

Remote and Open Access to Enterprise Data: Enhanced Distributed Data Access Through Multiple Engine Architecture

Jack Wallace, SAS Institute Inc., Cary, NC
Michael D. Jones, SAS Institute Inc., Cary, NC

ABSTRACT

Access to data from the SAS® System is constantly being expanded and enhanced: Transparent access to remote data is now available when the local and remote machines have different architectures. And another new feature opens access to SAS data to applications that utilize the ODBC API, such as Microsoft Word.

These new features are built on a modular system architecture that often provides more than one way to accomplish a given distributed data access task. To choose the best method, it is advantageous to understand the processing performed by the components of Multiple Engine Architecture. Future enhancements for remote and open data access are also dependent on this architecture.

This paper introduces the new features but concentrates on the architecture and its implementation. If you simply want to know how to use the features, read the introduction and selected references at the end of this paper. If your interest extends to optimizations and processing details, you should find the body of this paper informative.

Readers are also encouraged to also read the paper by Cheryl Garner, "The SAS® System: A Complete Client/Server Solution" (see the list of references at the end of this paper). That paper compares and contrasts remote library services to other client/server features of the SAS System. For many SAS applications, these other features may provide enhanced performance.

INTRODUCTION

The new features and their functionality are overviewed in this "INTRODUCTION". A better understanding of the features will be gained as the architecture of the system and its components is explored in the next part of the paper, "ODBC and Multiple Engine Architecture Basics". Techniques for optimizing distributed data access are primarily covered in "Accessing Distributed Data from Multiple Sources".

Remote Library Services

Remote Library Services allows a SAS program to access a SAS data library on another machine as if the data library were on the local machine. The program accesses the data library through a SAS server on the remote machine.

To access a remote library, the SAS program simply executes the LIBNAME statement it would use on the remote machine with an additional option which specifies the name of the server on the remote machine:

```
LIBNAME xyz 'physical-library-name' SERVER=server_name;
```

SAS files in the remote library can now be manipulated by SAS programs on the local machine.

Remote library access has been available through the SAS/SHARE® product for many years, but server access was restricted to SAS programs running on the same machine architecture. For example, a SAS program on one MVS machine could access a SAS data library through a server on another MVS machine. Since 6.07, the SAS program or server could execute on MVS while the other executed on CMS because both MVS and CMS execute on the same IBM® 370 machine architecture.

In the fall of 1993, experimental code was released which allowed access between a SAS program executing on one machine architecture and a server running on another architecture. For example, a Windows SAS program could access a library on a MVS server. This cross-machine architecture capability has been enhanced and is now production.

When accessing a data library through a server on a different machine architecture, only SAS data sets may be read and written. SAS catalogs and utility files can not be read and converted to another machine architecture by remote library services. Note that SAS views, including SAS/ACCESS® views of DBMS tables, are a type of SAS data set, so DBMS tables on remote machines can also be accessed through a server on a different machine architecture. This will be more apparent as we discuss the architecture later.

Open Access to SAS Data

SAS Institute has been interested in the open standards work for database access and are members of the SQL Access Group (SAG). Several years ago, Microsoft modified and enhanced an early draft of the SAG-CLI and incorporated it in their Windows Open Systems Architecture (WOSA) as the Open Database Connectivity (ODBC) interface. This is the first commercially available implementation with significant market penetration and Independent Software Vendor (ISV) support. (We'll see later that the ODBC implementation in Windows is very similar to the SAS Multiple Engine Architecture.) Many Windows applications and application tools such as Microsoft Word, Visual Basic, and the SAS System itself can utilize ODBC to access data in other DBMS systems.

An ODBC driver to access SAS data from ODBC applications was released in the summer of 94. This driver communicates with a SAS server to access SAS data. The server may execute on the same machine, a remote machine with the same machine architecture, or a remote machine with a different machine architecture. So, a Microsoft Word document (or other ODBC application) running on Windows 3.1 or Windows NT will be able to access SAS data on MVS (or on any machine in the network including mainframes, minis, and workstations.)

Some users have desired the ability to access SAS data from

other programming languages and tools. A library of routines is currently in development for passing queries to a SAS server from a C program. This runtime library is being implemented for the major Unix platforms and ports to other client hosts are currently being investigated. The library will allow non-SAS client programs written in C to access SAS data through a SAS server anywhere on the network. Similar interfaces for other programming languages are also under consideration.

Expanded Use of SQL Passthru

In order to fully implement the features above, we have expanded the use of SQL Passthru. Originally, SQL Passthru was designed to allow the SQL processor in the SAS system to accept SQL statements with the syntax of other DBMS systems and send those statements directly to the DBMS for execution. For example, PROC SQL can send SQL statements directly to a DB2® server for execution.

With remote library services, the SQL processor can send an SQL statement through a remote SAS server to a DBMS system on the remote machine.

The SAG-CLI and ODBC standards for open access to data are based on the SQL language. The SAS SQL processor has been enhanced to accept SAS SQL statements through the same interface that it uses to pass DBMS SQL statements to a DBMS. Other modules have been added to accept the ODBC API, convert it to appropriate SAS SQL statements, and send it to the SAS SQL processor for execution through this passthru API. That architecture will be explored further below.

ODBC and Multiple Engine Architecture Basics

It is interesting to compare and contrast the ODBC and Multiple Engine Architectures. This section overviews the architectures and highlights their similarities and differences.

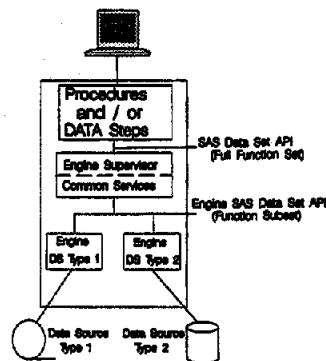
Architecture Layers

Figure 1 illustrates the basic components of ODBC and MEA.

