CHAPTER 3

A Performance Management Methodology

Although performance management and crisis management (including performance problem resolution) require different techniques and data collection, the same basic methodology can be used for both. Performance management uses the following steps:

- Assessment
- Measurement
- Interpretation and Analysis
- Identification of Bottlenecks
- Tuning or Upgrading the System

The flow chart in Figure 3-1 summarizes the performance methodology that will be discussed in detail, step by step.

3.1 Assessment

Assessment often involves asking lots of questions, as the following example shows.

From Bob's Consulting Log—I spent the entire morning asking people a lot of questions about the application, the system configuration, and how users perceived the performance problem. At lunch, the person who hired me said he was surprised that I had not yet logged onto the system. I told him my approach is to ask questions first and collect system data second. Later that day, I did obtain some data from the system, but the most important clue to the nature of the problem came from interviewing the users.

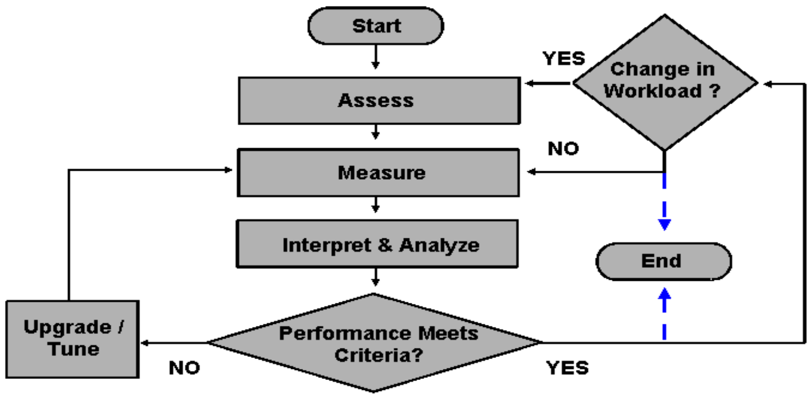

Figure 3-1 Performance Management Methodology

These questions are necessary to help you understand your limits as a performance professional, as well as those things that you can change. The following items must be learned during the assessment phase:

- System configuration
- Application design
- Performance expectations
- Known peak periods
- Changes in the system configuration or the application
- Duration of the problem, if applicable
- The options
- The politics

3.1.1 System Configuration

System configuration includes both hardware and software configuration. The number of disk drives, how data are distributed on the disks, whether the disks are used with mounted file systems or used in raw mode, the file system parameters, and how the application makes use of them—all of these factors are examined during a system configuration assessment. Memory size, the amount of lockable memory, and the amount of swap space are scrutinized in assessing the virtual memory system configuration. You will need to know the processor type and the number in the system. Finally, the kernel (operating system) configuration must be understood, and the values of tunable kernel parameters should be identified. Knowing all these items in advance will make carrying out the various performance management functions much easier.

3.1.2 Application Design

Understanding the design of the *application* is equally important. It may not be possible for you as a performance specialist to thoroughly understand the design and workings of the application. Optimally, there should be someone with whom the application design can be discussed.

Such things as inter-process communication (IPC) methods, basic algorithms, how the application accesses the various disks, and whether the application is compute- or I/O-intensive are examples of the knowledge you will need. For instance, with relational databases, it is important to understand whether or not the Relational Database Management System (RDBMS) software supports placing a table and the index that points to it on different disk drives. Some RDBMS products support this and some do not; when this capability is present, it is an important tuning technique for improving database performance. Modern design techniques such as creating multi-threaded applications and the deployment of applications on multi-processor systems require new skills to analyze how the application works and its use of system resources.

You should expect consumer complaints and comments to be in terms of the application. However, measurements will be based upon the available metrics, which are mostly kernel-oriented. If you can't translate system measurements into application-specific terms, it will be difficult to explain in a meaningful way what the measurements indicate in terms of necessary changes in the system.

3.1.3 Performance Expectations

You will need to learn the *performance expectations* of the system's users. It is very important to know, in advance, the measurable criteria for satisfactory performance. This is the only way to know when performance tuning is successful and when it is time to monitor performance rather than actively attempt to improve it. Objective rather than subjective measures must be elicited. Being told that response time must be less than two seconds and is currently five seconds or more is much more useful than being told that performance is "lousy."

