



Wiley Encyclopedia of
Operations Research and
Management Science

Queueing Disciplines

Journal:	<i>Wiley Encyclopedia of Operations Research and Management Science</i>
Manuscript ID:	EORMS-09-0054.R1
Wiley - Manuscript type:	Introductory
Date Submitted by the Author:	18-Apr-2009
Complete List of Authors:	Harchol-Balter, Mor; Carnegie Mellon University, Computer Science Dept.
Keywords:	scheduling policies, M/G/1, SRPT, SJF, Processor-Sharing



view

Queueing Disciplines

Mor Harchol-Balter
Computer Science Department
Carnegie Mellon University

(Introductory Level for EORMS)

This article introduces queueing disciplines, also known as *scheduling policies*, and describes their effect on the performance of computer systems, call centers, and other systems where queueing is involved.

1 Why Scheduling Matters

Try to remember the last time you asked yourself a question like: “Why is this computer system so slow?” or “Why is it taking so long for the phone company to pick up my call?” At the time, you probably came to the realization that delay was being caused by *other users* who were sharing the database, or who were simultaneously using the call center.

Given that in many situations we do not have resources all to ourselves, one can ask, “How can we reduce user waiting time?” One obvious answer is to purchase more resources. For example, buying more RAM or a faster CPU could help in a computer system. Likewise, adding (human) servers could reduce waiting time at a call center. Such solutions of course cost money.

The goal of *scheduling* is to reduce mean delay *for free!* By simply serving jobs in the “right order,” we can often reduce mean delay by an order of magnitude, without requiring the purchase of additional resources. How can this be? What is the “right order” in which jobs should be scheduled? The answer depends on parameters of the system being scheduled, like load and the job size distribution. These parameters and others will be explained in Section 2. Typical scheduling policies will then be defined in Section 3 and analyzed in Section 4. Finally, in Section 5 we will see some recent real-world success stories where scheduling has been used to greatly decrease average delays. The versatility of scheduling has made it the topic of thousands of research papers. While the majority of this paper introduces scheduling under the simplest queueing model (an M/G/1 queue) and considers only the simplest performance metric (mean response time), in Section 6 we briefly discuss more complex queueing models, performance metrics, and scheduling policies, and give pointers to research papers covering those topics.

2 Queueing Model

The majority of this paper will present results on scheduling policies for a single queue, the M/G/1 queue, shown in Figure 1. In the M/G/1 queue, it is assumed that jobs arrive according to a Poisson Process with some average rate λ jobs per second. A “job” may refer to a computing job or may be a customer placing a call into a call center. We depict a job as a person in the figure. As shown in the figure, jobs are not all the same: some jobs have large size (require a large amount of service) and others have small size. For the M/G/1, it is assumed that all job sizes are independent instances from a common general job size distribution (‘G’ = general). We will use the random variable X to denote a job’s size. The expected (mean) job size is $E[X]$, the second moment of job size is $E[X^2]$, and the squared coefficient of variation of job size, C_X^2 , is the variance normalized by the square

of the mean, namely,

$$C_X^2 = \frac{\text{var}(X)}{\mathbf{E}[X^2]}$$

The *load*, ρ , of the system is the time-average fraction of the time that the server is busy, and is defined by $\rho = \lambda \mathbf{E}[X]$, where we assume $0 \leq \rho < 1$. In the analysis of scheduling policies, we will assume that the job size distribution is continuous with finite variance. We use $f(\cdot)$ to denote its probability density function (pdf) and $F(\cdot)$ to denote its cumulative distribution function.

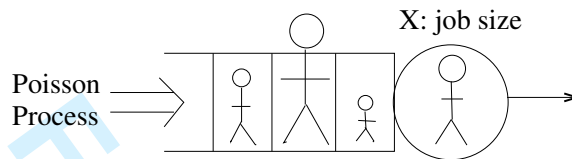


Figure 1: Illustration of an $M/G/1$ queue. Jobs (or customers) have different sizes (service demands).

In many applications, particularly computer science applications, it is frequently the case that empirical job size distributions exhibit *very high variability*, and are best modeled by a Pareto distribution, a LogNormal, or a Weibull with high C_X^2 . Some examples include Unix process CPU requirements (see [1, 2]) (where C^2 values of 25 to 49 have been measured), sizes of files transferred through the Web [3, 4], sizes of files stored in Unix filesystems [5], durations of FTP transfers in the Internet [6], and CPU requirements for supercomputing jobs [7]. In this article we will therefore focus a lot of attention on the effect of job size variability. *Decreasing failure rate* (hereafter referred to as DFR) is also prevalent in computer science applications. Roughly speaking, under DFR, the more service a job has received so far, the greater its remaining service requirement is likely to be. For example, the more CPU a UNIX job has used so far, the more CPU it is expected to use in the future [1], and the more bytes a flow has transmitted so far, the more bytes it is likely to transmit in the future [8].

3 Definition of Common Scheduling Policies

In this section, we define the most common scheduling policies for a single-server system. These can be subdivided along two axes, see Figure 2. *Blind* policies do not make use of a job's size when determining which job to run next, while *size-based* policies do. *Non-preemptive* policies can't preempt a job once it has started running, while *preemptive* policies allow jobs to be stopped and restarted at a later point, without losing intermediary work. All policies which we consider are *work-conserving*, meaning that the server is always busy working on a job whenever there are jobs to be served, and no work that is done is lost.