An ODBC application calls ODBC functions in the driver manager to submit SQL statements and receive results. The driver manager loads and transfers calls to the appropriate driver. The driver processes the ODBC function calls and translates them into the necessary function calls or actions to access a data source. The data source is the data that the application wants to access along with any operating system, database server, and communications software used to access the data.

SAS applications are composed among other things of procedures, DATA steps, and SQL code. These application components in turn access SAS data sets by calling routines in the engine supervisor. Depending on the type of SAS data source, the engine supervisor loads and calls an appropriate engine to access the data source. In figure 0 a tape data source and disk data source are depicted. Many other data source types are possible as we will see.

Multiple Engine Architecture



ODBC Architecture

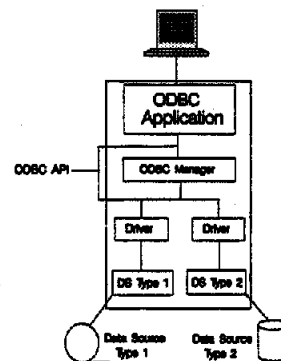


Figure 1

Application Programming Interfaces (API)

The Application Programming Interfaces really define the data model and methods to access data available to application programs from ODBC drivers or SAS engines. A full description of the API's is beyond the scope of this paper.

Because different data sources are capable of providing different functionality, not all drivers and engines provide the complete functionality of the API's. In both architectures, a part of the API are informational interfaces which the application can use to query the driver and engines about their conformance or support for other API functions.

ODBC API

As stated in the introduction, the ODBC interface between the application and the ODBC manager and driver is based on the SQL language. This includes routines to:

- connect to a DBMS,
- obtain descriptions of the tables and columns available,
- execute SQL statements,
- retrieve results,
- disconnect from a DBMS.

The interface between the ODBC manager and the driver is almost the complete ODBC API. The primary purpose of the manager is to load the appropriate driver for a data source. It does provide some initialization services and validates parameters and sequencing for application ODBC calls.

SAS API

The API between the application components and the engine supervisor for SAS data set access includes both navigational and SQL set based interfaces.

The navigational interface includes routines to:

- open a SAS data set,
- obtain descriptions of the variables (columns) and indexes if any,
- read, update, or add records (rows or observations),

- fetch or replace variable values in the record buffer,
- restrict access to records qualified by a where clause,
- position data set or index to a particular record or key value,
- close the SAS data set.

The navigational interface between the engine supervisor and the engine is a subset of SAS data set functionality. Common services in the engine supervisor removes the requirement to duplicate all functionality in each engine. In particular, routines to fetch and replace variable values in record buffers are common services and not passed to each engine. Instead, the engine transfers records between the engine supervisor and the data source.

Some functions may be executed by common services in the engine supervisor or by the engine. The engine and engine supervisor shake hands and give the processing to the engine if it can optimize the processing. For example, restricting access to records qualified by a where clause may be performed by the engine. Common services in the engine supervisor are used if the engine does not support WHERE clause processing.

The SQL set interfaces are similar to dynamic SQL and the ODBC API and is referred to as the SQL passthru API since the application is really communicating directly with the engine. Passthru interfaces are provided to:

- connect to a DBMS,
- prepare, describe, and open SQL SELECT statements,
- execute other SQL statements,
- retrieve results,
- disconnect from a DBMS.

Driver and Engine Classifications

It's convenient to classify drivers and engines into several groups that share similar attributes. Figure 2 shows basic driver and engine types:

- single-tier and multiple-tier drivers,
- native, view, and remote engines.

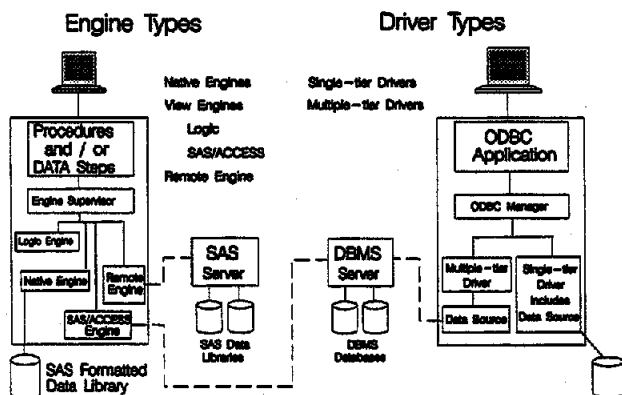


Figure 2

Single-tier Drivers and Native Engines

Single-tier drivers and native engines are similar in that they each directly access the data files associated with the data sources. They each contain the logic to process the API requests through operating system functions to access the appropriate data files, extracting the correct data, and delivering the requested results. While the operating system may use file server technology to store the files on a different machine, we assume the files are stored on the machine where the engine executes and do not consider that a form of remote access in this paper.

Single-tier drivers process SQL requests which are frequently restricted to single table access. They generally map to simple yet popular PC file formats such as DIF, WKS, DBF.

Native engines process navigational requests for data that is stored in a SAS system defined format. The most important native engine in any release is the BASE (or current version) engine. It implements all the full functionality of the current release SAS data set model in the most optimal manner for disk access. Compatibility engines support back release formats of the BASE engine. Sequential engines store SAS data sets on sequential access media such as tape. The transport engine stores SAS data sets in a format that can be read by another machine type. Only the BASE engine is guaranteed to directly support all the functionality of the current SAS data set model.

Multiple-tier Drivers and other SAS Engines

Multiple-tier drivers are similar to the remote engine and SAS/ACCESS engines in that they send requests to another process called a server for processing. The server process may execute on the same machine as the application or on a different machine.

