Probability in the Engineering and Informational Sciences, **34**, 2020, 507–521.
doi:10.1017/S0269964819000160

SCHEDULING IN A SINGLE-SERVER QUEUE WITH STATE-DEPENDENT SERVICE RATES

URTZI AYESTA
CNRS, IRIT, 2 rue C. Camichel, 31071 Toulouse, France
LAAS-CNRS, Université de Toulouse, CNRS, INP, Toulouse, France
IKERBASQUE - Basque Foundation for Science, 48011 Bilbao, Spain
UPV/EHU, Univ. of the Basque Country, 20018, Donostia, Spain
E-mail: urtzi.ayesta@irit.fr

BALAKRISHNA PRABHU
LAAS-CNRS, Université de Toulouse, CNRS, INP, Toulouse, France

RHONDA RIGHTER
University of California, Berkeley, CA 94720, United States

We consider single-server scheduling to minimize holding costs where the capacity, or rate of service, depends on the number of jobs in the system, and job sizes become known upon arrival. In general, this is a hard problem, and counter-intuitive behavior can occur. For example, even with linear holding costs the optimal policy may be something other than SRPT or LRPT, it may idle, and it may depend on the arrival rate. We first establish an equivalence between our problem of deciding which jobs to serve when completed jobs immediately leave, and a problem in which we have the option to hold on to completed jobs and can choose when to release them, and in which we always serve jobs according to SRPT. We thus reduce the problem to determining the release times of completed jobs. For the clearing, or transient system, where all jobs are present at time 0, we give a complete characterization of the optimal policy and show that it is fully determined by the cost-to-capacity ratio. With arrivals, the problem is much more complicated, and we can obtain only partial results.

Keywords: operations research, queueing theory

1. INTRODUCTION

In most practical service systems, capacity varies with the number of customers, or jobs. In some cases, it deteriorates as the system becomes more congested. For example, human servers may experience extra overhead, in terms of distractions, crowd control, stress, and fatigue, etc., with more jobs. On the other hand, in many telecommunications and computer applications, such as wireless systems or peer-to-peer file sharing, capacity may increase with the number of jobs. Moreover, capacity changes may not be monotonic; for some applications capacity first increases and then decreases with congestion [8]. Increasing and convex

© Cambridge University Press 2019 0269-9648/19 $25.00

https://doi.org/10.1017/S0269964819000160 Published online by Cambridge University Press

capacities can also occur, for example, in device-to-device systems that use opportunistic scheduling, where jobs can serve each other, and speed may be related to proximity. These capacity dynamics complicate scheduling control. For example, idling may be optimal, and the optimal scheduling policy may depend on the arrival rate.

In our model, there is a single server, with speed that scales with the number of jobs. That is, work in the system can decrease at maximum rate $C(i) \geq 0$ when there are i jobs in the system. If $C(i) = I\{i > 0\}$, we recover the classical single-server queue. However, our model is different from a multi-server queue, even if $C(i) = \min\{i, s\}$ for some $s = 2, 3, \ldots$, because we allow all the capacity to be dedicated to a single job. In other words, if we had multiple servers, they would be allowed to collaborate on the same job. Preemption (and processor sharing) is permitted without penalty, and without increasing work. Job sizes are observed upon arrival. We also permit a general holding cost function, where the cost depends on the number of jobs present, but not on which jobs are present. We assume known job sizes (total service time for the job).

There are many results for the problem with constant capacity. For example it is well known that (non-idling) SRPT (shortest-remaining-processing-time first) stochastically maximizes the departure process for a single-server constant-capacity system with a general arrival process and general service times [14]. However, when we allow the capacity to depend on the number of jobs, the problem, with otherwise general assumptions, is hard. In the next section, we will give several examples that exhibit counter-intuitive behavior for the objective of minimizing average sojourn time.

We first show that our scheduling problem, with and without arrivals, can be transformed, by allowing the option of holding completed jobs, into a problem in which jobs are served according to SRPT and the problem is to determine when to release them. For example, LRPT is equivalent to SRPT when jobs are held until the last one completes service, at which time all jobs are released simultaneously. In the case when there are no arrivals, we obtain an algorithm that characterizes the optimal release times, which turns out to be fully determined by the cost-to-capacity ratio. In particular we show that if $C(i)$ is *star-shaped* (*anti-star-shaped*), defined later, then LRPT (SRPT) minimizes the total holding cost. We then consider the case with arrivals. We show that if the cost-to-capacity ratio is linear, then all non-idling policies yield the same average cost. We further characterize the optimal policy in some special cases. For example, as long as capacity is increasing, LRPT (holding jobs to the end of the busy period) stochastically minimizes the mean busy period length for stable systems.

The rest of the paper is organized as follows. In Section 2 we review the most relevant related literature. In Section 3 we introduce the model description and we present various examples that illustrate the complexity and richness of the problem. In Section 4 we show that by adding the option of holding jobs there exists an optimal policy within the class of policies that deploys SRPT and the decision is when to release completed jobs. Section 5 covers the case without arrivals, and Section 6 considers arrivals.

2. RELATED LITERATURE

The problem of scheduling jobs with a constant speed or capacity is a classical area in operations research. If the service times are known, Schrage [14] showed that SRPT minimizes *sample-path-wise* the number of jobs in the system. If only the distribution of the service time is known, then, depending on its properties, the optimal policy might be FCFS (First-Come-First-Served) or LAS (Least-Attained-Service), see [11,12]. In a multi-class setting, another classical result is that the so-called $c\mu$-rule minimizes the holding cost among

https://doi.org/10.1017/S0269964819000160 Published online by Cambridge University Press

preemptive (with exponential service times) and non-preemptive policies, see for example [6]. For a survey on scheduling results we refer to Weiss [17] and Righter [10].