Understanding performance expectations is a complicated task. The perception of actual system performance and the definition of satisfactory performance will change depending upon one's perspective and role. Understanding expectations includes the need for eliciting success criteria. In another words, you need to know whether the users or owners of the application and system will agree with you that you have finished tuning the system and application.

3.1.3.1 Response Time

Users of the system typically talk about poor response time. They are mostly concerned with how long it takes to get a response from the system or application once the enter key is pressed. For instance, a user who is entering invoices into an accounts payable system expects to have data verification and data insertion completed within several seconds, at most. Engineers who use a computer-aided design application expect to see the image of the part being analyzed rotated in real-time. Otherwise, they get frustrated with what they consider to be poor performance. Response time is the most commonly used measure of system performance and is typically quoted as one of the following:

- An average of a fractional second (sometimes called sub second) or seconds
- A certain confidence level
- A maximum response time

Typical confidence levels are 90%–98%. For example, one would say that the desirable average response time is two seconds, 95% of the time, with a maximum transaction response time of five seconds.

What is a good value for response time? While the automatic answer "It depends" is certainly true, it is useful to have a range of values for response time to use as a guideline. Users of a text editor or word processing package don't want to have to wait to see the characters echoed on the screen as they are typed. In this case, sub-second response time of approximately 250 milliseconds would be considered optimal. In a transaction processing environment one must understand the environment before developing good response time values. Some of the factors are as follows:

- Transaction complexity
- Number of users
- Think time between transactions
- Transaction types and ratios

Transaction Complexity This deals with the amount of work which the system must perform in order to complete the transaction. If the transaction is defined as requiring three record lookups followed by an update, that transaction is much more complex than one that is a simple read. The complexity associated with typical Computer-Aided Design (CAD) applications is very high. Many CPU cycles and perhaps disk I/Os are necessary for simple interactive tasks, such as rotating a model of an object on the display.

Number of Users The number of users influences the sizing of any system that is required to support a given workload and response time.

Think Time As the think time between transactions increases, more work can be supported by the system in the idle periods between transactions. *Heads down* environments provide almost no think time between transactions. This is a typical data input environment. Conversely, customer service environments often provide very long think times, since most of time the customer service representative is speaking with the caller by telephone and casually accessing various databases as the telephone conversation proceeds.

Transaction Types and Ratios It is necessary to look at the types of transactions and the ratios of the number of each transaction type to the total. Read-intensive applications can provide rapid response times to queries. Insert- and particularly update-intensive applications require more CPU cycles and often disk I/Os to complete the transaction. Table 3-1 gives guidelines for acceptable response times.

Table 3-1 Typical Application Response Times (based on the author's experience)

Transaction Type	Acceptable Response Time in Seconds
Interactive CAD applications	< 1
Text editing or word processing	1/4
Read-intensive, low complexity	< 1
Read-intensive, medium to high complexity	1–2
Update-intensive, low to medium complexity	5
Update-intensive, high complexity	5–15
Long think-time environments	2–3
Batch run	N/A

Users perceive performance as poor when update response time exceeds 5 seconds and when there is no preparation to be done by the user to get ready for the next transaction. Read performance must be no more than 1–2 seconds to keep users satisfied.

After installing the computer system and the application, some system administrators or performance managers have been known to create dummy workloads on the systems before letting any users access the applications or the system. The initial users perceive a certain response time following their inputs. As more users are added, the dummy workload is reduced, thus providing a constant response time to the users. This trick attempts to address another issue with the perception of actual performance. Users prefer consistent responsiveness rather than variable responsiveness. If someone decides to run some resource-intensive batch jobs while interactive users are on the system, interactive performance will typically degrade. End-of-month processing will usually consume a very large amount of system resources, making it necessary to either keep the interactive users off the system while it is being run, or to run it in off-hours.

Users can tolerate and accept consistently poor response time rather than response that is good one minute and poor the next. The acceptable variance in response time changes as the average response time itself changes. Users will tolerate an average response time of 1.5 seconds with a variance of ± 1 second much less than they will tolerate an average response time of 3 seconds with a variance of ± .25 second. One problem associated with attempting to prevent variability in performance is that when throughput is favored in tuning the system, the chance of experiencing variability in response time is greatly increased.

