Paper # 4014

Designing Client/Server Applications for Performance

Dr. Mike W. Wang
Subo Ko
Information Systems Services and Technology
Hewlett-Packard Company
3000 Hanover Street, Palo Alto, CA 94304
Phone: (415) 857-4736
(415) 857-2540
Fax: (415) 857-5518

Introduction

In a typical client/server information collection and retrieval application system, there are client and server machines, networking components, and oftentimes databases to store data. With these multiple components, the performance of a client/server application system is much more difficult to control and measure than a non-client/server application. These multiple components also give rise to many challenges and opportunities in designing an application that will perform optimally in this environment.

In this paper, the authors present many different client/server application design guidelines and examples to illustrate how to achieve optimum performance. The authors also discuss how to design client/server applications to outperform the non-client/server approaches, how to identify false methods used in measuring client/server performance, and what are some of the desirable performance features of the middleware incorporated into the application.

Information Process Flow in a Client/Server Application

In a typical client/server application, there are four major components: the client program, the networking component, the server program, and the database component. Out of these four components, the application end users only see the displays of the client program. The other three components are "behind the scene" processes. The process flow in this environment usually begins with the client program asking the user for input. The user input then turns into information which consists of command(s) and data to describe the task(s) to be executed. This information will then travel through the networking component and be received by the server program. The server program then processes this information. If the tasks described by the information requires the access of a database, then the database engine is invoked and the required database operations are performed. Once the result of the database operations is obtained, it is passed to the server program. The server program then sends the result across network, and to the client program. The client program processes the result and displays it on the screen. This completes the information process flow for a typical client/server application.

In the Diagram 1, this process flow is described as steps 1, 2, 3, 4, 5, 6, and 7. Any tasks performed in a client/server application must go through these seven steps in the order described.

Now let us further consider the steps in the above mentioned information process flow. In each step of the process, there are computers (networking components are also made up of computers) doing the work. All these computer processes take time, with some of the tasks being completed in milliseconds. However, every component will contribute to the total time that it will take to processing a user request and receiving results. We can associate the
time taken to complete the tasks for each step as follows:

1. Client program processing time

This includes the time taken by the client program to display the initial screen, accept user input, and turn it into information to describe the tasks to be executed.

2. Networking time

The time spent in this step includes the time it takes to prepare the information into a format suitable for network
transmission, as well as the time taken for the information to be transmitted from the client machine to the server machine.

3. Server program processing time

This includes the time taken by the server program to receive and process the information for the client.

4. Database processing time

The time spent in this step includes the time taken by the database engine as it prepares to process the database request and the actual data retrieval time.

5. Server processing time for returned data

This is the time taken by the server program to retrieve the information from the database engine, possibly post-process the information, and send it to the network.

6. Networking time

The time spent in this step includes the time it takes to prepare the information suitable for network transmission and the time it takes to actually transmit the information to the client machine.

7. Client program processing and displaying the resulting data

The time it takes to receive, process and display the information on the client machine.

These seven steps are the basis for how to control and measure the performance of a client/server application program. In essence, to improve the performance of a client/server application is equivalent to improving the performance of each of the individual steps, and of the entire process as a whole.

Misconceptions in Client/Server Performance

The idea of fine tuning the components for each step of the client/server application data flow is obvious, and a correct approach towards improving the performance of the application. For instance, step number 4 listed above, database processing time, can potentially be the most time consuming task in a client/server data operation. Therefore, to improve the performance, it is vital to minimize the database processing time. In this case, the normally performed tasks in a non-client/server environment, such as designing an efficient database schema, normalizing the data for the database, creating indexing, and perhaps, spreading the database and indexes into different disk drives, using an efficient database engine, etc., must be taken into consideration in order to obtain an optimally performing client/server application.

By the same token, writing an efficient client program and an efficient server program will result in a better performing client/server application. Of course, increasing the bandwidth of the network (increasing the speed of the transmission) and selecting an efficient network product will most likely improve the performance of the application.

Along with the above described correct concepts and methods for improving a client/server application's performance, there are also misconceptions and incorrect methods, as discussed below, that are sometimes used in obtaining and interpreting the performance of a client/server application.

