to query this data set and distribute only the requested reports to each individual franchise.

Data and Application Requirements of Franchise Offices

Several functions have been distributed to the local franchise offices in order to ease the system load on the mainframe at Headquarters and to improve response time for the customer by locating the data at the point where it is used - at each franchise office counter. The systems that are in place at the individual franchises include a contracts system as well as a reporting system.

The contracts system includes a reservation database and a contract database that exist on the workstation at the franchise counter. When a customer needs to check out a car, the local reservation database is queried for the customer's reservation. If it is found in the local database, the information is completed and the record is stored in the contract database. If the reservation is not found in the local data base, a query is made to the Headquarters' master reservation database and, if found, the information is entered and the record is stored in the contract database. If no reservation exists, a blank entry screen is presented, the information is completed, and the record is stored in the contract database. In addition, if the customer wishes to return the rental car in a city other than the origin of rental, the contracts system transfers the completed contract to the destination city so that the information is available when the customer returns the car to the final destination.

The reporting system includes methods for accessing production reports that are produced on a regular basis by the Headquarters office as well as a method for generating *ad hoc* reports which require company-wide data maintained on the Headquarters' mainframe. Each franchise maintains a control data set that contains, among other things, the specific production reports that are important to that office. The reporting system first checks for the existence of a requested production report. If the previous night's run was successful, the report will be on the local system, and it will be displayed. If the report is not found, processing will be invoked on the mainframe to generate the report, the report will be moved to the local workstation, and then displayed. The *ad hoc* portion of the reporting system produces graphic output to compare the performance of this franchise to that of the other franchises within the company with respect to reservations as well as rental types. The graphs must be displayed locally but must be run against the master reservation database because it is the only source of information on all of the franchise offices.

Analysis of the Distributed Application Implementation

The applications just described were implemented in a two-phase process. In the first phase, the implementation was done using compute services and data transfer services. The second phase made use of RLS where appropriate. The following sections focus on several small pieces of the distributed application and discuss how they were implemented with compute services and/or data transfer services, and why each piece was or was not converted to make use of RLS in the second phase of the implementation.

• The Best Solution - Remote Library Services

First, we'll take a look at the contract system running at the local franchise office. Specifically, we'll look at the situation of a reservation not found in the local reservation database and the process of querying the Headquarters' master reservation database and finding the reservation there.

The following fragment contains SCL program statements that were coded in the first phase of the implementation in order to extract a single reservation from the master reservation database.

```
submit continue;
  rsubmit &rsessid;
  /* define remote shared library to */
  /* remote SAS session */
  options comamid=xms;
  libname master "sascgg.sugi18.library"
        server=mvs.c02xhsrv;
  /* transfer only requested reservation */
  /* to local temp ds */
  proc download data=master.reserv
        out=finlcall status=no;
        where resnum='&resnum'; run;
  endrsubmit;
proc append base=s18.reserv data=finlcall;
run;
endrsubmit;

/* open local reservation database */
wherecls = "resnum=" || "" || resnum || "";
dsid = open("s18.reserv");
call set(dsid);
rc = where(dsid, wherecls);
rc = fetchobs(dsid, 1);
```

Data transfer services are used in this implementation via the PROC DOWNLOAD statement. The DOWNLOAD procedure, in this case, makes use of a WHERE statement to copy only the requested reservation to the local workstation where it is then appended to the local reservation data base. The record is copied to the local system because it needs to be displayed in a local data entry window and then saved into the local contract database. WHERE processing is used to minimize the amount of data being transferred through the network and to provide the optimal response time to the customer waiting at the counter. Compute services could not address the needs of this part of the application because the remote data needs to be displayed and edited by the local GUI. Therefore, the processing must remain on the local system.

Now, let's re-examine this implementation with respect to RLS. The following SCL fragment represents the second implementation of this piece of the application, which takes advantage of RLS.

```
/* define remote shared library to local */
/* SAS session */
libname master "sascgg.sugi18.library"
              server=mvs.c02xhsrv;

/* build and apply where clause to speed */
/* up retrieval */
wherecls = "resnum=" || '*' || resnum || '*';
/* open remote Headquarters database */
dsid = open("master.reserv", 'u');
call set(dsid);
rc = where(dsid, wherecls);
rc = fetchobs(dsid, 1);
```

In this implementation, a LIBNAME statement is issued in the local SAS session to define the remote library containing the Headquarters' reservation database. Note that the remote library is being accessed through a SAS/SHARE server. The reservation is then moved to the local session (but not written to disk) when the SCL FETCHOBS function is executed. Note the WHERE and FETCHOBS function calls after the OPEN of the master reservation database.