We start by describing policies which are non-preemptive and do not make use of a job's size:

FCFS or FIFO – First-Come-First-Served also known as First-in-First-Out. Jobs are served in the order that they arrive, with each job being run to completion before the next job receives service. This policy is typically used in manufacturing systems, call centers, and supercomputing centers. In such settings it is often difficult to preempt a running job, and it seems “fair” to serve customers in the order that they arrive.

RANDOM – Random-Order-Service. Whenever the server is free, it chooses a random job in the queue to run next and runs that job to completion. This policy is mostly of theoretical interest.

LCFS – Last-Come-First-Served (non-preemptive). Whenever the server is free, it chooses the last job to arrive and runs that job to completion. This policy is used for applications where jobs are piled onto a “stack”

	Non-preemptive	Preemptive
Blind to Size	FCFS RANDOM LCFS	PS PLCFS FB
Uses Size	SJF	PSJF SRPT

Figure 2: Taxonomy of some common scheduling policies.

structure, with each new job being “pushed” on top of the stack. When the server becomes free, it is convenient to “pop” the job off the top of the stack.

The following policies are preemptive and do not make use of a job’s size:

PS – Processor-Sharing. Whenever there are n jobs in the system, each job is allocated $1/n$ th fraction of the server. Processor-Sharing has its roots in Round-Robin scheduling of computing jobs by the operating system CPU, whereby the CPU rotates among the jobs in the queue in a cyclic fashion, giving each job only a small fixed quantum of service before moving on to the next job, see [9]. If one imagines the limit as the quantum size goes to zero, one gets Processor-Sharing. Besides its use in modeling CPU processing, PS scheduling is also a good model of equal bandwidth sharing of a network link.

PLCFS – Preemptive-Last-Come-First-Served. At every moment, the server is always running that job that arrived last. When a new jobs arrives, it always preempts the job in service. This policy makes sense, for example, in a market information processing situation, where the most recent job carries the most relevant information (current stock price) and older jobs are less relevant.

FB or LAS or SET – Foreground-Background also known as Least-Attained-Service or Shortest-Elapsed-Time. This policy always serves the job which has obtained the least service so far. We say that a job’s *age* is the total service it has received so far. FB (LAS) always serves that job with the lowest age – resulting in PS if all jobs have the same age. This policy is often used in situations where the job sizes are not known, but it is known that the job size distribution has DFR, e.g., CPU scheduling or flow scheduling. Given unknown job size and DFR, serving the job with lowest age is equivalent to serving that job that has the lowest expected *remaining* service requirement. FB has been proven to minimize the mean response time and queue length distribution among blind policies, see [10].

We now move on to policies which make use of a job’s size, favoring shorter jobs in some way. Given that typically the great majority of jobs are “short,” favoring short jobs reduces overall mean response time (it is preferable to have long jobs wait behind short jobs, than the reverse).

SRPT – Shortest-Remaining-Processing-Time. This preemptive policy always runs that job which has the currently smallest remaining time. If a job arrives with shorter processing requirement than the job in service, then it preempts the job in service. SRPT is known to minimize the mean response time and the queue length distribution, see [11], however its use is limited to situations where job size and remaining size are known.

PSJF – Preemptive-Shortest-Job-First. This preemptive policy is very similar to SRPT, but favors that job with smallest *original* size rather than smallest *remaining* size. This can be a useful substitute for SRPT when remaining size is not known.

SJF – Shortest-Job-First. Under this non-preemptive variant of PSJF, whenever the server becomes free, it always grabs the shortest job to work on (shortest original size). SJF is used in non-preemptive settings, where one wants to get the benefit of favoring short jobs, but jobs can't be preempted.

4 Performance of Scheduling Policies in M/G/1

The *response time* (a.k.a., flow time or sojourn time) of a job is the time from when a job arrives until it completes service, and is denoted by T . Since some scheduling policies might favor short jobs, it is also instructive to consider $T(x)$, the response time for a job conditioned on that job being of size x . In this section we state results on mean response time, $\mathbf{E}[T]$, and on $\mathbf{E}[T(x)]$ for an M/G/1 queue under the above scheduling policies. While we don't have room to prove these results, we will provide high-level intuition behind each formula. We refer the reader to references such as [12, 13, 14] where complete proofs of these results can be found.

While the formulas for response time that we present below are all well-known, the comparative ranking between the different scheduling policies is not obvious from the formulas, many of which include triple nested integrals. We have therefore evaluated these formulas using MathematicaTM to compare the mean response time under the different policies as a function of load (see Figure 3) and as a function of C^2 (see Figure 4). In both figures, we assume a Weibull job size distribution. The Weibull, defined by $\bar{F}(x) = e^{-(\frac{x}{\lambda})^\alpha}$ with parameters λ and α , is convenient because when $\alpha < 1$, it has DFR and C^2 can be made as high as desired.

