By Alan Jay Smith

WORKLOADS (CREATION AND USE)

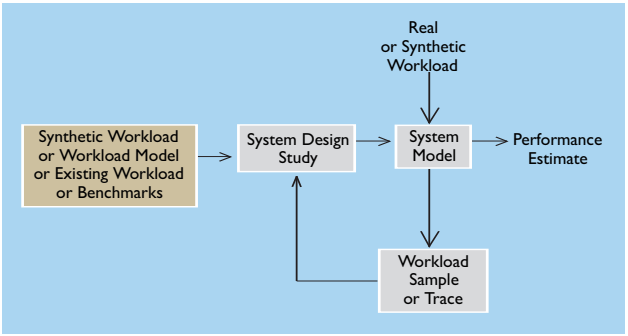
Workloads are used to estimate and measure system performance, but the results are only as good as the workloads and models being used.

Evaluating the performance of a (computer) system is meaningful only in the context of a workload, that is, what the system is being asked to do. Appropriate workloads vary with the type of system being considered (such as PC, workstation, mainframe, and supercomputer) and with the type of user and application. A discussion of workloads is also inseparable from consideration of benchmarks, or standardized workloads used for comparing the performance of different systems.

Workloads are valid to the extent that they allow one to CORRECTLY EVALUATE SYSTEM BEHAVIOR with respect to the feature(s) of interest.

Workloads are typically used in two main ways: to evaluate existing systems and to evaluate proposed designs and design alternatives for new systems. Workloads are used in many areas of science and engineering; for example, bridges are designed to cope with the traffic workload and with seismic and aerodynamic workloads. The Tacoma Narrows Bridge collapsed in 1940 from a failure to properly deal with the aerodynamic workload. The Brooklyn Bridge (completed in 1883) is still standing due to a massive overdesign by John Roebling, who knew that his workload knowledge was imprecise and limited.

Workloads and workload studies have always been recognized as important, and a series of annual conferences was recently instituted to consider the subject [5]; the figure outlines how workloads are used. Here, I provide an overview and discussion of the issues involved in collecting, selecting, analyzing, and using workloads for the purposes of computer system analysis, design, and performance evaluation.



Roles of workloads and benchmarks in designing, measuring, and evaluating systems and system designs.

Workload Abstraction

Workloads can be classified along a number of dimensions [1]. One is the degree of abstraction. The most abstract is a mathematical model (such as Poisson process traffic along a communications line, probabilistic routing in a network, and the LRU stack model for program behavior). Equivalent would be a stochastically generated workload (such as through a complex simulation), based perhaps on observed parameters but not consisting of real events. The usefulness of such workload models depends on two factors: how well a model matches the observed effect (not behavior) of real workloads; and whether the model has predictive power.

The former requires that the model affect the relevant system features in the same way the real or expected workload does or will, not that the model actually reproduces all aspects of the real workload. (It is, however, often difficult to estimate how an unknown future workload will affect a new, not yet constructed system.) The latter is the test typically used for physical theories. After a theory is found to account for all known observations, the next step is to determine if it predicts things that have not yet been observed. The LRU stack model for program behavior will not, for example, show a bursty fault process but will account pretty well for the behavior of set associative cache memories [3]. Such abstract models typically have the advantage of being compact and parameterizable.

Other types of workloads include collections of synthetic jobs and samples of real jobs (both of which are executable) and live workloads, that is, unreproducible samples of job streams as they arrive. In place of samples of complete jobs, an evaluator can use samples of the job characteristics of interest. For example, to study memory system behavior (such as caches and paging), we can use program address traces. To study I/O, we can use traces of I/O requests. To study branch target buffers, we can use program branch traces.

Different types of workloads are generally used for different purposes. Today, almost all workloads used for evaluation and research are real, either sample jobs or traces. To the extent that models are used, they are almost always derived from observations of real systems.