Predictability is another way of looking at consistency of performance. Predictability makes the job of the performance professional easier when forecasting future resource consumption. It also allows the appropriate setting of expectations for performance, as the following analogy shows.

The public transportation department announces that buses on a particular bus route are scheduled to arrive at each stop an average of every ten minutes. In a given thirty-minute period, three buses arrive all at once and the next one arrives forty minutes later. This schedule meets the stated criteria. However, it will make the people waiting for the fourth bus very unhappy. It would be much better for the waiting passengers if the buses were to arrive consistently ten minutes apart. It would also be perceived well if the buses were to arrive predictably at certain clock times.

3.1.3.2 Throughput

Information system *management personnel* are typically interested in throughput, rather than response time. If the demands of the organization require that a certain amount of work must be processed in an average day, then management is concerned whether the system can process that workload, rather than caring whether response time is one second or two seconds. Throughput is often quoted as work per unit time. Examples of throughput measures are:

- Three thousand invoices processed per hour
- Five CAD modeling runs per day
- All end-of-month processing must complete within three days
- Overnight processing must complete in the nine off-peak hours

It is not possible to develop guidelines for good throughput values. Throughput is driven by business needs, and the system must be sized to support those requirements. Capacity planning is done to ensure that as business requirements grow, the system will be able to handle the workload. It should be easy to predict in advance whether a system will be able to provide a specified throughput. The workload is considered a batch workload and the average time to complete a unit of work can be measured with a simple benchmark.

In reality, the situation is never that simple. Users are typically very vocal, and poor response time often reaches the ears of management. The point of this discussion is that the definition of performance may change from person to person; attitudes about response time and throughput must be examined to determine what users consider to be acceptable performance. Although the overall methodology is the same, tuning for response time and for throughput are different. Another way of putting this is that there is one strategy for approaching performance, but there are many different tactics.

3.1.4 Known Peak Periods

It is useful to identify *known peak periods* in advance, so that unusual spikes in the data can be readily explained. For instance, it is often mentioned that resource utilization in an office environment peaks at 11:00 a.m. and between 2:00–3:00 p.m. during the normal work day. Processing requirements typically grow at the end of the month or the end of a quarter. Expecting peaks at these times can save time when analyzing the data. Additionally, if the peaks are absent, it may be a clue that something unusual is preventing full system utilization.

3.1.5 Sudden Changes

Anyone who has worked in technical support has experienced callers complaining that the system "suddenly" is no longer working correctly or that performance has suddenly degraded. The following is a familiar dialogue:

From Bob's Consulting Log—

Consultant: Has anything changed in the application or the system?
Client: No, nothing has changed.
Consultant: Are you sure that nothing has changed?
Client: I'm quite sure nothing has changed.

(Three days later, after a lot of investigation ...)

Consultant: Did you notice that your database administrator (DBA) dropped some indexes?
Client: Oh! I didn't think **those** changes would make a difference.

The task of investigating changes that may have been made to the system is quite an art. However, the importance of this part of the assessment should not be minimized.

3.1.6 Duration of the Problem

When doing performance problem diagnosis, identifying the *duration of the problem* involves several issues:

- How long does the performance problem last?
- When does the performance problem occur?
- How long has the performance problem existed?
- When did it begin?

It is important to understand *the length of time that the problem lasts* to detect whether it occurs only sometimes, because of a spike in resource utilization, or constantly. In each of these situations, tuning requires a different approach.

Knowing *when the performance problem occurs* means that data collection can be planned and minimized, as an alternative to collecting performance data for days or weeks to capture data when the problem manifests itself.

Finally, *the length of time that the performance problem has existed* influences the probability of determining if anything in the system or application has been changed. If the problem has existed for a long time (weeks or months), it is very unlikely that any changes will be discovered. One can also question the seriousness of the situation if the users have been living with the problem for months.

3.1.7 Understanding the Options

Understanding the options lets you determine what recommendations should be offered. If there is no capital budget for purchasing computer hardware, you can look for other ways to resolve the performance problem. Perhaps tuning the operating system or application is a viable alternative to upgrading the CPU to a faster model. In contrast, if time constraints dictate that the problem must be resolved quickly and deterministically, then upgrading the CPU to a faster model would probably be more expeditious than spending an unpredictable amount of time attempting to improve or redesign the application.

