CAMBRIDGE

Cambridge University Press
978-1-107-07823-9 - Workload Modeling for Computer Systems Performance Evaluation
Dror G. Feitelson
Excerpt
More information

# 1

# Introduction

Performance evaluation is a basic element of experimental computer science. It is used to compare design alternatives when building new systems, to tune parameter values of existing systems, and to assess capacity requirements when setting up systems for production use. Lack of adequate performance evaluation can lead to bad decisions, which result either in an inability to accomplish mission objectives or an inefficient use of resources. A good evaluation study, in contrast, can be instrumental in the design and realization of an efficient and useful system.

There are three main factors that affect the performance of a computer system:

1. The system's design.
2. The system's implementation.
3. The workload to which the system is subjected.

The first two factors are typically covered with some depth in vocational training and academic computer science curricula. Courses on data structures and algorithms provide the theoretical background for a solid design, and courses on computer architecture and operating systems provide case studies and examples of successful designs. Courses on performance-oriented programming and on object-oriented design, as well as programming labs, provide the working knowledge required to create and evaluate implementations. But there is typically little or no coverage of performance evaluation methodology in general and of workload modeling in particular.

Regrettably, performance evaluation is similar to many other endeavors in that it follows the GIGO principle: garbage-in-garbage-out. Evaluating a system with the wrong workloads will most probably lead to irrelevant results, which cannot be relied on. This motivates the quest for the "correct" workload model [716, 256, 653, 19, 731, 103, 235, 635]. It is the goal of this book to help propagate the knowledge and experience that have accumulated in the research community regarding workload modeling and to make it accessible to practitioners of performance evaluation.

To read more: Although performance evaluation in general and workload modeling in particular are typically not given much consideration in vocational and academic



© in this web service Cambridge University Press www.cambridge.org

CAMBRIDGE

Cambridge University Press
978-1-107-07823-9 - Workload Modeling for Computer Systems Performance Evaluation
Dror G. Feitelson
Excerpt
More information



curricula, there has nevertheless been much research activity in this area. Good places to read about this are textbooks on performance evaluation, including the following:

- Jain [366] cites the arguments about what workload is most appropriate as the deepest rat hole that an evaluation project may fall into (page 161). Nevertheless, he does provide several chapters that deal with the characterization and selection of a workload.
- Law and Kelton [426] provide a very detailed presentation of distribution fitting, which is arguably at the core of workload modeling.
- Le Boudec [428] has perhaps the most practical and down-to-earth exposition of performance evaluation, including a discussion of model fitting and heavy-tailed distributions. And it has the advantage of being available for free from http://perfeval.epfl.ch/.
- The book on self-similar network traffic edited by Park and Willinger [536] provides good coverage of heavy tails and self-similarity.

In addition there are numerous research papers, many of which are cited in this book and appear in the Bibliography. For an overview, see the survey papers by Calzarossa and co-authors [103, 101]. Another good read is the classic paper by Ferrari [258]. This book has its roots in a tutorial presented at Performance 2002 [234].

## 1.1 THE IMPORTANCE OF WORKLOADS

The study of algorithms involves an analysis of their performance. When we say that one sorting algorithm is $O(n \log n)$, whereas another is $O(n^2)$, we mean that the first is faster and therefore better. But this is typically a worst-case analysis, which may occur, for example, only for a specific ordering of the input array. In fact, different inputs may lead to very different performance results. The same algorithm may terminate in linear time if the input is already sorted to begin with, but may require quadratic time if the input is sorted in the opposite order.

The same phenomena may happen when evaluating complete systems: they may perform well for one workload, but not for another.[1] To demonstrate the importance of workloads we next describe three examples in which the workload makes a large difference to the evaluation results.

### Example 1: Scheduling Parallel Jobs by Size

A simple model of parallel jobs considers them as rectangles in processors $\times$ time space: each job needs a certain number of processors for a certain interval of time. Scheduling is then the packing of these job-rectangles into a larger rectangle that represents the available resources.

[1] Incidentally, this is true for all types of systems, not only for computer systems. A female computer scientist once told me that an important side benefit of her chosen career is that she typically does not have to wait in line for the ladies room during breaks in male-dominated computer science conferences. But this benefit was lost when she attended a conference dedicated to encouraging female students to pursue a career in computer science.

