# 1 Introduction

'Performance can be bad, but can it ever be wrong?'

*Jim Kohn, SGI/Cray Research, Inc.*

## 1.1 Measuring performance

If the automobile industry had followed the same development cycles as the computer industry, it has been speculated that a Rolls Royce car would cost less than $100 with an efficiency of more than 200 miles per gallon of gasoline. While we certainly get more car for our money now than we did twenty years ago, no other industry has ever changed at the incredible rate of the computer and electronics industry.

Computer systems have gone from being the exclusive domain of a few scientists and engineers who used them to speed up some esoteric computations, such as calculating the trajectory of artillery shells, for instance, to being so common that they go unnoticed. They have replaced many of the mechanical control systems in our cars, thereby reducing cost while improving efficiency, reliability, and performance. They have made possible such previously science-fiction-like devices as cellular phones. They have provided countless hours of entertainment for children ranging in age from one to one hundred. They have even brought sound to the common greeting card. One constant throughout this proliferation and change, however, has been the need for system developers and users to understand the *performance* of these computer-based devices.

While measuring the cost of a system is usually relatively straightforward (except for the confounding effects of manufacturers' discounts to special customers), determining the performance of a computer system can oftentimes seem like an exercise in futility. Surprisingly, one of the main difficulties in measuring performance is that reasonable people often disagree strongly on how performance should be measured or interpreted, and even on what 'performance' actually means.

*Performance analysis* as applied to experimental computer science and engineering should be thought of as a combination of *measurement*, *interpretation*, and *communication* of a computer system's 'speed' or 'size' (sometimes referred to as its 'capacity'). The terms speed and size are quoted in this context to emphasize that their actual definitions often depend on the specifics of the situation. Also, it is important to recognize that we need not necessarily be dealing with complete systems. Often it is necessary to analyze only a small portion of the system independent of the other components. For instance, we may be interested in studying the performance of a certain computer system's network interface independent of the size of its memory or the type of processor. Unfortunately, the components of a computer system can interact in incredibly complex, and frequently unpredictable, ways. One of the signs of an expert computer performance analyst is that he or she can tease apart these interactions to determine the performance effect due only to a particular component.

One of the most interesting tasks of the performance analyst can be figuring out how to measure the necessary data. A large dose of creativity may be needed to develop good measurement techniques that perturb the system as little as possible while providing accurate, reproducible results. After the necessary data have been gathered, the analyst must interpret the results using appropriate statistical techniques. Finally, even excellent measurements interpreted in a statistically appropriate fashion are of no practical use to anyone unless they are communicated in a clear and consistent manner.

## 1.2 Common goals of performance analysis

The goals of any analysis of the performance of a computer system, or one of its components, will depend on the specific situation and the skills, interests, and abilities of the analyst. However, we can identify several different typical goals of performance analysis that are useful both to computer-system designers and to users.

- **Compare alternatives.** When purchasing a new computer system, you may be confronted with several different systems from which to choose. Furthermore, you may have several different options within each system that may impact both cost and performance, such as the size of the main memory, the number of processors, the type of network interface, the size and number of disk drives, the type of system software (i.e., the operating system and compilers), and on and on. The goal of the performance analysis task in this case is to provide quantitative information about which configurations are best under specific conditions.

- **Determine the impact of a feature.** In designing new systems, or in upgrading existing systems, you often need to determine the impact of adding or removing a specific feature of the system. For instance, the designer of a new processor may want to understand whether it makes sense to add an additional floating-point execution unit to the microarchitecture, or whether the size of the on-chip cache should be increased instead. This type of analysis is often referred to as a *before-and-after* comparison since only one well-defined component of the system is changed.
- **System tuning.** The goal of performance analysis in system tuning is to find the set of parameter values that produces the best overall performance. In time-shared operating systems, for instance, it is possible to control the number of processes that are allowed to actively share the processor. The overall performance perceived by the system users can be substantially impacted both by this number, and by the time quantum allocated to each process. Many other system parameters, such as disk and network buffer sizes, for example, can also significantly impact the system performance. Since the performance impacts of these various parameters can be closely interconnected, finding the best set of parameter values can be a very difficult task.
- **Identify relative performance.** The performance of a computer system typically has meaning only in the context of its performance relative to something else, such as another system or another configuration of the same system. The goal in this situation may be to quantify the change in performance relative to history – that is, relative to previous generations of the system. Another goal may be to quantify the performance relative to a customer's expectations, or to a competitor's systems, for instance.
- **Performance debugging.** Debugging a program for correct execution is a fundamental prerequisite for any application program. Once the program is functionally correct, however, the performance analysis task becomes one of finding performance problems. That is, the program now produces the correct results, but it may be much slower than desired. The goal of the performance analyst at this point is to apply the appropriate tools and analysis techniques to determine why the program is not meeting performance expectations. Once the performance problems are identified, they can, it is to be hoped, be corrected.
- **Set expectations.** Users of computer systems may have some idea of what the capabilities of the next generation of a line of computer systems should be. The task of the performance analyst in this case is to set the appropriate expectations for what a system is actually capable of doing.