3.1.8 Understanding the Politics

It may be necessary to *understand the politics of the organization* before revealing the cause of the problem or before recommending the changes to be implemented. Knowledge of the politics may help narrow the scope of the question you are trying to answer. It may be that the user organization wants to gain more control of the system, and it is trying to gather evidence to support this cause. The Information Technology (IT) department (also called Information Systems in some organizations) may be trying to allocate computing resources fairly, and it may not be possible to make changes that improve the situation for only one group. Finally, you may have been called in simply to provide objective data to justify the purchase of a larger, faster system.

3.2 Measurement

The next phase of the performance management methodology involves measuring actual system or application performance. For this phase, performance tools are used to perform the data collection and presentation.

Measurement is based upon the answers to several interrelated questions that must be answered before any data is collected.

- Which performance tools are available?
- What is the purpose of the measurement?
- Is the measurement baseline- or crisis-oriented?
- How long should the data be collected, and at what intervals?

- What data metrics are to be collected?
- How well are the metrics documented?
- How accurate are the data presented by the tool?
- Are certain system resources already saturated?
- How do these measurements relate to the organization's business needs?

3.2.1 Tool Availability

Some performance tools come standard with the HP-UX Operating System. Others are available as separately purchasable products. The availability of the various tools on the system being measured will constrain the answers to the other questions. Tool familiarity also affects the answer to this question. Comfort and experience with using a tool often limit the use of other tools even if other tools are more useful in a given situation. However, the best tool for a given purpose should be used to make the measurements, even if that tool must be purchased.

3.2.2 Purpose of the Measurement: Baseline versus Crisis

The purpose of the data collection must be determined in advance in order to select the correct set of tools for gathering the data. It is important to measure performance on the system when performance is acceptable. This type of measurement is called a *baseline measurement.* Think of a baseline as a signature or profile which can be used for purposes of comparison at those times when performance is not acceptable or is degrading. Baseline measurements require fewer metrics and a longer sampling interval, because a particular problem is not being investigated. Instead, the goals are to characterize system performance only, and to watch for trends.

An analogy to baseline measurements is a routine physical exam. The physician takes certain vital signs like blood pressure, pulse rate, temperature, and blood tests, including cholesterol counts. A visual inspection by the physician is correlated with the internal measurements and an historical record is kept to monitor trends over time. Any unusual symptoms are investigated immediately so that the physician can treat the problem before it becomes chronic.

Baseline measurements should be reviewed to develop conclusions about system performance without the immediate goal of altering performance by tuning the system or application. Baseline measurements should be archived for historical purposes. They can then be used to:

- Review performance trends over time
- Compare against current performance when investigating or diagnosing current performance problems
- Provide data for performance forecasting
- Develop and monitor service level agreements
- Provide data for capacity planning

Data collected and archived over time does not typically need to be as voluminous and detailed as for performance problem resolution.

Performance crises usually result from failing to manage performance. *Crisis measurements* (those typically made during performance problem diagnosis) require much more detail so that performance problems can be adequately investigated. The additional detail involves additional metrics as well as more frequent measurement, resulting in a much larger volume of data. The purpose of crisis measurements is to characterize *current* system performance in detail so that appropriate tuning can be done. Managing a performance crisis is much more difficult when there are no baselines against which a comparison can be made.

Baseline measurements should be made, archived, and reviewed periodically so that future performance crises can be prevented, and dangerous performance trends acted upon before they become serious. As the data ages, less and less detail is required. As changes in the system occur, for example, adding users or making changes in the application, new baseline measurements should be taken. Another tactic is to compare baselines from similarly configured systems, to help understand the variances before problems occur.

Baseline measurements can help provide the necessary translation between the language of performance tools and the needs of users. If the data can be presented and reviewed prior to a crisis, then communication during the crisis should be easier.

Baseline measurements should include information that is not related to existing sources. For instance, reviewing the "Other Application" category in MeasureWare can indicate if new work is being added, or whether the trend is to move work away from existing applications.

3.2.3 Duration and Interval of the Measurement

Some tools are good for displaying performance metrics in real-time. Other tools are better for collecting performance metrics in the background over a long period for casual analysis later. If performance problem diagnosis is the goal of the data collection and the problem manifests itself consistently or after a short time, then real-time performance tools would be the best choice. The amount of data produced by real-time performance tools is quite large. Therefore, this type of data should not be gathered for long periods of time since there is a large storage requirement, and it is difficult to review large volumes of data. Performance problems that cannot be readily reproduced, or those that occur unpredictably or only occasionally, will warrant longer-term data collection. Tools that provide summarization and detail capabilities are the best choice.