If the capacity can vary, most of the literature on speed scaling takes the scheduling policy as given (generally SRPT – shortest remaining processing time first or PS – processor sharing) and dynamically adjusts the speed depending on congestion (number in system) to minimize the costs that depend on both response time and energy (e.g., [2,3,18]). We take the dynamic speed scaling, $C(i)$, as given, and find the optimal scheduling policy. A related research area, in which the scheduling policy affects the capacity, is in opportunistic scheduling in wireless networks. Here different flows or users face different capacities, and the overall capacity region depends on the present flows. Sadiq and de Veciana [13] consider a system that can be modeled with nested capacity regions, and where the total capacity depends on the allocation to flows. They give a characterization of the optimality of a variant of SRPT in their setting. Bekker et al. [4] study a queueing model in which service rates and arrival rates depend on the work in system rather than the number in system, but they do not consider optimal scheduling. Another relevant application domain in which our result could potentially have an impact is in coupled speed scaling systems, see [7,9]. In these systems, the speed of the CPU is adjusted dynamically based on the number of jobs present in the system. For instance, modern processors typically support over a dozen discrete operating speeds, often with a factor of two (or more) between the slowest and the fastest speeds available.

A few researchers have studied the problem in which the capacity depends on the number of jobs in the system. Timmermans and Vredeveld [16] consider weighted flowtime minimization for a model with a fixed set of initial jobs and no arrivals and job-dependent weights, and machine speed that depends on the number of remaining jobs, and show that the problem is NP-hard. Gupta and Harchol-Balter [8] investigate admission control policies when $C(i)$ is initially increasing, but starts falling beyond a certain point due to resource contention and thrashing. In recent work, Berg, Dorsman and Harchol-Balter [5] investigate how to split a multi-core processor among jobs, which in turn determines the speed profile.

3. MODEL DESCRIPTION AND MOTIVATING EXAMPLES

We assume that the capacity in the system is $C(j) \geq 0$ when there are j jobs in the system, and we let $g(j) \geq 0$ be the total holding cost rate when there are j jobs in the system.

In the case without arrivals we are given a fixed set of K jobs with known service requirements, or job sizes. We consider the set Π of scheduling policies that are size-aware, pre-emptive and possibly idling, and the objective is to schedule the jobs (to choose $\pi \in \Pi$) in order to minimize the total holding cost:

$$V^{\pi} := \int_0^{\infty} g(N^{\pi}(s))ds, \tag{1}$$

where $N^{\pi}(s)$ is the number of jobs in the system at time s under π. As particular cases, when $g(j) = j$, minimizing the total cost is equivalent to minimizing the flowtime, or mean sojourn time, and when $g(j) = I\{j > 0\}$, our objective becomes minimization of makespan, or time to complete all jobs.

In the case of arrivals, we will assume that new jobs arrive according to some stochastic process with finite rate. The objective is then to determine the scheduling policy $\pi \in \Pi$ that minimizes the long-run average holding cost:

$$\mathbb{E}[V^{\pi}] := \limsup_{T\to\infty} \frac{1}{T} \int_0^{T} g(N^{\pi}(s))ds. \tag{2}$$

https://doi.org/10.1017/S0269964819000160 Published online by Cambridge University Press

TABLE 1. Example 1. The Optimal Values are underlined

	Capacity function				Makespan		Mean soujourn time	
Profile	# of jobs	$C(1)$	$C(2)$	$C(3)$	SRPT	LRPT	SRPT	LRPT
(a)	2	1	3	–	7	<u>3</u>	4	<u>3</u>
(b)	2	1	1.5	–	8	<u>6</u>	<u>5</u>	6
(c)	2	1	0.5	—	<u>12</u>	18	<u>9</u>	18
(d)	3	1	3	4.3	7.23	<u>2.33</u>	8.70	6.99

When $g(j) = j$, minimizing average holding cost is equivalent to minimizing the average response time or sojourn time, and when $g(j) = I\{j > 0\}$, the objective is to minimize the average length of the busy period.

We now present a few examples to illustrate the complexity of the problem.

Example 1 (No arrivals): In the first example, there are no arrivals to the system except the jobs that are present at time 0. First, we shall compare SRPT and LRPT when there are only two jobs for the three speed profiles (a), (b), and (c) as given in Table 1. In profiles (a) and (b), $C(i)$ is increasing while in (c) it is decreasing. Assume that one job of size 3 and one of size 6 are present at time 0. Table 1 gives the makespan as well as the mean sojourn times for each of these profiles under SRPT and LRPT. The values in the table illustrate the fact that which of SPRT or LRPT is better depends not only on the objective and whether $C(i)$ is increasing, but also, if $C(i)$ is increasing, on how it is increasing. In Section 5, we shall show that the optimal policy depends on both $C(i)$ as well as upon $C(i)/i$. Note that in profile (a) $C(i)/i$ is increasing while in profile (b) it is decreasing.

For profile (d) in Table 1, we assume that there are three jobs and no arrivals, and the sizes of jobs 1, 2, and 3 are 1, 3, and 6, respectively. We also consider the policy S-L (short then long) which serves job 1 to completion (so it follows SRPT for the first job), and then applies LRPT for the remaining two jobs. The makespan for S-L is 3.23 and the mean sojourn time <u>6.70</u>. In this case LRPT minimizes the makespan, but the best policy for the mean sojourn time is S-L; neither SRPT or LRPT minimizes the mean sojourn time for this example.

The next two examples illustrate some possibilities when there are arrivals, and the objective is the minimization of sojourn time, that is, $g(i) = i$.

Example 2: This example shows that the optimal policy may idle even when jobs are present. As we noted in the introduction, if the server speed is constant, because we allow preemption, it will never be optimal to idle. This is no longer the case with speed scaling. Suppose that $C(1) = 1$, $C(i) = 5$ for $i \geq 2$, and suppose jobs of size 1 arrive starting at time 0 and with interarrival time 10, and jobs of size 5 arrive starting at time 2 with interarrival time of 10. Then a non-idling policy will have mean long-run average sojourn time of 3. The optimal policy is to idle from time $1 - \varepsilon$, for infinitesimally small ε, until time 2, and then follow non-idling LRPT until both jobs are finished; this is repeated in each cycle. For this policy the long-run average sojourn time, as $\varepsilon \to 0$, is 2.

