

Optimizing Client-Server Performance through Data Warehouse Design

Eric C. Brinsfield, Meridian Software, Inc.

Abstract

Client-server systems enable users to access large volumes of data on seemingly unfriendly mainframes or servers, while offering a friendly graphical user interface (GUI) on the local PC. Even the best GUI design, however, cannot disguise the slow performance that results from poor client-server design. This paper will discuss how to utilize data warehousing concepts to improve client-server performance and will describe specific examples.

Introduction

Before we can talk about optimization and design, we need to clarify our terms and assumptions. I will first define client-server and data warehouse separately and discuss the performance issues for each separately. Within the definitions I will explain SAS Institute's software solutions. At that point we can discuss the interaction between client-server technology and data warehouse architecture.

As a preliminary step to design, I will discuss the collection of user requirements. Finally, I will suggest some key design principles that should improve client-server performance and discuss a case study with proven results.

Definitions and Assumptions

Definition of Client-Server

For this discussion, we will limit ourselves to SAS Institute's implementations of client-server technology. Generally, a client-server application provides a graphical user interface that runs on a PC or workstation that accesses and processes data on a separate server running the same or different operating system. The user does not usually know or need to be concerned about where the data resides or where the processing occurs.

Strictly speaking, a graphical user interface is not required for a client-server application, but a GUI is usually implied when people discuss a client-server system. As a SAS programmer, however, you can work in a client-server environment by developing code locally then submitting code remotely. In this case, you may or may not be utilizing a graphical user interface.

Many people think client-server means accessing data on a remote server from a local workstation. Unfortunately, client-server development is not that simple. The client-server designer must optimize where specific processing is performed rather than just linking to remote data and performing all manipulations on the local workstation.

SAS Tools for Client-Server

In the SAS world, we usually think of SAS/CONNECT as the tool that enables programmers to create client-server applications. In many cases, you also need additional products depending on what functions you decide to perform and where. For example, if you are developing a client-server application that runs on Windows NT and accesses data in an Oracle database on UNIX, you will need to license SAS/ACCESS to Oracle on the UNIX server, but not on the workstation. Base SAS software and SAS/CONNECT are, of course, required on both ends.

SAS Institute offers several new vehicles for client-server development including

- the ODBC interface
- Web-enablement tools.

With the ODBC interface, you can access other software vendors' products, which you can use as client-server gateways. You can develop SAS applications that access non-SAS databases or develop non-SAS interfaces that access SAS databases. In both cases, client-server strategies can be utilized.

With SAS Institute's new efforts in Web-enablement, you can now use a mixture of products to develop client-server applications over the Internet, an intranet, or an extranet. I will not elaborate on these options in this discussion.

Performance Factors in a Client-Server Application

Any software application involves data access, data processing or manipulation, and data display. The client-server system adds two additional activities:

- data transfer
- split processing.

I will focus on these two activities for potential client-server performance optimization, but first we need to define them.

Data transfer refers to the movement of data or results from the server to the client workstation, and sometimes back to the server. Usually, the data traverses a network connection or sometimes even a modem link. Obviously, the speed at which data moves from server to client is an important performance factor. If the volume of data transferred is large and the connection is slow, your client-server application will be slow. In addition, if your server stores data in a different format than your workstation does (EBCDIC to ASCII), speed may be reduced by conversion processing.

Split processing refers to the programmer's option to perform all or part of the processing after data extraction on the server or to perform all or part of the processing after download on the workstation. To improve performance, you should

- strive to perform CPU intensive processing on the appropriate machine, which is usually the server
- try to reduce the amount of data that must be downloaded, thereby reducing the transfer time even if the transfer rates are poor.

Obviously, client-server response time is a function of the interaction between the data transfer rate and the processing location.

Definition of Data Warehouse

According to Inmon, a data warehouse is "a collection of integrated subject-oriented databases designed to support the DSS function;

where each unit of data is relevant to some moment in time. The data warehouse contains atomic data and lightly summarized data."¹

Although this definition is accurate, it does not clarify the true purpose of the data warehouse, when it states that it is "designed to support the DSS function." Most databases are designed to ease data entry, reduce redundancy, and speed the retrieval of a single individual or entity. The data warehouse, on the other hand, is designed for fast retrieval of information and answers, which means that groups of records will be retrieved, manipulated, and analyzed. It also means that you may need to access data derived from multiple database sources, which accounts for the phrase "a collection of integrated subject-oriented databases."

As with many other types of database applications, the data warehouse architecture relies heavily on metadata or data that describes the attributes of your data. Advanced data warehouse software helps manage and utilize metadata.