If performance characterization or forecasting is the goal, longer-term trending tools would be the tools of choice. Chapter 5, "Survey of Unix Performance Tools" on page 51 will discuss in detail the individual tools and when they are best used. The duration of the measurement should greatly influence selection of the tool or tools that will be used to collect the data.

The measurement interval must also be determined. Sampling intervals for baseline measurements should be measured in minutes or hours rather than seconds to reduce the volume of data that will be collected. Sampling intervals for crisis measurements or for performance problem resolution need to be shorter and are measured in seconds. If performance problems are of short duration, i.e., tend to spike, sampling intervals must also be short: typically one to five seconds. The shorter the sampling interval, the higher the overhead of making the measurement. This is of particular concern if one or more of the system resources are already saturated.

3.2.4 Particular Metric Needs

Performance tools come from various sources, and people get used to using certain ones, depending on their background. These then become their favorite tools, whether or not they are best for the job. Some tools were written for specific purposes; for example, *vmstat* was written specifically to display virtual memory and CPU metrics, but not disk statistics. The tool chosen should be useful in diagnosing the performance issue at hand.

3.2.5 Metric Documentation

There are several hundred performance metrics available from the kernel. These metrics were developed over time, and some are better documented than others. Some of the metrics may have a one-line description that is incomprehensible. Only by reviewing kernel source code can one hope to determine the meaning of some of the more esoteric metrics. Of course, the availability of kernel source code is limited, as is the desire to review it.

For example, the manual page for the tool *vmstat* defines the field *at* as the number of address translation faults. Those of us who are familiar with hardware components of modern computers might readily conclude that this field counts the number of Translation Lookaside Buffer (TLB) faults. This would be a very desirable metric, since it would give one indication of how well the CPU address translation cache is performing. Unfortunately, only by reviewing the kernel source code can one determine that the *at* field in the *vmstat* report is really referring to the number of page faults, a virtual memory system metric.

Good documentation is needed to determine what metrics are important to the purpose of the measurement and to learn how to interpret them.

3.2.6 Metric Accuracy

In order to understand completely why some performance metrics are inaccurate, one must understand how the kernel is designed. For instance, although most CPU hardware clocks measure time in microseconds, the granularity of the HP-UX system clock is 10 milliseconds. Most Unix-based operating systems record CPU consumption on a per-process basis by noting which process was running when the clock ticked. That process is charged for consuming CPU time during the *entire* clock tick, whether or not it used all of the tick. So, the saying “garbage in, garbage out” applies to performance tools in a loose sense. If the source of the data is inaccurate, the data will be inaccurate.

The standard Unix performance tools use the kernel's standard sources of data. This method of determining global and per-process CPU utilization was fine in the early days of Unix when systems were much slower. Newer methods have been developed by Hewlett-Packard to more accurately characterize system performance. Tools developed by HP get their data from the new sources in the kernel, which provides for greater metric accuracy. The IEEE POSIX

(P1004) committee, the Open Group, and the Performance Working Group (PWG) are all reviewing HP's implementation for adoption into a standard that could be implemented in other versions of Unix.

3.2.7 Saturation of Certain System Resources

When one or more system resources are saturated, it may be necessary to concentrate on particular metrics in order to determine the source of the saturation. However, merely invoking a performance tool causes additional overhead. The tool chosen should be one that does not exacerbate the saturation of the resource of concern. Tool overhead will be discussed more fully in Chapter 4, "Kernel Instrumentation and Performance Metrics" on page 39 and Chapter 5, "Survey of Unix Performance Tools" on page 51.

3.2.8 Relationship Between the Metric and the Application

Metrics by their nature usually count how many times some piece of code or hardware is used. The relationship of a given metric to a particular application and to the needs of its users must be established by analysis and interpretation of the data, plus a complete understanding of the application.

