Scheduling in Practice

Ernst W. Biersack
Institut Eurecom
Sophia-Antipolis, France
erbi@eurecom.fr.

Bianca Schroeder
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213
bianca@cs.cmu.edu.

Guillaume Urvoy-Keller
Institut Eurecom
Sophia-Antipolis, France
urvoy@eurecom.fr

ABSTRACT

In queueing theory, it has been known for a long time that the scheduling policy used in a system greatly impacts user-perceived performance. For example, it has been proven in the 1960's that size-based scheduling policies that give priority to short jobs are optimal with respect to mean response time. Yet, virtually no systems today implement these policies. One reason is that real systems are significantly more complex than a theoretical M/M/1 or M/G/1 queue and it is not obvious how to implement some of these policies in practice. Another reason is that there is a fear that the big jobs will "starve", or be treated unfairly as compared to Processor-Sharing (PS). In this article we show, using two important real world applications, that size-based scheduling can be used in practice to greatly improve mean response times in real systems, without causing unfairness or starvation. The two applications we consider are connection scheduling in web servers and packet scheduling in network routers.

1. INTRODUCTION

Resource allocation in the very first computer systems was easy: computers were special purpose devices, used by only one user at a time. Scheduling of system resources quickly became more complex, when computer systems started to be shared simultaneously by multiple users and to run many concurrent processes. Today, many systems simultaneously serve thousands of clients, making the question of how to schedule resources between clients a challenging and critical aspect of system design.

Resource scheduling in modern IT systems serves a number of different goals. One of the key goals is to schedule system resources, e.g. CPU and disk, such as to provide each user the illusion of "owning the system" or a fair share of it. Another goal is to schedule requests such as to make efficient use of the system resources (e.g. minimize the movements of a disk head).

In this paper, we focus on a third aspect of scheduling: scheduling as a means to provide *shorter mean response times* across *all* requests in a system. The response time of a request is defined as the time from when a user submits a request until the user receives the complete response. We use two common real-world applications to illustrate how scheduling can greatly improve user experienced system performance, without requiring major changes to system hardware or software. We will begin with a study of SRPT scheduling for Web servers (Section 2). We will then move on to LAS scheduling for network routers (Section 3). Finally, we will discuss the impact of the system model (open vs closed) on the effectiveness of scheduling (Section 4).

2. SCHEDULING IN WEB SERVERS

2.1 How favoring short requests can help all

Much of the recently renewed interest in scheduling has been sparked by the following question, which was originally posed for Web servers serving static (GET file) requests: *"Is it possible to reduce the expected response time of* every *HTTP request at a Web server, simply by changing the order in which we schedule the requests?"*

Surprisingly, it turns out that the answer is yes. The idea is to replace the fair time-sharing scheduling policy used by traditional web servers with a different policy SRPT (Shortest Remaining Processing Time), which gives preference to requests for small files or requests with short remaining file size. In scheduling theory it has been known for a long time that scheduling jobs in the order of Shortest-Remaining-Processing-Time (SRPT) is optimal. However, SRPT hasn't been used in practice for fear of starvation: when analyzed under the M/M/1 queue, SRPT significantly penalizes long jobs.

Recent implementation work as well as new analytical results show that this fear is unfounded for many practical applications. Figure 1 shows results from an implementation study [17], which implements SRPT scheduling in an Apache web server and evaluates its performance under a trace-based workload. The implementation is done at the kernel level and involves controlling the order in which socket buffers are drained into the network, such that priority is given to connections with few remaining bytes to be sent. The SRPT server is compared to a standard (non-modified) Apache server, which fairly time-shares between all connections. We will therefore also refer to the standard server as a FAIR server.

As shown in Figure 1 (left), the mean response time of the new SRPT web server is significantly lower than those of the standard FAIR web server. For high loads, the SRPT server improves mean response times by nearly a factor of ten compared to the FAIR server. Most importantly, these improvements in mean response time do not come at the expense of hurting requests for large files. Figure 1 (right) shows the response time under the SRPT and the FAIR server as a function of the request size (percentile of the request size distribution). The graph shows that 99% of all requests benefit from SRPT scheduling, while requests for the largest 1% of all files are hardly penalized. The mean response time of the largest 1% of requests is nearly identical under FAIR and SRPT scheduling and requests for the very largest file are only slightly penalized (5% larger response time under SRPT compared to FAIR).

The explanation for this counter-intuitive result lies in the statistical distribution of web file sizes. While the traditional M/M/1 queueing model assumes an exponential job size distribution, web file sizes have been shown to exhibit highly variable distributions with heavy-tails. It turns out that for those highly variable distributions size-based scheduling that favors short jobs does not unfairly

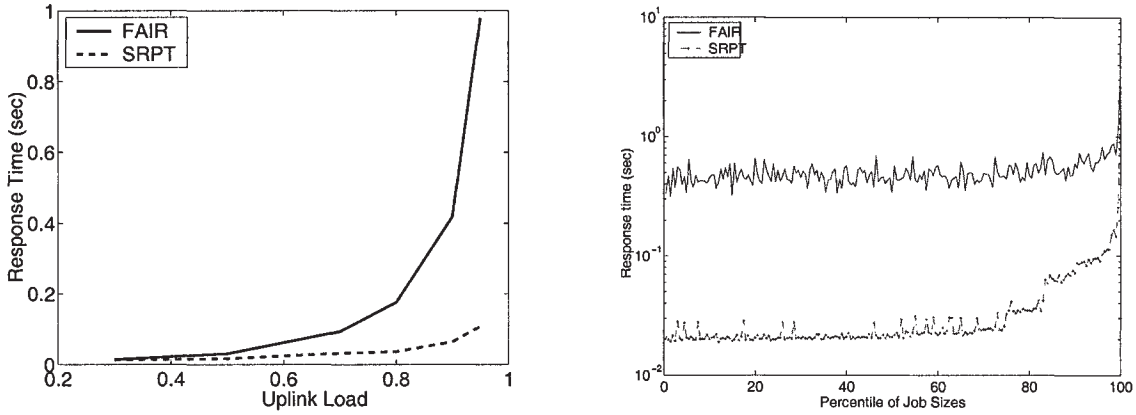


Figure 1: ***(Left) Shows the improvement in mean response time of SRPT scheduling over FAIR scheduling for static requests at a web server in a LAN setting. (Right) Shows mean response time as a function of the size of the requested file, where system load is fixed at*** $\rho = 0.8$.