The multiple-tier driver simply converts the ODBC functions to the appropriate API functions calls for the data source. The data source is then responsible for communicating the request to a database server and returning the results.

SAS Engine and SAS Driver Processing

This section discusses the actual processing of SAS engines and the SAS/ODBC Driver. While it is not necessary to know this material in order to use the engines or driver, it is useful when discussing the optimizations for accessing data from multiple sources.

View Engine Processing

Views are a type of SAS data set which are materialized from other data sources based on descriptions of the other data or processing instructions which are stored in a VIEW file. A VIEW file is processed by a particular view engine to materialize the SAS data set it represents.

Common processing of view engines is depicted in figure 3.

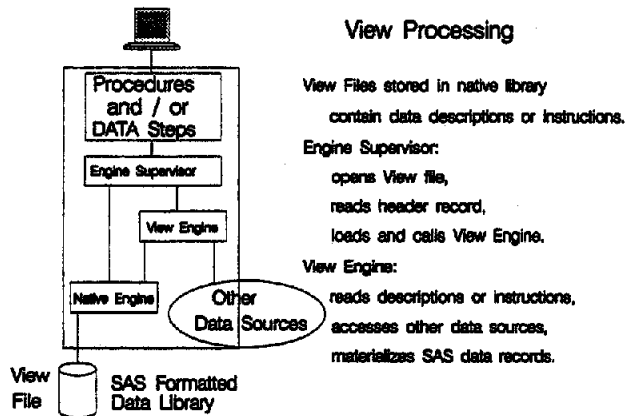


Figure 3

The application requests that the engine supervisor open a SAS data set with a particular name in a library that is accessed by a native engine.

The engine supervisor determines that the name references a file of type VIEW instead of type DATA. It calls the native engine to open the view file and read the view header record. The header record contains the name of the engine that processes this view. The supervisor loads and calls the view engine to complete opening the SAS data set. The view engine reads data descriptions or instructions from the view file and closes it. It opens other data sources to complete the open processing.

All subsequent access to the open SAS data set is passed to the view engine. It accesses data from the other data sources and materializes SAS data records for the engine supervisor.

In the remaining discussions of view engines, access to the VIEW file is ignored except where it plays a particular role in remote access.

In this paper, the term logic engine is introduced to refer to view engines that programmatically access data in other sources according to stored program logic in a VIEW file. Figure 4 depicts and describes the two logic engines in the SAS System today.

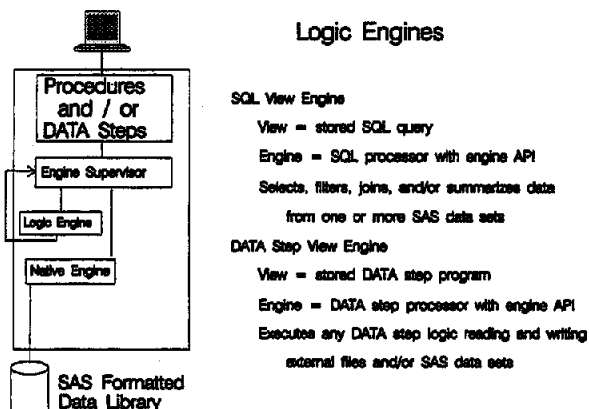


Figure 4

The SQL view engine processes stored SQL queries to select, filter, join, and summarize data from one or more other SAS data sets. The DATA step view engine processes stored DATA step programs which read and write data from either external files or SAS data sets. In both cases, the engine processor is a programmable component of the SAS system that is normally executed as an application and accesses SAS data sets through the engine supervisor. Figure 4 depicts that these processors still access other SAS data sets through calls back to the engine supervisor.

It is worth noting that the SQL processor will itself determine if some of the referenced SAS data sets are SQL VIEW files. In this case, the SQL processor iteratively reads in all referenced SQL VIEW files. It then transforms the combined SQL SELECT statement into an optimized access strategy. In version 6, it does not retransform and optimize the SELECT statement if an additional WHERE clause is specified after this initial optimization when the view is opened. These additional WHERE clauses are treated as filters of records that are returned by the original access strategy. In some cases this is optimal but in others a retransformation would significantly optimize a join strategy. You can force the WHERE clause to be used by defining a new view which applies the WHERE clause to the original view.

The DATA step view engine does not optimize access according to WHERE clauses specified for a view. The engine supervisor always filters the records that are returned from the DATA step view.

SAS/ACCESS engines are used to access data in other formats. In most cases, the data is formatted by a DBMS and the engine uses the DBMS to access the data as depicted in figure 5.

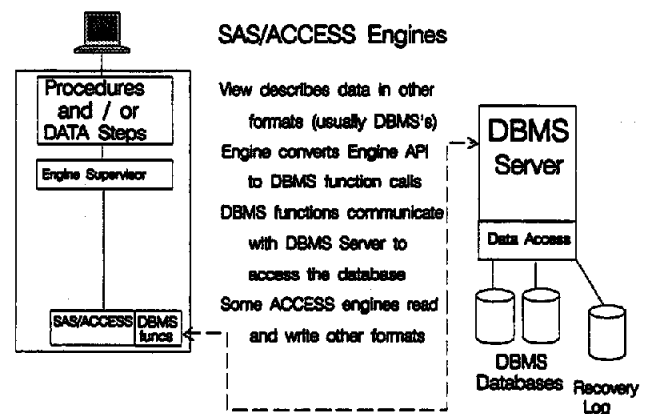


Figure 5

The view file itself contains descriptions of the data that is in the DBMS table. The engine converts requests to access records from the engine supervisor to the appropriate DBMS system function calls. The DBMS function calls usually communicate the request with a DBMS server to access the database. Record data (rows) are returned through the DBMS functions to the engine, engine supervisor, and application program.

