Payoff

Many IT organizations find themselves continually enhancing client/server applications that were never designed to operate in today's distributed environments. Understanding the goals and components of an effective performance-engineering plan helps IS managers ensure that they receive the information they need to make major decisions concerning hardware and software and that computer systems provide timely support to users.

Problems Addressed

The secret of software performance is unwittingly tied to and ultimately saved by the presence of ever more powerful hardware. Consider, for example, the case of a production application designed in 1991 to service 10 users in an early 486 Intel environment. When it exhibits slow performance, the server is upgraded to a faster 486 processor. This helps for a while, but as more users employ the software, performance sags again until Intel comes to the rescue with speedier processors. In 1994, the scope of the application changes, the software is enhanced, and additional users access the system. Although everyone is pleased with the system's functionality, performance once again becomes an issue. Luckily, dual-processing, P5 servers come online that help provide adequate response time.

Where does it all end? Although no one purposely builds software that does not run robustly or meet the performance needs of the business, IT organizations continually enhance client/server software that services an ever larger base of users. These legacy (if one daringly uses the term once solely reserved for mainframe software) applications were never designed to operate in the distributed environments they typically operate within.

Although there is no shortage of attention focused on client/server architectures, tools, system services (e.g., Relational Data Base Management System, transaction-processing monitors, and messaging middleware), and environments, performance engineering remains one of the least discussed and understood of the many arts and sciences surrounding client/server computing. The increasingly distributed nature of applications and the wave of Internet-enabled applications soon to hit the beaches render few topics more critical to today's IS managers. This article aims to refocus attention on the long-standing challenge of making client/server applications perform.

Software Performance Basics

There are four rules of software performance:

- A computer can never have too much memory or too fast a Central Processing Unit.
- There is no such thing as software that has been optimized enough.
- There is no such thing as too much network bandwidth or line speed.
- See rules 1-3.
One reasonable question regarding application performance might concern how long it will take to process a certain number of records. Another might ask about the minimum time needed to satisfy a query. Although these are excellent questions, they are impossible to answer. Asking them is like asking bike riders about the maximum speed they will reach on a certain day. By pedaling as hard as possible, the riders will go as fast as they can. But speed depends on other factors, such as whether a rider is pedaling uphill or downhill, and whether a rider got a good night’s sleep or is not feeling 100%.

Similarly, hundreds of questions—from both the software and hardware points of view—must be addressed during an application's design, development, and implementation stages. A sampling of questions includes: What type of servers should be deployed? How many servers are needed? How big should the servers be to handle the application volume? How can we tell if the servers are optimized for the Network Operating System and communications infrastructure that is in place? Are applications properly configured for a mixed-platform environment? Can traditional resource management functions (e.g., backup and security) be easily accomplished for an architecture under consideration? What workloads run at each node? What hardware is needed to maintain service due to workload growth? How many more users can each existing server support? If the number of transactions increases by n%, what will the server utilization’s become? What server do we place which workload on? Which workloads can be mixed on servers? What if we add more memory to the server? What if we change servers or operating systems? What happens when the transaction volume increases? Is the database optimized?

Desktop hardware, network infrastructure, communication schemes, generated code, database access, operating systems, and system services all play a role in determining how well an application executes. In most organizations, however, network and systems management are handled by two different staffs. The network management organization focuses on bridges, routers, and other networking hardware, and the systems group concentrates on the application side of things, pre- and postimplementation. Thus performance engineering, metering, and similar topics fall into the cracks.

Goals of Performance Engineering

The Mission Statement

An effective performance-engineering plan begins with a performance mission statement and the development of a performance-engineering job function. The mission statement outlines the goals of performance monitoring and planning, which might be stated as follows:

- To ensure that computer systems provide timely support to end users and that management personnel have the information they require to make major decisions concerning hardware and software systems. These objectives are accomplished by identifying, evaluating, developing, and sharing tools and techniques for measuring, evaluating, reporting, and optimizing application and system performance.

Effective Measuring and Reporting

Monitoring encompasses a variety of qualitative and quantitative factors. Qualitative monitoring focuses on all-or-nothing availability, including, for example, the availability of
an enabling subsystem, key hardware element, business-critical server, and network connection. Quantitative monitoring requires measurements and is intrinsically real time. It includes, for example, the degree of success in meeting response time service targets, transaction turnaround time, transaction throughput volumes, and items like queue depths and service delays.