penalize long jobs. A number of very recent theoretical papers formalize and generalize this result [4, 32, 44, 43, 45]. For example, the work in [4] shows that in many situations even the very largest job might prefer an SRPT system over a fair time-sharing system, depending on the job size distribution and system load. The work in [43] provides a scheme for classifying general scheduling policies with respect to their fairness when compared with processor sharing.

These new experimental and analytical results have motivated a number of generalizations and extensions on the idea of SRPT scheduling for web servers [9, 14, 21, 24, 31, 46, 47]. Some of this work [21, 24, 31], for example, proposes taking other request attributes in addition to file size into account, when making scheduling decisions. Others propose new hybrid policies, that combine features of the standard FAIR scheduler and the new SRPT scheduler [14].

2.2 Scheduling under overload

Interestingly, scheduling can also help with another key challenge that web sites face: *transient periods of overload.* Web traffic is known to be bursty and hard to predict, and hence even well-provisioned servers can experience transient periods of overload. During overload the number of connections at a server grows rapidly, leading to long response times and eventually rejected connections. The key idea behind scheduling for improving overload performance is that a traditional server, by time-sharing among all requests, is slowing down every request in the system, causing a large connection buildup. On the other hand, SRPT-based scheduling minimizes the number of connections at a server by always working on the connection with the smallest amount of work left.

The work in [35] demonstrates in extensive experiments that SRPT-based scheduling significantly improves both server stability and client experience during transient overload conditions. Figure 2 shows one of the results from this work. Both a traditional FAIR server and an SRPT server are run under a time-varying, trace-based workload, which alternates between periods of overload and periods of low load. Figure 2 (left) shows the mean response times observed under the FAIR system, while Figure 2 (right) shows the dramatically improved mean response times under SRPT scheduling. Note the improvement is close to an order of magnitude. Again, counter to intuition, the improvements in mean response times do not come at the expense of hurting the requests for large files, due to the heavy-tailed nature of web file size distributions.

2.3 Scheduling dynamic web requests

The previous two subsections have focused entirely on static web requests. Ideally, one would like to apply the same techniques to improve user-perceived performance for dynamic web requests, that is web requests whose responses are created on the fly. Since a common bottleneck in processing dynamic web requests is the database backend, applying the size-based scheduling idea to dynamic requests means scheduling database transactions to give priority to short transactions.

Unfortunately, implementing size-based scheduling for database transactions is more complicated than for static web requests for two reasons. First, giving priority to short transactions, in the spirit of SRPT, requires a way of knowing the length of a transaction before actually running it, which is a hard problem. We will see in Section 3 that there is a scheduling policy that favors short jobs without prior knowledge of the job size.

A second problem in scheduling database transactions is that existing database management systems do not support effective transaction prioritization for web-based transactional workloads. While existing prioritization tools are based on CPU scheduling, web-based database workloads are often lock-bound, hence requiring lock scheduling for effective transaction prioritization [22]. To overcome this problem, recent work has implemented and evaluated a number of different lock scheduling policies [23] for web-driven workloads. The authors find that a new lock scheduling policy POW (Preempt-on-Wait) can significantly improve the performance of high priority transactions without overly penalizing low priority transactions. For example, in systems where 10% of the transactions are high priority and the remaining 90% low priority, POW improves the mean response time of high priority transactions by a factor of 8, while increasing the mean response time of low priority transactions by less than 10%.

Other recent work proposes and implements a methods for scheduling transactions *outside* the database system, without directly controlling database internal resources [36, 37]. Surprisingly, the results from this work show that external transaction scheduling, when done right, can be as effective as internal (lock) scheduling, without introducing any negative side-effects, such as reduced system

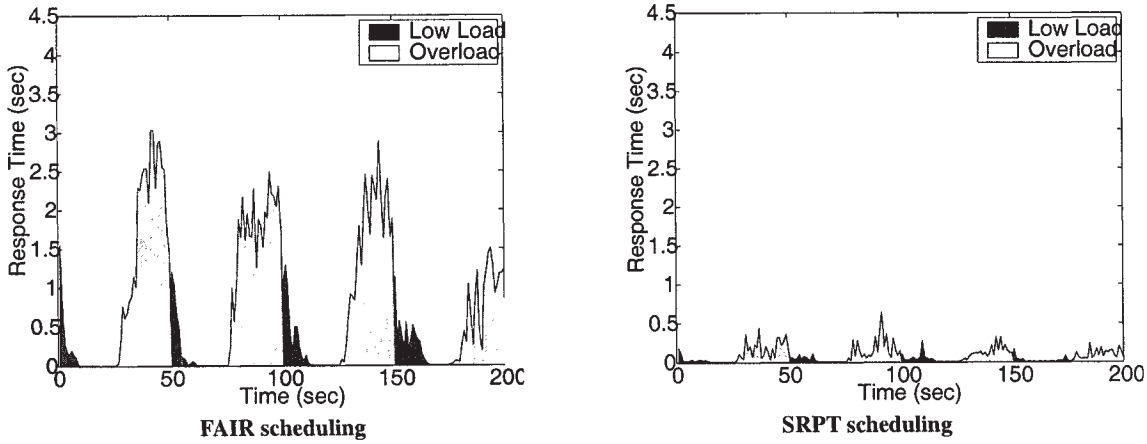


Figure 2: ***(Left) Mean response time under FAIR scheduling under transient overload. (Right) Mean response time under SRPT scheduling under transient overload. Load ρ alternates between 1.2 and 0.2, averaging at $\rho = 0.7$.***

throughput or increased overall response times.

