COMPUTER SYSTEM PERFORMANCE

COMPUTER
SCIENCE
SERIES

Computer System Performance

Herbert Hellerman
State University of New York at Binghamton

Thomas F. Conroy
IBM Corporation
Education Center

McGraw-Hill Book Company

New York St. Louis San Francisco Auckland Düsseldorf
Johannesburg Kuala Lumpur London Mexico Montreal New Delhi
Panama Paris São Paulo Singapore Sydney Tokyo Toronto

Computer System Performance

1 2 3 4 5 6 7 8 9 0 KPKP 7 9 8 7 6 5

This book was set in Press Roman by Scripta Graphica.
The editors were Kenneth J. Bowman and Michael Gardner;
The production supervisor was Charles Hess.
The drawings were done by J & R Services, Inc.
Kingsport Press, Inc. was printer and binder.

Library of Congress Cataloging in Publication Data

Hellerman, Herbert, date
Computer system performance.

(McGraw-Hill computer science series)
Includes bibliographical references.
1. Electronic digital computers–Evaluation.
I. Conroy, Thomas F., joint author. II. Title.
QA76.5.H447 001.6'4'044 74-19029
ISBN 0-07-027953-5

COMPUTER SYSTEM PERFORMANCE

Preface

Assessing the performance of a complex and expensive computer system is both a practical necessity and a formidable technical challenge. Performance is characterized by a set of precisely defined descriptors of efficiency that help determine how closely a system comes to meeting stated objectives. Because these may differ from system to system, and even conflict within the same system, it is important to understand not only the descriptors, but also the relationships between them and how they are influenced by choices that must be made in system architecture, design, and operations. The study of performance involves not only mastery of certain special definitions and techniques, like measurement and simulation, but also a deep understanding of many themes that thread the entire fabric of computer science and technology.

This book is concerned primarily with performance as opposed to function. The reader is assumed already to be familiar with functional aspects of machine organization and programming. However, some complex topics like operating systems are discussed, beginning with basic concepts. A performance viewpoint offers a splendid opportunity to cap an academic program in computer science or engineering, since it brings together considerations of hardware, software, human, and management factors. This book is therefore intended for senior undergraduate or graduate courses. It should also be of interest to working computer professionals in research, development, and operations.

The first chapter is an overview of major performance issues in computer systems. It is designed to introduce some of the terminology and problems in characterizing performance and to motivate interest in the subject.

The next two chapters cover selected topics in statistics and discrete mathematics. They supply all tools needed later and other material useful in reading the literature. A major objective is to explain fundamental concepts and to develop important skills in statistics, set notation, and applied probability. For the most part, the formulas given in text are supplied with their assumptions and derivations. For readers having a weak background in these subjects, Chap. 2 and 3 are essential and help make the book self-contained. This material and the Appendixes are also valuable for review and reference.

Chapters 4 and 5, in the middle of the book, deal with job processing and queueing models from three viewpoints: intuitive, simulation, and mathematical. Some of this material may be familiar to industrial engineers and operations-research specialists, but it should also be at the fingertips of computer professionals.

The last six chapters are concerned with performance issues that are very specific to computer systems. Chapter 6 discusses the relatively simple class of cases involving a single job or a single component. The material includes mix and kernal techniques of CPU evaluation, compiler performance, software measurement methods, and a simple model of a single job where processing and input-output are overlapped in various ways.

Since computer performance is intimately related to the way its resources are managed, especially by its operating system, Chap. 7 describes major design options of operating systems including relocation, loading, scheduling, and deadlock-handling features. Chapter 8 discusses the various versions of the complex IBM OS/360 operating system and illustrates the principles discussed in Chap. 7.

Chapter 9 is devoted to timesharing systems, beginning with the user viewpoint, proceeding to general architecture considerations, and finally to specific cases including MIT-CTSS, IBM-TSO, and IBM-APL.

Chapters 10 and 11 examine the theory and practice of virtual storage, which includes some of the most sophisticated and dynamic resource-allocation strategies in computer systems. The two design contexts of a fast CPU with a cache store and the main-store/drum-disk memory hierarchy are both discussed with most examples from IBM System/370 systems.

A set of exercises for each chapter is supplied at the end of the book. For those chapters covering mathematical-analytic techniques, these exercises include numerical and algebraic problems. For the last few chapters, which involve primarily descriptive material, most of the exercises require word answers. These are intended not only to help focus attention on important points in the text but also to stimulate critical thinking and familiarity with terminology, and to give readers practice in developing their skills of expression and exposition. The final exercises in many of the chapters are major projects, some of which require considerable library work and nontrivial programs to be written.

