Performance Evaluation: Experimental Computer Science at its Best

Peter J. Denning

Report Number:
80-368

http://docs.lib.purdue.edu/cstech/296

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries. Please contact epubs@purdue.edu for additional information.
PERFORMANCE EVALUATION: EXPERIMENTAL COMPUTER SCIENCE
AT ITS BEST*

Peter J. Denning

Computer Sciences Department
Purdue University
W. Lafayette, IN 47907

CSD-TR 368

June 1, 1981

*This work supported in part by NSF Grant MCS-78-01729. Part of this paper is based on the editorial, "What is Experimental Computer Science?, by P.J. Denning in Communications of ACM, October 1980, pp. 543-544.
Performance Evaluation: Experimental Computer Science at its Best*

Peter J. Denning

Computer Sciences Department
Purdue University
W. Lafayette, IN 47907

Abstract. Experimental science classifies knowledge derived from observations. The experimenter sets up an apparatus, uses it to collect data, and analyzes the data to sustain or refute hypotheses. The result of one line of investigation can be a model that becomes the apparatus for another line of investigation. Two examples from the area of performance evaluation are used to illustrate. The M44/44K project at IBM Watson Research Lab in the mid 1960s evaluated concepts of time sharing, especially about memory policies and program behavior, by implementing and measuring them on an IBM 7044. The study of queueing network models since 1971 illustrates how strong interaction between theory and experiment can produce a model (the Bard-Schweitzer mean value equations) that is sufficiently simple to serve as the starting point for new lines of investigation of system models.

*This work supported in part by NSF Grant MCS78-01729. Part of this paper is based on the editorial, "What is Experimental Computer Science?" by P. J. Denning in Communications of ACM, October 1980, pp. 543-544.
What is experimental computer science? This question has been widely discussed ever since the Feldman Report was published (1979 [16]). Many computer scientists believe that survival of their discipline is intimately linked to their ability to rejuvenate experimentation. The National Science Foundation instituted the Coordinated Experimental Research Program (CERP) in 1979 to help universities set up facilities capable of supporting experimental research. Other agencies of government are considering similar programs. Some industrial firms are offering similar help through modest cash grants and equipment discounts.

What is experimental computer science? Surprisingly, computer scientists disagree on the answer. A few believe that computer science is in flux — making a transition from theoretical to experimental science — and, hence, no operational definition is yet available. Some believe that it is all the non-theoretical activities of computer science, especially those conferring “hands-on” experience. Quite a few believe that it is large system development projects — i.e., computer and software engineering — and they cite MIT’s Multics, Berkeley’s version of Bell Labs’ UNIX, the ARPAnet, IBM’s database System R, and Xerox’s Ethernet-based personal computer network as examples.

These beliefs are wrong. There are well-established standards for experimental science. The field of performance evaluation meets these standards and provides examples of experimental science for the rest of the computing field.

Hypotheses, Apparatus, and Tests

Science classifies knowledge. Experimental science classifies knowledge derived from observations. The experimenters set up an apparatus, use it to collect data about a phenomenon, and then analyze the data to sustain or refute hypotheses about that phenomenon. The result of one line of investigation may be a model that becomes the apparatus for a later line of investigation.

The experimental apparatus may be a real system or subsystem — for example, the program implementing a hashing algorithm, an interactive computer system, or a paging algorithm. But the apparatus can also be a model — for example, a queueing network or a simulator of VM/370.

The hypothesis may concern a law of nature — for example, one can test whether a hashing algorithm’s average search time is a small constant independent of the table size by measuring a large number of retrievals. The hypothesis may concern characteristics of people — for example, one can test whether interactive computing improves programmer productivity by comparing the ability of control groups to solve problems with and without interactive terminals. The hypothesis may deal with design principles of computers — for example, one can determine which paging algorithm is best by controlled experiments with different algorithms managing the same workload. The hypothesis may concern the quality of models — for example, one can systematically measure the errors between response-time estimates calculated by a queueing network model and the real response times measured in a computer system.