Example 3: We now consider an example in which the optimal policy depends on the arrival rate, and where it could be something other than SRPT or LRPT. Here we assume $C(1) = 1$; $C(i) = 3/2$ for $i \geq 2$. Suppose batches of two jobs arrive at $t = (4 + \tau)n$, $n = 0, 1, \ldots$, where $\tau \geq 0$, so the interarrival times are $4 + \tau$, and service times are 1 and 5 time units. The

https://doi.org/10.1017/S0269964819000160 Published online by Cambridge University Press

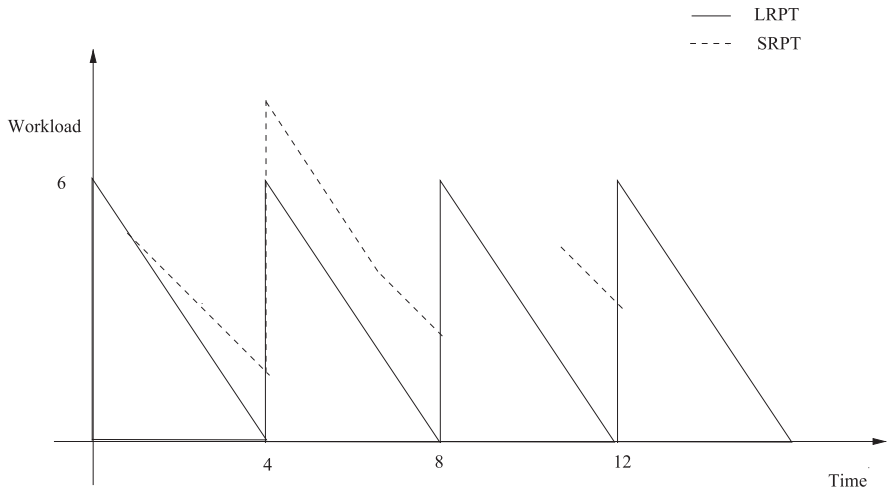


FIGURE **1.** Workload process with deterministic arrivals.

system will be stable if and only if the total arriving rate of work $((1/2 + 5/2)/(4 + \tau)$ is smaller than the maximal capacity 3/2, that is, if the traffic intensity, ρ, is such that

$$\rho = \frac{3}{4+\tau} < 1.$$

In this case we will not want to idle the server.

We first consider LRPT and SRPT, and show that which of these is preferred depends on the arrival rate. The workload process for SRPT and LRPT and $\tau = 0$ is shown in Figure 1.

For LRPT, the workload process is saw-toothed with a period $4 + \tau$ seconds, and the mean sojourn time is 4. For SRPT, the process is more complicated, and we present the full analysis in the appendix. We deduce that, for minimizing mean sojourn time,

$$\tau < 2/3 \Rightarrow \text{LRPT is better than SRPT;}$$
$$\tau > 2/3 \Rightarrow \text{SRPT is better than LRPT.}$$

In terms of the load, LRPT is better than SRPT for $9/14 < \rho < 1$, and SRPT is better than LRPT if $\rho < 9/14$.

The optimal policy may be something "in between" SRPT and LRPT. In particular, for $0 < \tau < 2/3$, starting at time 0 with two jobs, the optimal policy is to work on the two jobs in such a way that the second departure is at time $4 + \tau$, which means the first departure is at $4 - 2\tau$, and the mean sojourn time is $4 - \tau/2$. This could be implemented by, for example, working on job 2 (the size-5 job) for $3\frac{1}{3} - 2\tau$ time units, and then working on the size-1 job to completion, that is, for 2/3 time units, and then switching back to job 2 until time $4 + \tau$. This repeats for each cycle.

4. OPTIMAL POLICIES WITH A JOB HOLD OPTION

We now suppose that when idling is permitted we are allowed to "hold" jobs that are complete, they can remain in the system until we decide to release them, and we can release

https://doi.org/10.1017/S0269964819000160 Published online by Cambridge University Press

them at any time. When idling is not permitted, let us suppose that we are allowed to "hold" jobs that are complete as long as there is work in the system (i.e., up until the end of a busy period). This is in contrast to standard scheduling in which it is implicitly assumed that a job will always be released as soon as it is complete. We call such policies standard policies.

We define two new classes of policies:

- SRPT-hold policies are policies where jobs are always served according to non-idling SRPT, but each job may be held after it completes for an arbitrary amount of time. We shall denote the set of these policies by SRPTh.
- SRPT-work-hold policies are defined similarly, except that now jobs can only be held as long as there is non-zero work in the system. The set of these policies will be denoted by SRPTwh.

For an arbitrary policy π, for time t, let $N^\pi(t)$ be the number of jobs (either unfinished or finished and held) at t and let $M^\pi(t)$ be the cumulative number of released jobs by time t. We say two policies π_1 and π_2 are state-equivalent if we can construct sample paths of $\{N^{\pi_1}(t),\, M^{\pi_1}(t)\}_{t=0}^{\infty}$ and $\{N^{\pi_2}(t),\, M^{\pi_2}(t)\}_{t=0}^{\infty}$ on the same probability space so that they are indistinguishable, that is, they are identical almost surely. Note that this is equivalent to having all departures occur at the same times under both policies. We have the following state-equivalence between arbitrary policies and SRPTh or SRPTwh policies.

LEMMA 1: *For any standard idling policy there exists a state-equivalent* SRPTh *policy, and for any non-idling standard policy there is a state-equivalent* SRPTwh *policy.*

PROOF: We focus on the non-idling case for ease of notation; the idling case follows similarly. For an arbitrary non-idling policy π, let $R_i^\pi(t)$ be the i-th smallest remaining service time among the unreleased jobs at time t and let $W^\pi(t) = \sum_{i=1}^{N^\pi(t)} R_i^\pi(t)$ be the total unfinished work at t. Let $S^\pi(t) = (N^\pi(t),\, M^\pi(t),\, R_1^\pi(t), \ldots,\, R_{N^\pi(t)}^\pi(t))$ be the extended state at time t. For coupled sample paths for two policies π_1 and π_2 we say that $\{S^{\pi_1}(t)\}_{t=0}^{\infty} \prec \{S^{\pi_2}(t)\}_{t=0}^{\infty}$ w.p. 1, if, jointly for all t, and with probability 1, $N^{\pi_1}(t) = N^{\pi_2}(t)$, $M^{\pi_1}(t) = M^{\pi_2}(t)$, and remaining service times are smaller under π_1 in the majorization order. That is, for each t, the total workload is the same, while the partial sums of remaining service times are ordered for all $j = 1, \ldots,$ so that, with probability 1, $N^{\pi_1}(t) = N^{\pi_2}(t)$,

$$\sum_{i=1}^{j} R_i^{\pi_1}(t) \le \sum_{i=1}^{j} R_i^{\pi_2}(t),$$

$$W^{\pi_1}(t) = \sum_{i=1}^{N^{\pi_1}(t)} R_i^{\pi_1}(t) = \sum_{i=1}^{N^{\pi_2}(t)} R_i^{\pi_2}(t) = W^{\pi_2}(t).$$