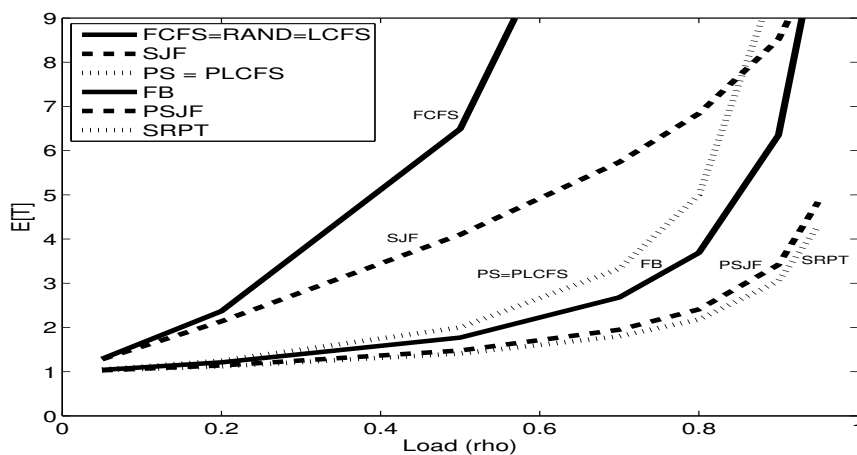


Figure 3: Mean response time as a function of load for the M/G/1 with various scheduling policies. The job size distribution is a Weibull with mean 1 and $C^2 = 10$.

In the analysis below, we will require a few terms that we haven't introduced yet. First, we will need to talk about the load made up of just those jobs of size $< x$. We will denote this by:

$$\rho(x) = \lambda \int_0^x t f(t) dt \quad (1)$$

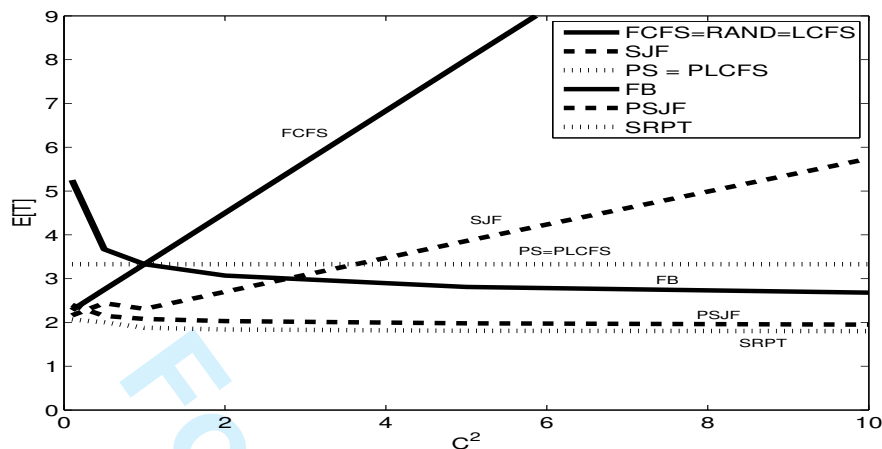


Figure 4: Mean response time as a function of variability (C^2) for the $M/G/1$ with various scheduling policies. Load is fixed at $\rho = 0.7$. The job size distribution is a Weibull with fixed mean 1 and changing C^2 .

Second, we will often talk about a *busy period*, which is defined to be the time from when a job arrives to an empty system until the system is once again empty. We use B to denote the length of a busy period in an $M/G/1$ queue (for any work-conserving policy), and $B(x)$ to denote the length of a busy period conditioned on the fact that the starting job is of size x . Then, it is well known (see, e.g., [15]) that the expected length of $B(x)$ and B increase with load ρ in the following way:

$$\mathbf{E}[B(x)] = \frac{x}{1 - \rho} \quad (2)$$

$$\mathbf{E}[B] = \frac{\mathbf{E}[X]}{1 - \rho} \quad (3)$$

First-Come-First-Served (FCFS)

A job's response time, T , under FCFS can be viewed as its *waiting time*, W (the time from when the job arrives until it first gets service) plus its service time, X . The mean waiting time is given by the well-known Pollaczek-Khinchin (P-K) formula:

$$\mathbf{E}[W]^{FCFS} = \frac{\lambda \mathbf{E}[X^2]}{2(1 - \rho)} \quad (4)$$

Of key importance in this formula is the variability of the job size distribution (the second moment term $\mathbf{E}[X^2]$). Specifically, even if the load, ρ , is low (say $\rho = \frac{1}{2}$), the mean waiting time can still be quite high if the second moment of the job size distribution is high.

The second moment term shows that mean delay (waiting time) is proportional to the variability of the job size distribution. This makes sense because when there are short jobs mixed in with long jobs, some short jobs will inevitably get stuck behind long jobs, causing large mean delay. Another way to think about (4) is to consider the excess (remaining) service time on the job which is currently in service when a random arrival occurs. One might think that the expected excess for the job in service is $\frac{\mathbf{E}[X]}{2}$, however this is only true when jobs have

deterministic job sizes. When job sizes are variable, a random arrival is more likely to see a larger-than-average excess, and, in fact, by the Inspection Paradox (see [16]), the expected excess turns out to be $\mathbf{E}[X^2]/2\mathbf{E}[X]$, which comes up in the derivation of formula (4) above.