© in this web service Cambridge University Press www.cambridge.org

CAMBRIDGE

Cambridge University Press
978-1-107-07823-9 - Workload Modeling for Computer Systems Performance Evaluation
Dror G. Feitelson
Excerpt
More information



It is well known that average response time is reduced by scheduling short jobs first (the SJF algorithm). The problem is that the runtime is typically not known in advance. But in parallel systems, scheduling according to job *size* may unintentionally also lead to scheduling by *duration*, if there is some statistical correlation between these two job attributes.

The question of whether such a correlation exists is not easy to settle. Three application scaling models have been proposed in the literature [740, 629]:

- *Fixed work*. This assumes that the work done by a job is fixed, and parallelism is used to solve the same problems faster. Therefore the runtime is assumed to be inversely proportional to the degree of parallelism (negative correlation). This model is the basis for Amdahl's law.
- *Fixed time*. Here it is assumed that parallelism is used to solve increasingly larger problems, under the constraint that the total runtime stays fixed. In this case, the runtime distribution is independent of the degree of parallelism (no correlation).
- *Memory bound*. If the problem size is increased to fill the available memory associated with a larger number of processors, the amount of productive work typically grows at least linearly with the parallelism. The overheads associated with parallelism always grow superlinearly. Thus the total execution time actually increases with added parallelism (a positive correlation).

Evaluating job scheduling schemes with workloads that conform to the different models leads to drastically different results. Consider a workload that is composed of jobs that use power-of-two processors. In this case a reasonable scheduling algorithm is to cycle through the different sizes, because the jobs of each size pack well together [418]. This works well for negatively correlated and even uncorrelated workloads, but is bad for positively correlated workloads [418, 449]. The reason is that under a positive correlation the largest jobs dominate the machine for a long time, blocking out all others. As a result, the average response time of all other jobs grows considerably.

But which model actually reflects reality? Evaluation results depend on the selected model of scaling; without knowing which model is more realistic, we cannot use the performance evaluation results. As it turns out, the constant time or memory-bound models are more realistic than the constant work model. Therefore scheduling parallel jobs by size with a preference for large jobs is at odds with the desire to schedule short jobs first and can be expected to lead to high average response times.

#### Example 2: Processor Allocation Using a Buddy System

Gang scheduling is a method for scheduling parallel jobs using time slicing, with coordinated context switching on all the processors. In other words, first the processes of one job are scheduled on all the processors, and then they are all switched simultaneously with the processes of another job. The data structure used to describe this is an Ousterhout matrix [528], in which columns represent processors and rows represent time slots.

An important question is how to pack jobs into rows of the matrix. One example is provided by the DHC scheme [245], in which a buddy system is used for processor allocation: each request is extended to the next power of two, and allocations are

© in this web service Cambridge University Press www.cambridge.org

CAMBRIDGE

Cambridge University Press
978-1-107-07823-9 - Workload Modeling for Computer Systems Performance Evaluation
Dror G. Feitelson
Excerpt
More information



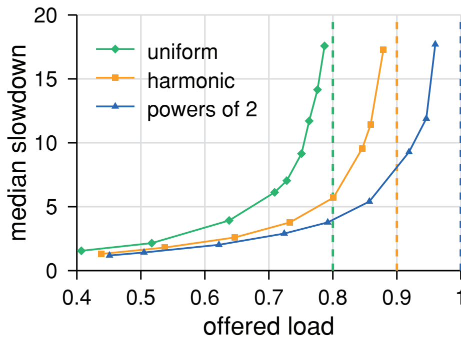


**Figure 1.1.** Simulation results showing normalized response time (slowdown) as a function of load for processor allocation to parallel jobs using DHC, from [245]. The three curves are for exactly the same system – the only difference is in the distribution of job sizes. The dashed lines are proven bounds on the achievable utilization for these three workloads.

always done in power-of-two blocks of processors. This scheme leads to using the same blocks of processors in different slots, which is desirable because it enables a job to run in more than one slot if its processors happen to be free in another slot.

