Performance Evaluation 144 (2020) 102147

Contents lists available at ScienceDirect

Performance Evaluation

journal homepage: www.elsevier.com/locate/peva

heSRPT: Parallel scheduling to minimize mean slowdown

Benjamin Berg [a,*,1], Rein Vesilo [b], Mor Harchol-Balter [a,2]

[a] Carnegie Mellon University, United States of America
[b] Macquarie University, Australia

ARTICLE INFO

Article history:
Available online 6 October 2020

Keywords:
Parallel scheduling
Server allocation
Optimization
Speedup curves
Slowdown
Flow time

ABSTRACT

Modern data centers serve workloads which are capable of exploiting parallelism. When a job parallelizes across multiple servers it will complete more quickly, but jobs receive diminishing returns from being allocated additional servers. Because allocating multiple servers to a single job is inefficient, it is unclear how best to allocate a fixed number of servers between many parallelizable jobs.

This paper provides the first optimal allocation policy for minimizing the mean slowdown of parallelizable jobs of known size when all jobs are present at time 0. Our policy provides a simple closed form formula for the optimal allocations at every moment in time. Minimizing mean slowdown usually requires favoring *short* jobs over long ones (as in the SRPT policy). However, because parallelizable jobs have sublinear speedup functions, system efficiency is also an issue. System efficiency is maximized by giving *equal* allocations to all jobs and thus competes with the goal of prioritizing small jobs. Our optimal policy, high-efficiency SRPT (heSRPT), balances these competing goals. heSRPT completes jobs according to their size order, but maintains overall system efficiency by allocating some servers to each job at every moment in time. Our results generalize to also provide the optimal allocation policy with respect to mean flow time.

Finally, we consider the online case where jobs arrive to the system over time. While optimizing mean slowdown in the online setting is even more difficult, we find that heSRPT provides an excellent heuristic policy for the online setting. In fact, our simulations show that heSRPT significantly outperforms state-of-the-art allocation policies for parallelizable jobs.

1. Introduction

Modern data centers serve workloads which are capable of exploiting parallelism. When a job parallelizes across multiple servers it will complete more quickly. However, it is unclear how to share a limited number of servers between many parallelizable jobs.

In this paper we consider a typical scenario where a data center composed of N servers will be tasked with completing a set of M parallelizable jobs, where typically M is much smaller than N. In our scenario, each job has a different inherent size (service requirement) which is known up front to the system. In addition, each job can utilize any number of servers at any moment in time. These assumptions are reasonable for many parallelizable workloads such as training neural

* Corresponding author.
E-mail addresses: bsberg@cs.cmu.edu (B. Berg), rein.vesilo@mq.edu.au (R. Vesilo), harchol@cs.cmu.edu (M. Harchol-Balter).

1 This author was supported by a Facebook Graduate Fellowship.

2 This author was supported by: NSF-CMMI-1938909, NSF-CSR-1763701, NSF-XPS-1629444, and a Google 2020 Faculty Research Award.

https://doi.org/10.1016/j.peva.2020.102147

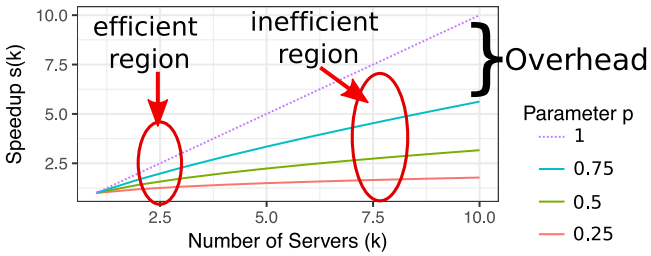

Fig. 1. A variety of speedup functions of the form $s(k) = k^p$, shown with varying values of p. When $p = 1$ we say that jobs are *embarrassingly parallel*, and hence we consider cases where $0 < p < 1$. Note that all functions in this family are concave and lie below the embarrassingly parallel speedup function ($p = 1$).

networks using TensorFlow [1,2]. Our goal in this paper is to allocate servers to jobs so as to minimize the *mean slowdown* across all jobs, where the slowdown of a job is the job's completion time divided by its running time if given exclusive access to all N servers. Slowdown is a measure of how a job was interfered with by other jobs in the system, and is often the metric of interest in the theoretical parallel scheduling literature (where it is also called stretch) [3], as well as the HPC community (where it is called expansion factor) [4].