3. SCHEDULING IN ROUTERS

3.1 Introduction

While SRPT scheduling is highly effective in improving mean response times in many systems, it is not applicable in routers since the "job size" is not known a priori. This section presents an alternative policy, LAS (Least-Attained-Service), for use in systems where size estimates are not available. LAS aims at mimicking SRPT by guessing the remaining service time of a job based on the service it has received so far. Below we provide some background on LAS and describe its application in traditional packet-switched networks as well as in wireless networks.

3.2 Background on LAS

LAS was proposed in the mid 1960s for time sharing computers to favor short interactive jobs at the expense of batch jobs. LAS is a preemptive scheduling policy that requires no prior knowledge of the job sizes. LAS gives service to the job in the system that has received the least amount of service. In the event of ties, the jobs having received the least service share the processor in a processor-sharing mode. A newly arriving job always preempts the job currently in service and retains the processor until it departs, or until the next arrival occurs, or until it has obtained an amount of service equal to that received by the job preempted on arrival, whichever occurs first.

First analytical results on the LAS queue (mean and Laplace transform of the response time), were obtained by Schrage [34] and Kleinrock [19]. Note that LAS is known under different names, such as Foreground Background (FB) or Shortest Elapsed Time (SET). Also, different flavors of the policy exist, depending on the number of priority queues, the use of preemption and on whether the minimum amount of service (service quantum) is null or strictly positive.

LAS belongs to the family of blind policies that do not know the job size in advance, in contrast to other size-based scheduling policies, like SRPT (Shortest Remaining Processing Time) that gives priority to the job that is the closest to completion. The common belief about LAS and other size-based scheduling policies favoring short jobs is that they might lead to the starvation of the longest jobs. However, in the context of Internet traffic where the traffic usually consists of a mix of a lot of small flows and a few large flows that account for a significant fraction of the mass, it has been observed that size-based policies can avoid the starvation of the longest flows.

In the remainder of this section, we briefly review theoretical results that back up those observations. Due to space constraints, we present only a few results and refer to the recent survey by Nuyens and Wierman [25] and the references therein for further details. Despite the existence of theoretical results on the LAS queue, networking researchers often rely on simulations to study these policies for the simple reason that a job as used in queuing theory is not a good model for a flow in the networking context. In the queuing theory, the term *job* defines an amount of work that arrives to the system all *at once*. Therefore, a *flow* of packets cannot be considered as a job since a flow does not arrive at a router at once. Instead, the source transmits a flow as a sequence of packets, possibly spaced out in time, that are in turn statistically multiplexed with packets from other flows. However, we will see in Section 3.5 that in the specific case of LAS, job level results and flow level results can agree.

3.3 Notation

Notation used in the rest of this paper is summarized in Table 1. Note that the set of IMRL (Increasing Mean Residual Life) distributions encompasses the set of DFR (Decreasing Failure Rate) distributions. All IMRL (and thus DFR) distributions have a CoV greater than 1.

3.4 Job level results for LAS

We present in this section, results for the mean response time and the conditional mean response time of an LAS queue. Unless otherwise stated, all results apply to an M/G/1/LAS queue.

3.4.1 Mean Response Times

LAS tries to mimic the SRPT policy by guessing the remaining service time of a job based on the service it has received so far. The reason behind this strategy is that SRPT is optimal (for a G/GI/1/LAS queue) in the sense that it minimizes the mean response time and the mean queue length among all policies. If LAS correctly guesses the remaining service time, LAS should thus be optimal among all blind policies. Several results have been proven along this line. It has been shown that LAS is optimal among all

X	Service requirement with mean $E[X]$ and variance $var(X)$
f	Density of X. We assume that f is continuous and with finite first and second moments.
F	Cumulative distribution function of X
λ	Arrival Rate of clients to the queue
$\rho \triangleq \lambda E[X]$	Input load to the queue
$\rho(x) \triangleq \lambda \int_0^x xf(x)dx$	Input load to the queue due the clients of size less or equal to x
$\rho_x \triangleq \rho(x) + \lambda x(1 - F(x))$	Input load to the queue due to clients of size less or equal to x and clients of size larger than x but truncated to x
T_p	Response time of policy p with mean $E[T_p]$
$E[T_p(x)]$	Mean conditional response time of a client of size x for policy p
$E[S_p(x)] \triangleq \frac{E[T_p(x)]}{x}$	Mean conditional slowdown of a client of size x for policy p
$CoV(X) \triangleq \sqrt{var(X)}/E[X]$	Coefficient of variation of X
DFR	Decreasing Failure Rate. For a DFR distribution, $\frac{f(x)}{1-F(x)}$ is a decreasing function of x
IMRL	Increasing Mean Residual Life. For a IMRL distribution, $E[X - x\|X \geq x]$ is an increasing function of x

Table 1: Notation

blind policies for the case of DFR service time distributions. However, LAS is not necessarily optimal for the class of IMRL service time distributions [1], or for all service time distributions with $CoV(X) > 1$.