SAS Tools for Data Warehousing

Generally speaking, the SAS system is a tool for data warehousing. Many of us have been creating data warehouses using the star schema for many years, because IT departments did not want analysts and report writers to read directly from their online transaction processing systems. Because we were creating our own databases, we had the luxury of designing the data for faster analytical access.

Today, SAS Institute offers several new tools, including the MDDB server for online analytical processing (OLAP) and the SAS/Warehouse Administrator for managing data warehouse tables, entities, and metadata. I will discuss MDDB server later, but will not discuss the data warehouse administrator.

Performance Factors in a Data Warehouse

For this discussion, performance in a data warehouse refers to the speed of data access or data retrieval. The performance of applications that use a data warehouse depends on three primary factors:

- the overall performance of the server
- the engine used to extract data from the warehouse
- the design of the data warehouse.

As software developers, we will assume that we have no control over the server performance, which is dependent upon the CPU power, the installed memory, the number of concurrent users, and other applications that also use the same server.

By *engine*, I am referring to the actual software that is searching through and reading from the data warehouse databases. In many cases, you will employ multiple engines, if some of the data is stored in a DBMS, such as Oracle, and some of the data is stored in SAS databases. To improve performance, design your data warehouse to extract data with the most efficient database engine for each specific table or data set. In other words, do not suffer the overhead of a full DBMS for all tables just because the DBMS is available.

Within SAS software, you should evaluate the choices of engines provided on the server operating system. For example, for any given table within your data warehouse, you should ask which is faster:

- a DATA step, SQL, a specific PROC, or SCL?

- a basic SAS data set, an indexed SAS data set, or a SAS MDDB?
- a virtual view or a materialized view?
- a SAS/ACCESS view, the SQL pass-through facility, or possibly ODBC?

The design of the data warehouse, which overlaps somewhat with engine selection, provides significant ways to improve, but also the most opportunities to degrade, the performance of your application. Your data warehouse must be designed to provide easy, uncomplicated access to a specific user or group of users. To achieve the fastest response times, try to decrease the size of tables that will be accessed by your application and reduce the number of joins or merges performed during the extract.

Efficient use of metadata could also be considered a primary performance factor, but I have included that under data warehouse design. Taking advantage of metadata can significantly reduce data retrieval times in addition to offering many maintenance benefits.

Bookshelves today are loaded with books explaining how to design data warehouses, so we will not go into detail about general data warehouse design issues. Instead, we will focus on the performance factors for client-server applications that can be solved by data warehouse design.

The Client-Server Data Warehouse Interaction

I have briefly identified performance factors that relate to client-server technology and data warehouse architecture independently. The interdependence is evident when you consider how data warehouse design could improve client-server performance by reducing

- data retrieval time
- total processing time
- the quantity of data transferred from the server to the client.

Designing the Application from Start to Finish

Evaluating the User's Requirements

Before you begin designing the data warehouse, you must understand the user's requirements. Although you need to be familiar with the available data, you should not drive the application by the data. Instead, drive the data warehouse design by the user's needs and expectations.

Before designing or coding anything, you should establish:

- What type of graphical user interface makes the users comfortable?
- What types of queries do they expect to make? Frequently? Occasionally?
- What data will be summarized? Frequently? Occasionally?
- What new variables, if any, should be derived during the queries?
- What time spans are the users interested in? Frequently? Occasionally?

applications development

- What types of comparisons will they make? Frequently? Occasionally?
- How well do the users know the data and/or the data structure?

In many situations, the users want fast response and only a limited set of query functions. In this case, you can design the interface and the data warehouse to reflect the narrow set of requirements. If the users want maximum flexibility, you will need to design both the interface and the data warehouse to accommodate that requirement.

Because you will be creating a data store that is designed specifically for your users, you may want to save summarized data or calculated variables in the data warehouse. If space becomes a problem, you should balance the frequency of access to specific types of queries with the cost of storing fast pre-summarized data for those queries. Infrequent queries can be given a lower priority for performance.

Designing the Client-Server Application – Round 1

Once you have evaluated the user's requirements, you can design your user interface, which enables you to determine what data is needed to satisfy user requests. In our example, we were fortunate, because the users knew what functionality they wanted. They provided sample windows and a limited set of queries that answered very reasonable questions.