Let π be an arbitrary non-idling policy, and let $\sigma \in$ SRPTwh be the SRPT-work-hold policy that we will construct, which releases jobs at the same time as π, with $S^\pi(0) = S^\sigma(0)$ (so $S^\pi(0) \prec S^\sigma(0)$). We will couple the processes under the two policies so that $\{S^\pi(t)\}_{t=0}^{\infty} \prec \{S^\sigma(t)\}_{t=0}^{\infty}$ w.p. 1. In particular, assume sample paths of the arrival processes (including associated job service times) are identical for both policies. We argue by induction on event times, which are the (common) arrival epochs as well as the departure epochs induced by π. Let us assume that up until and including an event time t_0, we have coupled the processes so that $\{S^\pi(t)\}_{t=0}^{t_0} \prec \{S^\sigma(t)\}_{t=0}^{t_0}$ w.p. 1. Let t_1 the time of the next event after t_0, which can be an arrival or a service completion under π. Let σ not release any jobs until time t_1. Between

https://doi.org/10.1017/S0269964819000160 Published online by Cambridge University Press

t_0 and t_1, the capacity remains constant, at $C(N^\pi(t_0))$, under both policies. Then, with probability 1, for $t \in [t_0, t_1)$, $N^\pi(t) = N^\sigma(t)$, $M^\pi(t) = M^\sigma(t)$, $W^\pi(t) = W^\sigma(t)$, and from [1, Proposition 1], we can also conclude that the majorization order for remaining service times is preserved, that is, $S^\pi(t) \prec S^\sigma(t)$, for $t \in [t_0, t_1)$. If the event at t_1 is an arrival, then clearly $S^\pi(t_1) \prec S^\sigma(t_1)$ w.p. 1 still holds. If the event at t_1 under π is a departure, of k jobs say, then necessarily k jobs have already finished their service under σ, because of the majorization order. We let σ also release k jobs, and again we will have $S^\pi(t_1) \prec S^\sigma(t_1)$ w.p. 1, thus completing the induction step.

For the idling case, the argument is similar, except that we have $W^\pi(t) \leq W^\sigma(t)$ w.p. 1, that is, we have a weak majorization order on the states. ■

Note that SRPT is an SRPTh policy that never holds a job, that is, it is the earliest release policy among all SRPTh policies, and non-idling LRPT is state-equivalent to an SRPTwh policy that holds all jobs until the end of the busy period, that is, it is the latest release policy among all SRPTwh policies. Within SRPTh policies, idling corresponds to not releasing any jobs when there is no work in the system, that is, all current jobs have completed processing. We note when there are no arrivals, we can arbitrarily closely approximate an SRPTh or SRPTwh policy with a standard policy.

With Lemma 1, by allowing the option of holding jobs, we have reduced our problem to one of determining the release times of completed jobs, and otherwise following SRPT (possibly with idling, or not releasing jobs, even when there is no work).

5. MODEL WITHOUT ARRIVALS

In this section we assume that all jobs are present at time 0, with no future arrivals, and job sizes are known. We give a complete characterization of the optimal policy for a general state-dependent cost function.

First, we observe that it is never optimal to idle when there are no future job arrivals. This observation together with Lemma 1 says that we can restrict our search for the optimal policy to the class of SRPTwh policies. This reduction thus determines the order in which jobs will be served in the optimal policy. The only remaining variables which can be used to vary the cost are the release times of completed jobs. Because jobs differ only in size, the choice of which of the completed jobs to be released does not influence the total cost. Assume that jobs are ordered in decreasing order of their sizes, that is, job 1 is the longest job and job K is the shortest job, and let x_i denote the size of job i. If there are two or more jobs with the same size, then they can be ordered arbitrarily amongst themselves without affecting the cost. Since SRPT is the preferred service discipline, job K will be the first one to enter service followed by job $K-1$ and so on until we get to job 1.

For some feasible policy π, let $T_i^\pi \geq 0$ be the amount of time there are i jobs in the system, so, abusing notation a bit, $\pi = (T_K^\pi, \ldots, T_1^\pi)$, and all jobs will be complete at time $\sum_{i=1}^K T_i^\pi$. Note that $T_i = 0$ is possible. For example, a policy could wait for two jobs to complete, and then release both of them, in which case $T_{K-1}^\pi = 0$. We have

$$V^\pi = \int_0^\infty g(N^\pi(t))dt = \sum_{i=1}^K g(i)T_i^\pi.$$

Let W_i^π be the amount of work done by time $\sum_{j=i}^K T_j^\pi$, so $W_i^\pi = \sum_{j=i}^K C(j)T_j^\pi$. Since a feasible policy cannot release more jobs than are completed, and jobs are served in SRPT

https://doi.org/10.1017/S0269964819000160 Published online by Cambridge University Press

order, for any feasible policy π, $W_i^\pi \geq \sum_{j=i}^K x_j$. We therefore have the following linear program to find the optimal policy.

THEOREM 1: *The optimal policy is the solution to the linear program:*

$$P : \min_\pi V^\pi = \sum_{i=1}^K g(i) T_i^\pi \text{ subject to}$$
$$\sum_{j=i}^K C(j) T_j^\pi \geq \sum_{j=i}^K x_j \quad \text{for } i = 1, \ldots, K$$
$$T_i^\pi \geq 0 \quad \text{for } i = 1, \ldots, K$$

Abusing notation a bit, we will also refer to the problem we have just defined as P. Now let us consider a new, constant-speed, problem, $\hat{P}$, with $C(i) \equiv 1$, and cost function $\hat{g}(i) = g(i)/C(i)$. The optimal policy for $\hat{P}$ is the solution to the linear program:

$$\hat{P} : \min_{\hat{\pi}} V^{\hat{\pi}} = \sum_{i=1}^K \hat{g}(i) T_i^{\hat{\pi}} \text{ subject to}$$
$$\sum_{j=i}^K T_j^{\hat{\pi}} \geq \sum_{j=i}^K x_j \quad \text{for } i = 1, \ldots, K$$
$$T_i^{\hat{\pi}} \geq 0 \quad \text{for } i = 1, \ldots, K$$

Let $\hat{\pi} = (T_K^{\hat{\pi}}, \ldots, T_1^{\hat{\pi}})$ be a feasible solution for $\hat{P}$, and let π be such that $T_i^\pi = T_i^{\hat{\pi}}/C(i)$. Then π is a feasible solution to P, and $V^\pi = V^{\hat{\pi}}$. We have the following corollaries.