In all of these situations, the effort involved in the performance-analysis task should be proportional to the cost of making the wrong decision. For example, if you are comparing different manufacturers' systems to determine which best satisfies the requirements for a large purchasing decision, the financial cost of making the wrong decision could be quite substantial, both in terms of the cost of the system itself, and in terms of the subsequent impacts on the various parts of a large project or organization. In this case, you will probably want to perform a very detailed, thorough analysis. If, however, you are simply trying to choose a system for your own personal use, the cost of choosing the wrong one is minimal. Your performance analysis in this case may be correctly limited to reading a few reviews from a trade magazine.

## 1.3 Solution techniques

When one is confronted with a performance-analysis problem, there are three fundamental techniques that can be used to find the desired solution. These are *measurements* of existing systems, *simulation*, and *analytical modeling*. Actual measurements generally provide the best results since, given the necessary measurement tools, no simplifying assumptions need to be made. This characteristic also makes results based on measurements of an actual system the most believable when they are presented to others. Measurements of real systems are not very flexible, however, in that they provide information about only the specific system being measured. A common goal of performance analysis is to characterize how the performance of a system changes as certain parameters are varied. In an actual system, though, it may be very difficult, if not impossible, to change some of these parameters. Evaluating the performance impact of varying the speed of the main memory system, for instance, is simply not possible in most real systems. Furthermore, measuring some aspects of performance on an actual system can be very time-consuming and difficult. Thus, while measurements of real systems may provide the most compelling results, their inherent difficulties and limitations produce a need for other solution techniques.

A simulation of a computer system is a program written to model important features of the system being analyzed. Since the simulator is nothing more than a program, it can be easily modified to study the impact of changes made to almost any of the simulated components. The cost of a simulation includes both the time and effort required to write and debug the simulation program, and the time required to execute the necessary simulations. Depending on the complexity of the system being simulated, and the level of detail at which it is modeled, these costs can be relatively low to moderate compared with the cost of purchasing a real machine on which to perform the corresponding experiments.

The primary limitation of a simulation-based performance analysis is that it is impossible to model every small detail of the system being studied. Consequently, simplifying assumptions are required in order to make it possible to write the simulation program itself, and to allow it to execute in a reasonable amount of time. These simplifying assumptions then limit the accuracy of the results by lowering the fidelity of the model compared with how an actual system would perform. Nevertheless, simulation enjoys tremendous popularity for computer-systems analysis due to its high degree of flexibility and its relative ease of implementation.

The third technique in the performance analyst's toolbox is analytical modeling. An analytical model is a mathematical description of the system. Compared with a simulation or a measurement of a real machine, the results of an analytical model tend to be much less believable and much less accurate. A simple analytical model, however, can provide some quick insights into the overall behavior of the system, or one of its components. This insight can then be used to help focus a more detailed measurement or simulation experiment. Analytical models are also useful in that they provide at least a coarse level of validation of a simulation or measurement. That is, an analytical model can help confirm whether the results produced by a simulator, or the values measured on a real system, appear to be reasonable.