The use of these functions is very important because it positions the remote file on the desired record. If a LOCATEC function were used in place of the WHERE and FETCHOBS functions, each reservation would have to be transferred to the local session and compared against the specified reservation number during the execution of the LOCATEC function, until a match was found. This would result in very poor performance and a misuse of RLS. This illustrates why it is important to remember that the data must still move through the network with RLS and that the capability should only be used on small to medium amounts of data. It is preferable to use RLS in this case rather than data transfer services because we are moving a small amount of data, and it can be locally edited and saved into the local contract database as needed.

• The Best Solution - Data Transfer Services

The next part of the application to be examined is the distribution of selected reservations from the master reservation database to each of the local franchise offices. The implementation involves the mainframe initiating a conversation with each of the franchise workstations and moving selected reservations to each workstation.

The following program enables distribution of reservations to a franchise office by using a WHERE statement to select the desired reservations. The actual application was implemented using the macro facility and a control data set to provide remote session ids for each franchise. The macro made it very easy to distribute reservations to any number of franchises as well as to add or delete franchises as needed. This example represents one iteration of the macro.

```
/*-----*/
/* Name: DISTRSV.PROGRAM */
/* */
/* This program distributes the */
/* reservations data set to the */
/* franchise offices. */
/*-----*/
INIT:

submit continue;
signon atlanta;

rsubmit;
  libname mres "d:\counter";
  libname backup "d:\counter\backup";

/* upload all reservations for a */
/* particular location */
rsubmit;
  proc upload data=hq.reserv
    out=update status=no;
    where origin="Atlanta";
  run;

/* sort uploaded ds for merge */
proc sort data=update;
  by resnum;
run;

/* backup existing data set */
proc copy in=mres out=backup;
  select reserv;
run;

/* merge new & existing data sets */
data mres.reserv;
  update mres.reserv update;
  by resnum;
run;
endrsubmit;
signoff;
endsubmit;
```

In the first phase of the implementation, moving a copy of each office's reservations to that office is clearly a better choice than maintaining the single centralized copy on the mainframe and requiring the franchises to access this copy of the data either through terminal emulation or by transferring single reservations many times throughout the day. The goals of reducing the mainframe workload and improving customer response time, the need to manipulate the reservation data in a local GUI, and the probability of little or no change to a reservation once it has been entered all support creating a full copy of each office's reservations on the local workstation.

When considering RLS for this part of the application you need to

consider the goals of offloading work from the mainframe, and how often and how much data access is involved. Each franchise is expected to frequently query and update reservation data throughout the day. When you multiply this by the number of franchise offices, the network overhead of RLS causes an unacceptable response time for the customer waiting to check out a car. Also, if the connection to the mainframe were temporarily lost, the franchise counter operations would be crippled. Clearly, it is still desirable to employ data transfer services to create local copies of the reservations designated for each office.

• The Best Solution - Compute Services

The next area to be considered is the *ad hoc* reporting part of the local franchise reporting system. The *ad hoc* reports require graphic output to highlight the performance of a specific office with respect to the rest of the franchise offices.

This sample segment of SCL code moves all of the processing, the PROC SORT and PROC SUMMARY preprocessing, and the actual graph generation to the remote mainframe for execution by the faster hardware and against the remote data. The graphic output is then displayed on the local workstation.

```
submit continue;
rsubmit &rsessid;
  proc sort data=master.reserv
    out=tmp; by origin;
  run;
  proc summary data=tmp vardef=n
    noprint; by origin;
    output out=tmp2;
  run;

  title 'Total Reservations';
  pattern1 value=solid color=green;

  goptions dev=grlink ftitle=centx
    ftext=simplex htitle=3;
  proc gchart data=tmp2;
    block origin /sumvar=_freq_
    noheading; label origin='00'x;
  run; quit;
endrsubmit;
endsubmit;
```

The initial implementation of this portion of the application fully exploits compute services. The only database containing information on all of the franchise offices resides on the Headquarters' mainframe. The mainframe can also provide the computing power necessary to process the large amount of data needed to produce the graphs. However, we want the graphs to be displayed on the local workstation and possibly saved for later reference. Compute services are the logical choice over data transfer services because the amount of data needed to generate the graphs is too large to be transferred and you do not need or want a copy of the entire reservation database on the local system.

Even for phase two of the implementation, when we consider the usefulness of RLS for this portion of the application, compute services still have a clear advantage. The large amount of data involved is an obvious indication that RLS are not the right choice. Therefore, compute services give the maximum benefit to this segment of the application.