3.2.9 Qualitative versus Quantitative Measurements

One last point about measurements. The preceding discussion involved quantitative measures of how the system is performing. Other, equally important measures of system performance are qualitative in nature. There is a management style called "Management by Wandering Around," or MBWA, in which the manager visits a wide variety of managed personnel and asks questions that help to monitor the pulse of the organization. In the performance arena, MBWA becomes "Measuring by Wandering Around." Talk to the users. Ask them how they perceive system and application performance. Find out *why* they might feel that performance is bad. Watch how they interact with the system. It's possible that the users may be interacting with the system in a way that the application designers never imagined, and that is causing the application to behave poorly.

Another type of qualitative measurement can be made by interacting with the system directly and noting its responsiveness or the lack of it. Logging on the system, initiating a simple command like *ls(1)* for a directory listing, and noting the response might result in a qualitative measure of interactive performance.

3.2.10 Summary

In summary, the reasons for making measurements are:

- Measurements are key to understanding what is happening in the system.
- Measurements are the *only* way to know what to tune.

In addition, it is necessary to measure periodically in order to proactively monitor system and application performance so that problems can be prevented or resolved quickly.

3.3 Interpretation and Analysis

After making the measurements, it is necessary to review the voluminous amount of data that were collected. Some of the performance tools present the data as tables of numbers. Other tools offer summarization of the data in graphical form. Regardless of the presentation method, interpretation of the data is complex. One must first understand what the metrics mean. Then, it is necessary to know how to interpret the numbers; in other words, what number indicates a performance problem. This is no easy task! **Rule #1** comes into play here. A good value or a bad value for a given performance metric *depends* upon many factors.

From Bob's Consulting Log—I was involved in a sizing exercise where we had to determine what system was needed to support the application running 125 users with a response time of < 5 seconds. We started out with a basic system and looked at the load average metric, which was 250. We also saw that the response time was well above 5 seconds (some took 5 minutes!), and we were only simulating 75 users.

After upgrading the CPU to a faster one, we re-ran the test, and the load average dropped to 125. We were now able to simulate all 125 users, but response time was still unsatisfactory. Finally, we upgraded to the most powerful processor currently available. Now, the load average was 75. Most people would cringe at that number, saying response time should be terrible. However, all transactions completed in under 5 seconds for all 125 users. The moral is: Don't be scared by large numbers. What really matters is that you meet the performance requirements of the application.

The value for a given metric that can be considered good or bad will be discussed in-depth in the tuning section for each major system resource.

Some of the general factors that affect setting rules of thumb for performance metrics are:

- Type of system: multi-user or workstation
- Type of application: interactive or batch, compute- or I/O-intensive
- Application architecture: single system or client/server, multi-tiered, parallel system
- Speed of the CPU
- Type of disk drives
- Type of networking

3.3.1 Multi-User versus Workstation

A multi-user system experiences many more context switches than a workstation normally does. Context switches consume some of the CPU resource. Additionally, the high number of users typically causes a lot of I/O, which puts demands on the CPU and disk resources. Workstations that are used for graphics-intensive applications typically have high user CPU utilization numbers (which are normal) but lower context switch rates, since there is only one user. Applications on a multi-user system usually cause random patterns of disk I/O. Workstation applications often cause sequential patterns of disk I/O. So, these factors affect the optimal values for the CPU and disk metrics.

3.3.2 Interactive versus Batch, Compute-Intensive versus I/O-Intensive

Workstations can support highly interactive applications, for example, X/Windows or Motif applications that require a lot of interaction. These can be business applications, such as customer service applications that provide several windows into different databases. Alternatively, technical applications such as CAD programs support interactive input to draw the part on the screen. Compute-bound applications on a workstation sometimes act like batch applications on a multi-user systems. Batch applications consume a large amount of CPU, as do compute-intensive applications. Interactive applications cause many more context switches and more system CPU utilization than do batch or compute-intensive applications. Batch applications use less memory than highly interactive applications. Compute-intensive applications can touch more pages of memory than individual I/O-intensive applications. The optimal values for the CPU and memory metrics are affected by these factors.

3.3.3 Application Architecture

An application may be architected in several ways. It can be monolithic, that is, one large program that runs entirely on a single system. Parallel applications are designed to make better use of the components of the single computer system (especially a Symmetric Multi-Processing (SMP) system) to improve throughput or response. In contrast, an application can be distributed as in a multi-tiered client/server environment. In this case, parallel processing does not necessarily provide the same benefits. For these reasons, understanding the architecture is necessary before deciding what measurements to make and how to interpret the data.