**Example.** The delay observed by an application program when accessing memory can have a significant impact on its overall execution time. Direct measurements of this time on a real machine can be quite difficult, however, since the detailed steps involved in the operation of a complex memory hierarchy structure are typically not observable from a user's application program. A sophisticated user may be able to write simple application programs that exercise specific portions of the memory hierarchy to thereby *infer* important memory-system parameters. For instance, the execution time of a simple program that repeatedly references the same variable can be used to estimate the time required to access the first-level cache. Similarly, a program that always forces a cache miss can be used to indirectly measure the main memory access time. Unfortunately, the impact of these system parameters on the execution time of a complete application program is very dependent on the precise memory-referencing characteristics of the program, which can be difficult to determine.

Simulation, on the other hand, is a powerful technique for studying memory-system behavior due to its high degree of flexibility. Any parameter of the memory, including the cache associativity, the relative cache and memory delays, the sizes of the cache and memory, and so forth, can be easily changed to study its impact on performance. It can be challenging, however, to accurately model in a simulator the overlap of memory delays and the execution of other instructions in contemporary processors that incorporate such performance-enhancing features as out-of-order instruction issuing, branch prediction, and nonblocking caches. Even with the necessary simplifying assumptions, the results of a detailed simulation can still provide useful insights into the effect of the memory system on the performance of a specific application program.

Finally, a simple analytical model of the memory system can be developed as follows. Let $t_c$ be the time delay observed by a memory reference if the memory location being referenced is in the cache. Also, let $t_m$ be the corresponding delay if the referenced location is not in the cache. The cache *hit ratio*, denoted $h$, is the fraction of all memory references issued by the processor that are satisfied by the cache. The fraction of references that *miss* in the cache and so must also access the memory is $1 - h$. Thus, the average time required for all cache hits is $ht_c$ while the average time required for all cache misses is $(1 - h)t_m$. A simple model of the overall average memory-access time observed by an executing program then is

$$t_{avg} = ht_c + (1 - h)t_m. \tag{1.1}$$

To apply this simple model to a specific application program, we would need to know the hit ratio, $h$, for the program, and the values of $t_c$ and $t_m$ for the system. These memory-access-time parameters, $t_c$ and $t_m$, may often be found in the manufacturer's specifications of the system. Or, they may be inferred through a measurement, as described above and as explored further in the exercises in Chapter 6. The hit ratio, $h$, for an application program is typically more difficult to obtain. It is often found through a simulation of the application, though. Although this model will provide only a very coarse estimate of the average memory-access time observed by a program, it can provide us with some insights into the relative effects of increasing the hit ratio, or changing the memory-timing parameters, for instance. ◇

The key differences among these solution techniques are summarized in Table 1.1. The *flexibility* of a technique is an indication of how easy it is to change the system to study different configurations. The *cost* corresponds to the time, effort, and money necessary to perform the appropriate experiments using each technique. The *believability* of a technique is high if a knowledgeable individual has a high degree of confidence that the result produced using that technique is likely to be correct in practice. It is much easier for someone to believe that the execution time of a given application program will be within a certain range when you can demonstrate it on an actual machine, for instance, than when relying on a mere simulation. Similarly, most people are more likely to believe the results of a simulation study than one that relies entirely on an analytical model. Finally, the *accuracy* of a solution technique indicates how closely results obtained when using that technique correspond to the results that would have been obtained on a real system.

The choice of a specific solution technique depends on the problem being solved. One of the skills that must be developed by a computer-systems performance analyst is determining which technique is the most appropriate for the

**Table 1.1.** A comparison of the performance-analysis solution techniques

| | Solution technique | | |
|---|---|---|---|
| Characteristic | Analytical modeling | Simulation | Measurement |
| Flexibility | High | High | Low |
| Cost | Low | Medium | High |
| Believability | Low | Medium | High |
| Accuracy | Low | Medium | High |

given situation. The following chapters are designed to help you develop precisely this skill.

## 1.4 Summary

Computer-systems performance analysis often feels more like an art than a science. Indeed, different individuals can sometimes reach apparently contradictory conclusions when analyzing the same system or set of systems. While this type of ambiguity can be quite frustrating, it is often due to misunderstandings of what was actually being measured, or disagreements about how the data should be analyzed or interpreted. These differences further emphasize the need to clearly communicate all results and to completely specify the tools, techniques, and system parameters used to collect and understand the data. As you study the following chapters, my hope is that you will begin to develop an appreciation for this art of measurement, interpretation, and communication in addition to developing a deeper understanding of its mathematical and scientific underpinnings.