• The Best Solution - Data Transfer Services Combined with Remote Library Services

The final portion of the application that will be discussed is the distribution of production reports from Headquarters to each of the franchise offices based on the information contained in the control data set maintained by each of the franchise offices. The implementation involves the mainframe initiating a conversation with each of the franchise workstations and transferring a set of reports to the franchise based on selection criteria.

The following program distributes reports to a franchise office. This is accomplished by reading a Headquarters' data set that provides the network location of a franchise, querying that office's control data set, and generating a text file with the statements necessary to transfer the requested reports. The actual application was implemented using the macro facility and a control data set to provide remote session ids for each franchise. The macro made it very easy to distribute reports to any number of franchises as well as to add or delete franchises as needed. This example represents one iteration of the macro.

```
/*-----*/
/*
/* Name: DISTRPT.PROGRAM
/*
/* This program runs on the HQ
/* machine and distributes reports
/* to the franchise offices.
/*
/*-----*/
INIT:

submit continue;
signon atlanta;

x "alloc fi(xferrpt)
  da('sasccg.sugil8.xferrpt') shr";

rsubmit;
  filename xferjob "xferjob.sas";
  libname rpt "d:counterreports";
  filename frptlib "d:counterreportsprod";

  data _null_;
    set rpt.preport end=finish;
    file xferjob;
    if _n_ = 1 then
      put "rsubmit;";

  /* only transfer desired reports */
  if (copy = "Y") then do;
    put "proc upload
      infile='sasccg.sugil8.name';
      put "outfile=franch(" name ")
        status=no;run;";
  end;
  if finish then
    put "endrsubmit;";
  run;
  /* download file to %include at HQ */
  proc download infile=xferjob
    outfile=xferrpt status=no;
  run;
endrsubmit;
```

```

/* upload desired reports */
%include xferrpt;

signoff;
endsubmit;

```

This program uses a combination of compute services and data transfer services. The DATA step to determine the reports that are being requested and to generate the PROC UPLOAD statements to transfer the reports is executed on the franchise system. This execution belongs on the franchise system. That is where the control data set used for report selection is maintained by each franchise, and there is no need to move these data sets to the mainframe system. The statements to transfer the reports are written to a text file which is then downloaded to the mainframe, invoked, and the appropriate reports are then distributed to the franchise office.

While the above implementation accomplishes the desired task, RLS offer a simpler, more efficient solution. Consider the control data set on the local franchise workstation. The amount of information needed by Headquarters from this data set is small - the value of one variable for no more than a dozen records. Headquarters does not need a physical copy of the data set, but only some information to be used in subsequent processing. Also, the subsequent processing - creating a file with the statements for actually moving the reports to the franchise - is much better appropriated to the mainframe to avoid having to transfer this file back to the mainframe for execution. For all of these reasons, RLS are utilized in the second implementation of this segment of the application, shown below.

```

/*-----*/
/* Name: DISTRPT.PROGRAM */
/*
/* This program distributes reports */
/* to the franchise offices by */
/* making use of RLS. */
/*-----*/
INIT:

submit continue;
signon atlanta;

x 'alloc fi(xferrpt)
   da('sascgg.sug118.xferrpt') shr';

rsubmit;
   filename frptlib 'd:counterreportsprod';
endrssubmit;
libname rpt 'd:counterreports'
          server=atlanta;
data _null_;
   set rpt.preport end=finish;
   file xferrpt;
   if _n_ = 1 then
      put 'rsubmit;';

   /* only transfer desired reports */
   if (copy = 'Y') then do;

```

```

put 'proc upload
    infile='sascgg.sug118.'name'';
put 'outfile=frptlib('name')
    status=no;run;';

end;
if finish then
   put 'endrssubmit;';
run;

/* upload desired reports */
%include xferrpt;

signoff;
endsubmit;

```

THE PRODUCTS THAT PROVIDE THE SERVICES

There are two products within the SAS System that provide the services described in this paper: SAS/CONNECT® software and SAS/SHARE® software. The following sections describe which product provides which service(s) along with a brief description on how each service is provided.

SAS/CONNECT Software

SAS/CONNECT software is defined as our primary cooperative processing product for multiple CPU environments. It allows the SAS application developer to optimize computer resources by choosing the appropriate CPU on which to execute the various components of an application. This may mean accessing remote data from local processing or moving the processing to the remote machine where the hardware or data resources reside. SAS/CONNECT software provides compute services, data transfer services, and RLS.