From the mean waiting time, it is easy to derive the mean response time as follows:

$$\mathbf{E}[T(x)]^{FCFS} = x + \mathbf{E}[W]^{FCFS} = x + \frac{\lambda \mathbf{E}[X^2]}{2(1-\rho)} \quad (5)$$

$$\mathbf{E}[T]^{FCFS} = \mathbf{E}[X] + \mathbf{E}[W]^{FCFS} = \mathbf{E}[X] + \frac{\lambda \mathbf{E}[X^2]}{2(1-\rho)} \quad (6)$$

Looking at Figures 3 and 4, we see that mean response time under FCFS is worse than any of the other policies when $C^2 > 2$. This is because of the huge role that job size variability plays in response time.

RANDOM and Last-Come-First-Served (LCFS)

While RANDOM and LCFS seem very different from FCFS, it turns out that their mean response time is the same of that of FCFS, and thus they lie on top of FCFS in Figures 3 and 4. In fact, an even stronger claim can be made, [14]:

Lemma 1 *For an M/G/1, any non-preemptive scheduling policy that does not make use of job size will yield the same distribution on the number of jobs in the system.*

The above lemma is easy to understand: Although the different non-preemptive policies may choose their next job differently at each completion point, the job is *not* chosen based on its size. Therefore the time until the next completion is equal in distribution under all such policies. Thus the number of arrivals between each completion is also stochastically the same across policies, resulting in the same stochastic process for the number of jobs in the system. It then follows by Little's Law [17] that the mean response time is also the same for all such policies. Hence:

$$\mathbf{E}[T]^{RANDOM} = \mathbf{E}[T]^{LCFS} = \mathbf{E}[T]^{FCFS} = \mathbf{E}[X] + \frac{\lambda \mathbf{E}[X^2]}{2(1-\rho)} \quad (7)$$

Note that although the distribution of the number of jobs in the system is the same for all blind non-preemptive policies, the distribution of response time is not the same. In particular it can be shown that [14]:

$$\text{var}(T)^{FCFS} < \text{var}(T)^{RANDOM} < \text{var}(T)^{LCFS}$$

Processor-Sharing (PS)

The PS policy has many pleasant properties. First, the distribution of the number of jobs in M/G/1/PS turns out to be the same as in an M/M/1/FCFS queue, see [16]. Letting $\mathbf{E}[N]$ denote the mean number of jobs, we have that

$$\mathbf{E}[N]^{PS} = \frac{\rho}{1-\rho} \quad (8)$$

The conditional mean response time for a job of size x is similarly attractive:

$$\mathbf{E}[T(x)]^{PS} = x/(1-\rho) \quad (9)$$

Another way of stating this is that, under PS, every job, regardless of its size, is slowed down by the same constant factor:

$$\text{Constant slowdown factor} = \frac{\mathbf{E}[T(x)]^{PS}}{x} = \frac{1}{1 - \rho}$$

Observe that $1/(1 - \rho) = \mathbf{E}[N]^{PS} + 1$, see (8). Intuitively it makes sense that an arrival is slowed down by a factor equal to the number of jobs it sees plus itself (roughly speaking).

Finally, from (9), we see that

$$\mathbf{E}[T] = \frac{\mathbf{E}[X]}{1 - \rho} \quad (10)$$

This is fascinating because of its complete independence of the job size variability. This property is best illustrated by the perfectly horizontal line in Figure 4, showing the insensitivity of mean response time to C^2 . The survey paper [18] presents more results on PS.

Preemptive-Last-Come-First-Served (PLCFS)

The PLCFS policy has surprisingly little to do with the LCFS policy, although they are both based on serving the last job to arrive. While the LCFS mean response time is highly affected by job size variability, we will see that the PLCFS mean response time is independent of job size variability.

Consider a new arrival of size x to an M/G/1/PLCFS queue. We will refer to this job as “job x .” All arrivals prior to job x will not get worked on while job x is in the system. However, all arrivals after job x , which occur before job x completes, will have priority over job x . Thus, we can view the response time of job x under PLCFS as the length of a busy period started by a job of size x , namely $B(x)$. From (3), it follows that:

$$\mathbf{E}[T(x)]^{PLCFS} = \frac{x}{1 - \rho} \quad (11)$$

$$\mathbf{E}[T]^{PLCFS} = \frac{\mathbf{E}[X]}{1 - \rho} \quad (12)$$

Thus the mean response time of the M/G/1/PLCFS is identical to that of the M/G/1/PS queue, in agreement with Figures 3 and 4. However PLCFS has an advantage over PS: typically many fewer preemptions. In fact, under PLCFS, each job creates at most 2 preemptions – one when the job arrives and one when it departs. Since preemptions are not free, this can be a real advantage.

Foreground-Background (FB) a.k.a. Least-Attained-Service (LAS)

Again consider an arrival of size x , which we refer to as “job x .” Under FB (a.k.a., LAS), the arriving job x immediately receives service (since it is the youngest job in the system). However, it may later have to share both with jobs already in the system and with new arrivals. Jobs in the system only affect job x *until they hit age x* , after which point they are invisible to job x . New arrivals while job x is in the system will all initially have priority over job x , and will each get to complete *up to x units of work* (actually the minimum of x and their size) before job x leaves.