We have used the manuscript as a text in a two-semester sequence in system performance. Since our particular curriculum required a separate course in probability and stastics, Chap. 2 and 3 were used only for a quick review, and thereafter for reference. Most of the first half of the first semester was spent on Chap. 4 and 5, where the simulation and mathematical job processing models were developed. Here, we found the use of the SIMJOB simulator program (outlined in Chap. 4) to be useful for student experimentation. Chapter 6 was then covered rather quickly and the last half of the term was devoted mainly to Chap. 7 and 8, which concern modern conventional operating systems, and a part of Chap. 9, which introduces timesharing systems. The second semester began with timesharing systems and continued with the final chapters on virtual storage. The last four weeks were reserved for student seminars on papers selected from the references or reports on student projects, most of which were of the simulation type. Depending on the interests and needs of the students and the preferences of the instructor, different orderings and selections from the text are, of course, possible.

We are pleased to acknowledge the diligent proofreading work of Mr. Steven Lake at State University of New York at Binghamton. We also wish to express our appreciation to Mrs. Shanna McGoff for her excellent efforts and patience in typing the manuscript.

Herbert Hellerman
Thomas F. Conroy

COMPUTER SYSTEM PERFORMANCE

Contents

CHAPTER 1

The Nature of Computer-Performance Evaluation

The evaluation of a computer system involves the following classes of considerations:

1. Performance (defined loosely as *measures of system speed* and resource use)
2. Cost
3. User convenience
4. Reliability

Perhaps the most fundamental property of modern data-processing systems is their generality. The theoretical basis of this generality is due to the work of A.M. Turing (1936). It can be shown that most commercially available computers are logically equivalent to each other, since they can compute the same wide class of *computable* functions, conditional only on the character-sets they can accept and print and on the size of storage available (but not fundamentally on the technologies or speed properties of the storage devices). This extreme generality of function is the source of many of the greatest difficulties in computer evaluation. This is because any system can be evaluated only with respect to its functions; the wider the class of functions, the more difficult the task of characterizing its "goodness."

The fact that performance varies in the same direction as cost is understandable, although the quantitative relationship is often not at all clear. The relation of user convenience to performance is also very difficult to quantify. Like most desirable properties, user convenience usually has its price in both system cost and performance, but this can be more than offset by increased human productivity. One reason for a speed penalty is that convenience requires the physical system to be disguised to user-oriented objectives through the use of resources in an indirect manner that is not the best match to their physical structure. Also, user convenience is achieved at least in part by system programs that automatically do tasks otherwise left to the individual programmer. It is difficult to design such facilities to perform optimally for the particular needs of all possible user programs. However, with care, system programs can often perform at least as well as corresponding programs written by most programmers.

It would be most desirable to have *quantitative* measures of the value of system features to user convenience. Unfortunately, these are rarely obtainable. For example, even today we have no measure of the worth of say the FORTRAN language over coding in Assembly language, although there is little doubt that the benefits in human productivity are very great. This kind of gap is a major obstacle in developing a science of system evaluation.

Since about 1969, the literature on computer performance has burgeoned. This reflects in part the maturing of the data-processing industry. Increasing attention is now being given to making wise selections from among several available devices and software components with comparable functions. The strong incentive of system economies coupled with considerable technical effort has not yet produced a science of system performance. Some of the obstacles still to be overcome include considerable differences among workers in performance in the definitions of various performance measures, the scarcity of analytic methods for relating various performance measures to each other, and the lack of general methods of scaling performance measures from one system to another. In other words, we do not have a *theory* of performance evaluation. Progress toward such a theory has been slow, not only because of technical problems in definitions and analysis, but also because performance, user convenience, reliability, and cost have important business implications. Many tools and results are consequently being held proprietary to the business firm that developed them.

In the remainder of this chapter, we shall briefly touch on several computer performance issues, many of which will be discussed in greater detail later in the book.