COROLLARY 1: *To solve the speed-scaled problem, P, we need to only solve the constant-speed problem, $\hat{P}$, and let $T_i^* = T_i^{\hat{\pi}^*}/C(i)$.*

COROLLARY 2: *The optimal job release times for both P and $\hat{P}$ will be a subset of job completion times, with jobs processed in SRPT order. At these times, either no jobs or all completed jobs will be released.*

PROOF: Because the optimal solution to a linear program is at the vertices of the constraint space (when the constraints are binding), the result follows. ■

Let s_i be the number of jobs that are in the system when job i (the i'th longest job) is in service, so $s_i \geq n - i + 1$, and $n = s_1 \geq s_2 \geq \cdots s_n$. From Lemma 1 and Corollary 2, any optimal policy $\pi \in \mathsf{SRPTwh}$ is uniquely characterized by its sequence $(s_i)_{i=1\ldots K}$. Note that under this characterization, the corresponding policy in the constant-speed model, $\hat{\pi}$, is the same as π, $\pi = \hat{\pi}$. Also, under the constant speed model, the total cost incurred while processing job i under π is $\hat{g}(i)x_i$ from the definition of s_i, so,

$$V^\pi = \sum_{i=1}^K \hat{g}(s_{s_i}) x_i = \sum_{i=1}^K \frac{g(s_i)}{C(s_i)} x_i. \tag{3}$$

Since a job cannot be recalled once it is released, it follows that, for any feasible policy,

$$\begin{aligned} i \leq s_i \leq s_{i+1}, \quad i = 1, 2, \ldots, K-1, \\ s_K = K. \end{aligned} \tag{4}$$

https://doi.org/10.1017/S0269964819000160 Published online by Cambridge University Press

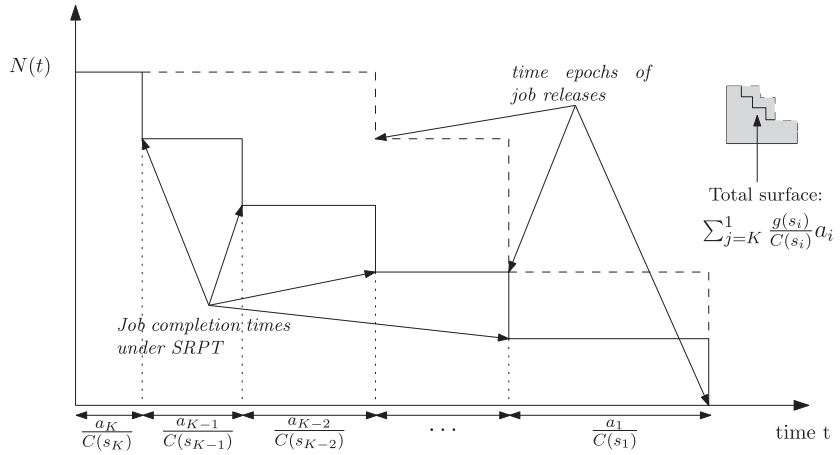


FIGURE **2.** Evolution of the number of jobs in the system over time. The dotted line is that of an arbitrary policy.

COROLLARY 3: *The optimal policy, for both P and $\hat{P}$, is determined by the solution $s^* = (s_K^*, \ldots, s_1^*)$ to the linear program that minimizes* (3) *subject to the constraints* (4).

Note that s^* is the same for both P and $\hat{P}$, but the actual job release times are different. A graphical illustration of Corollary 3 is given in Figure 2.

Consider the algorithm NA^* whose pseudocode is given below. The algorithm determines s_j^* for each j, and we shall prove that it outputs an optimal policy. We note that there could be several equivalent optimal policies based on which jobs are released and which among the several possible s_j^*'s are picked. If there are several minimizers for some

Algorithm NA^*: Optimal scheduling algorithm for no arrivals

Input : Vectors **g**, and **C**
$s_K \leftarrow K$; Serve job K
for *j=K-1:1* **do**
  $s_j^* \in \arg\min_{i:j\le i\le K} \frac{g(i)}{C(i)}$
  `/* choose arbitrarily one minimizer ensuring that` $j \le s_j^* \le s_{j+1}^* \le K$ `*/`
  Release $s_{j+1}^* - s_j^*$ jobs.
  `/* chosen arbitrarily */`
  Serve job j
end

job then we arbitrarily chose any one of these. We shall show that this does not influence the total cost.

For each job, NA^* myopically computes the best possible number in the system to be held while processing that job, to minimize the contribution of the job to the total cost. Using Corollary 3, we now prove that this myopic algorithm is globally optimal.

THEOREM 2: *Algorithm NA^* minimizes total holding cost.*

https://doi.org/10.1017/S0269964819000160 Published online by Cambridge University Press

PROOF: First, we shall show that $\mathbf{s}^*$ as computed by Algorithm NA^* gives a feasible policy as defined in (4). The first inequality $s_j^* \geq j$ follows from the definition because the minimizer is searched over the set of values that are larger than j. The second inequality follows from the fact that s_j^* is a minimizer over a larger set compared to the set over which s_{j+1}^* is a minimizer. Thus, there will always exist an $s_j^* \leq s_{j+1}^*$.

Next, by definition of s_j^*, for any other feasible policy $g(s_j)/C(s_j) \geq g(s_j^*)/C(s_j^*)$. It then follows that $\mathbf{s}^*$ minimizes (3) subject to (4).