Workload Validity

Workloads are valid to the extent that they allow one to correctly evaluate system behavior with respect to the feature(s) of interest. The standard for validity can thus be as strong as requiring that overall system behavior be comparable to what is expected in practice. It can also be as weak as simply requiring that if under the workload W algorithm A is better (or

specifically Z% better) than algorithm B for aspect X of the system, the same result would be observed in practice. For example, if for the sample workload W, branch predictor A is better than predictor B, then validity would require the same result in a real system with the actual future workload.

A number of factors can cause workloads to be invalid for their desired purpose:

System design. The workload itself can be affected by the system design so it becomes difficult or impossible to evaluate system changes. For example, adding a cache to a disk system affects the I/O response time and thereby the timing of subsequent I/O events [4]. Reportedly, the capacity of the initial Multics system was overestimated by a factor of 10 because the workload was assumed to be similar to that observed for the existing Compatible Time Sharing System; the actual workload was very different due to the vastly increased functionality of Multics relative to CTSS;

Changing technology. Technology changes (such as in memory size, cache size, disk size, and CPU speed) make the workload inappropriate for studies of future systems. For example, a trace of a 1995 file system can't possibly yield reliable absolute miss ratios for a 2007 file system, since the disks and amount of stored information have become orders of magnitude larger over that period. Conversely, prefetching studies based on old traces may still be quite valid;

Representation. Workloads may simply be unrepresentative, as when a scientific workload is used to evaluate the design of a system that spends most of its time running the operating system or servicing database queries;

Adaptation. Systems may adapt to workloads over time. For example, database systems are continuously optimized to perform well with the latest version of the Transaction Processing Performance Council (TPC) benchmarks [2], and compilers have been tuned to recognize and optimize the SPEC benchmarks. Thus once systems have been tuned for these benchmarks, performance estimates based on them are not accurate for the actual workloads likely to occur, since compilers are likely to be far less effective on code for which they are not specifically tuned; and

Lack of adaptation. Conversely, workloads may not be adapted to their systems, as when "dusty decks" are used to benchmark supercomputers. In actual use, programs run on supercomputers are usually carefully tuned for the system on which they run, and a benchmark comparison of two different supercomputers usually requires that each benchmark program be carefully optimized (or rewritten) for each system.

Workload characteristics determined by human behavior generally remain valid over time and across systems; people evolve slowly. Conversely, workload characteristics determined by technology (such as CPU speed, memory, and disk size) may quickly become out of date. But it may be possible to compensate for these factors (such as scaling) or by collecting relative rather than absolute measurements. To the extent that the validity of the estimates is suspect, it is valuable to confirm assumptions of validity when possible. For example, if we are estimating the performance of Java programs on the newest Pentium design by studying traces of Fortran programs for a VAX, it would be wise to consider why the results might or might not be valid and confirm the accuracy of the estimates at the end of the design process. But there is no reason to a priori reject such a workload use; programs are written by people, and there is no reason to think that the object code will look significantly different. It is just as fallacious to reject benchmarks that may well be valid as to accept benchmarks that are obviously inappropriate.

Sensitivity, or how much a system's performance changes with changes in the workload, is an important issue. To the extent that estimated performance is sensitive to changes in the workload, either performance estimates will be unreliable or the model or estimating methodology will be suspect.

System Models

The primary, if not only, use for workloads is the estimation of performance, which in turn can be used for design, tuning, evaluation, and purchase, as outlined in the figure. For example, the throughput of a computer system, the miss ratio of a cache, the delay on a bridge, and the barrier size required for a flood break are all performance measures as a function of a workload. The system model and the workload should generally be matched to the question at hand. If the cache design or the branch predictor is being studied, it is usually unnecessary to model the full system (such as components like the floating point unit and the I/O system).