Concerning the comparison between LAS and PS, Coffman and Denning ([8], page 189) conjectured that for service time distributions with $CoV(X) > 1$ the mean response time for LAS is lower than that of PS. Recently, Wierman et al. [42] have shown that this is not the case. However, for the more restrictive class of IMRL service time distributions, LAS achieves a smaller mean response time than PS [1].

More generally, an upper and a lower bound on the mean response time of LAS have been obtained in [25]:

THEOREM 1. $\frac{E[X]}{\rho} \leq E[T_{LAS}] \leq E[X]\frac{1-\rho/2}{(1-\rho)^2}$

The deterministic distribution constitutes a worst case where the upper bound on the mean response time of LAS is attained, while a number of distributions (of practical interest) lead to the lower bound.

A number of studies have also been conducted on the impact of the load on the performance of LAS. When the load reaches 1, both LAS and PS see their mean response time growing to infinity. However, the ratio of the average response time of PS to LAS grows unbounded, which means that the PS queue builds up faster than the LAS queue [25].

3.4.2 *Conditional Mean Response Times*

Let us first consider the mean conditional response time $E[T_{LAS}(x)]$. The extent to which the mean conditional response time of LAS is close to the corresponding value under SRPT depends both on the distribution and the load of the input traffic [29]:

THEOREM 2. *For all job sizes x and at load $\rho < 1$:*
$E[T_{SRPT}(x)] \leq E[T_{LAS}(x)] \leq \left(\frac{1-\rho(x)}{1-\rho_x}\right)^2 E[T_{SRPT}(x)]$

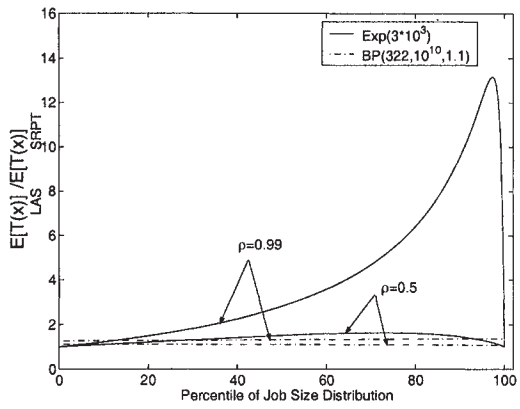


Figure 3: $\frac{E[T_{LAS}(x)]}{E[T_{SRPT}(x)]}$ as a function of the job size percentile

Figure 3 illustrates the previous theorem for an exponential distribution with a mean job size of 3000 units and a bounded Pareto distribution with an minimum value of 322 units, a maximum value of 10^{10} units and a shape parameter α of 1.1. The mean of the bounded Pareto distribution is also 3000 units. The CoV of the exponential distribution is 1 and the CoV of the bounded Pareto distribution is 284. One can see from Figure 3 that for a distribution with a large CoV like the Bounded Pareto distribution, the mean conditional response time of LAS is very close to the one of SRPT for all job sizes, almost irrespectively of the load. This is not the case for the exponential distribution where good agreement exists only for small to moderate load values.

The number of small jobs that experience a smaller response time under LAS than under PS has been studied in [43, 29, 6]. Combining the results in [43, 29, 6], one obtains:

THEOREM 3. *In an M/G/1 queue,*
$E[T_{LAS}(x)] \leq \frac{1-\rho/2}{1-\rho} E[T_{PS}(x)] \quad \forall x$
$E[T_{LAS}(x)] \leq E[T_{PS}(x)] \quad \forall x \text{ s. t. } \rho_x \leq \max\{\frac{\rho}{(1+\sqrt{1-\rho})}, \frac{2}{3}\rho\}.$

Brown [6] has further proven that for Pareto distributions with infinite second moment, the conditional mean response time for all job sizes is smaller under LAS than under PS.

In case of overload, some jobs still experience a finite service time under LAS [29]. This is in contrast with FIFO and PS where the conditional response time of all jobs goes to infinity.

THEOREM 4. *When $\rho > 1$, all jobs of size $x < x_\rho$, such that $\rho_{x_\rho} = 1$ have a finite response under LAS.*

Apart from response time, fairness has also received significant attention. The metric used to evaluate fairness is the mean conditional slowdown $E[S(x)]$. A policy is said to be fair if its mean conditional slowdown for all job sizes is smaller than the corresponding metric under PS. Wierman and Harchol-Balter [43] proposed a framework to classify scheduling policies with respect to their fairness. A striking result is that fairness of the maximum job size is the same for both LAS and PS. However, when the flow size distribution has a finite second moment, some large jobs (though not the largest job) always experience a slowdown greater under LAS than under PS [43].

3.5 Flow level results

While queuing theory offers an ideal framework to compare different scheduling strategies, the results obtained are not directly applicable to packet networks as we have already discussed above. However, in [30], we demonstrated that a router implementing LAS can be modeled by an M/G/1/LAS queue, provided a low packet loss rate (less than 2%), homogeneous round trip times, and identical packet sizes. The intuition behind this result is that since LAS is a priority scheduling policy, the behavior of an LAS router is similar to that of an LAS queue as long as the packet to be serviced next arrives at the router before the time instant where it is selected by the scheduler. As a FIFO router can be modeled as an M/G/1/PS queue, we can use the results presented in the previous section on the comparison between LAS and PS at job level to compare LAS and FIFO scheduling in a router.