The quality of the produced packing obviously depends on the distribution of job sizes. The DHC scheme has been evaluated with three different distributions: a uniform distribution in which all sizes are equally likely, a harmonic distribution in which the probability of size $s$ is proportional to $1/s$, and a uniform distribution on powers of two. Both analysis and simulations showed significant differences between the utilizations that could be achieved for the three distributions (Figure 1.1) [245]. These differences correspond to different degrees of fragmentation that are inherent to packing jobs that come from these distributions. For example, with a uniform distribution, rounding each request size up to the next power of two leads to a 25% loss to fragmentation – the average between no loss (if the request is an exact power of two) and a nearly 50% loss (if the request is just above a power of two, and we round up to the next one). The DHC scheme recovers part of this lost space, so there is actually only 20% loss, as shown in Figure 1.1.

Note that this analysis tells us what to expect in terms of performance, provided we know the distribution of job sizes. But what is a typical distribution encountered in real systems in production use? Without such knowledge, the evaluation cannot provide a definitive answer. Empirical distributions have many small jobs (similar to the harmonic distribution) and many jobs that are powers of two. Thus using a buddy system is indeed effective for real workloads, but it would not be if workloads were more uniform with respect to job size.

### Example 3: Load Balancing on a Unix Cluster

A process running on an overloaded machine will receive worse service than a process running on an unloaded machine. Load balancing is the activity of migrating a running process from an overloaded machine to an underloaded one. When loads are balanced, processes receive equitable levels of service.

One problem with load balancing is choosing which process to migrate. Migration involves considerable overhead. If the process terminates soon after being migrated, that overhead has been wasted. In addition, the process cannot run during the time it is being migrated. Again, if it terminates soon after the migration, it would have been better off staying in its place.

© in this web service Cambridge University Press www.cambridge.org

CAMBRIDGE

Cambridge University Press
978-1-107-07823-9 - Workload Modeling for Computer Systems Performance Evaluation
Dror G. Feitelson
Excerpt
More information



Thus it would be most beneficial if we could identify processes that may be expected to continue to run for a long time and then select them for migration. But how can we know in advance whether a process is going to terminate soon or not? The answer is that it depends on the statistics of process runtimes.

It is well known that the exponential distribution is memoryless. Therefore if we assume that process runtimes are exponentially distributed, we cannot use the time that a process has run so far to learn how much longer it is expected to run: this expectation is always equal to the mean of the distribution. In mathematical terms the probability that the runtime $T$ of a process will grow by an additional $\tau$, given that it has already run for time $t$, is equal to the probability that it will run for more than $\tau$ in the first place:

$$\Pr(T > t + \tau \mid T > t) = \Pr(T > \tau)$$

But the runtimes of real processes on Unix systems, at least long-lived processes, are not exponentially distributed. In fact, they are heavy-tailed [434, 319]. Specifically, the probability that a process run for more than $\tau$ time has been found to decay polynomially rather than exponentially:

$$\Pr(T > \tau) \propto \tau^{-\alpha} \qquad \alpha \approx 1$$

This means that most processes are short, but a small number are very long. If we condition the probability that a process will run for additional time on how much it has already run, we find that a process that has already run for $t$ time may be expected to run for an additional $t$ time: the expectation actually grows with how long the process has already run! (The derivation is given in Section 5.2.1.) This makes the long-lived processes easy to identify: they are the ones that have run the longest so far. Selecting processes for migration based on runtime will be much better than selecting at random, because a random process will most likely be very short. But note that selection based on runtime depends on a detailed characterization of the workload, which in fact is valid only for the specific workload that is indeed observed empirically.

### Sensitivity to Workloads

These three examples are, of course, not unique. There are many examples in which workload features have a significant effect on performance. Importantly, not every workload feature has the same effect: in some cases it is one specific workload feature that is the most important. The problem is that it is not always obvious in advance which feature is the most important; even if it *seems* obvious, we might be wrong [237, 438, 248]. This motivates the practice of conservative workload modeling, where an attempt is made to correctly model all known workload features, regardless of their perceived importance [248]. Alternatively, it motivates the use of real workloads to drive evaluations, because real workloads may contain features that we do not know about and therefore cannot model.

## 1.2 TYPES OF WORKLOADS

Workloads appear in many contexts and therefore have many different types.