From the above observations, we can see that if $f(y)$ denotes the probability density function for job size, then we need to define a new probability density function $f_x(y)$ where jobs of size greater than x have been replaced

by jobs of size x . We define:

$$f_x(y) = \begin{cases} f(y), & y < x \\ 1 - F(x), & y = x \end{cases}$$

We define X_x to be a random variable with density function $f_x(y)$, and define $\rho_x = \lambda \mathbf{E}[X_x]$. Then the expected response time for a job of size x is given by:

$$\mathbf{E}[T(x)]^{FB} = \frac{x + \frac{\frac{1}{2}\lambda \mathbf{E}[X_x^2]}{1-\rho_x}}{1-\rho_x} = \frac{x(1-\rho_x) + \frac{1}{2}\lambda \mathbf{E}[X_x^2]}{(1-\rho_x)^2} \quad (13)$$

$$\mathbf{E}[T]^{FB} = \int_0^\infty \mathbf{E}[T(x)] f(x) dx \quad (14)$$

Observe that in appearance (13) does not seem all that different from (5), the corresponding equation for FCFS. There is still a dependence on the second moment of the job size distribution, however the job size is now X_x , with lots of extra mass on x and no mass above x . Also the result is scaled by $(1 - \rho_x)$, due to delay incurred by waiting on later-arriving jobs.

In practice, however, these small changes can have a large effect, particularly when the job size distribution has DFR. Figure 3 shows that the mean response time for FB can be quite close to that of SRPT, which is optimal, and quite far from FCFS. Furthermore, Figure 4 shows that the mean response time for FB can actually decrease with increased job size variability, particularly when increased variability implies even stronger DFR behavior, as in the case of the Weibull job size distribution or the Pareto distribution. The survey paper [19] presents more results on FB.

As a final point, we note that the fact that the curves for FCFS, PS, and FB all cross in Figure 4 at the point where $C^2 = 1$ is no accident. For exponentially distributed job sizes ($C^2 = 1$), it is easy to argue that the continuous-time Markov chain tracking the number of jobs in the system is identical under all these policies.

Shortest-Job-First (SJF)

The waiting time for a job of size x under SJF is given below:

$$\mathbf{E}[W(x)]^{SJF} = \frac{\lambda \mathbf{E}[X^2]}{2(1-\rho(x))^2} \quad (15)$$

where $\rho(x)$, defined in (1), denotes the load made up by jobs of size less than x only.

The sharp resemblance of the above formula to FCFS, see (4), may seem surprising, but should not be. While SJF favors short jobs, even a short job still has to wait for the server to first free up. Thus the waiting time for a job of size x , $\mathbf{E}[W(x)]^{SJF}$ still includes the excess term, involving $\mathbf{E}[X^2]$, that we saw in FCFS. The only difference is that now we have $(1 - \rho(x))^2$ in the denominator rather than $(1 - \rho)$. The intuition is as follows: When a job of size x arrives, it first has to wait for the server to finish serving its current job (the excess). Then job x has to wait behind those jobs in the queue whose size are $< x$ – these contribute a $(1 - \rho(x))$ factor in the denominator. Finally job x has to wait behind all future arrivals of size $< x$ during its waiting time – these contribute another $(1 - \rho(x))$ factor. Note that in FCFS, job x doesn't have to wait behind future arrivals.

The mean response time for SJF is given as follows:

$$\mathbf{E}[T(x)]^{SJF} = x + \mathbf{E}[W(x)]^{SJF} \quad (16)$$

$$\mathbf{E}[T]^{SJF} = \mathbf{E}[X] + \int_0^\infty \mathbf{E}[W(x)]^{SJF} f(x) dx \quad (17)$$

From Figures 3 and 4, we see that the performance of SJF is far worse than that of the other policies that favor short jobs (PSJF and SRPT), and in fact most closely resembles that of FCFS. This is because the $\mathbf{E}[X^2]$ term dominates mean response time in SJF, as it does in FCFS. The second thing to notice is that SJF appears to “get better” as load gets high. This is because the $(1 - \rho(x))^2$ in the denominator depends on $\rho(x)$, which can be far lower than ρ , for high ρ , causing the denominator in SJF (even with its squared effect) to be larger than that in FCFS.

Preemptive-Shortest-Job-First (PSJF)

PSJF is the preemptive version of SJF. There are two components to an arrival’s response time: the time the arriving job waits until it *first* receives service (we call this *initial waiting time*), and the time from when the arrival first receives service until it completes (we call this *residence time*). Since a job in service can be preempted by smaller arrivals, a job’s residence time typically consists of way more than its service time. The mean response time for a job of size x is shown below:

$$\mathbf{E}[T(x)]^{PSJF} = \frac{\frac{\lambda}{2} \int_0^x t^2 f(t) dt}{(1 - \rho(x))^2} + \frac{x}{(1 - \rho(x))} \quad (18)$$

The first term in (18) indicates the initial waiting time for an arrival of size x . This is very similar in form to the waiting time for SJF. The key difference is that the $\mathbf{E}[X^2]$ term has been replaced by $\int_0^x t^2 f(t) dt$, which represents the contribution to the second moment made up only of jobs of size $< x$. To understand this change, observe that an arrival of size x no longer has to consider any jobs whose original size is $> x$, since the arrival has immediate priority over those jobs; this is in contrast to SJF, where a small arrival is still affected by the excess of the job in service, no matter what its size.