The key concepts here are an apparatus for collecting data, a hypothesis, and systematic analysis to see whether the data supports the hypothesis. There is considerable flexibility in the types of hypotheses and apparatuses that may
Ideas versus Products

The whole point of science is to discover which ideas are important. Experiments are essential: to understand ideas and convince others of their value. Once an idea is assimilated by the community, the experiments behind it may be forgotten. This is true even of mathematics: results are reproved to improve understanding of the underlying principles, good results have many proofs, social processes with empirical overtones help identify and simplify the best concepts, and the best theorems eventually become definitions. [14]

No scientific discipline can be productive in the long term if its experimenters merely build components. Building a complex apparatus in the lab is a technological effort that may require great skill. But unless the apparatus is used to obtain significant new knowledge, the research is judged not to be substantive and is soon forgotten. Development projects primarily enhance the personal skills of the developers and make products available for consumption, but they are not primarily aimed at increasing our sum of knowledge. This is why engineering development projects such as Multics or the ARPAnet are not inherently experimental science. They produce apparatuses.

Two Examples

It is no accident that the best examples of experimental computer science can be found in our field, which we call performance evaluation and sometimes systems modeling. The primary aim of our work is the construction, validation, and evaluation of computer-system models, which are robust enough to be used for prediction.

I will cite two examples of experimental science in our field. The first is the M44/44X project at IBM Watson Research Lab in the middle 1960s; this project evaluated concepts of time sharing, especially memory policies and program behavior, by implementing and measuring them. This project is a paradigm of experimental work in computer systems architecture. The second example is the study of queueing network models since 1971; this line of investigation illustrates how strong interaction between theory and experiment can lead to a conceptually simple model that may serve as the starting point for future lines of investigation. This process is a paradigm of how yesterday's theorems can become tomorrow's definitions. between theory and experiment.
(Space limits me to two examples. I mean no offense to my colleagues who have worked on other projects in the true spirit of experimental science.)

The M44/44X Project

The M44/44X project was conducted at the IBM Research Center in Yorktown Heights, NY, in the middle 1960s. Its purpose was to evaluate the emerging concepts of time sharing systems by reducing them to practice and measuring them. The central principle of its architecture was a set of virtual machines, one for each user. The main machine was an IBM 7044 (M44 for short) and each virtual machine was an experimental image of the 7044 (44X for short). Virtual memory and multiprogramming were used to implement the address spaces of the 44X's in the memory hierarchy of the M44. This machine served as the apparatus for numerous experiments about memory policy and program behavior.

O'Neill (1967 [20]) described the system architecture and early experience with it. It is interesting that they recognized the problem of thrashing and solved it with a load controller.

Les Belady conducted a series of projects to understand the behavior of paging algorithms, the effects of page size, and the costs of storage fragmentation. His comprehensive paper (1966 [2]) significantly increased our knowledge of paging algorithms applied to individual programs in fixed allocations of main memory. Belady studied half a dozen policies and their variants. He concluded that LRU is better than FIFO, but that a simple variant of FIFO (the forerunner of today's CLOCK) gave a good approximation to LRU. He invented the optimal algorithm (MIN) and compared it with the realizable ones. He measured the effects of page size on performance, including the amount of superfluous information on pages (words not referenced after the page is loaded in main memory). His study was a model for similar experiments in other systems that independently corroborated his basic findings. The paper is still read today.

In studies the extension of his results to multiprogramming, Belady discovered that system performance is improved by varying the space allocated to individual programs: variable partitioning is more efficient than fixed. He and Carl Kuehner (1967 [3], 1969 [4]) proposed a model for this that exploited that concave-up shape of the lifetime curve of a program. (The lifetime curve gives the mean virtual time between page faults when a given amount of space is allocated to the program.)

In joint work with Robert Nelson and Jerry Shedler (1969 [5]), Belady observed that the FIFO policy has anomalous behavior — that is, it may increase paging in response to increased space allocation. (They demonstrated that adding one page to memory may double the paging rate.) This work influenced the later work of Mattson et al. (1970 [19]), whose "stack algorithms" are well behaved.
Brian Randell continued the work on storage fragmentation (1969 [21]). He concluded that internal fragmentation is more serious than external fragmentation. He proposed a new addressing mechanism (partitioned segmentation) that allocated a large segment as a "head" consisting of pages and a "tail" consisting of a segment shorter than the page size.