© in this web service Cambridge University Press www.cambridge.org

### 1.2.1 Workloads in Different Domains

The previous three examples are all from the field of scheduling by an operating system, where the workload items are jobs that are submitted by users. This type of workload is characterized by many attributes. If only the scheduling of the CPU is of interest, the relevant attributes are each job's arrival and running times. If memory usage is also being investigated, the total memory usage and locality of reference also come into play, because memory pressure can have an important effect on scheduling and lead to swapping. I/O can also have a great effect on scheduling. Modeling it involves the distribution of I/O sizes and how they interleave with the computation. For parallel jobs, the number of processors used is an additional parameter, which influences how well jobs pack together.

The level of detail needed in workload characterization depends on the goal of the evaluation. For example, in the context of operating system scheduling, it is enough to consider a process as "computing" for a certain time. But when studying CPU architectures and instruction sets, a much more detailed characterization is required. The instruction mix is important in determining the effect of adding more functional units of different types. Dependencies among instructions determine the benefits of pipelining, branch prediction, and out-of-order execution. Loop sizes determine the effectiveness of instruction caching. When evaluating the performance of a complete CPU, all these details have to be correct. Importantly, many of these attributes are input dependent, so representative workloads must include not only representative applications but also representative inputs [204].

I/O provides another example of workloads that can be quite complex. Attributes include the distribution of I/O sizes, the patterns of file access, and the use of read vs. write operations. It is interesting to note the duality between modeling I/O and computation, depending on the point of view. When modeling processes for scheduling, I/O is typically modeled as just taking some time between CPU bursts (if it is modeled at all). When modeling I/O, computation is modeled as just taking some time between consecutive I/O operations. Of course, it is possible to construct a fully detailed joint model, but the number of possible parameter combinations may grow too much for this to be practical.

Application-level workloads are also of interest for performance evaluation. A prime example is the sequence of transactions handled by a database. Databases account for a large part of the usage of large-scale computer systems such as mainframes and are critical for many enterprise-level operations. Ensuring that systems meet desired performance goals without excessive (and expensive) overprovisioning is therefore of great importance. Again, reliable workload models are needed. For example, the transactions fielded by the database of a large bank can be expected to be quite different from those fielded by a database that provides data to a dynamic web server. The differences may lie in the behavior of the transactions (e.g., how many locks they hold and for how long, how many records they read and modify) and in the structure of the database itself (the number of tables, their sizes, and relationships among them).

The workload on web servers has provided a fertile ground for research, as has network traffic in general. Of major interest is the arrival process of packets. Research in the early 1990s showed that packet arrivals are correlated over

© in this web service Cambridge University Press www.cambridge.org

CAMBRIDGE

Cambridge University Press
978-1-107-07823-9 - Workload Modeling for Computer Systems Performance Evaluation
Dror G. Feitelson
Excerpt
More information



long periods, as manifested by burstiness at many different time scales. This finding was in stark contrast with the Poisson model that was routinely assumed until that time. The new models based on this finding led to different performance evaluation results, especially with respect to queueing and packet loss under high loads.

The world wide web is especially interesting in terms of workload modeling because the workloads seen at the two ends of a connection are quite different. First, there is the many-to-many mapping of clients to servers. A given client only interacts with a limited number of servers, whereas the population as a whole may interact with many more servers and display different statistics. Servers, in contrast, typically interact with many clients at a time, so the statistics they see are closer to the population statistics than to the statistics of a single client. In addition, caching by proxies modifies the stream of requests en route [270, 265]. The stream between a client and a cache has more repetitions than the stream from the cache to the server. Uncacheable objects may also be expected to be more prevalent between the cache and the server than between the clients and the cache, because in the latter case they are intermixed with more cacheable objects.

### 1.2.2 Dynamic vs. Static Workloads

An important difference between workload types is their rate of events. A desktop machine used by a single user may process several hundreds of commands per day. This may correspond to thousands or millions of I/O operations and to many billions of CPU instructions and memory accesses. A large-scale parallel supercomputer may serve only several hundred jobs a day from all users combined. A router on the Internet may handle billions of packets in the same time frame.

Note that in this discussion we talk of *rates* rather than *sizes*. A size implies that something is absolute and finite. A rate is a size per unit of time and implies continuity. This is related to the distinction between static and dynamic workloads. A *static workload* is one in which a certain amount of work is given, and when it is done that is it. A *dynamic workload*, in contrast, is one in which work continues to arrive all the time; it is never "done."