What makes this problem difficult is that jobs receive a concave, sublinear speedup from parallelization—jobs have a decreasing marginal benefit from being allocated additional servers (see Fig. 1). Hence, in choosing a job to receive each additional server, one must keep the overall efficiency of the system in mind. The goal of this paper is to determine the optimal allocation of servers to jobs where all jobs follow a realistic sublinear speedup function.

It is clear that the optimal allocation policy will depend heavily on the jobs' speedup—how parallelizable the jobs being run are. To see this, first consider the case where each job is *embarrassingly parallel* (see Fig. 1), and can be parallelized perfectly across an arbitrary number of servers. In this case, we observe that the entire data center can be viewed as a *single server* that can be perfectly utilized by or shared between jobs. Hence, from the single server scheduling literature, it is known that the Shortest Remaining Processing Time policy (SRPT) will minimize the mean slowdown across jobs [5]. By contrast, if we consider the case where jobs are hardly parallelizable, a single job receives very little benefit from additional servers. In this case, the optimal policy is to divide the system equally between jobs, a policy called EQUI. In practice, a realistic speedup function usually lies somewhere between these two extremes and thus we must balance a trade-off between the SRPT and EQUI policies in order to minimize mean slowdown. Specifically, since jobs are *partially* parallelizable, it is still beneficial to allocate more servers to smaller jobs than to large jobs. The optimal policy with respect to mean slowdown must split the difference between these policies, figuring out how to favor short jobs while still respecting the overall efficiency of the system.

In this paper, we present the optimal allocation policy with respect to mean slowdown, which balances the tradeoff between EQUI and SRPT. We call this policy *high efficiency SRPT*. Our analysis considers the case where all jobs are present in the system at time 0. While this is a common case in practice, it is also interesting to consider an online version of the problem, where jobs arrive over time. Unfortunately, it has been shown that, in general, no optimal policy exists to minimize mean slowdown in the online case [6]. We demonstrate that heSRPT, which is optimal in the offline case, provides an excellent heuristic policy, Adaptive-heSRPT, for the online case. Adaptive-heSRPT often performs an order of magnitude better than policies previously suggested in the literature.

Our model

Our model assumes there are N identical servers which must be allocated to M parallelizable jobs. All M jobs are present at time $t = 0$. Job i is assumed to have some inherent size x_i where, without loss of generality (WLOG),

$$x_1 \geq x_2 \geq \cdots \geq x_M.$$

In general we will assume that all jobs follow the *same* speedup function, $s : \mathbb{R}^+ \to \mathbb{R}^+$, which is of the form

$$s(k) = k^p$$

for some $0 < p < 1$. Specifically, if a job i of size x_i is allocated k servers, it will complete at time

$$\frac{x_i}{s(k)}.$$

In general, the number of servers allocated to a job can change over the course of the job's lifetime. It therefore helps to think of $s(k)$ as a *rate*[3] of service where the remaining size of job i after running on k servers for a length of time t is

$$x_i - t \cdot s(k).$$

[3] WLOG we assume the service rate of a single server to be 1. More generally, we could assume the rate of each server to be μ, which would simply replace $s(k)$ by $s(k)\mu$ in every formula.

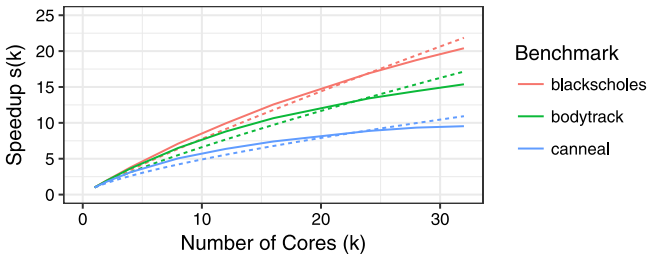

Fig. 2. Various speedup functions of the form $s(k) = k^p$ (dotted lines) which have been fit to real speedup curves (solid lines) measured from jobs in the PARSEC-3 parallel benchmarks [7]. The three jobs, blackscholes, bodytrack, and canneal, are best fit by the functions where $p = .89$, $p = .82$, and $p = .69$ respectively.