The second term above is the residence time for a job of size x . Observe that this has the form of the duration of an M/G/1 busy period, started by a job of size x , where the system load is $\rho(x)$, see (3). This should make perfect sense, since a job of size x starts its residence time when there are no jobs of size $< x$ in the system, however this job is then interrupted by every arrival of size $< x$, and doesn’t get to depart until there are again no jobs of size $< x$ in the system.

The mean response time for PSJF is given as follows:

$$\mathbf{E}[T]^{PSJF} = \int_0^\infty \mathbf{E}[T(x)]^{PSJF} f(x) dx \quad (19)$$

As can be seen by Figure 3, the mean response time under PSJF is far lower than that under SJF, simply because the $\mathbf{E}[X^2]$ factor is gone. As expected, Figure 4 shows that PSJF is far less sensitive to variability in the job size distribution than is SJF.

Shortest-Remaining-Processing-Time (SRPT)

The mean response time for a job of size x under SRPT is very similar to that under PSJF, as shown below:

$$\mathbf{E}[T(x)]^{SRPT} = \frac{\frac{\lambda}{2} \int_0^x t^2 f(t) dt + \frac{\lambda}{2} x^2 (1 - F(x))}{(1 - \rho(x))^2} + \int_0^x \frac{dt}{1 - \rho(t)} \quad (20)$$

1
2
3
4
5
6
7
8 The first term represents the initial waiting time for a job of size x . The only difference from PSJF is the
9 additional term involving $x^2(1 - F(x))$. To get a rough feel for where this term comes from, observe that under
10 SRPT, jobs of size $> x$ can still contribute to the waiting time of an arrival of size x , if those jobs of size $> x$
11 have remaining time $\leq x$ at the time job x arrives. Thus the $(1 - F(x))$ fraction of jobs of original size $> x$ can
12 contribute at most x^2 to the second moment portion of the initial waiting time of job x .

13
14 The second term represents the residence time for a job of size x . This again resembles the residence time for
15 PSJF, however it is actually a lot smaller. The point is that, as the remaining size of job x drops, the job has to
16 compete with fewer and fewer interruptions (less load), because only those jobs with lower remaining size can
17 bother it. Thus, the residence time for a job of size x can actually be viewed as a sum of busy periods, each
18 started by a job of size dt , where each successive busy period fights against less and less load.

19
20 As expected from [11], Figure 3 shows that the mean response time under SRPT is superior to all other policies.
21 Also as seen in Figure 4, the response time actually decreases a slight bit with increased variability. Importantly,
22 PSJF (with its lower initial waiting time and higher residence time) is a very close approximation to SRPT with
23 respect to overall mean response time.

24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60

5 Scheduling in Practice

It is clear from Figures 3 and 4 that the choice of scheduling policy can dramatically affect mean response time. In situations where the job size variability is high, preemptive policies can be orders of magnitude more effective than non-preemptive policies. When load and variability are high (e.g., $\rho = 0.9$ and $C^2 = 10$, as shown in Figure 3), there is also significant differentiation among preemptive policies, e.g., SRPT offers more than a 3-fold improvement over PS.

These dramatic improvements, at the cost of no additional resources, have not been lost on the computer systems community, which has incorporated smart scheduling in a wide variety of applications including web servers [20], routers [21], wireless networks [22], operating systems [23], and many more. Below we elaborate on just two such examples:

In a series of papers [20, 24, 25], Harchol-Balter et al. look at the problem of speeding up the mean response time in retrieving files at a web server. They find that the bottleneck resource for static “Get FILE” requests is the limited bandwidth in the web server’s access link. Traditionally, all requests share this bandwidth fairly in a manner resembling PS scheduling. The authors argue that scheduling requests in SRPT order would decrease mean response time, since file sizes at web sites are known to exhibit highly variable Pareto distributions. However there are two potential stumbling blocks to implementing SRPT scheduling at web servers: First, one needs to know the “size” of the jobs. Second, there is a fear that by favoring short jobs, one might “starve” long jobs, or certainly treat them very unfairly. The authors deal with the first stumbling block by showing that the “size” of a job, in this case the time to retrieve a file, is well-approximated by the size of the file, which is known. The authors deal with the starvation issue in [26, 27], where they prove a counter-intuitive theorem, showing that, when job sizes follow a heavy-tailed distribution (like the Pareto or Bounded-Pareto), the expected response time of *every* job (including the largest-size job) is actually smaller under SRPT than under PS, provided load is not too high. Motivated by this “all-can-win” fairness result, the authors in [20] proceed to implement SRPT scheduling of HTTP requests by modifying the Linux kernel of an Apache web server to change the order that the server’s socket buffers are drained into the server’s access link, in accordance with SRPT scheduling. They demonstrate that this change improves mean response time by a factor of 5 for higher load, without penalizing requests for large files.