Even as Belady and his colleagues experimented with the architecture of memory management policies, David Sayre, Frances Gustavson, Barbara Brawn, and E. Mankin were studying the virtual memory hypothesis: the performance of properly tuned automatic memory management is better than the performance of the best handcrafted manual overlay strategy (1968 [7], 1969 [25], 1970 [8]). They compared programs run on the M44 with the automatic memory manager on and off to conclude that the hypothesis is correct for programs exhibiting locality of reference. This set of experiments laid to rest the remaining doubts about the efficacy of virtual memory.

Since the time of the M44 experiments, approximately 200 researchers around the world have contributed to the experimental effort to understand and optimize virtual memory operating systems (Denning 1970 [15], 1980 [16]). Aside possibly from the experimental work by Yon Bard and his colleagues on the CP-67 and VM/370 operating systems at IBM Scientific Center in Cambridge, MA, I am not aware of any similar project. The M44/44X project is a unique milestone along the highway of experimental computer science.

Queueing Network Models

The theory of stochastic queueing networks was developed in the 1950s and 1960s by Jackson, Gordon, and Newell [17]. This theory captured the interest of the computing community in 1971, when Jeff Buzen pointed out their application to the central server system and showed how to efficiently compute standard performance metrics in this model [9, 10]. Since that time, a significant portion of the systems modeling community has been studying the queueing network hypothesis: the queueing network model is an accurate, robust description of standard performance metrics for a wide class of common computer systems.

The principal result of the Jackson-Gordon-Newell theory is that the steady-state probability of seeing state \( \mathbf{n} = (n_1, \ldots, n_K) \) in a closed network containing \( N \) customers and \( K \) stations is of the product form:

\[
p(n) = \frac{1}{G(N)} \prod_{k=1}^{K} F_k(n_k),
\]

where

\*This collective wisdom has unfortunately not much influenced the design of large mainframe machines - e.g., the VAX-11 does not have usage bits. I regard this as evidence of the power of the belief that engineering, not science, is the driving force behind progress in computing.
\[ F_k(n) = V_k S_k(n) S_k(n-1) \cdots S_k(1) \]

\( V_k \) is the mean number of visits by a customer to station \( k \), \( S_k(n) \) is the mean time between service completions when \( n \) customers are present at station \( k \), and

\[
G(N) = \sum_{\alpha \mu \kappa} \prod_{k=1}^{K} F_k(n_k)
\]

is a normalizing constant. This result is formidable not only visually -- it is computationally infeasible because of the enormous size of the state space.

Buzen's discovery (1971 [10], 1973 [11]) was a simple recursion formula for calculating \( G(N) \) and then, from it, the other performance metrics. In the case of load-independent stations, Buzen's formula is

\[
g(n, k) = g(n, k-1) + D_k g(n-1, k)
\]

where \( D_k = V_k S_k \) is the average total processing demand per customer at station \( k \). The normalizing constant is \( G(N) = g(N, K) \), the utilization of station \( k \) is

\[
U_k(N) = D_k \frac{G(N-1)}{G(N)}
\]

and the mean queue length at station \( k \) is attainable from the recursion

\[
Q_k(N) = U_k(N)(1 + Q_k(N-1)). \quad N = 1, 2, \ldots
\]

where \( Q_k(0) = 0 \). The mean response time per visit to station \( k \) can be obtained from

\[
R_k(N) = S_k(1 + Q_k(N-1))
\]

Because the algorithms for these formulae are compact and efficient, they are easy to program, even on hand calculators. Many validation studies were undertaken to compare the accuracy of these easily used models with data from real systems. It was soon found that the formula for \( U_k \) will typically estimate the actual utilization to within 5% and the formula for \( Q_k \) will typically estimate the actual mean queue length to within 25%. [17] In fact, the accuracy of these models is now so well trusted, that most analysts suspect an error in the model or its parameters if the formulae do not produce answers within these tolerances.
In 1975 Baskett, Chandy, Muntz, and Palacios extended the product form solution to include multi-class networks [6]. Each class has its own parameters \( \{V_k\} \) and \( \{S_k(n)\} \). The Buzen algorithms were extended for this case by Reiser and Kobayashi (1975 [23]). Jeff Buzen and I proposed operational analysis to help explain why all these models work well even when the stochastic assumptions do not seem to apply (Denning & Buzen 1978 [12, 17]).