We choose the family of functions $s(k) = k^p$ because they are (i) sublinear and concave, (ii) can be fit to a variety of empirically measured speedup functions (see Fig. 2) [7], and (iii) simplify the analysis. Note that [8] assumes $s(k) = k^p$ where $p = 0.5$ and explicitly notes that using speedup functions of another form does not significantly impact their results.

In general, we assume that there is some policy, P, which allocates servers to jobs at every time, t. When describing the state of the system, we will use $m^P(t)$ to denote the number of remaining jobs in the system at time t, and $x_i^P(t)$ to denote the remaining size of job i at time t. To describe the state of each job at time t, let $W_i^P(t)$ denote the total amount of work done by policy P on job i by time t. Let $W_i^P(t_1, t_2)$ be the amount of work done by policy P on job i on the interval $[t_1, t_2)$. That is,

$$W_i^P(t_1, t_2) = W_i^P(t_2) - W_i^P(t_1).$$

We denote the completion time of job i under policy P as T_i^P. When the policy P is implied, we will drop the superscript.

We will assume that the number of servers allocated to a job need not be discrete. In general, we will think of the N servers as a *single, continuously divisible resource*. Hence, the policy P can be defined by an *allocation function* $\boldsymbol{\theta}^P(t)$ where

$$\boldsymbol{\theta}^P(t) = (\theta_1^P(t), \theta_2^P(t), \ldots, \theta_M^P(t)).$$

Here, $0 \le \theta_i^P(t) \le 1$ for each job i, and $\sum_{i=1}^M \theta_i^P(t) \le 1$. An allocation of $\theta_i^P(t)$ denotes that under policy P, at time t, job i receives a speedup of $s(\theta_i^P(t) \cdot N)$. Completed jobs are allocated 0 servers.

For each job, i, we define a corresponding *weight* $w_i > 0$. We then define the *weighted flow time* for a set of jobs under policy P, F^P, to be

$$F^P = \sum_{i=1}^{M} w_i \cdot T_i.$$

By manipulating the weights associated with each job, one can derive metrics with different intuitive meanings. We say that weights *favor small jobs* if larger weights are always assigned to smaller jobs. That is, if

$$w_1 \le w_2 \le \cdots \le w_M.$$

The class of weighted flow time metrics where weights favor small jobs includes several popular metrics. We are primarily interested in the mean slowdown of a policy P, $\overline{S}^P$, which is defined as

$$\overline{S}^P = \frac{1}{M} \cdot \sum_{i=1}^{M} \frac{T_i^P}{x_i/s(N)}.$$

The objective of minimizing mean slowdown is equivalent to minimizing weighted flow time when $w_i = \frac{1}{x_i/s(N)}$. Clearly, weights favor small jobs in this setting. Similarly, the objective of minimizing mean flow time is equivalent to minimizing weighted flow time when $w_i = 1$ for each job, i.

For the sake of generality, we consider the problem of finding the policy P^* which minimizes the weighted flow time for a set of M jobs when weights favor small jobs. Let F^* be the weighted flow time under this optimal policy. We will denote the allocation function of the optimal policy as $\boldsymbol{\theta}^*(t)$. Similarly, we let $m^*(t)$, $x_i^*(t)$, $W_i^*(t)$ and T_i^* denote the corresponding quantities under the optimal policy.

Why server allocation is counter-intuitive

Consider a simple system with $N = 10$ servers and $M = 2$ identical jobs of size 1, where $s(k) = k^{.5}$, and where we wish to minimize mean flow time (which is equivalent to slowdown in this case). A queueing theorist might look at this

problem and say that to minimize flow time, we should use the SRPT policy, first allocating all servers to job one and then all servers to job two. However, this causes the system to be very inefficient. Another intuitive argument would be that, since everything in this system is symmetric, the optimal allocation should be symmetric. Hence, one might think to allocate half the servers to job one and half the servers to job two. While this equal allocation does maximize the efficiency of the system by ensuring that neither job receives too many servers, it does *not* minimize their mean flow time. Theorem 5 will show that the optimal policy in this case is to allocate 75% of the servers to job one and 25% of the servers to job two. In our simple, symmetric system, the optimal allocation is very asymmetric! Note that this asymmetry is *not* an artifact of the form of the speedup function used. For example, if we had instead assumed that the speedup function s was Amdahl's Law [8] with a parallelizable fraction of $f = .9$, the optimal split is to allocate 63.5% of the system to one of the jobs. Instead, the optimal policy balances the tradeoff between using an efficient equal allocation and using the inefficient SRPT policy which favors small jobs. If we imagine a set of M arbitrarily sized jobs, one suspects that the optimal policy again favors shorter jobs, but it is not obvious how to calculate the exact allocations for this policy.