The differences between static and dynamic workloads may have the following subtle implications for performance evaluation.

- A dynamic workload requires the performance evaluator to create a changing mix of workload items (e.g., jobs). At the very minimum, doing so requires an identification of all possible jobs and data regarding the popularity of each one, which may not be available. With static workloads you can use several combinations of a small set of given applications. For example, given applications $A$, $B$, and $C$, you can run three copies of each in isolation or a combination of one job from each type. This is much simpler, but most probably further from being realistic. Benchmarks (discussed in the next section) are often static.
- A major difficulty with dynamic workloads is that they include an arrival process, which has to be characterized and analyzed in addition to the workload items themselves. A static workload does not impose this additional burden. Instead, it is assumed that all the work arrives at once at the outset.

© in this web service Cambridge University Press www.cambridge.org

CAMBRIDGE

Cambridge University Press
978-1-107-07823-9 - Workload Modeling for Computer Systems Performance Evaluation
Dror G. Feitelson
Excerpt
More information



- Distributions describing static or dynamic workloads may differ. For example, a snapshot of a running system may be quite different from a sample of the input distribution, due to correlations of workload attributes with residence time. Thus if the input includes many short jobs and few long jobs, sampling from a trace of all the jobs that were executed on the system (effectively sampling from the input distribution) will display a significant advantage for short jobs. But observing a snapshot of the live system may indicate that long jobs are more common, simply because they stay in the system longer and therefore have a higher probability of being seen in a random sample.
- Perhaps the most important difference occurs because performance often depends on the system's state. A static workload being processed by a "clean" system may be very different from the same set of jobs being processed by a system that had previously processed many other jobs in a dynamic manner. The reason is system aging (e.g., the fragmentation of resources) [638].
- Aging is especially important when working on age-related failures (e.g., those due to memory leaks). Static workloads are incapable of supporting work on topics such as software rejuvenation [691]. A related example is the study of thrashing in paging systems [173]. Such effects cannot be seen when studying the page replacement of a single application with a fixed allocation. Rather, they only occur due to the dynamic interaction of multiple competing applications.

Several of these considerations indicate that static workloads cannot be considered as valid samples of real dynamic workloads. This book focuses on dynamic workloads.

### 1.2.3 Benchmarks

One of the important uses of performance evaluation is to compare different systems to each other, typically when trying to decide which to buy. However, such comparisons are meaningful only if the systems are evaluated under equivalent conditions and, in particular, with the same workload. This motivates the canonization of a select set of workloads that are then ported to different systems and used as the basis for comparison. Such standardized workloads are called *benchmarks*.

Benchmarks have a huge impact on the computer industry, because they are often used in marketing campaigns [722]. Moreover, they are also used in performance evaluation during the design of new systems and even in academic research, so their properties (and deficiencies) may shape the direction of new developments. It is thus of crucial importance that benchmarks be representative of real needs. To ensure the combination of representativeness and industry consensus, several independent benchmarking organizations have been created. Two of the best known are SPEC and TPC.

SPEC is the Systems Performance Evaluation Consortium [654]. This organization defines several benchmark suites aimed at evaluating computer systems. The most important of these suites is SPEC CPU, which dominates the field of evaluating the microarchitecture of computer processors. This benchmark comprises a set of applications divided into two groups: one emphasizing integer and symbolic processing, and the other emphasizing floating point scientific processing. To ensure that

© in this web service Cambridge University Press www.cambridge.org

CAMBRIDGE

Cambridge University Press
978-1-107-07823-9 - Workload Modeling for Computer Systems Performance Evaluation
Dror G. Feitelson
Excerpt
More information



the benchmark is representative of current needs, the set of applications is replaced every few years. New applications are selected from those submitted in an open competition.

TPC is the Transaction Processing Performance Council. This organization defines several benchmarks for evaluating database systems. Perhaps the most commonly used is TPC-C, which is used to evaluate online transaction processing. The benchmark simulates an environment in which sales clerks execute transactions at a warehouse. The simulation has to comply with realistic assumptions regarding how quickly human users can enter information.