Remote Engine Processing

The remote engine is used to access SAS files through a SAS server for one of two reasons:

- to share update access to the SAS file with other SAS programs reading or updating the same file,
- to access a SAS file on a remote machine where the server is executing.

As depicted in figure 6, the remote engine communicates the engine function request to a mirror task for the user in the server. The mirror task calls the real engine to access the data.

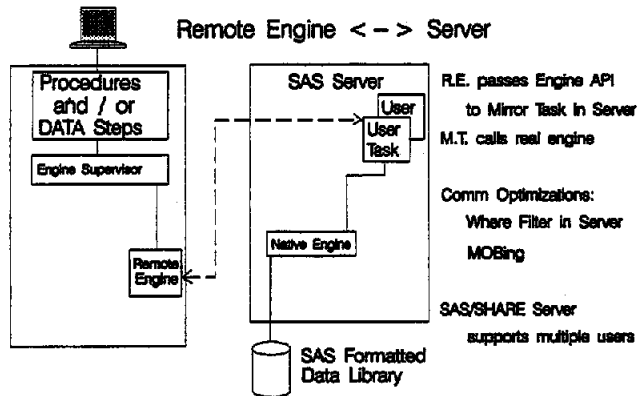


Figure 6

Accessing a SAS file through a server is always slower than directly accessing the file because of the extra code to communicate the request and pass it to the engine in the server. The response time cost of this communication increases when the server is executing on a different machine than the application program and varies amongst different communications protocols and network topologies. Optimizing the communications costs is essential in remote data access applications.

The remote engine does optimize access time by reducing the number of communications between the remote engine and the server. Not all engine calls result in calls to the server. If the call is requesting descriptive information retrieved when the SAS file is opened, no new communications are required. In addition, the server returns information about each record with the retrieval call. It returns multiple records in one call for sequential input access (and for random input access if the TOBSNO= option was specified). Subsequent requests for information or records which the remote engine already has in its control blocks or buffers, do not result in communications to the server.

The application program or user can also reduce the number of communications by utilizing WHERE processing. The remote engine always tells the engine supervisor that it can optimize WHERE processing and passes the WHERE clauses to the server. The server then shakes hands with the real engine to determine if the engine or the server will qualify records with the WHERE clause. In either case, only records that qualify are ever sent back across the communications link to the remote engine.

As stated in the Introduction, the remote engine and server were required to execute on the same machine architecture until experimental code was released in the fall of 1993. That restriction has now been lifted for SAS data set access. The

remote engine and server now transfer information on each others architecture when communications are established. If the architectures differ, machine code is generated to translate the data streams with the following priorities:

1. the remote engine assumes responsibility (translates directly to and from the server representation) if it can,
2. the server assumes responsibility (translates directly to and from the remote engine representation) if it can,
3. they share responsibility (each side translates to and from a transport format).

Only those pieces of data which need to be translated are translated. By generating machine code, the fastest translation path length is produced.

SAS ODBC Driver Processing

ODBC application access to SAS data is depicted in figure 7. It's easy to see that the SAS ODBC driver is a multiple-tier driver. The data source components of the SAS implementation consist of a SAS server accessed through a Remote Engine Emulator and Communications Access Method Emulators.

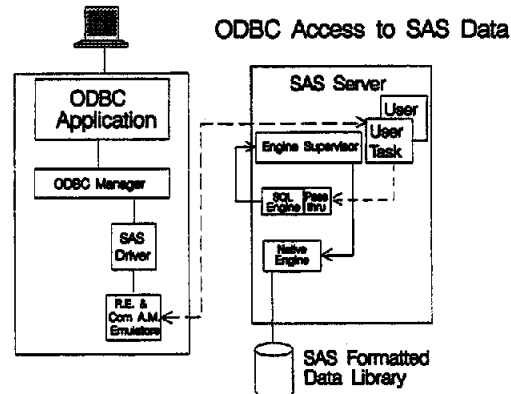


Figure 7

The SAS ODBC driver converts the ODBC API to the passthru API and calls the remote engine emulator. The remote engine emulator formulates the correct communications messages and calls the appropriate communications access method emulator to send the request to the SAS server and receive the response. In the SAS server, the communications and requests look almost identical to requests from a SAS application using the passthru API. The request is passed to the Passthru API of the SQL processor in the SQL engine. It then processes the SQL query or statement as it would from any other source. The results are returned from the server through the emulators to the driver.

Accessing Distributed Data from Multiple Sources

Multiple Engine Architecture and the many engines in the SAS system make it possible to access data from multiple sources in many different ways. We must first understand ways to merge or join data from multiple sources in a non-distributed environment before we can fully discuss the distributed case.

Joining/Merging Data in a Single SAS session

Two cases of interest are joining SAS data with DBMS data and joining DBMS data with DBMS data.

Joining SAS Data with DBMS Data

A logic engine can be used to join or merge data from multiple SAS data sets. Some of the SAS data sets may themselves be SAS/ACCESS views of DBMS data. Figure 8 illustrates a logic engine accessing data from a native engine SAS data file and a SAS/ACCESS view of a DBMS table.

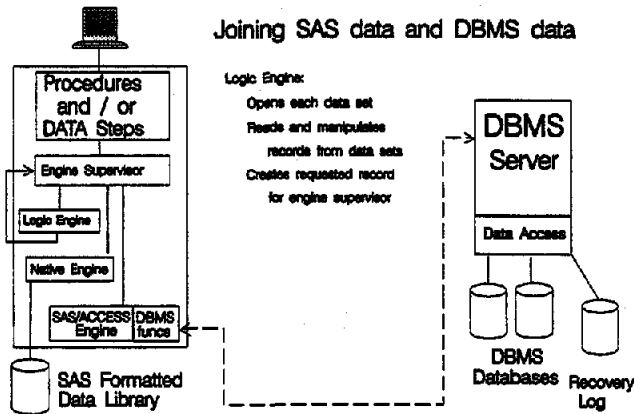


Figure 8

An SQL view simply references the SAS data file and the SAS/ACCESS view as the SAS data sets in the FROM clause of the SELECT statement (and specifies the proper joining conditions in the WHERE clause).