In our example, we developed a physician profile system that displayed several measures of hospital and physician performance. This enabled our client's users to compare their hospital's performance to any selected hospital included in a medical database that contained statistics for participating hospitals. The primary window into the system displayed departmental statistics for our client's hospital and one other (default) hospital. From this window, the user could select a different hospital for comparison or accept the current "Other" hospital. Once the other hospital is accepted, the user can drill down within a department to see physician statistics for two hospitals within that department or for two time periods for the same hospital and department.

From the physician profile window, the user can display bubble plots, trend plots or reports on

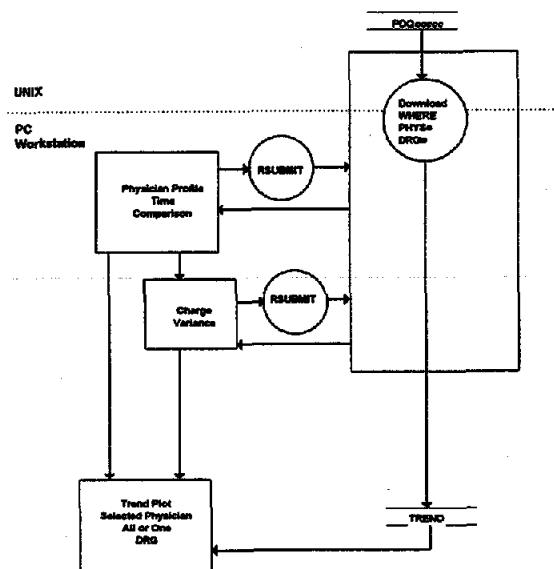
- the entire department
- one or more selected physicians
- several combinations of comparisons between department and physicians for all DRGs or one DRG (Diagnosis Related Groups).

All reports and graphs used statistics from either the current year or the past year, except the trend plot, which required quarterly statistics for the past 4 years. By compiling and evaluating all of the types of permitted queries, we could

- develop preliminary data flow diagrams showing the movement of data from server to client
- identify what data needed to be extracted or derived from our data warehouse.

With this information, we could design our data warehouse. Figure 1 shows the path of data from server to client for a single request for a trend plot for one physician and one DRG. Our preliminary diagram assumes that we can subset and download in the same step thereby transferring a small amount of data directly to the TREND data set needed to create the trend plot. This graph can be requested from the Physician Profile window or the Charge Variance window. In each

case, the download step is initiated by a remote submit (RSubmit). The trend plot window displays a plot using whatever data is found in the TREND data set.



➤ Figure 1 Sample Data Flow Diagram

Designing the Data Warehouse

The data warehouse, being a collection of integrated databases, can house different tables that store information at different levels of granularity. Serving all users, the organizational tables within the warehouse contain data at the greatest level of detail, which may even match the operational data from which it was derived.

In our example, the operational data resided at each hospital in disparate forms and most likely served as the patient billing and records information. The consolidated data that represents the organizational level of the data warehouse was provided in the medical database. The organizational level data was not, however, in a structure that facilitated the types of client-server queries we wanted to make, so we needed to construct additional tables.

Client-server applications usually connect to a data warehouse through a data mart or OLAP layer, where OLAP stands for online analytical processing. I will use the terms data mart and OLAP tables interchangeably. The OLAP tables are customized for specific users and may contain detailed data as well as summarized data.

The OLAP layer should provide easy access to multiple dimensions of the data. This means the user should be able to view data summed or grouped by any combination of variables. Obviously, the designer should evaluate which dimensions make sense or the data mart will explode. In our case, we knew exactly which dimensions were needed to satisfy the user interface requirements.

The OLAP tables could contain only a specific subset of data or they could have additional calculated variables that do not exist elsewhere. In other words, the OLAP tables are so customized for the target users that you do not need to worry about their effect on other users. Clearly, the data mart becomes our essential vehicle for optimizing client-server performance.

OLAP can be implemented in several ways. The method you choose will depend on software availability, the client-server application, and expected performance. Inmon identifies three primary approaches to providing OLAP:

- storing data using a star schema and developing code to access it
- using a commercially provide OLAP tool that can access normalized relational data
- using a multidimensional database.

I will cover only the first and the last methods in this discussion. The star schema is a structure that experienced SAS programmers have been using for years. Basically, the core of the architecture is a central fact table (data set) that consists of analytical data, whose values can be summed, and a unique set of key BY variables that identify the detail level of the data. Associated with or surrounding fact tables are groups of dimension tables, whose key variable matches a BY variable in each fact table. They contain additional variables that are dependent upon the key variable and would be unnecessarily redundant if they were stored in the fact table.