Note that the choice among the minimizers does not influence the total cost as it depends only upon $g(s_j^*)/C(s_j^*)$ which is the same for all the minimizers. ■

The optimal policy serves jobs $K, \ldots, 1$ in order. When job j ends service, there are s_j^* jobs still in the system. The optimal policy then releases $s_{j+1}^* - s_j^*$ jobs. If no jobs are released until the last job ends service, then we have standard LRPT, whereas if every job is released immediately upon completion of service, we have standard SRPT. This motivates us to consider special cases of cost functions for which either LRPT or SRPT is optimal. We will refer to $C(i)/i$ as the *per-job capacity*. We note that $C(i)$ is said to be *star-shaped* if $C(i)/i$ is increasing in i, and *anti-star-shaped* if $C(i)/i$ is decreasing in i [15]. We assume, without loss of generality, that $C(0) = 0$. Under this condition, if $C(i)$ is increasing and concave it is also anti-star-shaped (its per-job capacity is decreasing), and if it is increasing and convex it is also star-shaped (its per-job capacity is increasing). The converses are not true. If $C(i)$ is decreasing for $i \geq 1$ then it is anti-star-shaped (its per-job capacity is decreasing), because we assume $C(i) \geq 0$.

The following result can be deduced from Theorem 2.

COROLLARY 4: *Among policies that may be preemptive and may idle:*

(i) If $g(i)/C(i)$ is decreasing (increasing), then LRPT (SRPT) minimizes total holding cost. Therefore,

(ii) If the per-job capacity is increasing, then LRPT minimizes mean sojourn time.

(iii) If the per-job capacity is decreasing, then SRPT minimizes mean sojourn time.

(iv) If $C(i)$ is increasing (decreasing), then LRPT (SRPT) minimizes the makespan.

(v) If $g(i)/C(i) = b$ for some constant b, then total holding costs are the same for any non-idling policy, and it will not be optimal to idle when the objective is cost minimization.

(vi) If there exists a k_0, $1 \leq k_0 \leq K$ such that k_0 minimizes $g(i)/C(i)$ over $1 \leq i \leq K$, and $g(i)/C(i)$ increases for $i \geq k_0$, then an S-L k_0 policy is optimal, that is, follow SRPT until there are only k_0 remaining jobs, and then follow LRPT.

PROOF: We have (i) and (v) from Algorithm NA^*, because $s_i^* = K$ ($s_i^* = i$), for $i = 1, \ldots K$, if $g(i)/C(i)$ is decreasing (increasing), and, if $g(i)/C(i)$ is constant, s_i^* can be any integer between i and K. Result (vi) follow similarly from Algorithm NA^*. Finally, (ii) and (iii) follow from (i) with $g(i) = i$, and (iv) follows from (i) with $g(i) = I\{i > 0\}$. ■

Note that Corollary 4 (iv) tells us that under the model of Gupta and Harchol Balter [8], where the capacity first increases and then decreases in the number in system, the optimal policy is an S-L k_0 policy.

We may also want upper bounds on total holding cost for any policy. For a non-trivial problem formulation, we restrict ourselves to policies in which there is no idling. The proof of the following corollary is similar to that above.

https://doi.org/10.1017/S0269964819000160 Published online by Cambridge University Press

COROLLARY 5: *Among preemptive, non-idling policies:*

(i) If $g(i)/C(i)$ is decreasing (increasing), then SRPT (LRPT) maximizes total holding cost. Therefore,

(ii) If the per-job capacity is increasing, then SRPT maximizes mean sojourn time.

(iii) If the per-job capacity is decreasing, then LRPT maximizes mean sojourn time.

(iv) If $C(i)$ is increasing (decreasing), then SRPT (LRPT) maximizes the makespan.

(v) If $g(i)/C(i) = b$ for some constant b, then total holding costs are the same for any non-idling policy, and it will not be optimal to idle when the objective is cost minimization.

Remark 1: Note that when there are no arrivals, LRPT also minimizes the variability in sojourn times.

6. MODEL WITH ARRIVALS

As we saw in section 3, adding arrivals to our model can significantly complicate both the analysis and the results. As illustrated in Example 2, the optimal policy might idle, waiting for an arrival, in order to profit from the increment of capacity available in larger states. The important characterization provided in Lemma 1 is still true, that is, even with arrivals there exists an optimal policy that is either SRPTh or SRPTwh. That is, jobs will be served according to SRPT and the problem reduces to the question of when to release jobs that have already completed service. Idling now corresponds to not releasing all jobs when there is no work to do.

In this section we extend parts of Corollaries 4 and 5, but the proofs are quite different. We assume the arrival process and job sizes are arbitrary, but independent of the policy and system state, and job sizes are observed upon arrival. Throughout this section, we also assume that the system's parameters are such that a steady-state distribution exists and is well-defined.

6.1. The Linear Cost-To-Capacity Case

We first extend Corollary 5 (v) to arrivals.

THEOREM 3: *If we have a linear cost-to-capacity ratio, that is, $g(i)/C(i) = b$ for some constant b, then all non-idling policies yield the same long-run average cost per job, which will be b times the average job size. It will not be optimal to idle.*

PROOF: We assume a finite number of arrivals, N, and use induction on N to show that the total cost is equal to b times the job sizes of jobs. Let us fix the (remaining) service times of all current jobs and all future arrivals, as well as the arrival times for future arrivals. Suppose there are n jobs available at time 0. Let x_i be the remaining service time of the i'th current job (ordered arbitrarily), and let a_i and y_i be the arrival and job size of the i'th future arrival. Let $V_N^\pi(\mathbf{x}, \mathbf{y}, \mathbf{a})$ be the total cost, starting at time 0, for all current and future jobs, for an arbitrary non-idling policy π. For $N = 0$, the result follows from Corollary 4, and it is easy to show that $V_0^\pi(\mathbf{x}) = b\sum_{i=1}^n x_i$. As our induction hypothesis, suppose that $V_N^\pi(\mathbf{x}, \mathbf{y}, \mathbf{a}) = b\sum_{i=1}^n x_i + b\sum_{i=1}^N y_i$ for all non-idling π. Now we show the result for $N + 1$.

https://doi.org/10.1017/S0269964819000160 Published online by Cambridge University Press