An important aspect of experimental design is to design system models that allow the experimental

It is often very difficult to determine whether a specific type of workload will BEHAVE DIFFERENTLY from known and previously studied workloads.

results to be understood. Whenever possible, the number of parameters that are simultaneously varied in a system model should be minimized. A one-dimensional plot is a lot easier to understand than an N-dimensional plot, and such simple designs make it much easier to find and recognize anomalies and identify the effects of validity issues (workload or model) on the results.

Workload Characterization

Workloads are often used to directly evaluate systems, but it is frequently helpful or satisfactory to characterize a workload (in appropriate ways) and use the characterization itself for evaluation. Such characterization efforts may in and of themselves lead to insights into system design that may be difficult to obtain through brute-force experimentation. Determining that an arrival process is (or is not) Poisson, bursty, or self-similar or contains a time-varying trend may provide as much or more insight into a system design than simply playing a trace of arrivals against various alternative designs.

Relative Performance	
Benchmark	Runtime Ratio
Los Alamos	1.616
Baskett	1.139
Erasthostenes	0.818
Linpack	1.611
Livermore	1.526
Mandelbrot	1.555
NAS Kernels	1.765
Shell	0.671
Smith	1.678
Whetstone	2.841

Ratio of runtimes for the benchmarks in the figure when each benchmark is run on two low-end workstations.

Workloads can be characterized either generally or with a specific goal in mind. For example, in [11], file-reference patterns were studied to see if they could be Poisson, which would have specific implications for file-migration algorithms. However, workload characterization studies are generally useful and interesting only to the extent that there is a goal in mind; a "general characterization" without a goal usually provides a lot of data but little information. (That is not to say that sometimes a "general characterization" might serendipitously discover something useful.) However, workload collection itself is almost always worthwhile, although undirected workload collection may fail to collect the data needed for a later study.

Techniques for workload characterization vary with the type of workload and the issue being studied. Thus while general techniques (such as "cluster analysis" and "selecting representative samples") are generally applicable, their use becomes specific only with an understanding of the substantive issues under consideration. A mastery of operations research and statistical techniques is no substitute for genuine understanding of the actual systems issues.

Problems and Their Workloads

Workload collection. Data collection is often the most difficult problem in any scientific study, whether in computer system performance evaluation, physics, or psychology. (I cover how each type of workload could be collected later in my discussion of workload types and performance studies.)

CPU benchmarking. CPU performance estimates were originally generated by mainframe computer manufacturers using internally developed workloads and published only after reasonable internal validation, often after approval by the legal department. With the introduction of high-performance microprocessors in the 1980s, performance estimates were more often generated by marketing departments, with little if any validation, reportedly sometimes in direct contradiction to internal performance estimates. This led to the establishment of SPEC (originally the System Performance Evaluation Cooperative, later Corporation), which collected and published a set of CPU computational benchmarks in 1989 and subsequently revised the set several times, most recently in 2006 (www.spec.org).

Computational benchmarks should be selected to either be individually representative of specific workloads or representative in the aggregate. They also should be composed of public-domain code, in a standard programming language compatible with most or all existing commercial compilers, have no machine dependencies, preferably do little if any I/O, and preferably not be optimizable to an unrepresentative extent by optimizing compilers or preprocessors. Companies were known to have tuned their

compilers to produce highly optimized code for these standard benchmarks, which is why SPEC dropped the Matrix 300 benchmark. Other standard benchmarks include Dhrystone, Whetstone, Livermore Loops, and Linpack [7]. To evaluate PCs, *PC Magazine* (www.pcmag.com, April 10, 2007) used 3D Mark 06, Company of Heroes (game), Windows Media Encoder 9, Cinebench, and Photoshop CS2 Action Set.

It is important to appreciate that observed performance can be very sensitive to the benchmarks or workloads used. The table here lists data collected for (but not included in) [7], showing that the performance of two low-end workstations varied by a factor of more than 4.2 depending on which benchmark was used.

The issues in selecting workloads for CPU benchmarking have to do with finding programs that meet these criteria, whose characteristics are believed to be understood (suggesting that proprietary benchmarks not be used), and the behaviors of which span the anticipated workload.