A DATA step view might either:

- reference the SAS data file and the SAS/ACCESS view in a MERGE statement with a BY statement to specify the merging criteria, or
- programmatically access data from each source through multiple SET statements.

It is worth noting that the DATA step processor in MERGE processing with a BY statement will send WHERE statements to the SAS/ACCESS engine and DBMS server to select rows that match particular BY group values read from the SAS data file. This may be more optimal than reading all rows from the table when there are few BY groups in the SAS data file, lots of rows in the DBMS table, and the DBMS table has an index on the BY variables. It may be less optimal in other cases.

The SQL processor will use indexes to optimize join processing, but the SAS/ACCESS engines in version 6 can not surface full index processing like the BASE engine. This means that the SQL processor cannot optimize access with knowledge of DBMS

indexes in version 6.

Joining DBMS Data with DBMS Data

Logic engines are an effective solution for joining or merging data from two or more SAS/ACCESS views of database tables when:

- the database tables are in different servers, or
- the server can not duplicate the program logic in the view.

As depicted in figure 9, two separate and independent opens of DBMS tables results from a logic view accessing two SAS/ACCESS views.

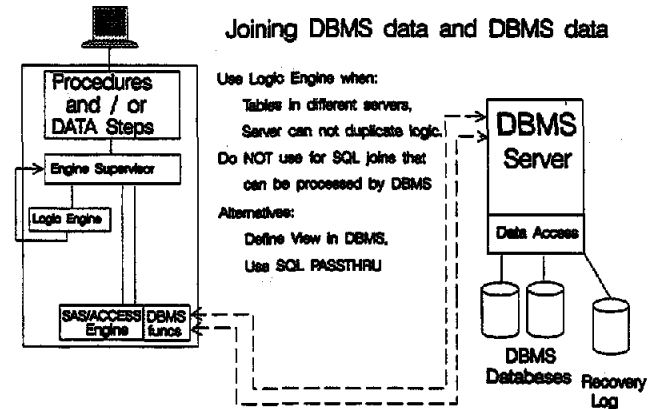


Figure 9

The SQL view engine and DATA step view engines behave in the same manner as when joining SAS data files with SAS/ACCESS views which was described above.

If the view logic is an SQL join of database tables which can be processed by the DBMS server, then it is usually far more efficient to allow the DBMS to process the join than to bring all the data back to a logic engine in the SAS system to process the join. This is consistent with a general optimization rule for remote data access that says: "Move data selection and manipulation logic as close to the data sources as possible."

There are two possible ways to get the DBMS server to process an SQL join. One way is to define a view in the DBMS itself that joins the base tables. In this case, a single SAS/ACCESS view that describes the view of the joined DBMS tables can be accessed without an intervening logic engine in the SAS session. However, there is overhead to defining a view in a DBMS which you may not want to incur for a one time join strategy. You may also find you are not allowed to define DBMS views in the server. For these cases, you should use the passthru facility of SAS SQL depicted in figure 10.

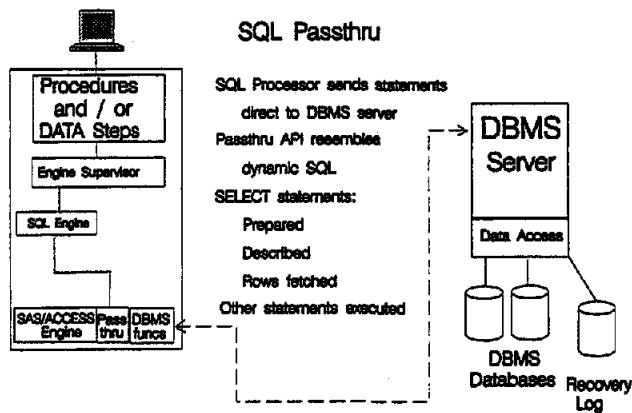


Figure 10

The SQL processor sends statements in the DBMS's dialect of the SQL language directly to the DBMS server through the passthru API in the SAS/ACCESS engine.

To establish a connection directly with the DBMS, you issue a connect statement to the SQL processor specifying the SAS/ACCESS engine name and explicit DBMS options in parentheses. For example:

```
CONNECT TO DB2 (SSID=DB2T);
```

Once you have a connection, you can pass DBMS syntaxed SELECT statements to the DBMS by enclosing them in ()'s and preceding them with SELECT * FROM CONNECTION TO dbms. For example:

```
SELECT * FROM CONNECTION TO DB2
  (SELECT NAME, COLCOUNT
   FROM SYSDM.SYSTABLES
   WHERE CREATOR='MYAUTHID');
```

Or you can execute other statements. For example:

```
EXECUTE (CREATE TABLE FOO
  (BAR CHAR(3),
   GOOMBAH NUMERIC))
  BY DB2;
```

You terminate a connection with the disconnect statement. For example:

```
DISCONNECT FROM DB2;
```

SQL passthru statements can be used either directly from PROC SQL or stored in SQL views for execution by the SQL view engine when the view is accessed as a SAS data set by other SAS applications.

Joining/Merging Data that is Remote

The general optimization rule for remote data access that says: "Move data selection and manipulation logic as close to the data sources as possible." is the underlying theme of this section.

Local Data with Remote Data

Joining local SAS data with remote data can be accomplished in the same manner as joining SAS data with DBMS data described above. That is, you can use a DATA step or SQL view engine in the local SAS session to reference both local SAS data sets and

remote SAS data sets as illustrated in figure 11.