Why finding the optimal policy is hard

At first glance, solving for the optimal policy seems amenable to classical optimization techniques. However, naive application of these techniques would require solving $M!$ optimization problems, each consisting of $O(M^2)$ variables and $O(M)$ constraints. Furthermore, although these techniques could produce the optimal policy for a single problem instance, it is unlikely that they would yield a closed form solution. We instead advocate for finding a closed form solution for the optimal policy, which allows us to build intuition about the underlying dynamics of the system.

To understand the source of this complexity, we will first consider the problem of minimizing the total flow time of a set of just two jobs of sizes x_1 and x_2 respectively. If we assume that the optimal policy completes job 2 first, we can write an expression for the total flow time of the two jobs as follows:

$$T_1 + T_2 = \frac{2 \cdot x_2}{s(\theta_2(0))} + \frac{\left(x_1 - \frac{s(\theta_1(0)) \cdot x_2}{s(\theta_2(0))}\right)}{s(N)}.$$

The first term in this expression describes the total flow time accrued between time 0 and time T_2. The duration of this period is $\frac{x_2}{s(\theta_2(0))}$, and because there are two jobs in the system during this period, the total flow time accrued during the period is $\frac{2 \cdot x_2}{s(\theta_2(0))}$. The second term encodes the remaining time required to finish job 1 after job 2 completes. This expression assumes that job allocations only change at the time of a departure, which we will prove formally in Theorem 1. More interestingly, however, this expression *implicitly encodes* the idea that the policy finishes job 2 before job 1. If, on the other hand, job 1 finishes first, the expression would change to

$$T_1 + T_2 = \frac{2 \cdot x_1}{s(\theta_1(0))} + \frac{\left(x_2 - \frac{s(\theta_2(0)) \cdot x_1}{s(\theta_1(0))}\right)}{s(N)}.$$

This has two main implications. First, it is possible that the allocation policy which minimizes a particular expression for total flow time does not complete jobs in the order encoded in the expression. In this case, the value of the expression is meaningless—it does not equal the total flow time of the jobs under the computed policy. Hence, to find the optimal policy with respect to a particular completion order, one must minimize the corresponding expression for total flow time subject to constraints which maintain the completion order of the allocation policy. Second, because the objective function depends on the completion order of the jobs, there is no single function to try to minimize to recover an optimal policy. Instead, we are interested in the *global* minimum across all of the $M!$ completion orders, each of which has its own *local* minimum that can be obtained by constrained minimization.

When there are only two jobs, it is tractable to directly solve for the optimal policy via known constrained optimization methods. One can use Lagrange multipliers with two constraints—one to ensure a valid allocation policy and one to enforce the completion order. One must solve two such optimization problems corresponding to each of the potential completion orders. However, given a set of M jobs with $M!$ possible completion orders, this naive approach would require solving $M!$ optimization problems, each consisting of $O(M^2)$ variables used to define an allocation policy and $O(M)$ constraints. Even if some of this complexity could be elided by, for instance, proving the completion order of the optimal policy (see Theorem 2), Lagrange multipliers are unlikely to emit a closed form given such a complex objective function and large set of constraints.

Additionally, one may think to apply classical techniques from the deterministic scheduling literature. Specifically, one technique is to show that the problem of minimizing weighted flow time can be reduced to solving a linear program where the feasible region is a polymatroid [9,10]. The polymatroid technique can be used to show that a system is *indexable* (the optimal policy is a simple priority policy) or even *decomposable* (job priorities do not depend on the other jobs in the system). This technique can be used to provide a greedy algorithm for obtaining the optimal policy. Unfortunately, the polymatroid technique is not straightforward to apply in our case.