By induction,

$$V^{\pi}_{N+1}(\mathbf{x},\mathbf{y},\mathbf{a}) = G^{\pi}(a_1) + bx^{\pi} + b\sum_{i=1}^{N+1} y_i,$$

where x^{π} is the total unfinished work remaining at time a_1 from the jobs present at time 0, and $G^{\pi}(a_1)$ is the total cost incurred between times 0 and a_1. Let d_i be the times between departures of jobs that depart by time a_1 under π, $i = 0, \ldots, m$ for some $m \le n$, and $\sum_{i=1}^{m} d_i \le a_1$. Then

$$\begin{aligned} x^{\pi} &= \sum_{i=1}^{n} x_i - \Bigg[C(n)d_1 - C(n-1)d_2 - \cdots - C(n-m+1)d_m \\ &\quad - C(n-m)\left(a_1 - \sum_{i=1}^{m} d_i\right)\Bigg], \\ G^{\pi}(a_1) &= g(n)d_1 + g(n-1)d_2 + \cdots + g(n-m+1)d_m \\ &\quad + g(n-m)(a_1 - \sum_{i=1}^{m} d_i) = \sum_{i=1}^{n} x_i - bx^{\pi} \end{aligned}$$

so

$$V^{\pi}_{N+1}(\mathbf{x},\mathbf{y},\mathbf{a}) = b\sum_{i=1}^{n} x_i + b\sum_{i=1}^{N+1} y_i.$$

A similar argument shows that we should not idle, because idling until time $\min\{\delta, a_1\}$ for some $\delta > 0$ and not thereafter (by induction), has cost $g(n)\delta + b\sum_{i=1}^{n} x_i + b\sum_{i=1}^{N+1} y_i > V^{\pi}_{N+1}(\mathbf{x},\mathbf{y},\mathbf{a})$. ■

We can therefore think of the linear cost-to-capacity queue as being equivalent to an infinite server queue with linear costs, and, regardless of the scheduling discipline, the long-run average cost rate is λb times the mean job size, where λ is the long-run average arrival rate. Note that this result does not depend on job sizes being i.i.d., and is true even if job sizes are not observed upon arrival, among standard policies that do not use size information and release jobs upon completion. The result also does not require any further capacity conditions to ensure finite sojourn times.

6.2. Optimizing the Busy Period Length

We now extend Corollaries 4 (iv) and 5 (iv) from makespan to busy periods. Using Lemma 1, we assume, that we are permitted to hold completed jobs. A busy period ends the first time there are no jobs in the system, which, if idling is permitted, may be later than the first time the total work in the system hits 0 if we hold completed jobs. In the theorem, by LRPT we mean non-idling LRPT, so all jobs are released if there is no remaining work in the system.

For the busy period to be a.s. finite we need extra capacity conditions. For example, a sufficient condition would be that there exists n_0 and C such that $C(i) \ge C$ for $i > n_0$, and that the long-run arrival rate, λ, is such that $\lambda < C$. However, because our result below is about the current busy period starting in some finite state, a capacity condition is not needed, strictly speaking. For example, for (i) the result is trivially true when the busy period is infinite (even under a possible "best" policy).

https://doi.org/10.1017/S0269964819000160 Published online by Cambridge University Press

THEOREM 4:

(i) If $C(i)$ is increasing, then LRPT stochastically minimizes the length of the current busy period.

(ii) If $C(i)$ is increasing (decreasing), then SRPT (LRPT) stochastically maximizes the length of the current busy period among non-idling policies.

(iii) If $C(i)$ is decreasing, then SRPT stochastically maximizes the departure process, $\{D(t)\}_{t=0}^{\infty}$, where $D(t)$ is the number of departures by time t, (and therefore it minimizes the current busy period length, and it minimizes mean sojourn time).

(iv) If $C(i)$ is increasing until $i = K$ and then decreasing, following LRPT while $i \leq K$ and following SRPT while $i > K$ stochastically minimizes the length of the current busy period.

PROOF:

(i) From Lemma 1, idling corresponds to not releasing all jobs as soon as there is no work in the system, and not releasing those jobs just extends the busy period, so idling is clearly not optimal for minimizing the current busy period length. Not following LRPT corresponds to releasing a job before the end of the busy period. Fix an initial state and a sample path of arrival times and job sizes, τ_i, σ_i, $i = 1, 2, \ldots$. If policy π releases a job at some time t before all the work has emptied, then from t until the end of the LRPT busy period π will have fewer jobs than LRPT, and hence lower capacity and more work. Therefore, its current busy period must be larger than that under LRPT.

(ii) Suppose $C(i)$ is increasing. Again, fixing a sample path, throughout an SRPT busy period the number of jobs is smaller, and hence server speed is smaller, under SRPT than any other policy. Therefore, the SRPT busy period will end later. The argument for the other cases is similar.

(iii) For a fixed sample path with $C(i)$ decreasing, SRPT now maximizes capacity over its busy period, so work gets done faster, and, for any fixed capacity profile, SRPT also produces the earliest possible release of jobs.

(iv) This follows from arguments similar to those of (i) and (iii). ■

From (i) and (iii) of Theorem 4 it is not hard to show the following.

COROLLARY 6: *If we have Poisson arrivals and i.i.d job sizes, and if $C(i)$ is increasing (decreasing), $g(i)$ is decreasing (increasing), and long-run costs are finite under some policy, then LRPT (SRPT) stochastically minimizes long-run average costs.*

7. CONCLUSIONS

We have investigated the optimal scheduling problem when the capacity is state-dependent. This is a modeling abstraction that has applications in server-farms, peer-to-peer downloading, wireless scheduling, and other situations. In full generality the solution space can be very complex. For instance, we have presented examples that illustrate that idling may be optimal, or that the optimal policy could depend on the load, and could be something other than SRPT or LRPT.

One of our main results is that there exists an optimal policy that always serves jobs according to SRPT, but may hold jobs after they complete. The key question becomes

https://doi.org/10.1017/S0269964819000160 Published online by Cambridge University Press

determining the optimal moment to release jobs from the system. In the case without arrivals we have shown that the optimal releasing moment crucially depends on the cost-to-capacity ratio. For the case with arrivals we have obtained partial results, but various interesting questions remain open.

Acknowledgment

Research partially supported by the French "Agence Nationale de la Recherche (ANR)" through the project ANR-15-CE25-0004 (ANR JCJC RACON).