Evaluating the Data

Using SAS procedures such as FREQ, MEANS, or UNIVARIATE, you can analyze the number of observations that will be associated with each dimension or BY-group combination in order to calculate a mean or expected number of observations that will be generated from any query. If you know your data, you can just estimate that number.

For example, in our first window of the physician profile system, we display a row of statistics for each department at two hospitals. If each hospital has an average of 30 departments, our window typically manages 60 observations. This dimension of pre-summarized statistics in the data mart will contain approximately 30 times the number of hospitals in the database.

In the physician profile window, we can display statistics for

- all physicians for a selected department in the client hospital and all physicians in the same department at another hospital for the same time period
- or, physicians for a selected department in the selected hospital for a different time period.

Assuming an average of 15 physicians per department and two periods (years) per department, we would expect $30 \times 15 \times 2 = 900$ observations for each hospital in this dimension. The graphical user interface, however, only has to manage roughly 30 observations.

We continued this analysis for each dimension needed for the application. The largest dimension appears in the data that is needed to produce the trend graphs per physician and DRG. With 30 departments and 15 physicians and an average of 100 DRGs per department and 16 quarters, we have 720,000 observations per hospital. Fortunately, one comparative trend plot only needs 32 of those observations at a time.

Designing the Data Mart or OLAP Tables without MDDB Server

Applying the results of our data evaluation, we designed our OLAP tables. The original application was developed prior to the availability of MDDB Server, so we developed our data mart using SAS data sets. To design with performance in mind, we considered the following issues:

- Should all data for this application be stored in a single star with a large fact table that houses all dimensions for OLAP?

As you can see from the evaluation, storing all data in one fact table with all dimensions and all hospitals would create an extremely large SAS data set. Even with indexing, retrieval times would be long, so we decided to utilize multiple fact tables. This did not create a problem, because the tables would be created during a batch process executed weekly or even monthly.

- How should the table be split?

Because our application always restricted the user's choices to data from one or two hospitals, the first split point was obvious. Splitting the table by hospital reduced the size of the table dramatically. In addition, retrieval times became independent of the number of hospitals that were added to the data warehouse.

Theoretically, the hospital split should have been sufficient, but we wanted to guarantee fast retrieval times. Therefore, we took the added step of dividing the table further into a fact table for each dimension, which mapped directly to each of the possible reports and graphs. This structure ties well to the client-server application, as we will discuss later.

- How do we track so many tables?

The data warehouse metadata provides easy access to fact table locations or names. With a single check of the metadata, the application can easily and reliably retrieve the correct data based on the user's actions.

- When should calculated variables be created?

Because we were storing all of the dimensions including summarized data, it made sense to store the calculated variables in the data mart. Not only did this move the calculation effort from the retrieval process to the load process, but also it placed the logic for creating these values in one central location.

Designing the Data Mart or OLAP Tables with MDDB Server

Although SAS MDDB Server offers high speed OLAP, we still needed to evaluate some of the same questions as we did for the SAS data sets. The hospital level split still makes sense, even when using an MDDB. By splitting the data by hospitals, our client has the additional option of archiving data for hospitals that would not be used frequently. This is still true with an MDDB. Therefore, even with the optimization built into the SAS MDDB, we recommend splitting the OLAP tables by hospital. One other consideration is the amount of memory that is required to build an MDDB. The bigger the database the more memory you need, so splitting by hospital reduces the chance of encountering this limit.

The SAS MDDB is designed to handle multiple dimensions, so by using MDDB Server we do not need to split the data beyond the hospital level as we did with the SAS data sets. This, however, is where the client-server process and the flow of data interacts with the data warehouse design.

Designing the Client-Server Application -- Round 2

At this point, we need to refine our original client-server design based upon our data warehouse decisions. We can return to our data flow diagrams to optimize where specific processing occurs (on client or on server) and determine when and what data is downloaded. The data flow diagrams make it easier to visualize how to minimize the quantity of data that is downloaded.

Without MDDB

In our original implementation, the separate fact tables were ideal for fast client-server processing, because each request initiated retrieval from a small table and file transfers for only a small number of observations.

In the worst case, for example, a trend plot over 16 quarters comparing one physician to the overall department total for one DRG, requires extracting 16 observations from approximately 720,000 physician observations and 16 observations from approximately 48,000 departmental observations. No calculations are performed on the server.

For tabular listings, no other processing is necessary on the client. For graphs, however, the data is downloaded and the graphs are produced on the client. It is much faster to download raw data and generate graphs on the client than to download graphic images created on the server. In addition, by generating the graphs on the target client, you do not need to worry about the target display or plotting device. The choice of plotting device can be controlled at the client workstation avoiding compatibility complications on the server.