3.3.4 Performance of the CPU

The higher performing the CPU, the greater the CPU resource that is available. Applications can get more work done during the time-slice they are allocated, and may be able to satisfy their current need for the CPU resource completely, rather than having to wait for another turn. This factor affects the *run queue* metric.

3.3.5 Type of Disks

Newer technology disk drives are faster, have higher capacity, and can support more I/Os per second than can older disks. The number of disk channels also affects the number of disk I/Os that are optimal. Caches on disks can dramatically improve the overall response times of disk I/Os.

3.3.6 Type of Networks

Networks continue to get faster and faster as well. High speed networks can move a lot of data between servers with high bandwidth and low latency. The network bandwidth has actually increased much more significantly over the years than the bandwidth of disks. Improvements in link throughput and connectivity technologies such as Dense Wave Division Multiplexing (DWDM) fiber links allow for high speed and low latency between systems separated by miles.

3.4 Identifying Bottlenecks

With all these factors in mind, the goal of data interpretation and analysis is to determine if there is a particular bottleneck. Once a particular bottleneck is identified, tuning to improve performance can be initiated. Characteristics of bottlenecks are:

- A particular resource is saturated.
- The queue for the resource grows over time.
- Other resources may be starved as a result.
- Response time is not satisfactory.

3.4.1 Resource Saturation

Resource saturation is often thought of as 100% utilization. The entire resource is consumed. Additional requests for the resource are required to wait. However, this is not sufficient as proof of a bottleneck. Two examples reinforce this point.

Disk utilization is determined by periodically monitoring whether there are any requests in the queue for each disk drive. The total number of requests in the queue is not factored into the disk utilization metric. Although 100% disk utilization is an indicator of a busy disk, it does not mean that the disk cannot support more I/Os.

The idle loop in the kernel is used to compute CPU utilization. On a workstation executing a compute-intensive application, CPU utilization is probably 100% for a long period of time. However, if no other processes are waiting for the CPU, and response time is satisfactory, there is no bottleneck.

There are utilization metrics for the CPU, memory, network, and disk resources.

3.4.2 Growing Resource Queue

Resource queue growth over time is a strong indicator of a bottleneck, in conjunction with the utilization metric. The queue for a resource tends to grow when demand increases and when there is not enough resource available to keep up with the requests. It is easier to develop rules of thumb for queue metrics than for utilization metrics.

3.4.3 Resource Starvation

Resource starvation can occur when one resource is saturated and another resource depends upon it. For instance, in a memory-bound environment, CPU cycles are needed to handle page faults. This leaves less of the CPU resource for application use.

3.4.4 Unsatisfactory Response Time

Unsatisfactory response time is sometimes the final arbiter of whether or not a bottleneck exists. The CPU example given above demonstrates this point. If no other processes are waiting for the CPU, and the application produces the results in a satisfactory time period, then there is no bottleneck. However, if the CPU is saturated and response or throughput expectations are not being met, then a CPU bottleneck exists.

3.4.5 Bottleneck Summary

Multiple metrics should always be reviewed to validate that a bottleneck exists. For example, the CPU utilization metric is a measure of saturation. The run queue metric is a measure of queue growth. Both metrics are needed to establish that a CPU bottleneck is present. Multiple tools should be used to validate that there is not a problem with a particular tool yielding misleading data. Consider the following analogy.

A three-lane highway is built to accommodate a certain maximum traffic flow, for example, twenty vehicles per minute distributed across the three lanes. This would be considered 100% utilization. Additional traffic entering the highway would be forced to wait at the entrance ramp, producing a queue. Suppose that a tractor/trailer overturns and blocks two lanes of the highway. Now, the same amount of traffic must funnel through the one remaining open lane. With the same number of cars on the road, the highway is now more than saturated. The queue builds at the entrance ramps. Resource starvation occurs, since two lanes are closed. The time it takes to travel a given distance on the highway now becomes unacceptably long.

Once a bottleneck is identified, it can possibly be alleviated. However, alleviating one bottleneck may result in the emergence of a new one to investigate.