First Misconception

With the concept that client/server applications are made up of multiple components, it is very easy to think that if each component in a client/server application is tuned to perform optimally, the entire application will perform optimally. In reality, the fact is that fine tuning each component is only the first step towards achieving an optimally performing client/server application. This leads to the first misconception in client/server application performance, namely, that fine tuning every component will lead to an optimally performing client/server application. While it is true that fine tuning every component is necessary for obtaining an optimally performing client/server application, we must also consider the relationships between the individual component and the entire process as a whole. These relationships are vital in determining the performance of a client/server application. They are so important that most design guidelines for achieving better performance for a client/server application discussed in the later sections of this paper are all based on how to appropriately design these relationships.

Second Misconception

When the performance of a client/server application is viewed as inadequate, it is easy to blame the network as the source of the problem. This leads to the second misconception in the performance of a client/server application.

While it is true that network will contribute to some delay of the information passed back and forth between the client and server machines, and sometimes network traffic congestion will degrade the network performance, the slowness of an application response time is not always the network's fault.

With the TCP/IP network, most of the infrastructure is run at a speed of 10 megabits per second. In a fine tuned network, i.e., other applications are running fine, a single application is not likely to cause a network slow down and result in an inadequate response time for the application. With a proper application design and reasonable expectations for the response time, the performance of a client/server application can usually achieve its optimal design goals.

Third Misconception

In order to obtain the performance statistics for a client/server application, it is very easy for developers to write a client program to repeatedly send the same commands to the server for execution. It is very convenient for a program to use a FOR or DO types of loop statement to perform repetitive operations. For example, a program continuously issues 200 INSERT SQL commands to simulate 200 users using the application. After obtaining the total time taken by the program to complete the 200 SQL commands, the developer claims to have estimated the response time for 200 users using the application simultaneously. This is the third common misconception.

As described in the previous section of this article, there are 7 sequential steps that information (sometimes an SQL database command) needs to go through in a client/server application. In each of these 7 steps, the processing is completed by computer hardware and software, whether it's a PC, a networking switch, a networking gateway, an operating system, or a server program. Every component in this chain will need time to process the information and will have to queue the information for its turn before it can be processed. For 200 sequentially issued SQL commands to be processed, the total time will be the sum of the time it takes for each
component to complete its tasks for each of the 200 SQL commands. So, this is an additive time phenomenal and can be illustrated by Diagram 2:
As shown in the diagram, the time it takes to process all the commands is in a positive direct relationship with the number of commands processed. In other words, if one command takes 1 second to process, 200 commands will take 200 seconds.

Let us now consider the situation where there are 200 users, each using their own PCs. They are all using the same client/server application and each one of them issues one SQL command to the server machine. In this example, step number 1 in their information process flow is done on their own PCs. Thus, theoretically, the time it takes to complete step number 1 for the 200 users can happen simultaneously. That is, if step number 1 takes 0.1 second to complete, it will take exactly the same 0.1 second for 200 users to complete step number 1. This phenomenal is also true for step number 7, and partially true for steps number 2 and 6.

Since in most of the cases, the server machine in a client/server environment is fast, it will take little time for the application to complete steps number 3 and 5. If we are issuing INSERT SQL commands, the time it takes for a decent database engine to process this INSERT command is minimal. Also, for INSERT SQL commands, the length of the command and the result of this SQL command will be short. Thus, it will take negligible time for network to transmit. Thus when 200 users issue an INSERT SQL command to the server simultaneously, the response time for all users will have a relationship as described in Diagram 3:

Notice that in Diagram 3, the response time is initially flat for a number of users. Depending on the network and server capacity, this curve shows that a large number of users can be sustained before performance is impacted for all users. This phenomenal is observed due to the fact that each PC is taking different paths to reach the server machine. If it is assumed that the network has adequate capacity and the server is powerful enough to process the SQL
commands with minimum time, this flat curve can take a lot of user requests before it turns upward.

So, the next time you hear people claim that a client/server model is not suitable for processing your data, be aware of the above misconception. This is particular true for applications classified as so-called on-line transaction processing (OLTP). A typical OLTP application usually involves the processing of short SQL commands such as INSERT, DELETE, and UPDATE. Thus be aware of the performance simulation methods used for these applications.

Design Guidelines for Optimal Performance

We have seen that for a client/server application to complete a task, information must go through 7 steps. Each step involves the processing of the information and the queuing before being processed. Thus, for a given task, we can picture that there is a long processing and queuing pipe in a client/server application. Designing a client/server application to reduce the total length of the pipe for a given task is a way to achieve the optimal performance of the application. This processing and queuing pipe idea leads to the following design guidelines for optimal performance:

1) Reduce the "ping-pong" effect

The most devastating performance killer in a client/server application is the constant sending and receiving of small chunks of information. Because of the seven steps that every chunk of information must go through, a "ping-pong" type of application design will add the length of the processing and queuing pipe, and thus slow down an application dramatically.