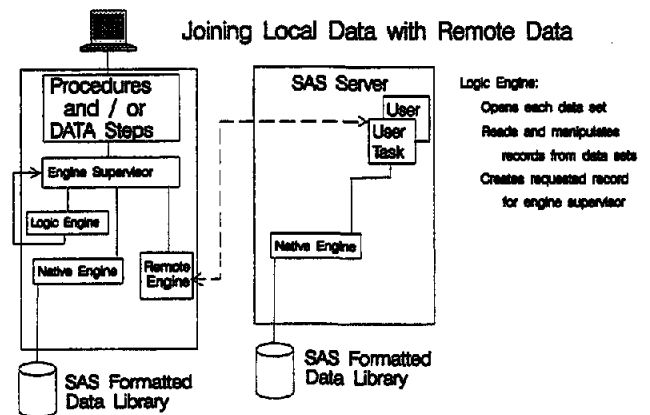


Figure 11

In the section on logic engine processing, it was noted that both the SQL view engine and the DATA step view engine would treat a WHERE clause applied to a view as a filter of the records returned. It was pointed out that the SQL view engine would use the WHERE criteria in an optimized access strategy if a new view was defined that applied this WHERE clause to the original view. This technique may be even more important when one of the SAS data sets is accessed remotely as in figure 11. It will cause the WHERE clause criteria to be pushed across to the server so that only appropriate records are sent back to the SQL engine in the local session.

All Remote Data

When all the data to be joined is in a remote server environment, it is often most efficient to join the data in the server environment. The remote engine / server architecture and MEA makes it possible to execute view engines in the server environment. Figure 12 illustrates execution of a logic engine in the server environment.

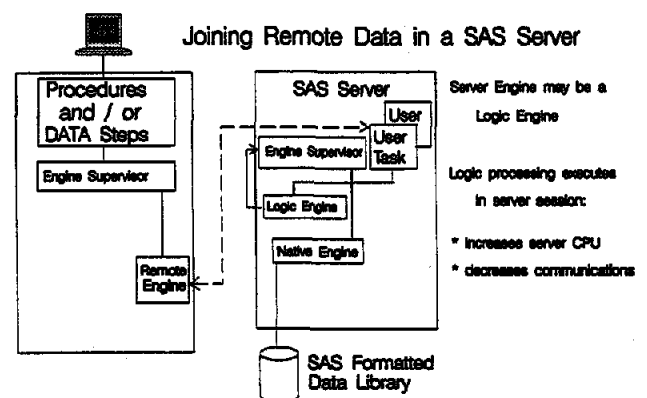


Figure 12

Using logic engines in the server environment may increase server CPU consumption but will usually decrease communications between the remote engine and server. This decrease is especially significant when the view processor produces a small set of data records from large data sets.

View engines are normally executed in the server environment if the view definition is stored in a library accessed through the server. You must be careful when utilizing the technique of defining a new view with a WHERE clause applied to an existing SQL view in a server library. The new view must also be defined in the server library in order for the WHERE clause to be included in the optimized access strategy.

If you do not want the view engine to execute in the server session, add the option RMTVIEW=NO to the LIBNAME statement that accesses the library with the VIEW file through the server. RMTVIEW=NO tells the engine supervisor to read the VIEW file header into the local session and invoke the specified view engine there. It also allows the SQL processor to iteratively read remote view definitions, thus preventing the SQL view engine from being executed in the server. If the server is a different machine architecture, the VIEW file can not be accessed by the local engine supervisor and the server will execute the view engine regardless of the RMTVIEW= option setting.

Accessing Remote DBMSs

SAS applications may be able to access remote DBMS data in two ways. If the DBMS system facilities support remote data access, then it's possible that a SAS/ACCESS view accessing the DBMS local functions can access a remote DBMS server. Alternatively, the SAS application can use the remote engine and server architecture to execute a SAS/ACCESS engine in a remote server to access a DBMS on the server machine.

DBMS Facilities for Remote Data Access

It's beyond the scope of this paper to describe all the capabilities of each DBMS system for remote data access. There are however two basic models; direct client to remote server communications, or client to local server to remote server communications.

The former model is similar to that used by the SAS remote engine to server. Client applications directly establish communications links to the remote server without any intervening local database server. Most DBMS systems implement this model and require specific client software for networked access through proprietary communications message formats and protocols to their DBMS server. The ODBC and MEA architectures allow for this by converting the APIs to the DBMS APIs in the drivers and engines, allowing the DBMS functions to handle communications.

The intervening server model is illustrated in figure 13.

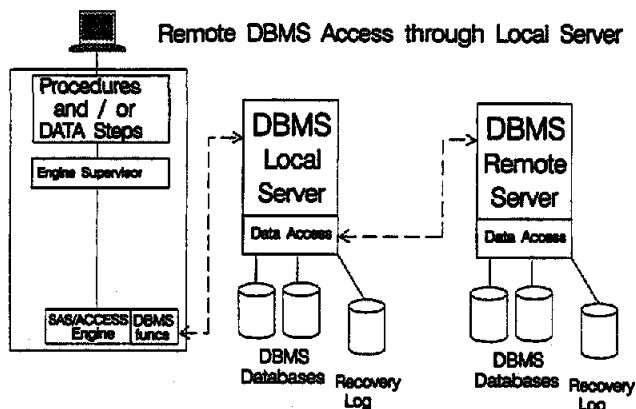


Figure 13

The application program uses DBMS functions to access a "local" server. The local server determines that the database to access is on a remote server machine and routes the access request to the remote machine. Note that the definition of local versus remote server may differ by environment. In a LAN environment, the term local server may include any server on the local LAN with the same basic machine architecture as the client application. The remote server is generally a server either on a different machine architecture or accessed across a WAN.

