Workload Characterization: Beyond the Book

Before we can determine whether a system is schedulable and what kinds of timing constraints can be met, we must characterize its workload. Your book does a good job of talking about the typical forms of workloads encountered in traditional real-time systems design, but we want you to be familiar with workload characterization outside of real-time systems, in particular how we encounter it in the context of queuing systems and networks.

Charactering a measured workload

Ideally, we can either specify a workload of a system directly, but often we can only measure the workload of an existing system and extrapolate from that to the workload of the new system.

Workload measurement is outside of the scope of this class. Its output, however, is simply *traces* of measured data. It is possible to use the traces directly in designing and testing the system. However, there is a tradeoff. While a trace is a very accurate representation of a workload *instance*, it may not be a good representative of the workload *ensemble* (the set of all possible traces). In fact, it must be the case that the workload is an *ergodic process* before one long trace is representative of the system. Few workloads are ergodic. We could use many traces to overcome this issue, but then we have the complexity of dealing with them whenever we want to test our system.

Typically, we take many traces and *summarize* them using statistical methods, creating *workload models* that we can use to test the system quickly and over a wide range of *workload parameters*. Jain, Chapter 6 covers some of the techniques that are used to do this summarization.

Periodic, Sporadic, Aperiodic

Your book classifies tasks according to these three terms. Recall that

- A task is a generator of jobs
- A job has an execution time or *service time* or *size*
- A job has a release time or *arrival time*
- A job has some sort of timing constraint
- Jobs form a graph whose edges represent *dependencies* or communication. A job's dependencies must be satisfied before it can be run.

Using these definitions, a hard real-time periodic task is deterministic and can be described as (P,T,S,D), where P is the arrival time of its first job, T is the period between job arrivals, S is the size of a job, and D is the deadline of a job relative to its arrival time. Many tasks can be fit into this model because we can treat the parameters as upper bounds.

A sporadic task is one which does not generate periodic jobs. Rather, the *interarrival time* between job arrivals is a random variable that is bounded from below. This means that the jobs of the task cannot arrive too often. Because of this bound, it is possible to consider these tasks within a convention hard real-time framework.

The book's puts all other kinds of tasks into the aperiodic grab-bag. The purpose of this handout is to explain the important distinctions that exist in this class of tasks. You are also reading Jain, Chapter 30 to get the classical view of this. The handout focuses on "modern queuing theory", post-dating the mid-90s.

Arrival processes

Generally speaking, aperiodic tasks belong in the domain of queuing theory. The goal of queuing theory is much like that of circuit analysis. Given a *queuing network* (a system of queues feeding servers that feed other queues), a *scheduling policy* for the queues (simplistically: the order in which jobs are pulled from a queue and fed to its server), and an *arrival process* for jobs into the queuing network, queuing theory seeks to give us analytic expressions for things like the average time of a job in the system, the variance of the time of a job in the system, and how numbers like this vary as jobs arrive faster (or are larger). If you've taken a circuits theory, you'll find your old friend Kirchoff is very important here.

In many cases, it is impossible (or very difficult) to find analytic expressions for performance characteristics. The ease of finding analytic expressions tends to be highly dependent on the arrival model.

Note that the term arrival process is intended to invoke *stochastic process*, a key idea in statistics, particularly time-series analysis.

We will talk more about queuing theory later in the course

Simplest stochastic arrival process

The simplest stochastic arrival process is one in which job interarrival times[1] and the job sizes are chosen from i.i.d.[2] exponential[3] distributions. The arrivals looks like they are scattered with uniform randomness over absolute time.

Exponential distributions have a couple of really important things going for them:

- They have finite variance.
- They are memoryless. If X is a random variable from an exponential distribution and you know that X>z, it doesn't help you better estimate the probability that X>2z.

[1] Having just exponentially distributed interarrival times gives you a *Poisson process.*
[2] Independent, identically distributed. Identically distributed just means that the distribution is not a function of time. Independent means that the distribution does not depend on its output. Getting one random variable from it does not tell you anything about what the next one will be.
[3] An exponential distribution is characterized entirely by its mean.

In the queuing theory literature, you will find that the simplest queuing network is an M/M/1 queue. The "M/M" means "exponential interarrival, exponential job size" and the "/1" means "a single, infinite capacity queue feeding a single server" . The properties noted above mean that M/M/1 and, indeed, most queuing networks fed with an "M/M" arrival process are solvable.

Jain, Chapter 30 goes into much more detail about this.

Unfortunately, such processes are actually quite rare in actual reality.