Effective application-performance measuring and monitoring encompasses the consideration of diverse workloads on multiple platforms. Once benchmarks are established and performance is tracked across the enterprise, the information should be reported to appropriate levels of end-user and IT management for action. Initial issues to be addressed include:

- Delivery of service.
  - Response times.
  - Turnaround times.
  - Transaction rates.
- Lockout or delay conditions.
  - Network traffic.
- Disk capacities.
  - CPU utilization.
  - Rate/queue/utilization of response-critical elements.
  - Status of hardware elements, software subsystems, and connections.

The many measurements made during monitoring are intended for after-the-fact analysis that focuses on inferring the need for operational, tactical, or strategic action. Thus, for example, if there is no possibility of increasing Central Processing Unit resources, there is little need to measure CPU utilization. Success or effectiveness of a measurement plan is based on the productivity of the people who receive and analyze the data. If a person spends 10 hours reading the data just to suggest the possibility (after further study) of one minor tactical action, the cost of wasted time may be too high.

Measurement should therefore aim to detect conditions or situations requiring action, and reporting based on such measurements should clearly highlight such items. Examples of valuable, action-oriented pieces of information include such statements as “average CPU utilization is at 83% of capacity” or “accounts receivable transactions have average response time of 3.2 seconds.”

It is important to avoid the measure-everything syndrome. Measuring what might be needed to solve every conceivable problem is subject to the law of diminishing returns. It takes a considerable amount of time, effort, and DASD to collect the data, which, once collected, needs to be analyzed. Extreme amounts of data yield results that are at a minimum confusing if not contradictory and at worse misleading.

If routine monitoring indicates an absence of problems, nothing else is needed. However, even if there is no problem today, many professionals in the business of
delivering production systems choose to worry about problems that might occur tomorrow. Given perfect clairvoyance, we can predict what problem will appear next and gather the information needed to resolve it. Mere humans, however, might find it more useful to collect the kind of data that can be used to discover slow-growing threats to service delivery. This data includes:

- Activity and performance details.
- Details of queue congestion.
- Analysis of key workloads and the affect of workloads on one another.
- Details of each service interruption (e.g., date, time, duration, impact, and cause).

**Benchmarking and Network Tuning**

Benchmarking client/server performance requires taking both a component-based and a systemic view of an application. In other words, it is not enough to focus solely on performance at the Central Processing Unit or network level; metrics must be gathered for each component and blended into a planning model for the entire application system.

Networks can be tuned based on the analysis of collected information and a determination as to where a problem lies. For example, increasing server cache memory or segmenting the network with multiple server cards may improve throughput. Although not always possible, splitting applications across servers divides the application workload across multiple processors. Maximizing network bandwidth (e.g., from 56K bps to 256K bps) offers many benefits as does using faster Network Interface Card (NICs) like fast Ethernet (e.g., transitioning workstations from 10M bps to 100M bps). Using native application programming interfaces (APIs) across all platforms and environments minimizes translation time. Of course, because many performance problems are related to data acquisition, optimizing database access and structured query language (SQL) statements must be a priority.

Collecting the right information is only the first step. Measurements are meaningful if they can be quantified and correlated against applications strategic to the business. This is a key point. The goal is not to measure performance but to make applications perform as robustly as possible within the constraints of the environment.

Performance must be considered early in the design/development phase. The advent of Rapid Application Development allows for prototyping and benchmark performance testing to be performed simultaneously to the benefit of both.

The main purpose of this testing is to gather the metrics necessary to determine the application's implementation strategy, including recommendation of the number of database servers to deploy, definition of the location of servers vis-a-vis users, client hardware configurations, preparation of recovery and availability plans, and identification of metrics for collection throughout the system's production life.

Early on, the benchmark environment is designed to represent a subset of the production environment. Although there are limitations to this approach, it does provide a basis for scaling the results to the implementation environment, especially concerning the actual number of users and transaction volumes. Caution and conservatism are the bywords to follow in pursuit of the five key objectives of the performance benchmark test.
Objectives of Benchmark Tests

The five test objectives are:

- Minimizing costs regarding servers and client workstations.
- Managing end-user expectations
- Identifying bottlenecks within the application and hardware architecture.
- Baselining performance to establish reasonable throughput and response time from critical system components.
- Defining planning metrics for ongoing capacity planning and system performance management following the transition to production.

Minimizing Costs.

Costs should be minimized for the data base and application servers and for the purchase and maintenance of client workstations. Each server requires a significant investment in hardware and systems software and, in the case of the data base and certain other software, in ongoing licensing costs.