Specifically, one generally proceeds by showing that a problem satisfies the so-called *generalized conservation laws* [9,10]. However, these conservation laws require scheduling policies which are *work-conserving*, meaning the total rate at

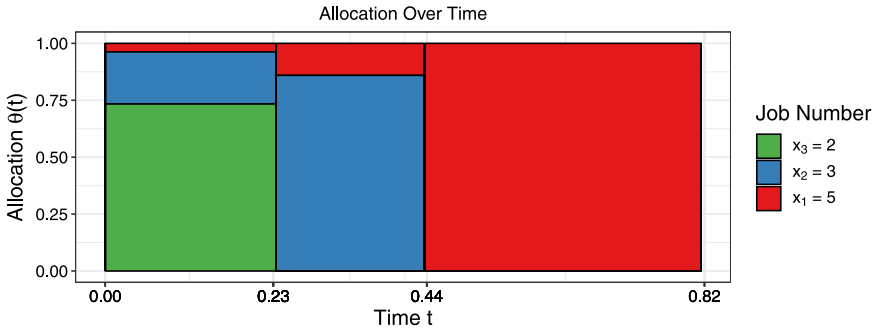

Fig. 3. Allocations made by the optimal allocation policy with respect to mean slowdown, heSRPT, completing a set of $M = 3$ jobs where the speedup function is $s(k) = k^{.5}$ and $N = 100$. Job 3 completes at time $t = .23$, job 2 completes at time $t = .44$ and job 1 completes at time $t = .82$. Jobs are finished in Shortest Job First order. Rather than allocating the whole system to the shortest job, heSRPT optimally shares the system between all active jobs.

which the system completes work remains constant over time. Allocation policies in our setting are not work conserving in general, and the allocations used by the optimal policy do not maintain a constant work rate (see Fig. 3). Additionally, due to the form of the speedup function $s(k) = k^p$, the region of achievable weighted flow times is not a polytope. Furthermore, the optimal policy we derive in Theorem 5 will show that our problem is not decomposable for the case of minimizing weighted flow time where weights favor small jobs. While there may exist a reduction of our problem that would allow one to establish some form of conservation laws, it is far from obvious what the "conserved quantity" would be for our problem which would allow for application of these techniques.

Prior work has investigated the related problem of scheduling flows in networks where the system capacity evolves over time [11]. However, this work derives an optimal policy only in the case when the work rates of each job are constrained to lie in a series of polymatroids that describe the system capacity at each moment in time. This assumption does not hold in our setting where the work rate of each job is defined by the speedup function $s(k) = k^p$.

Hence, standard techniques do not appear to be compatible with our goal of finding the optimal policy with respect to weighted flow time.

Contributions

In Section 3, we provide a complete overview of our results, but below we highlight our main contributions.

- To derive the optimal allocation function, we first develop a new technique in Section 4 to reduce the dimensionality of the optimization problem. This dimensionality reduction leverages two key properties of the optimal policy. First, in Section 4.1, we show that the optimal policy must complete jobs in shortest-job-first order. Then, in Section 4.2, we prove the scale-free property of the optimal policy which illustrates the optimal substructure of the optimal policy. These insights reduce the problem of solving $M!$ optimization problems of $O(M^2)$ variables and $O(M)$ constraints to the problem of solving one, unconstrained optimization problem of exactly M variables.
- In Section 4.3, we solve our simplified optimization problem to derive the first closed form expression for the optimal allocation of N servers to M jobs which minimizes weighted flow time when weights favor small jobs. At any moment in time t we define

  $$\boldsymbol{\theta}^*(t) = (\theta_1^*(t), \theta_2^*(t), \ldots, \theta_M^*),$$

  where $\theta_i^*(t)$ denotes the fraction of the N servers allocated to job i at time t. Note that $\theta_i^*(t)$ does not depend on N. Our optimal allocation balances the size-awareness of SRPT and the high efficiency of EQUI. We thus refer to our optimal policy as *high efficiency SRPT* (heSRPT) (see Theorem 5). We also provide a closed form expression for the weighted flow time under heSRPT (see Theorem 6).
- In Section 5.2, we numerically compare the optimal policies with respect to both mean slowdown and mean flow time to other heuristic policies proposed in the literature and show that our optimal policies significantly outperform these competitors.
- Finally, Section 5.3 turns to the online setting where jobs arrive over time. We propose an online version of heSRPT called *Adaptive-heSRPT* which uses the allocations from heSRPT to recalculate server allocations on every arrival and departure. Adaptive-heSRPT significantly outperforms competitor policies from the literature in simulation, often by an order of magnitude.