## 1.5 Exercises

1. Respond to the question quoted at the beginning of this chapter, 'Performance can be bad, but can it ever be wrong?'
2. Performance analysis should be thought of as a decision-making process. Section 1.2 lists several common goals of a performance-analysis experiment. List other possible goals of the performance-analysis decision-making process. Who are the beneficiaries of each of these possible goals?

3. Table 1.1 compares the three main performance-analysis solution techniques across several criteria. What additional criteria could be used to compare these techniques?
4. Identify the most appropriate solution technique for each of the following situations.
   (a) Estimating the performance benefit of a new feature that an engineer is considering adding to a computer system currently being designed.
   (b) Determining when it is time for a large insurance company to upgrade to a new system.
   (c) Deciding the best vendor from which to purchase new computers for an expansion to an academic computer lab.
   (d) Determining the minimum performance necessary for a computer system to be used on a deep-space probe with very limited available electrical power.

# 2 Metrics of performance

'Time is a great teacher, but unfortunately it kills all its pupils.'

*Hector Berlioz*

## 2.1 What is a performance metric?

Before we can begin to understand any aspect of a computer system's performance, we must determine what things are interesting and useful to measure. The basic characteristics of a computer system that we typically need to measure are:

- a *count* of how many times an event occurs,
- the *duration* of some time interval, and
- the *size* of some parameter.

For instance, we may need to count how many times a processor initiates an input/output request. We may also be interested in how long each of these requests takes. Finally, it is probably also useful to determine the number of bits transmitted and stored.

From these types of measured values, we can derive the actual value that we wish to use to describe the performance of the system. This value is called a *performance metric*.

If we are interested specifically in the time, count, or size value measured, we can use that value directly as our performance metric. Often, however, we are interested in normalizing event counts to a common time basis to provide a speed metric such as operations executed per second. This type of metric is called a *rate metric* or *throughput* and is calculated by dividing the count of the number of events that occur in a given interval by the time interval over which the events occur. Since a rate metric is normalized to a common time basis, such as seconds, it is useful for comparing different measurements made over different time intervals.

Choosing an appropriate performance metric depends on the goals for the specific situation and the cost of gathering the necessary information. For example, suppose that you need to choose between two different computer systems to use for a short period of time for one specific task, such as choosing between two systems to do some word processing for an afternoon. Since the penalty for being wrong in this case, that is, choosing the slower of the two machines, is very small, you may decide to use the processors' clock frequencies as the performance metric. Then you simply choose the system with the fastest clock. However, since the clock frequency is not a reliable performance metric (see Section 2.3.1), you would want to choose a better metric if you are trying to decide which system to buy when you expect to purchase hundreds of systems for your company. Since the consequences of being wrong are much larger in this case (you could lose your job, for instance!), you should take the time to perform a rigorous comparison using a better performance metric. This situation then begs the question of what constitutes a good performance metric.

## 2.2 Characteristics of a good performance metric

There are many different metrics that have been used to describe a computer system's performance. Some of these metrics are commonly used throughout the field, such as MIPS and MFLOPS (which are defined later in this chapter), whereas others are invented for new situations as they are needed. Experience has shown that not all of these metrics are 'good' in the sense that sometimes using a particular metric can lead to erroneous or misleading conclusions. Consequently, it is useful to understand the characteristics of a 'good' performance metric. This understanding will help when deciding which of the existing performance metrics to use for a particular situation, and when developing a new performance metric.

A performance metric that satisfies all of the following requirements is generally useful to a performance analyst in allowing accurate and detailed comparisons of different measurements. These criteria have been developed by observing the results of numerous performance analyses over many years. While they should not be considered absolute requirements of a performance metric, it has been observed that using a metric that does not satisfy these requirements can often lead the analyst to make erroneous conclusions.

1. **Linearity.** Since humans intuitively tend to think in linear terms, the value of the metric should be linearly proportional to the actual performance of the machine. That is, if the value of the metric changes by a certain ratio, the actual performance of the machine should change by the same ratio. This proportionality characteristic makes the metric intuitively appealing to most people. For example, suppose that you are upgrading your system to a system