Memory workloads. Memory system analysis was originally directed to main memory paging and, once paging became less important, shifted to cache memory design. Memory system studies are almost always conducted using program address traces, or the sequence of memory addresses referenced by a program, usually tagged as instruction fetch, data read, and data write. Such traces can be generated by a number of techniques, including: a CPU simulator; a hardware monitor; instrumented microcode; instrumented object code; and a trace trap facility.

The issues in using such traces include:

Memory space. Making sure that the memory space addressed is large enough to stress the memory system; this may require tracing very large and long program executions, then sampling or compressing them into a form that allows for many simulations to be run in a reasonable amount of time. Techniques have also been developed to stitch together sampled segments from long traces; and

Representative workloads. Selecting programs believed to be representative of the actual workload. Note that different workloads may yield very different results; for example, for one workload considered in [11], the supervisor state miss ratio was four times higher than the user state miss ratio.

An important and distinct class of memory workload is related to parallel programs run on multiprocessors. The important aspect of parallel workloads relates to access patterns to shared data that can't be derived or estimated from studies of sequential workloads. It is increasingly important as multicore processor chips become common.

Another special type of workload relates to multimedia. Multimedia workloads involve audio and video, often running on dedicated and/or embedded hardware. I distinguish these workloads from other CPU and memory workloads due to their importance and because it has been speculated that they are significantly different from other workloads. Existing multimedia workloads are described in [8], which also presented the Berkeley Multimedia Workload assembled from public-domain real applications. At least with regard to memory reference behavior, it has been shown [9] that multimedia workloads do not in fact differ significantly from other workloads.

Note that it is often, a priori, very difficult to determine whether a specific type of workload will behave differently from known and previously studied workloads. For example, there is some evidence (see, for example, [10]) that programs written in different languages (such as Fortran and Lisp) for different architectures (such as DEC VAX and IBM 370) and different applications do not differ significantly in many of their characteristics, so having the latest workloads for the target system may not be an important issue. (This relates to the comment I made earlier that code is written by people and that people evolve slowly.) It has been repeatedly suggested to me that object-oriented code behaves differently from code written in earlier programming languages, but I am aware of no evidence for this. Likewise, many people still claim that certain real workloads are so poorly behaved that CPU caches are ineffective, but I find no evidence for this as well.

CPU benchmarks/workloads. Benchmarks in this class are used to address questions in pipeline design, branch prediction, and other aspects of CPU architecture. The same techniques used for memory benchmarks can generate CPU architecture benchmarks (CPUABs), but the constraints are less severe. Cache benchmarks need to be large enough to fill the cache, but CPUABs need to be large enough only to fill a branch target buffer or all of the stages of a pipeline. Thus, useful workloads can be assembled from many relatively short segments.

Database workloads. Database systems are typically evaluated with the TPC series of benchmarks [2, 4], which consist of standard database configurations and a standard series of artificial queries. Over time, real systems have become larger and more sophisticated, so it has been necessary to develop larger and

more complex benchmarks. Furthermore, vendors have sought to optimize their processing of the TPC benchmarks, and the benchmarks have had to be revised to stay ahead of this tuning effort.

File and I/O workloads. File and I/O systems can be studied using traces of I/O activities (such as reads, writes, opens, closes, and renames). However, they are often very difficult to collect, since they are often obtained only by modifying the operating system or by relying on debug and accounting packages (such as GTF and SMF on the IBM System 370 and its successor systems). (Physical I/O traces can be obtained through hardware monitors.) Debug and accounting package records can be unsatisfactory or very difficult to use because they seldom contain the information needed for the studies envisioned, and thus massive amounts of post-processing and/or modifications to the packages may be needed to make the trace data usable for the desired purpose.

The effect of looking at real I/O workloads, as opposed to imagined workloads, is illustrated in [6], which showed that in contrast to the many published disk-arm scheduling studies that assumed uniformly distributed cylinder accesses and long queue lengths, the disk arm in real systems seldom moves. Other studies have also showed that queue lengths are seldom longer than one or two requests.