Managing End-User Expectations.

Purchased applications allow for little control over many performance characteristics. Applications built internally eventually reach a state where little more can be done from the performance point of view or, as is more likely the case, available resources must be used for other purposes. Test results should therefore be used to demonstrate the most likely performance that system users can expect.

Identifying Bottlenecks.

Performance delay occurs throughout the enterprise environment, which comprises the workstation, network, and server(s), and hardware, software, and system services. Complex transactions, a poorly designed data base or data access strategy, and limited bandwidth and Central Processing Unit cycles are just a few of the possible causes. Although understanding how data is manipulated is detailed, tedious work, it is important to determine where bottlenecks might occur when the system is implemented and as the user community grows.

Baselining Performance.

The performance and throughput of system components must be baselined during the benchmark to provide a measuring stick against which to evaluate future performance in the production environment. This is similar to regression testing when application enhancements are tested before going into production.
Defining Planning Metrics.

Testing that is formulated and performed early and often provides the opportunity to gather performance metrics and evaluate their utility for capacity planning and performance management.

Because performance issues often take a back seat to other more pressing development tasks, it is important to meet the testing objectives with a minimum of time and effort. This is best done by following a methodology, either purchased or developed in-house, that is tailored to the specific requirements and scenarios of the application. For example, it is not necessary to spend a lot of time testing remote access if there will be few remote users.

Types of Benchmark Tests

A variety of technical testing scenarios should be executed during the benchmark. Each scenario is designed to highlight the performance characteristics and limitations of one specific, isolatable component in the client/server architecture. Commonly performed transactions, data base queries, calculations, and other functions provide the means of testing individual components. SQL optimization and the data base design can be tightened and tuned through this procedure.

Although different applications require different types of tests, the following five general categories of tests should be executed:

- Shared file server test.
- Local file test.
- Remote access test.
- Clean room test.
- Volume test.

Shared File Server Test.

The shared file server test aims to determine the implications of implementing the application using the intended server to store and process application code and data. Testing access from or to remote facilities is also performed in the native production environment.

It is not unusual for the development organization to have a different access mechanism or a simulated environment that differs from the user communities’ environment. However, accurate reflection of what the system will do when it goes into production requires use of the same protocols and infrastructure users will employ. If, for example, users will employ an Ethernet link with Transmission Control Protocol/Internet Protocol (TCP/IP), the benchmark should be similarly configured.

Local File Test.

The local file test evaluates the performance improvements that can result from storing application code and data directly on the client workstation. This configuration often reduces both LAN traffic and the time necessary to load screen forms and application code.
Generally, some code and data are stored locally, and other code and data are stored remotely, on the server. This test helps pinpoint what should go where and what tradeoffs are worthwhile from a performance point of view.

**Remote Access Test.**

The remote access test evaluates the application's performance for end users located at sites across a WAN or requiring dial-up support. Client software may be executed from several remote sites across a WAN. To determine the average network latency that can be expected from a remote site, data accessed across a WAN or remotely should be measured and compared to data accessed during the shared file test. Because mobile system users expect the same performance regardless of their location, they should be made aware of the change in performance characteristics as they access application components from various locations and gateways.

**Clean Room Test.**

The clean room test illustrates the best possible performance available for the application. The client workstation is connected directly to the same network segment as the database server. Ideally, this network segment is segregated from all other traffic, and all applicable application code is stored locally on the client.

The clean room test is useful both as a theoretical target and for managing user expectations. For example, if the application takes seven seconds to bring up the main menu in this test, it is unreasonable for users to expect better performance in the production environment.

**Volume Test.**

The volume test evaluates the application's performance during a simulated peak period with heavy data, transaction, and user volume. Applications are unique and exhibit individual usage and volume patterns that change based on a variety of internal and external factors. Applications that exhibit seasonal patterns are especially worthy of note. Consider a swimwear firm's newly developed order-processing system. If the system is developed in December, volume testing must be based on anticipated system and network utilization during the spring and summer months.

A functional definition is needed to identify the typical workload that will appear during busy or peak periods, including complex transactions, queries, or processes that originate from a mouse click, secondary transaction, or other possibly external source (e.g., online financial service). Asking users how they will interact with the application to accomplish their assignments during the defined busy period helps ensure that tests are properly focused.

Applications also, of course, exhibit different behavior at different times of the day. Although it is often difficult to simulate heavy volume (i.e., transaction- or concurrency-based), it is important to work around this limitation. Test software is available to simulate these types of conditions. It is vital to consider the number of anticipated users both when the system is initially implemented and sometime in the future when the user population, and associated transaction volumes, have grown.