Similarly, Biersack et al. (see [8] and the references therein) consider flow scheduling at a bottleneck router, again with the goal of minimizing mean response time. However here the problem is complicated by the fact

1
2
3
4
5
6
7
8 that the flow durations are not known a priori, so it is not possible to bias towards “short” flows (flows with few
9 packets). Biersack et al. reason that since flow durations have a Pareto distribution (with DFR), favoring “young”
10 flows is a good approximation to favoring “short” flows. They implement an FB-like policy which favors those
11 flows which have transmitted the fewest packets so far, demonstrating an order of magnitude improvement over
12 the traditional PS scheduling of flows. However there are fairness issues with FB that are not present in SRPT,
13 which they resolve by further tailoring of their FB policy.
14

16 6 Other topics in scheduling and references

17
18
19 This document has offered a very brief introduction to scheduling in the $M/G/1$ queue. However what is presented
20 here is just the tip of the iceberg in terms of what is known about scheduling.
21

22 First of all, this document only considers mean response time. In fact, the full Laplace transform of response time
23 (and thus higher moments of response time) is known for all the policies discussed herein and is usually almost
24 as easy to derive (see [13, 14]), although the high-level intuition is less easy to see from the transform. The tail
25 behavior of response time has also been studied for many policies (see the survey paper [28]). For example, it
26 is known that for heavy tailed job size distributions, the response time tails for SRPT, PS, FB, and PLCFS all
27 mimic the tail of the job size distribution, whereas the response time tails for FCFS and SJF mimic the tail of the
28 excess of the job size distribution. Other performance metrics, like “fairness,” have also recently become very
29 important in computer systems (see [27, 29] for an overview).
30

31 There are many scheduling policies which we did not have room to mention. In particular there are many variants
32 of PS that come up in networking (see the survey [30]). We also have not had room to delve into multi-class
33 queues, where each class i of jobs has its own arrival rate λ_i and its own job size distribution, F_i , and jobs
34 in one class might have preemptive, or non-preemptive, priority over another class, see [15]. Also, while we
35 have studied individual policies in this paper, there’s a growing body of work that deals with broad classes of
36 scheduling policies. In particular the SMART class was introduced in [31] and defined so as to encompass a
37 wide range of policies which favor short jobs.
38

39 Scheduling theory also exists for architectures far more general than the $M/G/1$ queue. Many computer systems
40 follow a closed-loop architecture, where there is a fixed number of jobs, N , and a new job creation is only
41 triggered by a job completion. The impact of scheduling in closed-loop systems has been shown to be far smaller
42 than that in the open systems we have considered thus far [32]. By contrast, scheduling can have huge impact
43 in server farm architectures (multi-server systems), see [33], where scheduling decisions need to be made both
44 at the central router level and at the individual host level. Lastly, fluctuations in load, e.g. alternations between
45 very high and low load, can also greatly intensify the effect of scheduling, as seen in [25, 34].
46

47 Finally, while everything we have mentioned so far involves a stochastic setting, there is an entire field of online
48 scheduling where the metric is worst-case response time and policies are ranked according to their “competitive
49 ratio” when compared with the optimal policy (see the survey paper [35]).
50

52 References

- 53
54
55 [1] Mor Harchol-Balter and Allen Downey. Exploiting process lifetime distributions for dynamic load balancing. In
56 *Proceedings of ACM SIGMETRICS*, pages 13–24, Philadelphia, PA, May 1996.
57 [2] W. E. Leland and T. J. Ott. Load-balancing heuristics and process behavior. In *Proceedings of Performance and ACM*
58 *Sigmetrics*, pages 54–69, 1986.
59 [3] Mark E. Crovella and Azer Bestavros. Self-similarity in World Wide Web traffic: Evidence and possible causes.
60 *IEEE/ACM Transactions on Networking*, 5(6):835–846, December 1997.