1-1 MOTIVATIONS FOR EVALUATION

Computer system evaluation is of interest to several classes of people, each with its own perspective. A rough classification is:

1. Hardware-software architects and designers
2. Managers of system installations and their system programmers
3. Users (application programmers and analysts)

The scope narrows in the order of the above listing. The designer must deal with factors influenced by the entire spectrum of possible system use; the manager at a particular installation is concerned with many users, but only in a restricted environment; and the individual user is interested in his single program or a small collection of programs. As the width of interest becomes more specific, performance objectives and measures can be stated more precisely. However, less freedom is also available to manipulate resources and meet particular objectives. Thus the designers have a relatively large number of choices, while the installation manager working with a delivered system is confined to available resources. However, many modern computers are delivered with "open" parameters in their operating systems so that each installation can make important efficiency decisions, such as the devices to be used for program residence and *scheduling* options, in accordance with the properties of its workload. Judicious setting of these can be a major factor in good performance. Although at present most of this "tuning" is done manually, future operating systems will be "adaptive," i.e., they will sense the resources available, monitor activity, and assign resources automatically. However, even here it will still be necessary for users (e.g., installation managers) to convey their performance objectives to the system.

The individual user is most constrained in resource allocation, yet he often has a vital influence on performance. A recent specific case in our personal experience showed an improvement (by a *factor* of 50) in run time for a certain program by changing the iteration scheme. This factor is comparable to the average internal speed ratio between many large and small machines! Almost any experienced observer can give similar examples. The fact is that the range of program quality is in practice exceedingly large; this reflects the great differences in human skill in problem solving, which is really what programming is all about.

1-2 THE SYSTEM AS A COLLECTION OF RESOURCES

A modern computer system is best considered as a collection of *resources*. These and their typical present implementations include:

1. *Storages:*
   - (*a*) Registers and buffers; logic circuitry, logic arrays, and small fast core, film arrays.
   - (*b*) Main storage; film core, integrated transistor circuits.
   - (*c*) Auxiliary storage; core, drum, disk, delay line, tape.
2. *Processing Units:*
   - (*a*) Arithmetic-logical and program control operations.
   - (*b*) Transmission controllers.
3. *Transducers* (converters of one physical form of information to another):
   - (*a*) Card-tape readers, punches.
   - (*b*) Typewriters, printers, displays.
4. *System Programs*:
   - (*a*) Subroutine libraries.
   - (*b*) Language processors.
   - (*c*) Resource-management utilities.

The economics of technology has thus far resulted in system structures that are heterogeneous in speed and cost of substructures, especially storages. Although technology is changing, the fact that different technologies yield different cost-speed characteristics seems to be a safe extrapolation from all past experience. It follows that whatever the particular values of speed-cost improvement, future (like past) systems will consist of mixed technologies. Management of this hierarchy will remain a central issue of system performance.

Since resource allocation is programmable, it has traditionally been left to each individual user in each of his programs. This is a "fair" although burdensome task in a nonshared system where each user has all resources available to him during his run. In a shared (multiprogrammed) system, multiple-user demands converge at unpredictable times on common resources. Careful space allocation is now essential, since too large a share given to one user deprives others of this vital resource. These conflicts must be resolved by the *system*, since the individual user lacks the knowledge of resource states and the incentive and responsibility to manage resources for other users. Major resources are thus allocated and scheduled by a system program which we shall call the *supervisor*. To the user this program might as well be part of the hardware, and although many options are available to him and his installation, important performance factors are determined by the organization of the supervisor. The system-supplied programs mentioned above are used extensively by most programs, often in ways not obvious to the programmer. A realistic evaluation must take into account the performance of these facilities.

Finally, it is well to recognize the important role of the human operator as a "component" of system performance. Manual operations in the machine room (tape-disk mounting, printer-paper bursting, etc.) as well as console interaction can and do have significant effects on overall system performance.

1-3 PERFORMANCE MEASURES [Refs. BA,GO,HE2,LU]

Quantitative measures of performance are meaningful only when the following are clearly specified:

1. The system configuration (all device and software parameters)
2. The workload or job stream
3. The definition of the measures

The first of these is obvious, and little can be said in general. Configuration information can be very voluminous, although most of the performance effects are usually due to a small fraction of the total number of parameters. The difficulties in specifying and extrapolating workloads was mentioned earlier and will be a recurring theme of this book (see Sec. 1-4). For now, job i in a stream will be specified by a number-pair (A_i,X_i) giving its arrival and execute or service time. If all A_i are omitted, all jobs arrive at time = 0.

The definition of performance measures involves careful consideration of the objectives of the system, especially with regard to human expectations. Two measures in very common use are *thruput* and *response time*.

Thruput is a commonly used measure for "batch" systems and is expressed in units of jobs per minute, that is, the number of jobs in a stated job stream divided by the total time to process the stream:

$$H = \frac{n}{T}$$