An experiment was performed on an application where a character is sent to the server machine, and based on this character the server program returns a particular word to the client program. After a redesign of this application, the client program sent 30 characters together in one chunk. When the server program received
these characters, it processes them all and sends the resulting 30 words together in one chunk to the client program. This redesigned application's response time was about 30 times faster than the previous application.

Essentially, in the above experiment, the total length of the processing and queuing pipe was reduced by 30 times when 30 characters were batched together and sent all at once. This reduction in the length of the pipe leads to the 30 times performance improvement.

This astounding result is accomplished by only reducing the "ping-pong" effect of the application. Since the network and server machines have adequate capacities in this experiment, there is no difference in performance between processing 1 character or 30 characters at a time.

2) Pack more information together into one network transmission

Packing as much information as possible together into one operation essentially reduces the length of the processing and queuing pipe. It also reduces the overhead, such as the network headers that networking components must add and strip. Thus, this guideline improves performance, especially in cases where short messages are involved.

3) Reduce the amount of information to be sent across the network

If possible, design an application with program logic that sends and receives as little as possible between the client and the server programs. This practice improves performance by reducing the number of characters transmitted through the network. For example, do not send detailed records between the client and server programs, if the detailed records are not the final form of information needed. Send only the necessary information. The information to be sent across the network can be summary data; or can be processed and condensed data. This guideline will reduce the traffic between the client and the server programs.

4) Separate the application logic so that both the client and the server machines do their share of processing

Assuming that the server machine has a faster processing speed than the client machine, this design guideline takes advantage of the server speed and makes the server do the majority of the processing. For example, if data sorting or other heavy computations are necessary, do it on the server machine. On the other hand, if the server machine is loaded down with other processing, make the client machine do the heavy processing work. Therefore, the design guideline is to separate the application logic between the use of the client and the server machines, and achieve a benefit of using both machines' processing power.

The following two guidelines deal with the processing overhead and redundancy possibly existing in client/server applications:

5) Maintain the client and the server logical connection

To logically connect a client program and a server program in a TCP/IP network, a developer usually uses a socket based networking API. In this case, the server machine must create (fork, for UNIX operating system) a process, start the server program, check the security of the user sign on, etc.. These tasks take a fairly large amount of time when compared with other tasks performed by the server machine. In this case, the design guideline is to logically connect the client and the server programs only once and use the same logical connection for subsequent information exchanges between the client and the server machines. Of course, do not include the connection operations in a loop if it is not necessary.

6) Keep the database open

Once a database is open, leave it open for subsequent database operations. Opening a database takes considerable time when compared with other operations. These tasks involve creating a process on the system, checking user security, updating lock information, opening log files, etc.. Thus, if possible, only open the database once for all operations.

7) Use a 3GL for input and output operations

When a client program is written using a 4GL, the performance can be improved if intensive input or output operations are performed by a 3GL routine in the client program. This guideline holds true especially for 4GL programs which require table building tasks. As an example, we have used a C language routine in a Microsoft Visual Basic program for building a table and saw the response time improved 5 fold.

Other application design guidelines which will help the performance of an application are described in the "Performance Features of Middleware" section of this paper.

Designing Client/Server Applications to Outperform Non-client/server Applications

In the preceding sections of this paper, we have shown that the use of the programming capability and the CPU cycle power of both the client and the server machines can help us to obtain a better performance for an application. This added flexibility in designing a client/server application allows a developer to design applications that perform significantly better than a non-client/server counterpart.

Another characteristics that exists in a client/server environment is the distributed nature of the application. With this property, we can use more than one server machines to complete a given task, and thus to allow developers to design client/server applications that will outperform non-client/server approaches. The following two design examples
are used to illustrate this effect:

1) Use multiple server machines in an application system

When the tasks performed by the server program in a client/server application can be split over multiple server machines, the overall performance of the application will be better than using only a single server. In this case, multiple server machines are used simultaneously, thus providing a larger CPU cycle capacity than a single server machine. This design method is called Parallel Processing Client/Server Design Structure. Refer to reference number 1 for more details about this design structure.