An example of this model is implemented by IBM's relational database systems. A client application on OS/2[®] typically accesses a local DB2/2[™] server which may route the request to a remote DB2 or SQL/DS[™] server.

Remote Data Access Through a SAS server

A SAS application can access a DBMS through a SAS server by defining the SAS/ACCESS view in a server library. This is illustrated in figure 14.

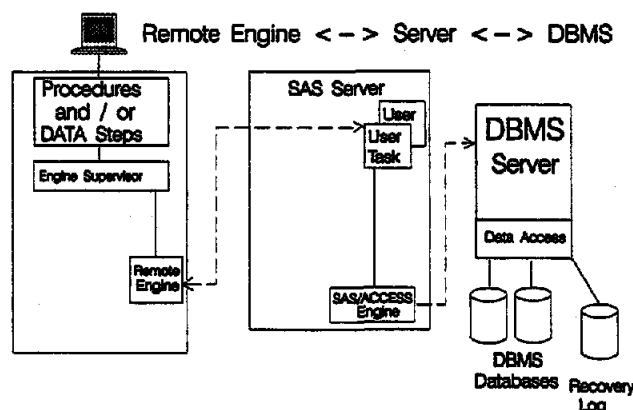


Figure 14

This combination works when the application and both servers are on a local machine but is not a recommended solution for that environment. In that environment, the SAS/ACCESS engine in the SAS application session can directly access the DBMS server and will result in less communications. To directly access the DBMS server from the SAS application session, either put the SAS/ACCESS VIEW file in a library not accessed through the server or add the option RMTVIEW=NO to the LIBNAME statement that accesses the library through the server.

The combination is useful when the two servers are on a remote machine and:

- the DBMS facilities don't provide a way to access the remote DBMS server so a local SAS/ACCESS engine can not be used, or
- you don't license the necessary software (SAS/ACCESS and/or DBMS network access software) on the application machine, or
- you want to join or merge the DBMS data with other data on the remote machine as depicted in figure 15.

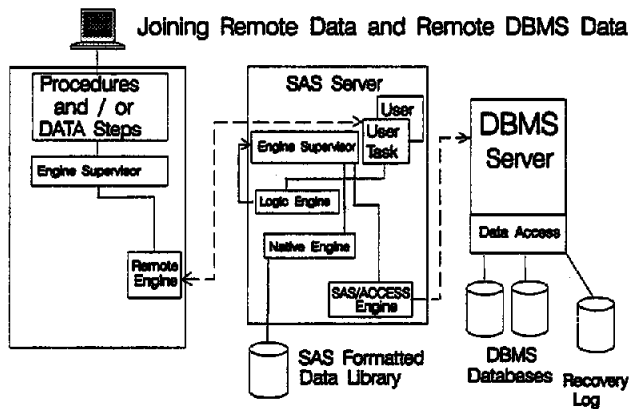


Figure 15

By moving the logic processing to the same machine where the data sources reside, network communications with the application session are reduced.

Just as it is useful in a single machine environment to pass join processing to the DBMS if it can do it (see DBMS data with DBMS data above), it is best to do so in a remote server environment. Figure 16 illustrates using the SQL processor in the local session to send DBMS syntaxed statements through the remote engine, server, and SAS/ACCESS engine in the server to the DBMS for execution.

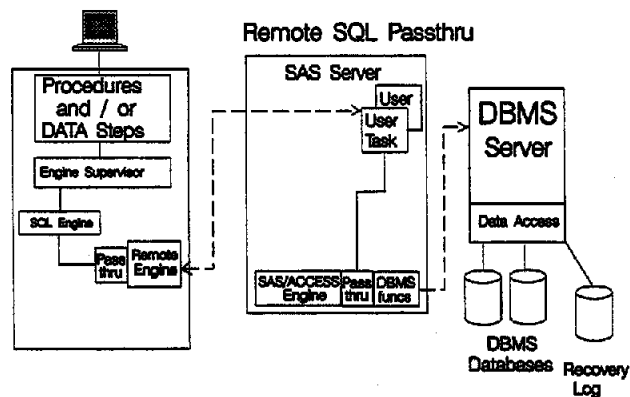


Figure 16

To make this connection, the passthru CONNECT statement specifies the REMOTE engine and in parentheses the name of the server, the name of the DBMS (SAS/ACCESS engine) and DBMS arguments. For example:

```
CONNECT TO REMOTE
(SERVER=SASSERVE
 DBMS=DB2
 DBMSARG= (SSID=DB2T))
AS DB2T;
```

Once a connection is established, the same SQL PASSTHRU statements described earlier can be used to send statements for execution in the remote DBMS system.

It is worth noting that a multi-user server appears to the DBMS as a single user application and DBMS authorization is based on the access rights of the server in the DBMS. You can still provide security for the DBMS in this environment by limiting access to the server with server access password and/or utilizing passwords on SAS/ACCESS views in the server.

In a multi-user server the DBMS's do not treat each user as a separate transaction environment that can be independently committed or rolled back. For this reason, update access to DBMS's through a multi-user server is prohibited.

We have introduced a new single-user server with remote library services. The single-user server runs in a remote SAS/CONNECT® session and only serves data access requests from the local SAS/CONNECT session. Since the remote SAS session runs with the authorization of the userid that created it, the DBMS authorization controls for that userid are effective. In addition, update access is allowed through the single-user server since the one local SAS session is executing the application.

CONCLUSION

MEA is Extremely Flexible

The modular structure of Multiple Engine Architecture allows you to accomplish tasks in many ways by various combinations of the engines. Figure 17 combines most of the previous SAS figures into one. I hope you can reference this diagram and paper to effectively combine the facilities available today in the SAS System.