3.6 LAS and TCP

LAS scheduling in packet switched networks such as the Internet can be applied to flows. A flow is defined as a group of packets with a common set of attributes such as (source address, destination address, source port, destination port). For each flow F, the amount of bytes S_F served so far needs to be tracked in order to compute the service priority of the next packet of flow F. S_F is initialized to 0 when the flow starts and incremented by P when a packet of size P belonging to flow F arrives. In the "basic" version of LAS, the *service priority* of a flow is given by S_F. The flow that has sent the least amount of bytes will have the highest service priority. Recall that the lower the value of the service priority, the higher the priority, and the closer to the head of the queue the packet is inserted.

We have already seen that there are some differences between LAS applied to jobs as compared to flows and we want to emphasize one more point. Routers have a finite buffer space and the implementation of LAS in routers must extended to not only decide in which order to serve packets but also which packet to drop when the buffer is full. When a packet arrives to a queue that is full, LAS computes the service priority of that packet, inserts the packet at its appropriate position in the queue, and then drops the packet that is at *the end of the queue*.

LAS for scheduling packets is attractive if the following conditions hold:

- The load of the link over which LAS schedules the transmission of packets is sufficiently high. Under low and moderate load ($\rho < 0.7$) there is not much difference in terms of conditional mean service times between the various scheduling policies such as LAS, PS or FIFO. It will be sufficient to deploy LAS scheduling in the routers serving congested links to achieve a significant reduction in the end-to-end response time for most flows. The links that are congested (highly loaded) are often located at the edge of the network. One place where the deployment of LAS can make a difference is for scheduling the transmission of packets over the broadband access link of home users.

- The flow size distribution generally exhibits a high coefficient of variation[1]. Internet traffic exhibits a high coefficient of variation since most of the flows are short, while more than 50% of the bytes are carried by less than 5% of the largest flows [26, 7].

Due to the closed-loop nature of TCP, the treatment given by LAS to an individual packet is likely to affect when TCP transmits the following packets of that flow. In fact, LAS interacts very favorably with the congestion control and error control of TCP and accelerates the transfer of most flows:

- Congestion Control: A new TCP flow starts in what is known as "slow start", where its congestion window is initialized to a very small value and then doubled after every round of transmission. The duration of a round is essentially determined by the time between the transmission of the packet and the reception of the acknowledgment, and comprises packet transmission time, propagation delay, and queuing delay. Since under LAS the first packets of a flow will experience no or negligible queuing delay, the duration of a round will be shorter under LAS than under FIFO and the congestion window will increase faster.

- Error Control: LAS gives buffer space priority to the first packets of each flow, which means that these packets should not experience any loss. Avoiding loss is very important since any packet loss may temporarily prevent the sender from emitting new packets until the loss has been repaired. Loss repair for the first packets of a TCP flow typically occurs after a timeout and requires much more time as compared to loss repair using duplicate ACKs (see [28]).

We have implemented LAS under Linux and are able to handle data rates of several tens of Mbit/s without any special tuning. The Linux kernel contains a complete architecture for support of QoS with components such as flow classification, packet dropping, metering, marking, and shaping, which makes it easy to experiment with new scheduling policies. The implementation of LAS keeps track for each flow of how many bytes have been served, computes the service priority to obtain the priority value for each packet and, most importantly implements a priority queue that inserts an arriving packet according to its priority value.

3.7 Performance of LAS vs FIFO in a single bottleneck case

We present the result of one experiment where LAS is used to schedule transmission over a bottleneck link of 320 Kbit/s that experiences an average offered load of 0.91. The round trip time is 40 ms and a total of 500,000 flows have been transmitted. The flow sizes are drawn from a bounded Pareto Distribution with minimum flow size of 1.5 KByte and maximum flow size of 233 MByte and a coefficient of variation of 33.41. For more results see [33]. The comparison of LAS and FIFO in Figure 4 shows that LAS reduces the mean response time of most of the TCP flows by an order of magnitude and more.

However, during our experiments we also discovered that LAS can "starve" large TCP flows more frequently than FIFO. This problem is due to the fact that some long TCP flows get preempted for a very long period of time by flows that have received less service. If a TCP flow gets preempted and some of its packets are stuck in the queue of the bottleneck link, the sender will timeout after a time RTO, and resend these packets. At each retransmission, the sender doubles its RTO, adopting an exponential back-off strategy. After 15 attempts, the TCP connection is aborted. The time until abort corresponds on average to $RTO*(\frac{1}{2}\sum_{i=1}^{15} 2^i) = RTO*(2^{15}-1)$. Since the minimal initial RTO value in TCP ranges from 200 ms to

[1]Although we have seen before that a high CoV may not be sufficient to obtain good performance, theoretical distributions used to model user traffic such as the Bounded Pareto distribution, have been observed to be very effective in reducing the response time of most flows when LAS is used as compared to FIFO (see section 3.2 of [29]).

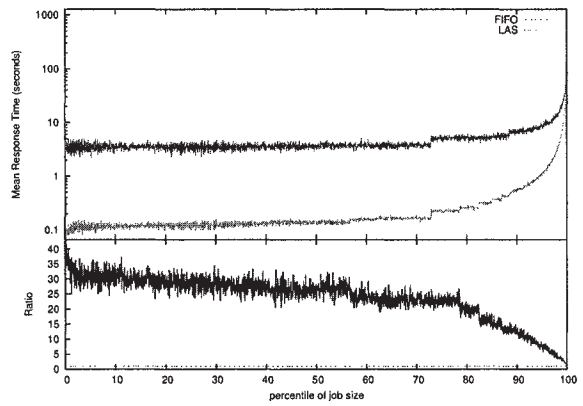


Figure 4: Conditional mean response time of FIFO and LAS and their ratio.

1 sec (see RFC 2988), it will take at least 90 minutes before a connection is aborted.