System benchmarks. Somewhat different from the workloads discussed here are workloads (or benchmarks) designed to evaluate the overall system. They need to exercise the CPU, I/O system, network connections, and any other component expected to have a significant effect on performance. Evaluations of the overall system are typically part of the system-acquisition process. The danger of this type of benchmarking is that even though the entire system is purportedly being measured, one or a small set of bottlenecks may actually be limiting system throughput; thus an analysis of performance as a function of system parameters can be very important. For example, adding memory may increase the system cost by 2% and throughput by 20%, something it would be very important to know. And as in every other case discussed here, it is important that the benchmarks chosen reliably represent the workload most likely to actually occur.

Among the other workloads that have been used, proposed, and/or discussed in the literature are mail server workloads, Java program workloads, Web server workloads, e-commerce workloads, network workloads, power-consumption workloads, and commercial workloads. In general, the need for performance estimation requires a suitable workload, and new workloads continue to be generated as needed.

A very important issue is that with the exception of overwhelmingly common specialized applications (such as multimedia), special hardware is seldom constructed. So what is important in system design is not designing to an ever-increasing variety of specialized workloads but to the common aspects among the various workloads to be supported by the underlying system; the aim is to maximize performance across the inputs to be expected.

Conclusion

Workloads are an essential component of any system evaluation effort, whether for design, research, or purchase. Finding or developing valid, appropriate, manageable workloads is often difficult and time consuming. That's why I've discussed some of the issues relating to workloads and provided some pointers to the literature. C

References

1. Ferrari, D. *Computer Systems Performance Evaluation.* Prentice Hall, Englewood Cliffs, NJ, 1978.
2. Gray, J., Ed. *The Benchmark Handbook, Second Edition.* Morgan Kaufman, San Mateo, CA, 1993.
3. Hill, M. and Smith, A.J. Evaluating associativity in CPU caches. *IEEE Transactions on Computers 38,* 12 (Dec. 1989), 1612–1630.
4. Hsu, W. and Smith, A.J. The performance effect of I/O optimizations and disk improvements. *IBM Journal of Research and Development 48,* 2 (Mar. 2004), 255–289.
5. IEEE International Symposium on Workload Characterization (San Jose, CA, Oct.). IEEE Press, 2006.
6. Lynch, W. Do disk arms move? *Performance Evaluation Review 1,* 4 (Dec. 1972), 3–16.
7. Saavedra-Barrera, R., Smith, A.J., and Miya, E. Machine characterization based on an abstract high-level language machine. *IEEE Transactions on Computers 38,* 12 (Dec. 1989), 1659–1679.
8. Slingerland, N. and Smith, A.J. Design and characterization of the Berkeley multimedia workload. *ACM Multimedia Systems Journal 8,* 4 (2002), 315–327.
9. Slingerland, N. and Smith, A.J. Cache performance for multimedia applications. In *Proceedings of the International Conference on Supercomputing* (Sorrento, Italy, June 17–21, 2001), 204–217.
10. Smith, A.J. Cache evaluation and the impact of workload choice. In *Proceedings of the 12th International Symposium on Computer Architecture* (Boston, June 17–19, 1985), 64–75.
11. Smith, A.J. Cache memories. *Computing Surveys 14,* 3 (Sept. 1982), 473–530.
12. Smith, A.J. Analysis of long-term file reference patterns for application to file migration algorithms. *IEEE Transactions on Software Engineering SE-7,* 4 (July 1981), 403–417.

Alan Jay Smith (smith at eecs.berkeley.edu) is a professor in the computer science division of the Department of Electrical Engineering and Computer Sciences at the University of California, Berkeley.

The research discussed here is partially supported by NXP and Philips Semiconductors, Toshiba Corporation, and the State of California MICROProgram.