References

1. Aalto, S. & Ayesta, U. (2009). SRPT applied to bandwidth-sharing networks. *Annals of Operations Research* 170: 3–19.
2. Bansal, N., Kimbrel, T., & Pruhs, K. (2004). Dynamic speed scaling to manage energy and temperature. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science*, pp. 520–529.
3. Bansal, N., Kimbrel, T., & Pruhs, K. (2007). Speed scaling to manage energy and temperature. *Journal of the ACM* 54: 1–39.
4. Bekker, R., Borst, S.C., Boxma, O.J., & Kella, O. (2004). Queues with workload-dependent arrival and service rates. *Queueing Systems* 46: 537–556.
5. Berg, B., Dorsman, J.-P., & Harchol-Balter, M. (2018). Towards Optimality in Parallel Job Scheduling. In *Proceedings of ACM SIGMETRICS 2018*, Los Angeles, CA, June 2018.
6. Buyukkoc, C., Varaya, P., & Walrand, J. (1985). The $c\mu$ rule revisited. *Advances in Applied Probability* 17: 237–238.
7. Elahi, M. & Williamson, C. (2018). On saturation effects in coupled speed scaling. In *Quantitative Evaluation of Systems* (A. McIver & A. Horvath Eds.), Cham, Switzerland: Springer International Publishing, pp. 407–422.
8. Gupta, V. & Harchol-Balter, M. (2009). Self-adaptive Admission Control Policies for Resource-sharing Systems. In *Proceedings of ACM SIGMETRICS*, pp. 311–322.
9. Marin, A., Mitrani, I., Elahi, M., & Williamson, C. (2018). Control and optimization of the SRPT service policy by frequency scaling. In *Quantitative Evaluation of Systems* (A. McIver & A. Horvath, Eds.), Cham, Switzerland: Springer International Publishing, pp. 257–272.
10. Righter, R. (1994). Scheduling. In *Chapter in Stochastic Orders* (J.G. Shaked & M. Shanthikumar Eds.), New York: Academic Press, pp. 381–432.
11. Righter, R. & Shanthikumar, J.G. (1989). Scheduling multiclass single server queueing systems to Stochastically maximize the number of successful departures. *Probability in the Engineering and the Informational Sciences* 3: 323–333.
12. Righter, R. & Shanthikumar, J.G. (1992). Extremal properties of the FIFO discipline in queueing networks. *Journal of Applied Probability* 29: 967–978.
13. Sadiq, B. & de Veciana, G. (2010). Balancing SRPT prioritization vs. opportunistic gain in wireless systems with flow dynamics, ITC 22.
14. Schrage, L. (1968). Letter to the Editor – A proof of the optimality of the shortest remaining processing time discipline. *Operations Research* 16(3): 687–690.
15. Shaked, M. & Shanthikumar, J.G. (2007). *Stochastic orders*. New York: Springer.
16. Timmermans, V. & Vredeveld, T. (2015). Scheduling with state-dependent machine speed, WAOA Workshop?. In *Approximation and Online Algorithms* (L. Sanità & M. Skutella Eds.), Cham, Switzerland: Springer, pp. 196–208.
17. Weiss, G. (1995). A tutorial in stochastic scheduling. In *Scheduling Theory and Its Applications* (P. Lenstra, E.G. Chretienne, Z. Coffman & J.K. Liu, Eds.), Wiley.
18. Wierman, A., Andrew, L.L.H., & Tang, A. (2009). Power-aware speed scaling in processor sharing systems. In *IEEE INFOCOM*, 2007–2015.

APPENDIX: ANALYSIS OF EXAMPLE 3

First consider the case when $\tau = 0$. Then, at $t = (4 + n)^+$, that is, right after the n'th arrival, the remaining service times of the jobs in the system are 1, 5, and x_n, where $1 < x_n < 5$, and x_n is an

https://doi.org/10.1017/S0269964819000160 Published online by Cambridge University Press

increasing sequence that follows the recursion:

$$x_{n+1} = \frac{2}{3}x_n + \frac{5}{3}, \quad n \geq 0,$$

with $x_0 = 0$.

In the interval $[4n, 4(n+1))$, the jobs of size 1 and of x_n are served entirely under SRPT. The job of size x_n finishes service at time $4n + \frac{2}{3}(1 + x_n)$ and the job of size 5 enters service. This job is served partially before the next arrival, at which point its remaining service time at $t = 4(n+1)^+$ is given by x_{n+1}. Since $\lim_{n\to\infty} x_n \to 5$, when $n \to \infty$ the workload process of SRPT converges to a periodic process of period 4. The remaining service times of jobs at the beginning of each cycle are 1, 5, and 5, where one of the size-5 jobs arrived at the beginning of the previous previous cycle. The job of size 1 will leave after 2/3 units, one of the size-5 jobs will leave at the end of the cycle, and the other will leave at the end of the next cycle, that is after 8 units. The mean sojourn time is then $(2/3 + 8)/2 > 4$.

For $\tau \geq 0$, the workload process under SRPT will converge to a cycle with a period of $4 + \tau$ units. If $\tau \geq 5/3$, cycles will not overlap, and the mean sojourn time is $(4/3 + 5)/2 < 4$. Suppose that $\tau < 5/3$. At the beginning of a cycle (that is, just after an arrival), the remaining service times will be 1, 5, and $5 - 3\tau$. Within a cycle, the job of size 1 will enter service immediately and will leave at time 2/3. At this point the job of remaining size $5 - 3\tau$ will enter service and will leave at $4 - 2\tau$ units. During the remaining 3τ units, the job of size 5 will be served at rate 1, which will leave it with $5 - 3\tau$ remaining service at the beginning of the next cycle. This job will spend $(4 + \tau) + (4 - 2\tau) = 8 - \tau$ time units in the system. Thus, the mean sojourn time of jobs will be

$$\frac{\frac{2}{3} + 8 - \tau}{2} = 4 + \frac{1}{3} - \frac{\tau}{2}.$$

One then deduces that

$$\tau < 2/3 \Rightarrow \text{LRPT is better than SRPT;}$$
$$\tau > 2/3 \Rightarrow \text{SRPT is better than LRPT.}$$

In terms of the load, we can conclude that LRPT is better than SRPT for $6/7 < \rho < 1$, and SRPT is better than LRPT if $\rho < 6/7$.

Also, it is worth noting that in the long run, for $0 \leq \tau < 5/3$, under SRPT there are always three jobs in the system whereas under LRPT there are always two jobs in the system. This is not true in the transient phase in which the number of jobs with SRPT oscillates between 1 and 3, whereas there are always two with LRPT. Thus, LRPT is better in terms of the long-run average number of jobs, but it is not better across the sample path in the transient phase.

https://doi.org/10.1017/S0269964819000160 Published online by Cambridge University Press