3.5 Tuning or Upgrading

Once the bottleneck is identified, there are two choices. Either the amount of the resource can be increased, or the demand for the resource can be reduced. Tuning techniques will be discussed along with the particular bottlenecks. However, there are some general tips that can be applied to tuning bottlenecks:

- Determine whether the data indicate a particular bottleneck.
- Devise a simple test for measuring results.
- Do not tune randomly.

03PerfMeth.fm Page 35 Wednesday, February 18, 2004 1:38 PM

- Use heuristics and logic in choosing what to tune.
- Look for simple causes.
- Develop an action plan.
- Change only one thing at a time.
- Prioritize the goals.
- Know when to stop tuning.

3.5.1 Determine Whether There Is a Particular Bottleneck

This is the first tip to apply. Using the multiple characteristics of a bottleneck, determine if the data indicate a particular bottleneck. There is no use in tuning disk I/O on a system that is CPU-bound.

3.5.2 Devise a Simple Test for Measuring Results

When something in the operating system or application is tuned, it is advisable to quantitatively measure whether performance has improved or degraded. The test should be simple, short, and repeatable, so that it can be performed as each tuning step is taken.

3.5.3 Do Not Tune Randomly

There was once a system administrator who always tuned the size of the buffer cache, no matter what the bottleneck. Tuning the wrong thing can make the situation worse rather than better. Use knowledge of how the kernel works to help decide the cause of the bottleneck. Always try to visualize what the data *should* look like and compare it to the actual data before assuming that a cause has been found.

3.5.4 Use Heuristics and Logic

Use heuristics to select the most likely cause of the bottleneck. Experience can point to a particular bottleneck rather quickly. Watching the access lights on the disk drives can quickly indicate which disk drives may be saturated even before you look at the metrics. Use the qualitative measures as well as the quantitative metrics. Logically work through the possible causes and potential tuning solutions. Experience in tuning systems will tell which tuning alternatives offer the largest probability of success.

3.5.5 Look for Simple Causes

By looking for simple causes, easier and less expensive solutions can be tried. This is the K. I. S. S. principle: Keep it Simple, Stupid! Simple causes tend to occur more frequently, and simple solutions should be tried first. For example, tuning a system parameter is easier and less costly than modifying the design of an application. Either tactic might result in performance improvements, but one is clearly a better solution than the other.

3.5.6 Develop an Action Plan

Write up an ordered, prioritized list of measurements and tuning actions. The list can sometimes be turned over to other people for execution. Include contingencies in case some of the steps do not result in improvement. Analyze the results after performing each step of the plan.

3.5.7 Change Only One Thing at a Time!

Otherwise, the cause of the bottleneck cannot be found. If you attempt to tune multiple things at once, the benefits from one type of tuning can be counteracted by the degradation caused by a different type of tuning.

3.5.8 Prioritize Goals

Often, you may have multiple goals when developing a tuning solution. Prioritizing the goals can solve the problem caused by different goals requiring different solutions. It is usually possible to tune for response time or for throughput, but not both. In a transaction-processing environment, one should tune for the most frequent type of transaction, or the most important.

3.5.9 Understand the Limits of the Application's Architecture

Some applications cannot fully utilize a system. An example is a single-threaded process flow designed with no parallelism. Adding a CPU to such an application may not increase performance.

Instrumenting an application (adding code to measure specific events or code paths) to provide performance metrics can also create this problem if the instrumentation is not created with parallelism in mind. For example, incrementing a single global counter may be a choke point, as in the following code design:

```
while not done {
    do transaction work
    lock
    increment global counter
    unlock
   }
```

3.5.10 Know When to Stop

Finally, knowing when to stop tuning is very important. Setting the performance expectations in advance is the easiest way to establish completion. Getting the user to agree to success criteria in advance is important for deciding not only when to stop tuning, but when you can declare success. Baselines can be used to help establish that performance has returned to a normal level. The “eternal hope syndrome” will not occur if the performance expectations are met through tuning.

From Bob’s Consulting Log—I spent several days at a customer site improving performance by a factor of 50%. Once the customer saw the extent of the improvement, she insisted I do additional work to improve things even more. Soon it became clear that we had already maxed out the performance improvement and then wasted several more days trying to eke out just a little bit extra.

It is not useful to spend a lot of time on gaining 1% improvement in performance. Optimizing a part of the application that is executed only once and consumes less than 5% of the total execution time is just not worth it.