With the split table design, we only download data when we need it. The transfers are initiated per user request and with impressive speed, because the number of actual observations downloaded are small. We considered further optimizing the download process by tracking what data was already downloaded, so if the user wanted to redisplay the same table, a redundant download would not be necessary. The time savings would not have been sufficient to offset the additional complexity in the code and additional cost to the client, so we did not pursue that level of optimization.

When I use the term *download*, I am referring to the act of running PROC DOWNLOAD or issuing a remote LIBNAME and using PROC COPY. We could have used Remote Library Services and left the data on the server side. Instead, our application enables the user to sort the data within any tabular listing, which makes it more efficient to use a copy of the data on the client side. DOWNLOAD runs quickly, because we transfer such small data sets.

With MDDB

We are still considering this option, but the deadline for the paper preceded the completion of performance testing of this approach.

When using a SAS MDDB, we need to determine whether to

- access the MDDB through Remote Library Services, leaving the MDDB on the remote side
- download an entire hospital-level MDDB
- download each dimension needed when needed
- extract the required observations from the MDDB on the remote side and download the subset.

Generally, with client-server applications, you do not want to link to a remote file and initiate processes that will run on the client. For example, you should never issue a remote LIBNAME to a remote file and run PROC SORT on the client against a data set in that library. In this case SAS will download the entire file, sort the data set, and upload the sorted data. The sort will not occur on the remote server.

MDDB Server on the other hand is designed to work with Remote Library Services. You can leave the MDDB on the remote server and

access the data with a remote LIBNAME statement (issued on the client). Unfortunately, our application was customized to use the data in a different way than MDDB Server is designed to support. We could redesign our application to take advantage of this feature.

If an entire MDDB were downloaded, the users would experience a delay initially, but would benefit from high speed direct access to the MDDB on their own machine. MDDBs can be extremely large, so downloading the entire MDDB is not usually a reasonable option, unless the user has plenty of disk space and wants to disconnect from the network.

One of the primary advantages of the MDDB is the fast access to or transport of any dimension in the MDDB. Downloading a dimension could provide what we need, but in most cases it would still provide more than we need for this type of application.

We could use the Remote Compute Facility in SAS/CONNECT and extract the desired observations from the MDDB using the SASSFIO engine. Unfortunately, the SASFIO engine is still in test mode.

One of the performance decisions we made for our first implementation involved pre-calculating several derived variables and storing them in our data mart. The SAS MDDB provides offers several statistics that can be stored in the MDDB or dynamically generated, but our new variables were ratios or more complicated indices, which could not be simply summed across dimensions. This poses a problem for our optimization efforts. We will need to calculate these values as we extract the data, which can degrade performance by adding an additional pass through the data.

Summary and Conclusions

To improve client-server performance (response time), you should strive to minimize the amount of data that is transferred across slow connections, either network or modem. Also, study the processes within your application to distribute the processing to the most efficient processor, either the server or the client. Data warehouses can affect both of these performance factors.

Evaluate your user's requirements carefully to determine if a data mart customized for this application will increase performance, while still providing the flexibility desired. In the data mart, consider storing summarized data in order to reduce processing time during user queries. Also, consider physically splitting your data mart into separate OLAP tables to speed data retrieval. Remember that physical splits can reduce overall flexibility of the application, so evaluate this option carefully.

Use data flow diagrams to visualize the movement of data from the server to the client. After designing your data mart, optimize your client-server logic by refining the data flow diagrams.

If you need a true multidimensional OLAP solution, consider using the SAS MDDB Server, which provides all of the speed and features required for OLAP. If your application does not need all of the capabilities provided by the MDDB Server, you should still be able to utilize data warehouse and OLAP techniques with SAS data sets to optimize your client-server system.

Reference Information

Contact

Eric C. Brinsfield
Meridian Software, Inc.
12204 Old Creedmoor Road

Raleigh, NC 27613

Phone: 919.518.1070

FAX: 919.518.1170

email: merceb@meridian-software.com

WWW site: <http://www.meridian-software.com>

Bibliography

SAS/CONNECT Software: Usage and Reference, Version 6, Second Edition, 1994. SAS Institute, Inc.

Trademarks

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

Windows NT™ is a trademark of Microsoft Corporation.

¹ Inmon, William; Welch, J.D.; Glassey, Katherine L; *Managing the Data Warehouse*, 1997, Wiley Computer Publishing; pages 365-366