It should be clear from this example that only under very rare circumstances a TCP flow will be aborted and the higher the data rate of the bottleneck or the larger the RTT, the less likely the event. In any case, there is a very simple fix against starvation of TCP connections: Limit the duration during which the transmission of packets for a flow can be suspended. We can modify LAS to obtain *LAS-threshold* where the service priority assigned to any packet cannot take values larger than *threshold*. Such a scheme is discussed by Feng and Misra under the name of FLIPS [10] and as LAS-FCFS in Rai's thesis (section 6.6. of [27]) and has been implemented and evaluated in [33].

Under *LAS-threshold*, the packets with service priority values less than *threshold* will be served according to LAS, the remaining packets are handled in a "background" FIFO queue, reducing the duration of a flow preemption. Finding the value for *threshold* is not too difficult since the time before a TCP connection is aborted is so large. In our implementation we saw that 300 KByte was a good value for *threshold* [33].

In fact, *LAS-threshold* is just one possible modification to LAS and several other modifications are possible that allow to take into account the needs of applications such as IP telephony or audio/video streaming. There are for instance *LAS-fixed* where all packets of a flow have the same fixed service priority value and *LAS-log* where the service priority grows with the logarithm of the number of bytes served, i.e. much slower than under normal LAS where it grows linearly. For more details, analytic results and also on how the different LAS policies interact see [30].

3.8 LAS in wireless networks

In this section, we discuss the use of LAS in a wireless context, with a focus on 802.11 networks. Pure ad-hoc networks or infrastructure networks suffer from two issues that LAS could alleviate. First, the relatively low bandwidth available in those environments, as compared to wired networks, as well as the bandwidth variability tend to create bottlenecks. Second, TCP suffers from poor performance and unfairness in wireless networks [18, 12].

In [11], the use of LAS in static 802.11 networks is investigated. Small multi-hop ad-hoc networks with a chain or grid topology are considered along with a single hop infrastructure network. We demonstrate here the benefits of LAS by reporting the results obtained for a chain topology. Let us consider a chain topology with four nodes where three long-lived FTP connections are established between a node at one end of the chain and each other node in the chain. Let us first assume that there is no hidden node. In this scenario, TCP suffers from unfairness as the throughput decreases with the number of hops in the path [16]. Simulation results reported in [11] show that LAS enforces fairness between the FTP connections. These results highlight that the bad performances of TCP in wireless networks are not solely due to the TCP algorithms but also to the interaction between TCP and FIFO.

Note that some transient starvations are observed when the three FTP connections do not start simultaneously, as has been observed in the previous section for a wired setting. However, the same type of argument applies here which is that in a realistic case with flows of different sizes (and not long-lived flows), these starvations should not last long enough to break the underlying TCP connections.

The case of a chain topology with one hidden node [13] (the last FTP receiver) is also addressed in [11]. It turns out that LAS can partly enforce fairness in this case as compared to FIFO where the last FTP receiver completely starves. The intuition behind this result is that the last FTP connection, which is penalized by the presence of the hidden node in its path (the last node in the chain), is granted a higher priority under LAS at the other (non hidden) nodes of the path. Thus, the last FTP connection has a chance to catch up with the other ones.

Though preliminary, those results highlight the benefits brought by the use of LAS in wireless networks.

3.9 Related work

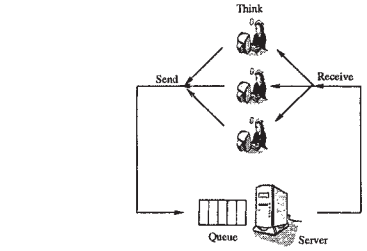
There are several papers [15, 2] that try to improve the performance of short flows by using two FIFO queues. The first K packets of a flow are enqueued in the first queue and the remaining ones of a flow are enqueued in the second queue. Each queue is served in FIFO order and packets from the second queue are only served if the first queue is empty. Two-queue based disciplines reduce the mean transfer times of the short flows compared to FIFO scheduling, however not to the extent that LAS is able to do.

4. THE EFFECT OF CLOSED VERSUS OPEN SYSTEM MODELS ON SCHEDULING

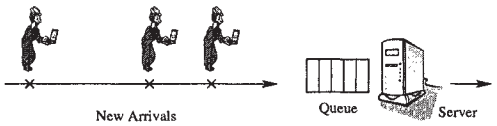
Most researchers are well aware of the fact that the performance of scheduling policies is greatly influenced by factors such as the job size distribution and the system load. However, little attention has been paid to the impact of whether the system under study follows a *closed* versus an *open* system model.

Figure 5(a) depicts a **closed system** configuration. In a closed system model, it is assumed that there is some fixed finite number of users, who use the system forever. This number of users is typically called the *multiprogramming level* (MPL). Each of the users repeats these 2 steps, indefinitely: (a) submit a job, (b) receive the response and then "think" for some amount of time. In a closed system, *a new request is only triggered by the completion of a previous request.*

Figure 5(b) depicts an **open system** configuration. In an open system model, each user is assumed to submit one job to the system, wait to receive the response, and then leave. The number of users queued or running at the system at any time may range from zero to infinity. The differentiating feature of an open system is that a *request completion does not trigger a new request: a new request is only triggered by a new user arrival.* The arrival times of users could for example be generated from a Poisson process or taken from a trace.



(a) Closed system



(b) Open system

Figure 5: Illustrations of the closed and open system model.

While nearly all queuing theoretic results on the performance of scheduling policies assume an open system model, in experimental work the system model is implicitly dictated by the workload generator used. In the case of web workloads, for example, there are popular workload generators that are based on a closed system model [5, 40, 41], as well as generators that follow an open system model [3, 20, 39].