By this time the models were being applied for large systems -- many customers, stations, and job classes. Numerical instabilities were encountered in the algorithms under high loads \( N \) (due to overflow or underflow in the calculations of the normalizing constant). This inspired Reiser and Lavenberg (1978; sec [24]) to propose new recursions, called mean value analysis, that avoided these problems:

\[
R_k(N) = S_k(1 + Q_k(N-1)), \quad k = 1, \ldots, K
\]

\[
X_0(N) = N / \sum_{k=1}^{K} V_k R_k(N)
\]

\[
Q_k(N) = X_0(N) V_k R_k(N), \quad k = 1, \ldots, K
\]

where \( X_0(N) \) is the overall system throughput. Starting from \( Q_k(0) = 0 \), these equations are iterated for \( N = 1, 2, \ldots \) until the desired load is reached. Because these equations do not calculate normalizing constants, they are not susceptible to overflow or underflow. (However, if these equations are extended to calculate the queue-length distributions, numerical problems may reappear (Chandy and Sauer 1980 [13]), which has inspired further work by Reiser (1980 [22]) on hybrid approaches.)

In using these equations, many intermediate values will be calculated and discarded en route to the mean values of response time, throughput, and queue length for a large load \( N \). Yon Bard, in consultation with Paul Schweitzer, proposed to approximate the mean queue length \( Q_k(N-1) \) by a linear proration down from \( Q_k(N) \): [1, 12]

\[
Q_k(N-1) = \frac{N-1}{N} Q_k(N)
\]

With this substitution, the parameter \( N \) becomes superfluous and the mean value equations become:
\[ R_k = S_k \left( 1 + \frac{N-1}{N} Q_k \right), \quad k = 1, \ldots, K \]

\[ X_0 = \frac{N}{\sum V_k R_k} \]

\[ Q_k = X_0 V_k R_k, \quad k = 1, \ldots, K \]

Starting from an initial guess of \( Q_k = \frac{N}{K} \), these equations are iterated until the value of \( Q_k \) converges. They are surprisingly accurate (Bard 1979 [1], Buzen & Denning 1980 [12]).

I have now arrived at my main point: The Bard-Schweitzer equations can be used as the starting point for a theory of queueing networks. The first equation has a simple intuitive explanation (the response time is the mean time per customer multiplied by the mean number of customers just after an arrival); the second and third equations are operational laws. A growing body of experimental evidence shows that these equations are sufficiently accurate and robust for many common systems. More experimental work remains, to precisely identify the class of systems for which these equations are good approximations.

Much of our understanding, experimenting, and experience of fifteen years with queueing networks has been distilled into a fine essence, captured by these equations. We can now present them to practitioners and students and know that they will be used well. This could not have happened without constant, complementary interaction between theoreticians and experimenters. We are witnessing a great scientific achievement.

Conclusion

The established standards of science can be used to distinguish true experimental research from engineering development projects in our field. The specialty of systems modeling contains the best examples of true experimental science. I have dwelt here on two. The M44/44X project is a paradigm of experimental study to evaluate the architecture of computer systems. The evolution of queueing network models is a paradigm of the interaction between theory and experiment, demonstrating that yesterday's theorems - the mean value equations - are tomorrow's definitions.
Although I have said that the systems modeling specialty contains the best examples of experimental computer science, I do not mean to imply that all work in this specialty is exemplary or that no work in other specialties is in the true spirit of experimental science.

In emphasizing that development projects are not necessarily experimental science, I do not mean to downgrade development projects. These projects can make important contributions to computer and software engineering, which are as important to the computing field as theory and experiment.

Acknowledgements

I am grateful to Michael Harrison, Domenico Ferrari, and Jeff Brumfield for inspirations for some of what is written here.

References


18. J. Feldman, editor, "Rejuvenating experimental computer science," *Communications of ACM* 22, 3 (September 1979), 497-502. See also the ACM Executive Committee position, same issue, pages 503-504.


25. D. Sayre, "Is automatic folding of programs efficient enough to displace manual?" *Communications of ACM* 12, 12 (December 1969), 658-660.