The SPEC CPU suite and TPC-C are essentially complete, real applications. Other benchmarks are composed of kernels or of synthetic applications. Both of these approaches reduce realism in the interest of economy or focus on specific aspects of the system. *Kernels* are small parts of applications in which most of the processing occurs (e.g., the inner loops of scientific applications). Their measurement focuses on the performance of the processor in the most intensive part of the computation. *Synthetic applications* mimic the behavior of real applications without actually computing anything useful. Using them enables the measurement of distinct parts of the system in isolation or in carefully regulated mixtures; this is often facilitated by parameterizing the synthetic application, with parameter values governing various aspects of the program's behavior (e.g., the ratio of computation to I/O) [93]. Taking this to the extreme, *microbenchmarks* are small synthetic programs designed to measure a single system feature, such as memory bandwidth or the overhead to access a file. In this case there is no pretense of being representative of real workloads.

Benchmarking is often a contentious affair. Much argument and discussion are spent on methodological issues, as vendors contend to promote features that will show their systems in a favorable light. There is also the problem of the benchmark becoming an end in itself, when vendors optimize their systems to cater to the benchmark, at the possible expense of other application types. It is therefore especially important to define what the benchmark is supposed to capture. There are two main options: to span the spectrum of possible workloads or to be representative of real workloads.

Covering the space of possible workloads is useful for basic scientific insights and when confronted with completely new systems. In designing such benchmarks, workload attributes have to be identified and quantified, and then combinations of realistic values are used. The goal is to choose attributes and values that cover all important options, but without undue redundancy [193, 387]. The problem with this approach is that, by definition, it only measures what the benchmark designers dreamed up in advance. In other words, there is no guarantee that the benchmarks indeed cover all possibilities.

The other approach requires that benchmarks reflect real usage and be representative of real workloads. To ensure that this is the case, workloads have to be analyzed and modeled. This should be done with considerable care and an eye to detail. For example, when designing a benchmark for CPUs, it is not enough to consider the instruction mix and their interdependencies – it is also necessary to consider the interaction of the benchmark with the memory hierarchy, as well as its working set size.

© in this web service Cambridge University Press www.cambridge.org

CAMBRIDGE

Cambridge University Press
978-1-107-07823-9 - Workload Modeling for Computer Systems Performance Evaluation
Dror G. Feitelson
Excerpt
More information



Although benchmarks are not discussed in detail in this book, the methodologies covered are expected to be useful as background for the definition of benchmarks.

To read more: Two surveys on benchmarks were written by Weicker [723, 722]. One of the reasons they are interesting is that they show how benchmarks change over time.

## 1.3 WORKLOAD MODELING

Workload modeling is the attempt to create a simple and general model, which can then be used to generate synthetic workloads at will, possibly with slight (but well-controlled!) modifications. The goal is typically to be able to create workloads that can be used in performance evaluation studies, and the synthetic workload is supposed to be similar to those that occur in practice on real systems. This is a generalization of the concept of benchmarks, which is applicable when the consensus regarding the precise workload is less important.

### 1.3.1 What It Is

Workload modeling always starts with measured data about the workload. This data is often recorded as a trace, or log, of workload-related events that happened in a certain system. For example, a job log may include data about the arrival times of jobs, who ran them, and how many resources they required. Basing evaluations on such observations, rather than on baseless assumptions, is a basic principle of the scientific method.

The suggestion that workload modeling should be based on measurements has been made at least since the 1970s [256, 653, 19]. However, for a long time relatively few models based on actual measurements were published. As a result, many performance studies did not use experimental workload models at all (and do not to this day). The current wave of using measurements to create detailed and sophisticated models started in the 1990s. It was based on two observations: one, that real workloads tend to differ from those often assumed in mathematical analyses, and two, that this makes a difference.

There are two common ways to use a measured workload to analyze or evaluate a system design [127, 256, 611]:

1. Use the traced workload directly to drive a simulation.
2. Create a model from the trace and use the model for either analysis or simulation.

For example, trace-driven simulations based on large address traces are often used to evaluate cache designs [634, 409, 400, 704]. But models of how applications traverse their address space have also been proposed and provide interesting insights into program behavior [682, 683].

The essence of modeling, as opposed to just observing and recording, is one of abstraction. This means two things: generalization and simplification.

Measurements are inherently limited. Collecting data may be inconvenient or costly, and instrumentation may introduce overhead. The conditions under which

© in this web service Cambridge University Press www.cambridge.org