Recent work [38] shows that the choice between an open and a closed system model can greatly impact results, in particular when it comes to the effectiveness of scheduling. In [38] implementation and simulation experiments together with theoretic support are used in order to identify a set of basic principles that capture the differences in behavior of closed, open, and a hybrid partly-open system models.

One key principle derived in [38] is that while open systems benefit significantly from scheduling with respect to response time, closed systems improve much less. Figure 6(a) shows the response time as a function of load for four different scheduling policies simulated in an open and a closed system. The four scheduling policies include two common non-size-based policies, First-Come-First-Serve (FCFS) and Processor-Sharing (PS), and two size-based policies, Preemptive-Shortest-Job-First (PSJF) and Preemptive-Longest-Job-First (PLJF). Under the open system, the disparity in performance between the scheduling policies grows to up to an order of magnitude for high loads. On the other hand under the closed system performance differences between the policies are comparatively moderate.

The above differences between open and closed system models are shown to be present across a range of applications, including web servers and database systems, and motivate the need for system designers to be able to determine how to choose if an open or closed model is more appropriate for the system they are targeting. The work in [38] therefore also presents a method that allows designers to determine whether a given system is better represented by an open or closed system model. The authors apply this method to the Web logs of more than ten large sites, and find that there is no "one fits all" solution: some sites are best modeled by an open system and others by a closed system, while some require a hybrid model.

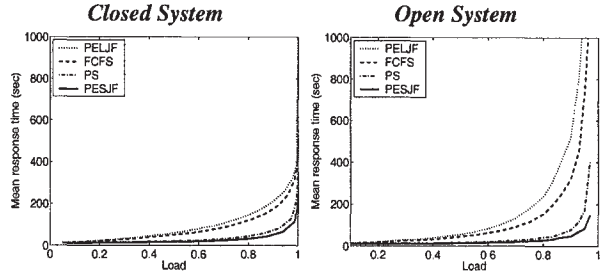


Figure 6: ***Model-based simulation results illustrating the different effects of scheduling in closed and open systems. In the closed system the MPL is 100, and in both systems the service demand distribution has mean 10 and a squared coefficient of variation of 8.***

5. CONCLUSIONS

In this paper, we have demonstrated through two real world examples that scheduling is a very powerful tool to improve user-perceived system performance.

We started by considering a Web server serving static requests. We showed that changing the connection scheduling from the traditional time-sharing scheduling to SRPT-based scheduling can greatly reduce mean response times at a busy web server. We also showed that this change in the connection scheduling significantly improves both server stability and client experience during transient overload conditions. Most importantly, these improvements do not come at the expense of hurting requests for large files.

In the second example, we have discussed LAS as a scheduling policy in routers. While LAS requires a small amount of per flow state in the routers, our experience has shown that LAS is easy to implement and interacts very nicely with TCP. LAS can be incrementally deployed and when used to schedule packet transmission over broadband access lines or wireless links, the user-perceived performance will be significantly improved. Also, its different priority functions offer a wide variety of possibilities for service differentiation.

6. REFERENCES

[1] S. Aalto and U. Ayesta. On the non-optimality of the fb discipline for imrl service times. *Journal of Applied Probability*, pages 523–534, 2006.

[2] K. Avrachenkov, U. Ayesta, P. Brown, and N. Nyberg. Differentiation between short and long tcp flows: Predictability of the response time. In *Proc. IEEE INFOCOM*, March 2004.

[3] G. Banga and P. Druschel. Measuring the capacity of a web server under realistic loads. *World Wide Web*, 2(1-2):69–83, 1999.

[4] N. Bansal and M. Harchol-Balter. Analysis of SRPT scheduling: Investigating unfairness. In *Proc. of ACM SIGMETRICS*, 2001.

[5] P. Barford and M. Crovella. The surge traffic generator: Generating representative web workloads for network and server performance evaluation. In *In Proc. of the ACM SIGMETRICS*, 1998.

[6] P. Brown. Comparing FB and PS scheduling policies. *SIGMETRICS Perform. Eval. Rev.*, 34(3):18–20, 2006.

[7] K. Claffy, G. Miller, and K. Thompson. The nature of the beast: Recent traffic measurements from an internet

backbone. In *Proceedings of INET*, July 1998.

[8] E. G. Coffman and P. J. Denning. *Operating Systems Theory*. Prentice-Hall Inc., 1973.

[9] S. Deb, A. Ganesh, and P. Key. Resource allocation between persistent and transient flows. *IEEE/ACM Trans. Netw.*, 13(2), 2005.

[10] H. Feng and M. Misra. Mixed scheduling disciplines for network flows. In *The Fifth Workshop of Mathematical Performance Modeling and Analysis (MAMA 2003), San Diego, California, USA*, June 2003.

[11] D. Ferrero and G. Urvoy-Keller. Size-based scheduling to improve fairness and performance in 802.11 networks. Technical Report RR 2125, Institut Eurecom, France, Dec. 2006.

[12] M. Franceschinis, M. Mellia, M. Meo, and M. Munaf. Measuring TCP over wifi: A real case. In *1st workshop on Wireless Network Measurements (Winmee)*, Riva Del Garda, Italy, 2005.

[13] Z. Fu, H. Luo, P. Zerfos, S. Lu, L. Zhang, and M. Gerla. The impact of multihop wireless channel on TCP performance. *Mobile Computing, IEEE Transactions on*, 4(2):209–221, 2005.

[14] M. Gong and C. Williamson. Simulation evaluation of hybrid SRPT scheduling policies. In *Proc. of IEEE MASCOTS*, 2004.