- 1
2
3
4
5
6
7
8 [4] Mark E. Crovella, Murad S. Taqqu, and Azer Bestavros. Heavy-tailed probability distributions in the world wide web. In *A Practical Guide To Heavy Tails*, chapter 1, pages 1–23. Chapman & Hall, New York, 1998.
- 9 [5] Gordon Irlam. Unix file size survey - 1993. Available at <http://www.faqs.org/faqs/os-research/part2/section-12.html>, September 1993.
- 10 [6] Vern Paxson and Sally Floyd. Wide-area traffic: The failure of Poisson modeling. *Transactions on Networking*, pages 226–244, June 1995.
- 11 [7] Bianca Schroeder and Mor Harchol-Balter. Evaluation of task assignment policies for supercomputing servers: The case for load unbalancing and fairness. *Cluster Computing: The Journal of Networks, Software Tools, and Applications*, 7(2):151–161, April 2004.
- 12 [8] Ernst W. Biersack, Bianca Schroeder, and Guillaume Urvoy-Keller. Scheduling in practice. *Performance Evaluation Review, Special Issue on “New Perspectives in Scheduling”*, 34(4), March 2007.
- 13 [9] Leonard Kleinrock. *Queueing Systems*, volume II. Computer Applications. John Wiley & Sons, 1976.
- 14 [10] R. Righter and J. Shanthikumar. Scheduling multiclass single server queueing systems to stochastically maximize the number of successful departures. *Probability in the Engineering and Information Sciences*, 3:967–978, 1989.
- 15 [11] L. E. Schrage. A proof of the optimality of the shortest remaining processing time discipline. *Operations Research*, 16:678–690, 1968.
- 16 [12] Mor Harchol-Balter. Online Lecture Notes for Performance Modeling Class at CMU. <http://www.cs.cmu.edu/~harchol/Perfclass/class05.html/>.
- 17 [13] Adam Wierman. *Scheduling for Today’s Computer Systems: Bridging Theory and Practice*. PhD thesis, Carnegie Mellon University, 2007.
- 18 [14] Richard. W. Conway, W. L. Maxwell, and Louis W. Miller. *Theory of Scheduling*. Addison-Wesley Publishing Company, 1967.
- 19 [15] Hideaki Takagi. *Queueing Analysis – A Foundation of Performance Evaluation*, volume 1: Vacation and Priority Systems, Part 1. North Holland, 1991.
- 20 [16] Sheldon M. Ross. *Stochastic Processes*. John Wiley and Sons, New York, 1983.
- 21 [17] John D. C. Little. A proof of the queuing formula $L = \lambda W$. *Operations Research*, 9:383–387, 1961.
- 22 [18] S. F. Yashkov. Processor-sharing queues: some progress in analysis. *Queueing Systems Theory and Application*, 2:1–17, 1987.
- 23 [19] Misja Nuyens and Adam Wierman. The foreground-background queue: A survey. *Performance Evaluation*, 65(3–4):286–307, March 2008.
- 24 [20] Mor Harchol-Balter, Bianca Schroeder, Nikhil Bansal, and Mukesh Agrawal. Size-based scheduling to improve web performance. *ACM Transactions on Computer Systems*, 21(2):207–233, May 2003.
- 25 [21] I. A. Rai, G. Urvoy-Keller, M. Vernon, and E. W. Biersack. Performance modeling of las based scheduling in packet switched networks. In *ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, 2004.
- 26 [22] R. Mangharam, M. Demirhan, R. Rajkumar, and D. Raychaudhuri. Size matters: Size-based scheduling for mpeg-4 over wireless channels. In *SPIE and ACM Proceedings in Multimedia Computing and Networking*, pages 110 – 122, 2004.
- 27 [23] H. Feng, V. Misra, and D. Rubenstein. Pbs: a unified priority-based cpu scheduler. In *ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, 2007.
- 28 [24] Mark Crovella, Bob Frangioso, and Mor Harchol-Balter. Connection scheduling in web servers. In *USENIX Symposium on Internet Technologies and Systems*, pages 243–254, Boulder, CO, October 1999.
- 29 [25] Bianca Schroeder and Mor Harchol-Balter. Web servers under overload: How scheduling can help. *ACM Transactions on Internet Technologies*, 6(1), February 2006.
- 30 [26] Nikhil Bansal and Mor Harchol-Balter. Analysis of SRPT scheduling: Investigating unfairness. In *Proceedings of ACM SIGMETRICS*, 2001.
- 31 [27] Adam Wierman and Mor Harchol-Balter. Classifying scheduling policies with respect to unfairness in an M/GI/1. In *Proceedings of ACM SIGMETRICS*, San Diego, CA, June 2003.
- 32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60

- 1
2
3
4
5
6
7
8 [28] Onno Boxma and Bert Zwart. Tails in scheduling. *Performance Evaluation Review, Special Issue on "New Perspectives in Scheduling"*, 34(4), March 2007.
- 9
10 [29] Adam Wierman. Fairness and classifications. *Performance Evaluation Review, Special Issue on "New Perspectives in Scheduling"*, 34(4), March 2007.
- 11
12 [30] Samuli Aalto, Urtzi Ayesta, Sem Borst, Vishal Misra, and Rudesindo Nunez-Queija. Beyond processor sharing. *Performance Evaluation Review, Special Issue on "New Perspectives in Scheduling"*, 34(4), March 2007.
- 13
14 [31] Adam Wierman and Mor Harchol-Balter. Nearly insensitive bounds on SMART scheduling. In *ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, 2005.
- 15
16 [32] Bianca Schroeder, Adam Wierman, and Mor Harchol-Balter. Closed versus open system models: a cautionary tale. In *Proceedings of Networked Systems Design and Implementation (NSDI)*, 2006.
- 17
18 [33] Mor Harchol-Balter, Mark Crovella, and Cristina Murta. On choosing a task assignment policy for a distributed server system. *IEEE Journal of Parallel and Distributed Computing*, 59:204–228, 1999.
- 19
20 [34] Varun Gupta, Mor Harchol-Balter, Alan Scheller-Wolf, and Uri Yechiali. Fundamental characteristics of queues with fluctuating load. In *ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, 2006.
- 21
22 [35] Kirk Pruhs. Competitive online scheduling for server systems. *Performance Evaluation Review, Special Issue on "New Perspectives in Scheduling"*, 34(4), March 2007.
- 23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60