A throughput analysis offers a view of the end-to-end processing characteristics of the enterprise, from client workstation to server. For example, testing might indicate that each client Central Processing Unit can process 500 transactions per second. The network
interface cards handle 2,400 packets per second. The LAN is capable of carrying 1,200 packets per second before saturation. The server's CPU can handle 120 transactions per second, and the server's single disk performs 30 I/Os per second. In this case, the system is bottlenecked by the server’s disk throughput and will perform only as many transactions as 30 I/Os per second permits. Several excellent tools are available for collecting this type of data along with concurrency and transaction-load information.

In addition, several operating systems provide utilities to help monitor a variety of performance characteristics. In the UNIX operating system, for example, server performance data is gathered using the standard UNIX administrator facilities. System-level statistics are collected using the `vmstat` command; a five-second interval for the length of the test is a good starting point. Disk I/O statistics are captured with the `iostat` command, and as is the case with `vmstat`, a five-second interval for the length of the test is a good starting point. Finally, process-specific resource usage (e.g., CPU utilization, memory access, and paging) statistics are gathered using the `ps` command. Specifically, `ps-aux` collects statistics on all processes and is automatically called every five seconds by a shell script.

Collating and analyzing the collected benchmark data is no simple task, but the results are surprising and rewarding. Knowing how much memory, DASD, bandwidth, or CPU cycles are required helps IS managers and their staffs to purchase hardware more systematically and often helps reduce hardware costs. For example, 100 Pentium 133 MHz workstations with 24 MB of memory may be reduced to only 50 of this type and 50 Pentium 75 MHz workstations with 16MB of memory. Even if costs are not reduced, the hardware is more appropriately matched to the requirements; for instance, faster server, increased server DASD is accessed by slower workstations with less memory.

Once analysis is completed, a performance model is created to evaluate what-if scenarios, such as which regions to locate servers, how much memory is required for workstations/servers when phase 2 is implemented, and where performance bottlenecks might occur. It is also possible to establish reasonable performance expectations for various classes of users (e.g., local and remote) and to project performance on a variety of client workstation configurations (486 versus 586 processors, 8MB versus 16MB of memory). Remote 486-based laptop users, for example, should expect response time in the five-second range for certain types of transactions and under two seconds for other transactions. This is valuable information that helps users anticipate how an application will perform in circumstances similar to their own. System users who know what to expect are much less likely to call the help desk or to complain about performance.

**Modeling and Analysis**

Both client and server have their own network of resources (e.g., DASD, terminals, peripherals/adapters, processor). Predicting performance of distributed networks is complex owing to the number of nodes, their interrelationships, the options for networking connections (both local and remote), the number of components, and the diversity of environments (especially where client and server are running under different operating systems). The objective is to optimize the performance of individual applications, groups of applications running simultaneously, and the network enterprise as a whole, contiguous unit. Thus the effects of changing transaction volumes, adding new applications, moving applications between systems, and modifying hardware and network configurations must be predicted. This process requires that models of each node be built and evaluated and the
performance of key applications enhanced through hardware review, software revisions, and infrastructure changes.

The results should show the distribution and make-up of actual or potential response times and allow for simultaneous comparison of data from several models based on the predicted performance of the network under differing circumstances. The constituent portion of each network member has to be approached differently, because the transactions, services required or supported, and transmissions comprise either different elements or similar elements reaching the node at different times.

The critical part of analyzing large amounts of performance data is determining what is significant. Complex network configurations and high-traffic volumes necessitate that most performance analysis currently focus on the network itself rather than on the applications operating across the network. Multiple applications executing concurrently often leave the network as the weak link in the infrastructure architecture. This is a generalization, however, and the components of the application itself or other pieces of hardware (e.g., data base servers) also require monitoring and analysis. Network analysis generally focuses on high network I/O, network congestion, and faulty hardware. Network analysis performed on a per-segment basis is most useful.

The following five actions are necessary to gather the information for the development of a realistic model:

- Defining workload characterization.
- Surveying users for upcoming resource requirements.
- Forecasting workload levels.
- Modeling potential new configurations.
- Generating reports for management.