[15] L. Guo and I. Matta. Differentiated control of web traffic: A numerical analysis. In *SPIE ITCOM'2002: Scalability and Traffic Control in IP Networks*, August 2002.

[16] P. Gupta and P. R. Kumar. The capacity of wireless networks. *Information Theory, IEEE Transactions on*, 46(2):388–404, 2000.

[17] M. Harchol-Balter, B. Schroeder, N. Bansal, and M. Agrawal. Size-based scheduling to improve web performance. *ACM Transactions on Computer Systems (TOCS)*, 21(2), 2003.

[18] V. Kawadia and P. R. Kumar. Experimental investigations into TCP performance over wireless multihop networks. In *E-WIND'05*, pages 29–34, 2005.

[19] L. Kleinrock. *Queuing Systems, Volume II: Computer Applications*. Wiley, New York, 1976.

[20] Z. Liu, N. Niclausse, and C. Jalpa-Villanueva. Traffic model and performance evaluation of web servers. *Performance Evaluation*, 46(2-3):77–100, 2001.

[21] D. Lu, H. Sheng, and P. A. Dinda. Size-based scheduling policies with inaccurate scheduling information. In *Proc. of IEEE MASCOTS*, 2004.

[22] D. T. McWherter, B. Schroeder, A. Ailamaki, and M. Harchol-Balter. Priority mechanisms for OLTP and transactional web applications. In *Proc. of IEEE ICDE*, 2004.

[23] D. T. McWherter, B. Schroeder, A. Ailamaki, and M. Harchol-Balter. Improving preemptive prioritization via statistical characterization of OLTP locking. In *Proc. of IEEE ICDE*, 2005.

[24] C. D. Murta and T. P. Corlassoli. Fastest connection first: A new scheduling policy for web servers. In *Proc. of the 18th International Teletraffic Congress (ITC-18)*, 2003.

[25] M. Nuyens and A. Wierman. The foreground-background queue: a survey. Technical report, under submission, 2006.

[26] V. Paxson and S. Floyd. Wide-area traffic: The failure of poisson modelling. *IEEE/ACM Trans. Netw.*, 3:226–244, June 1995.

[27] I. Rai. *QOS Support in Edge Routers*. PhD thesis, Télécom Paris, Institut Eurecom, France, Sept. 2004.

[28] I. Rai, E. W. Biersack, and G. Urvoy-Keller. Size-based scheduling to improve the performance of short TCP flows. *IEEE Network Magazine*, 19(1):12–17, January-February 2005.

[29] I. A. Rai, G. Urvoy-Keller, and E. W. Biersack. Analysis of las scheduling for job size distributions with high variance. In *Proc. ACM SIGMETRICS*, pages 218–228, June 2003.

[30] I. A. Rai, G. Urvoy-Keller, M. Vernon, and E. W. Biersack. Performance models for las-based scheduling disciplines in a packet switched network. In *ACM SIGMETRICS*, June 2004.

[31] M. Rawat and A. Kshemkayani. SWIFT: Scheduling in web servers for fast response time. In *Second IEEE International Symp. on Network Comp. and App.*, 2003.

[32] D. Raz, H. Levy, and B. Avi-Itzhak. A resource-allocation queueing fairness measure. In *Proc. of ACM SIGMETRICS*, 2004.

[33] A. Salvatori. LAS scheduler implementation and performance analysis, institut eurecom, Feb. 2005.

[34] L. E. Schrage. The queue m/g/1 with feedback to lower priority queues. *Management Science*, 13(7):466–474, 1967.

[35] B. Schroeder and M. Harchol-Balter. Web servers under overload: How scheduling can help. *ACM Transactions on Internet Technology (TOIT)*, 6(1):20–52, 2006.

[36] B. Schroeder, M. Harchol-Balter, A. Iyengar, and E. Nahum. Achieving class-based QoS for transactional workloads. In *Proc. of IEEE ICDE*, 2006.

[37] B. Schroeder, M. Harchol-Balter, A. Iyengar, and E. Nahum. How to determine a good multi-programming level for external scheduling. In *Proc. of IEEE ICDE*, 2006.

[38] B. Schroeder, A. Wierman, and M. Harchol-Balter. Open vs. closed systems: A cautionary tale. In *Proc. of Usenix NSDI*, 2006.

[39] Standard Performance Evaluation Corporation. SPECweb99. `http://www.specbench.org/osg/web99/`.

[40] Transaction Processing Performance Council. TPC benchmark W (web commerce). Number Revision 1.8, February 2002.

[41] G. Trent and M. Sake. WebStone: The first generation in HTTP server benchmarking. `http://www.mindcraft.com/webstone/paper.html`.

[42] A. Wierman, N. Bansal, and M. Harchol-Balter. A note on comparing response times in the M/GI/1/FB and M/GI/1/PS queues. *Operations Research Letters*, 32(1):73–76, Jan. 2004.

[43] A. Wierman and M. Harchol-Balter. Classifying scheduling policies with respect to unfairness in an M/GI/1. In *Proc. of ACM SIGMETRICS*, 2003.

[44] A. Wierman and M. Harchol-Balter. Classifying scheduling policies with respect to higher moments of conditional response time. In *Proc. of ACM Sigmetrics*, 2005.

[45] A. Wierman, M. Harchol-Balter, and T. Osogami. Nearly insensitive bounds on smart scheduling. In *Proc. of ACM SIGMETRICS*, 2005.

[46] S. J. Yang and G. de Veciana. Enhancing both network and user performance for networks supporting best effort traffic. *IEEE/ACM Trans. Netw.*, 12(2), 2004.

[47] J. Zhou and T. Yang. Selective early request termination for busy internet services. In *Proc. of WWW'06*, 2006.