The multiple server machine design approach can be used in CPU intensive applications such as large simulations or graphics applications. In this design, one machine can do a portion of the simulation or graphics rendering and the other machine can do some other part of the work. The design structure of this type of application is illustrated in Diagram 4:

In this example, if the speed of the server machines are identical, it is possible for two server machines using a parallel processing design structure to outperform a single machine design (non-client/server design) by a 2 to 1 ratio. Of course, if more than 2 server machines are used, the ratio will probably be even higher.

2) Use multiple database engines on the server machines

When an application can split and place its information in multiple databases on multiple server machines, it is possible that the performance of a multiple database design can be many times better than a single machine design (non-client/server design). This design can be particularly useful in multiple site applications or multiple timeframe (multiple months and/or years) applications. Diagram 5 shows a multiple database design:

The example in Diagram 5 is a design structure with 1 client machine and 3 server machines, with each server machine having its own database. If the entire body of information for the application is evenly distributed over the three databases, it is possible that the performance of the database operations could be up to three times faster as compared with a single database on a single server case (non-client/server design).

As an example, this design approach is used in a marketing history database application in our facility. In this application, the data are divided by month for a one year time span. Then, there are 12 server machines with a database on each machine. Each database contains one month of marketing history data. When a report needs the information for the whole year, the client program sends requests to all 12 database servers at the same time. The database servers will work and retrieve the information simultaneously. Thus, in some experiments, we have seen the data retrieval time was 12 times faster than in a single server machine single database design
(non-client/server).

Performance Features of Middleware

In a client/server application, middleware refers to a software tool that facilitates the communication between the client machine and the server machine. Such a tool can be developed in-house or provided by a vendor. When it is provided by a vendor, it usually comes as programmatic callable subroutines. These subroutines are sometimes called an application program interface (API). The major goal of this API is to provide easy to use routines which facilitate the communication between the client and server machines; and thus eliminates the needs for programmers to have to code the lower level communication layer in an application. Therefore, for its simplicity and fast development time, most client/server developers are using ready-made API to construct their client/server applications.

Some vendor's middleware provide not only the communication function between the client and the server machines, but also provide the function of accessing databases on the server machine. Most database vendors provide database access APIs for client programs. The client/server application design structure using middleware is described in detail in the article "Client/Server Application Design Structure" referred to as reference #1 by this article.

As examples of using middleware, Diagrams 6 and 7 illustrate a typical use of middleware in two-tier (Diagram 6) and three-tier (Diagram 7) client/server applications:

Note that in a two-tier client/server application, developers only develop the client program. The server program, in this design is supplied by database vendors. In a three-tier client/server application, developers must develop a server program in addition to the client program. As you can see from the above diagrams, middleware is intertwined with the client and the server programs, and thus plays a vital role in the construction and the performance of client/server applications. Some of the desirable performance features of middleware are discussed in the following text:

1) User controllable buffer size

The buffer size in this context refers to the memory size allocated to store the data being transferred over the network between the client and server machines. When the buffer size can be changed by the program, application developers can accumulate more than one message before the actual transmission takes place. By buffering more than one piece of information into one send or receive operation, the total number of sends and/or receives between the client and server programs can be reduced. This reduces the length of the processing and queuing pipe for the application. As discussed in the previous section, this capability of the middleware can potentially dramatically improve the application response time.

2) Bulk send and receive

The bulk send and receive feature refers to the capability of the middleware to continuously transmit the data until possibly exhausting all the data in the operation. For instance, multiple rows of data in a database can be returned by one operation for an SQL SELECT operation. A bulk transfer of all the rows of data will reduce the total number of send/receive operations, which in turn reduces
the amount of handshaking between the client and the server machines, and improves the performance of the application.

The bulk send and receive feature is very important for applications that require a large number of data records to be transmitted between the client and server machines. This feature is particularly helpful in information access types of applications where a large number of rows are transmitted from the database to the client machines. By using this feature in experiments conducted by the authors, some applications have shown a 10 fold reduction in response time.

3) Bulk INSERT/UPDATE/DELETE database operations

When a logical connection is maintained to the database, the SQL commands are issued, and the results are sent back, for one command at a time. In this case, if the results of the SQL commands are short in length, it will cause a "ping-pong" effect for multiple SQL commands. This is particularly true in the case where a large number of INSERT/UPDATE/DELETE SQL commands are issued. These commands are typically short in length and the results returned will only be short status messages to indicate the successful or unsuccessful result of the operations. Therefore, if a middleware can handle a bundle of multiple SQL commands in one send operation and return the results back in one send operation, then the "ping-pong" effect is eliminated. Thus, this feature is a very desirable performance feature for the middleware to have, especially in an online transaction processing (OLTP) environment.