Power-law job sizes

Suppose that you have a trace of job service times on some system and you plot a histogram of the service times. You see that histogram decreases quickly. You fit an exponential curve to the histogram and it seems to fit quite well. You are delighted that your measurements show a "/M" workload model.

You may well be wrong.

It turns out that job sizes in *many* contexts actually follow a *power-law distribution*. That is, the histogram is likely much better fitted with a function like x^{-B} than e^{-x}, where x is the job size. If you plot x^{-B} and e^{-x} in linear-linear scale, they look nearly identical, but this masks several important differences:

- x^{-B} is NOT memoryless
- x^{-B} contains MANY more large jobs than e^{-x}
- x^{-B} has *infinite variance*

These differences have profound effects on scheduling policies, often completely changing the answer to high level questions. For example, it makes no sense to migrate running jobs whose sizes are chosen from an exponential distribution, while it makes a lot of sense to do so for power-law sized jobs. With exponential jobs, several online scheduling policies that are optimal for all jobs with respect to average response time lead to starvation of some jobs, while for power-law jobs, the starvation does not occur.

When looking at a histogram for some trace data, it is important to check for exponential versus power-law job sizes. If you plot the histogram with a log scale vertical axis and linear scale horizontal axis, you will see a straight line if you have exponential job sizes. If you plot it with both axes in log scale, you will see a straight line if you have power-law job sizes.

We often model a power-law job size distribution with a Pareto distribution. When it is the case that job sizes are practically limited from above or below, we typically use a Bounded Pareto distribution.

Analysis of queuing networks with Pareto job sizes, or for that matter, job sizes of any distribution, is a matter of current research.

Correlated arrivals

Often, we assume exponential interarrival times. We can, of course, model interarrival times using other distributions. If we do so, however, queuing network analysis becomes much harder and in many cases is a matter for current research.

A larger issue with interarrival times is the assumption of independence. In practice, it is very often the case that interarrival times, regardless of their distribution, are not independent. In other words, the existence of an arrival changes the probability that a second arrival will occur within some period of time.

Correlated arrivals are often called “bursty”, meaning that arrivals occur in bursts. Seminal work in networks in the ‘90s showed that bursts not only occur, but that they occur at all timescales. There are 10 ms bursts, 10 hour bursts, etc. This phenomenon is often called *long-range dependence* and is typically explained as (stochastic) *self-similarity* because that is the only explanation that avoids *nonstationarity*, which we explain shortly.

Interestingly, self-similarity is *generated* in systems in which power-law-sized jobs are mixed together. Consider Jain, Chapter 30’s example of an M/M/1 queue as a generator of Poisson arrivals. An M/G/1 queue is not. With power-law job sizes, such a queue can be shown to generate correlated, self-similar departures.

Note that, regardless of the arrival distribution or correlation and the job size distribution or correlation, the number of jobs in a queue is correlated over time. We can make use of this correlation to decide whether or not to place a job on a particular queue. Correlation over time is analyzed using an *autocorrelation function.*

We often need to express correlation using very simple models. One that is common in networking is the *leaky bucket.* Here, the idea is that we have a bucket of n bits that leaks at some rate b. b defines the average rate at which work arrives, while n reflects the largest burst that can be dumped on the network at once.

More correlations

Just as job arrivals may be correlated over time, job sizes can be correlated over time. Job sizes could conceivably be correlated with interarrival times. For example, as a user increases the rate at which he works, the jobs he generates might become smaller in order to get finished in time.

The most complete characterization of correlation is *cointegration*, which captures correlations over time including from job size to interarrival. Cointegration would show is if, for example, the size of a current job is influenced by the interarrival time of a job 10 jobs ago.

For the most part, not much is known about these more general correlations. If they exist, an online scheduler could exploit them to improve its performance.

Nonstationarity

In a stationary process, the outputs (job size, interarrival time) vary, but the stochastic process that produces them does not. Nonstationarity simply means that the stochastic process itself is a function of time. So, for example, we might have exponential interarrival times, but the mean of the distribution might increase with time, or abruptly change from one epoch to the next.

Nonstationarity creates tremendous problems for workload characterization, queuing network analysis, and system design. For this reason, we often try to avoid it, modeling systems as being self-similar. However, it is not known how much is lost in doing so. It is conceivable that nonstationarity has as profound an effect on the answers to fundamental questions as self-similarity and power-law job sizes did.

For an example of nonstationarity, consider, for example, the *Slashdot effect.* If a web page on a particular web server suddenly becomes popular, is the server's workload anything like it was before the occurrence?