Rules of thumb—simple, well-known observations with obvious application to performance evaluation—can help staff collect information quickly. One rule of thumb, for example, is that Windows 95 client workstations should never have less than 16MB. Rules of thumb are excellent for day-to-day operations of installations, but they are not much help in forecasting hardware and software upgrading requirements. Thus they are useful in estimating limits but limited when used to represent complex interrelationships, like the impact of upgrading a Central Processing Unit. They are most useful when they are system- and application-specific (e.g., a specific application operating on a particular 486 workstation).

Statistical forecasting offers a more sophisticated alternative to rules of thumb. Forecasting is usually applied to workload volume (e.g., number of transactions per day) and utilization. It involves predicting, based on past history, items such as CPU, disk, and channel utilizations, as well as workload volumes. Queuing models use mathematical formulas that express relationships between workloads and performance measures. They can represent complex environments at a single instant in time but are rather difficult to define. Complex linear and nonlinear relationships should be determined to best approximate performance factors.

The modeling process results in answering the fundamental question of what hardware/software environment is needed to support a given workload(s) at a prescribed
volume while ensuring that service-level objectives (e.g., response time and turnaround time) are being met.

Workload characterization is another key concept of examining all of the executing entities (e.g., programs, processes, tasks, and applications) on a system and classifying these into a small number of sets. For example, although there might be thousands of transactions that execute on a system in a given day, the transactions may be classified into a handful of sets based on the type of application being performed. That is, all of the payroll transactions would fall into one class and all of the accounts-receivable transactions would fall into another class.

Sometimes it is beneficial to characterize work by similar function or resource consumption. For example, all transactions using a small amount of CPU time coupled with doing a very small number of I/Os fall into one class, and those transactions using a high amount of CPU time coupled with doing a large number of I/Os would fall into a different class. The objective here is to reduce the potentially large number of items into a much smaller summarized set. This makes it much easier to determine the cause and effect relationship between applications, resources consumed, and the positive or negative effect of potential changes. It also allows a dollar amount to be placed on different summarized sets to ensure the maximum return on investment from, at least, the performance point of view.

If a summarized set of transactions are running on a particular server, and they have access to its required data on that server, additional network access is not required. Ideally, transactions in a mission-critical application should be designed in such a way as to minimize network access, because that, in turn, minimizes response time by eliminating unnecessary network server nodes. The problem of process and data dependencies should also be considered. Because transactions in client/server environments often visit many servers before completion, there may be a dependency as to which workloads must complete first in order to be synched with other executing workloads. It quickly becomes obvious that there are many determinant factors in deciding how to mix and match application characteristics from a performance perspective.

**Recommended Course of Action**

The exponential increase in the deployment of distributed client/server applications across the enterprise is just the tip of the iceberg. Lurking below the waterline are the myriad transactions, messages, DASD requirements, Central Processing Unit utilization, data access/delivery, memory requirements, and communication issues waiting to slow an application down. Anticipating demands placed on the infrastructure, emerging software environments, hardware platforms, and the unexplored opportunities offered by the Internet is a delicate balancing act.

Because applications designed to efficiently utilize their resources while minimizing the impact across the network are a must, IS managers should take a proactive approach to performance engineering. First, they should ensure that baselines are established, because it is impossible to evaluate abnormal network activity when past activity has not been tracked. Exceptionally large networks necessitate a focus on exceptions, because there is simply too much data to analyze and little time to waste on nonproblem areas. Finally, IS managers and professionals new to performance management should take care to correlate significant events rather than look at problems in isolation. Performance engineering requires considering both the forest and the trees.
IS managers should establish and support a specific plan of action with identifiable and measurable performance goals. The plan should also include a slot for a software performance engineer, someone who understands the user environment, application infrastructure, and underlying topology. Collecting the appropriate information, effectively evaluating it, and then knowing what changes, if any, should be made is not a simple undertaking.

The key pieces of data that require analysis include response times, transaction rates, network traffic from competing resources and applications, CPU utilization, and the rate/queue/utilization of response-critical elements from node to node. A variety of tools and techniques should be used to collect and aggregate this information. Some of these are available with utilities built into operating systems.

Performance engineering for client/server environments involves many challenges. Instrumentation, data collection, and analysis are key across platforms. Modeling tools must allow topologies to be easily defined and address heterogeneous combinations of hardware and system software platforms. Although infrastructure issues and the sheer volume of collected performance data complicate analysis, no mission-critical applications should be deployed in an organization that has not addressed some of the basic system performance issues. It is time to stop relying on faster silicone and next-generation hardware by confronting performance head-on.

**Author Biographies**

Steven Rabin

Steven Rabin is chief technologist at American Software, Inc., in Edgartown MA.