4) Parallel processing capability

As we have indicated earlier, parallel processing is one way for a client/server application, through proper design of the program and/or the database, to significantly outperform non-client/server approaches. Thus, this feature is very desirable in selecting a middleware.

5) Three-tier design structure capability

When the middleware offers the capability of designing three-tier client/server applications, there are many performance benefits that can be realized by using this design structure. Three-tier design in this context means that developers can develop both the client and the server programs.

We have seen in the previous section of the paper that when client and server programs works together, we can design an application which reduces the length of the processing and queuing pipe; thus improve the performance dramatically. The following examples further illustrate, through proper design, three-tier client/server program can provide improved performance.

When the application can split the portion of the program which involves heavy CPU cycles and disk I/O usage to be run on the server machine, performance of the application is most likely improved. This is usually because in a client/server environment, the server machine is much faster than the client machine. With a faster machine doing more work for a giving task, the performance of the application will be improved.

As a second case where the performance of the application is improved by using a three-tier design structure, consider the situation where the SQL commands are known by the developer ahead of time and can be embedded (hard-coded) in the server program. In this case, developers can take advantage of this knowledge and embed the SQL commands in the server program. Embedding SQL commands in a program will allow the data structures for the database to be built during server program compilation time rather than at execution time. This method will potentially provide faster database access times for applications which require the retrieval/update of a single record (row) of keyed database elements (columns). When the number of records retrieved by the database engine increases or the data element retrieved is not a keyed database element, this
performance advantage may not be significant. This is due to the fact that the data structure for the database needs to be built only once for a given SQL command. The time it takes to build a data structure is overshadowed by the larger amount of time required to retrieve the records.

6) Local database manager

The database manager is a server program provided by the middleware that access the database. It usually resides on the same machine as the database. In a three-tier client/server application, the database manager usually resides on the same machine as the user's server program, thus data transferred between the database manager and the user's server program will not travel onto the network in a TCP/IP network environment. This is good for performance. However, some of the networking tasks such as performing socket calls, building TCP/IP headers, detecting its destination to be its own machine, etc., must still be performed. Therefore, if the database manager can be linked to the user's server program, and these networking tasks are eliminated all together, the performance of the application will be improved.

The middleware feature of allowing the data manager to be linked into the user's server program without the need of including the networking protocol in program execution is called a local database manager. In this way, the database manager acts as a local routine to the user's server program.

Conclusions

The goal of obtaining good performance for client/server applications tends to be more complex than for non-client/server applications. The added networking layer for the client machine, the actual network itself, and the server machine make performance tuning more difficult. However, the authors have
demonstrated that with proper application designs, we can achieve the goal of obtaining performance improvements by many times, compared with non-client/server applications. We can also design client/server applications which minimize the impact of networking components. Thus, the performance of some of the client/server applications will be better than the non-client/server applications with the networking layer. We can obtain superior performance by using not only the server machine but also the client machine as a part of the computing components. As the client machine works together with the server machine, they can give us optimally performing applications.

Readers have also seen the effects of the middleware. The performance features of the middleware can add to or limit the application's design and thus performance. In most cases, careful selection of the middleware is one of the most important factors in determining the optimum or mediocre performance of a client/server application.

References

1. Client/Server Application Design Structure; Mike Wang, Interex 1995; Interact October 1995; HP Omni October 1995
2. Using PC, HP3000, HP9000, and IBM Machines in Distributed Client/Server Applications; Mike Wang, Interex 1994; Interact February 1995
3. HP Intelligent Warehouse Architecture Primer; HP publication, February 1995
4. Allbase/XL SQL Reference Manual; HP; 1989
5. Microsoft ODBC Programmer Guide; Microsoft; 1993
6. Microsoft Visual Basic; Microsoft; 1993

Acknowledgments

The middleware used in the examples of this article has the code name SOLID. This middleware was designed and developed by Arthur Walasek, Christine Hsu, Phil Blocher, Keith Freiberger, Don McKee, Supin Ko and Subo Ko of Hewlett-Packard. They have devoted much wisdom and hard work, time and time again, to maximize the usefulness, performance and simplicity of this middleware. Special thanks also go to Chuck Sieloff, Jack Domme, and Guy Randazzo for their support and commitment in making SOLID a middleware used in HP internal business applications, in the Intelligent Access product, the Otter product, the HP Information Access product, and the HP Intelligent Warehouse product.