Compute services are provided through the use of the RSUBMIT facility. The RSUBMIT facility causes SAS program statements to be sent from a local SAS session to a SAS session on a remote system for execution. All results of that execution are then returned to the local SAS session for display and management. The RSUBMIT facility is implemented in several ways. The RSUBMIT command provides an interactive means for moving processing to a remote session. The RSUBMIT statement allows you to alternate between local and remote pieces of execution within a single SAS job submission. There is also a SUBMIT REMOTE implementation within SCL to allow the applications programmer to embed remote functionality within an application.

Data transfer services are provided by the UPLOAD and DOWNLOAD procedures. These procedures can be used to transfer SAS data sets, catalogs, graphics catalogs, entire data libraries, as well as external files between local and remote systems. These procedures also support WHERE statement processing for dynamic data subsetting.

The newest feature of SAS/CONNECT software, RLS, is provided through the use of a LIBNAME statement that specifies a valid remote session id as the SERVER= value. This functionality enables you to define a libref in a local SAS session that points to data residing on a remote platform and to access this data transparently from your local SAS execution.

SAS/SHARE Software

SAS/SHARE software is defined as a set of advanced database features necessary for multi-user concurrent data update

applications utilizing SAS storage formats. In the future, we intend to add database features such as audit trail and integrated integrity constraints to this software. For more information about these features, refer to the paper "Database Features Extend the Scope of SAS/SHARE Software" listed in the section "Where to Find More Information."

Historically, the SAS/SHARE client and server have run on either the same machine or on two machines of like architecture. Now, with RLS, SAS/SHARE software provides concurrent update access to SAS data files when the user session and server are running on machines that have different internal representations of data.

Do I Need SAS/CONNECT Software or SAS/SHARE Software?

There are well-defined differences between the functionality of SAS/CONNECT software and SAS/SHARE software. Mapping these characteristics with the requirements of an application should enable you to determine which product you need to get the job done. In many cases, you might find that you want to combine functionality from both products.

The most obvious distinguishing characteristic of SAS/SHARE software is that it is the only piece of the SAS System that provides multi-user update access to SAS data libraries or SAS files. On the other hand, SAS/CONNECT software is the only piece of the SAS System that gives you compute services, data transfer services, and single-user RLS.

The following two basic guidelines define which product is required on a machine based on the role of that machine.

1. If a machine needs to provide multi-user access to SAS data either for concurrent update access or to serve data to multiple remote users from one SAS session, then you need SAS/SHARE software on this machine.
2. Otherwise, you need SAS/CONNECT software on this machine because it includes the remote engine required to access a remote multi-user server as well as compute services, data transfer services, and single-user RLS.

CONCLUSION

Distributed applications come in a variety of forms, but they all have the need to access data, manage data, analyze data, and present the information obtained from the analysis. The successful implementation of a distributed application requires breaking the application down into specific segments, defining where the initial data sources reside, and identifying what information needs to be available to the user interface. Once these issues are understood, educated decisions must be made about where the data should reside and where the processing should take place for each segment of the application. The guidelines detailed in this paper will enable you to make educated decisions about implementing your application.

The SAS System offers a complete set of tools that provide you with the flexibility to implement the segments of your application to best meet your goals. Compute services, data transfer services, and now, RLS, let you maximize resource usage, response time, and enduser productivity.

Remember that whether you use RLS or data transfer services, the data must still move over the network. Be careful to program your application to balance the costs of moving data via RLS, moving data via data transfer services, computing on the remote host, and computing on the local host.

WHERE TO FIND MORE INFORMATION

For more information about SAS/CONNECT software and SAS/SHARE software:

- *SAS/CONNECT Software: Usage and Reference, Version 6, First Edition*
- *SAS Technical Report P255, SAS/CONNECT Software: Changes and Enhancements, Release 6.09*
- *Preliminary Document, SAS/CONNECT Software: Remote Library Services*
- *SAS/SHARE Software: Usage and Reference, Version 6, First Edition*
- *Preliminary Document, SAS/SHARE Software: Cross-Architecture Access*
- *SAS/SHARE software host documentation*
- *Proceedings of the SAS User's Group International Seventeenth Annual Conference, "Maximizing Computer Resources with Cooperative Processing and Distributed Data Access,"* pages 641-650
- *Proceedings of the SAS User's Group International Seventeenth Annual Conference, "Database Features Extend the Scope of SAS/SHARE Software" pages 335-340*
- *Proceedings of the SAS User's Group International Eighteenth Annual Conference, "Tuning Tips for Applications That Use SAS/SHARE Software"*

SAS, SAS/ACCESS, SAS/CONNECT, and SAS/SHARE 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.