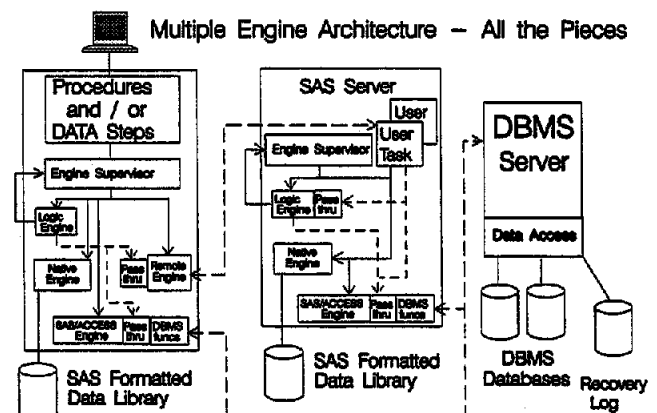


Figure 17

SAS Server and MEA extends capabilities of ODBC Applications

While the primary motivation of the SAS/ODBC Driver implementation is to allow OPEN access of SAS formatted and managed data from other non-SAS applications as some of our users have requested, it's important to remember that Multiple Engine Architecture in a server environment allows you to join and merge data from many sources and in many ways. Figure 18 illustrates all the additional capabilities from an ODBC application environment.

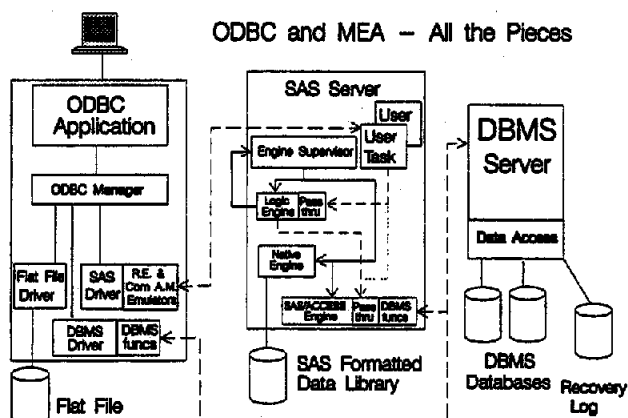


Figure 18

From the SQL logic engine you can:

- access SAS data files through the native engine,
- access external files through DATA step processor with a DATA step view,
- access DBMS data through SAS/ACCESS views,
- access DBMS data through SAS SQL views containing SQL passthru statements for the DBMS, and
- join and merge data from these sources through SQL statements or DATA step programming logic.

The architecture will also allow an ODBC application to send SQL statements through the server and a SAS/ACCESS passthru interface to a DBMS without using the SQL processor in the server execution as depicted by the line from the user task directly to the SAS/ACCESS passthru API. However, the SAS/ODBC Driver does not provide the same level of ODBC conformance when by passing the SAS SQL processor. We've found many ODBC application packages need level 1 conformance. Therefore, we have not expended resources testing this capability. You may find it adequate for writing custom ODBC applications for known databases and tables. If you are interested in trying this, contact Technical Support at SAS Institute.

Future Plans

We plan to support more standard API's for open data access to SAS data as they are adopted by the software community and requested by our customers. We will also consider providing a proprietary SAS API callable from C programs in environments where no standard API is emerging. If you have an interest in a particular standard and operating system - please convey your interest to both the author and the SAS Marketing division at SAS Institute.

We also intend to study ways to improve our processing and so that we can make more intelligent processing decisions for you. For example:

1. We want to be able to retransform and optimize SQL access strategies when a new WHERE clause is given. However, we don't want to do that if we've already accessed all the data and have a relatively small result set that could be filtered with the new WHERE clause. Different decisions are appropriate for different cases and may produce different result sets in a

concurrent access environment.

2. We are considering adding another key type to the SAS data model in version 7 that could be used for equality matching only. Some SAS/ACCESS engines will then identify DBMS indexes as these restricted use keys. The SQL processor will then use them and the engine will convert this restricted key interface to the appropriate WHERE clause. Again, there are cases where this is more efficient and others where it is less efficient. Some of the criteria will depend on the DBMS.

Ultimately, we need to give the programmer the ability to force one decision or another even if we improve our ability to optimize the default behavior.

REFERENCES

Barnhart, Andy, (1994), "Sharing Enterprise Data with ODBC," *Proceedings of the Nineteenth Annual SAS Users Group International Conference*, Cary, NC: SAS Institute Inc.

Beatrous, Stephen and Clifford, William, (1988), "Version 6 SAS® Data Base System Architecture: Current and Future Features," *Proceedings of the Thirteenth Annual SAS Users Group International Conference*, Cary, NC: SAS Institute Inc.

Garner, Cheryl, (1994), "The SAS® System: A Complete Client/Server Solution," *Proceedings of the Nineteenth Annual SAS Users Group International Conference*, Cary, NC: SAS Institute Inc.

SAS Institute Inc. (199x), *SAS/ACCESS® Interface to <your favorite DBMS> : Usage and Reference, Version 6, First Edition*, Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1990), *SAS/CONNECT™ Software: Usage and Reference, Version 6, First Edition*, Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1992), SAS Technical Report P-255, *SAS/CONNECT™ Software: Changes and Enhancements, Release 6.09* Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1991), *SAS/SHARE® Software: Usage and Reference, Version 6, First Edition*, Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1993), SAS Technical Report P-260, *SAS/SHARE® Software for the MVS Environment, Release 6.08* Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1993), SAS Technical Report P-261, *SAS/SHARE® Software for the CMS Environment, Release 6.08* Cary, NC: SAS Institute Inc.

ACKNOWLEDGEMENTS

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

IBM, DB2, DB2/2, OS/2, and SQL/DS are registered trademarks or trademarks of International Business Machines Corporation.

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