2. Prior work

Despite the prevalence of parallelizable data center workloads, it is not known, in general, how to optimally allocate servers to a set of parallelizable jobs. The state-of-the-art in production systems is to let the *user* decide their job's

allocation by reserving the resources they desire [12], and then to allow the system to pack jobs onto servers [13]. Users reserve resources greedily, leading to low system efficiency. Many workloads consist of *malleable* jobs which have the capability to change their degree of parallelism as they run [14,15]. We seek to improve the status quo by allowing the *system* to choose each job's server allocation at every moment in time.

The closest work to our results is [2], which considers jobs which follow a realistic speedup function and have known sizes. [2] also allows server allocations to change over time. [2] proposes and evaluates heuristic policies such as HELL and KNEE, but they make no theoretical guarantee about the performance of their policies.

Another related work, [16], assumes that jobs follow a concave speedup function and allows server allocations to change over time. However, unlike our work, [16] assumes that job sizes are *unknown* and are drawn from an exponential distribution. [16] concludes that EQUI is the optimal allocation policy. However, assuming unknown exponentially distributed job sizes is highly pessimistic since this means job sizes are impossible to predict, even as a job ages.

The performance modeling community has considered scheduling in multi-server systems with the goal of minimizing mean response time [17–21] or slowdown [22,23], but this work has not considered systems where a single job can run on multiple servers. An exception to this is the work on Fork-Join [24,25] in which a single job is composed of multiple tasks that may run in parallel. In Fork-Join models, however, the level of parallelism of jobs is fixed and is not chosen by the scheduling policy.

The SPAA/parallel community has studied the problem of allocating servers to jobs which follow arbitrary speedup functions in order to minimize flow time [26–29]. Like our paper, [26] considers jobs of known size while [27–29] consider jobs of unknown size. These papers use competitive analysis, which assumes that job sizes, arrival times, and even speedup functions are adversarially chosen. They conclude that a variant of EQUI is $(1+\epsilon)$-speed $O(1)$-competitive when job sizes are unknown [28]. When job sizes are known, a combination of SRPT and EQUI is $O(\log P)$-competitive, where P is the ratio of the largest job size to the smallest [26]. The SPAA community also considers minimizing mean slowdown with non-parallelizable jobs [30–32]. Sadly, O(1)-competitive policies for slowdown do not exist in general [6].

Instead of considering speedup functions, the SPAA community often models each job as a DAG of interdependent tasks [33–36]. This DAG encodes precedence constraints between tasks, and thus implies how parallelizable a job is at every moment in time. It is not clear how to optimally schedule a *single* DAG job on many servers [37]. The problem only gets harder if tasks are allowed to run on multiple servers [38,39]. Our hope is that by modeling parallelism using speedup functions, we can address problems that would be intractable in the DAG model.

Our model shares some similarities with coflow scheduling [40–44] where one allocates a continuously divisible resource, link bandwidth, to a set of flows to minimize mean flow time. However, here there is usually no explicit notion of a flow's speedup function. The most applicable work here is [45], which explores the tradeoff between efficiency and opportunistic scheduling in wireless networks. This work considers mean flow time, not mean slowdown. Section 5.1 shows how our results generalize for other metrics, such as mean flow time, in addition to mean slowdown.

3. Overview of our results

Our goal is to determine the optimal allocation of servers to jobs at every time, t, in order to minimize the weighted flow time of a set of M jobs of known size where weights favor small jobs. We derive a closed form for the optimal allocation function

$$\boldsymbol{\theta}^*(t) = (\theta_1^*(t), \theta_2^*(t), \ldots, \theta_M^*(t))$$

which defines the allocation for each job at any moment in time, t, that minimizes weighted flow time. We now provide an overview of the main theorems from this paper.

We begin by showing that the optimal policy completes jobs in *Shortest-Job-First* (SJF) order. The proof of this theorem uses an interchange argument to show that any policy which violates the SJF completion order can be improved. Because jobs follow a concave speedup function, a standard interchange proof fails. Our proof requires careful accounting, and interchanges servers between many jobs simultaneously. This claim is stated in Theorem 2.

Theorem 2 (*Optimal Completion Order*)**.** *The optimal policy completes jobs in Shortest-Job-First (SJF) order:*

$$M, M-1, M-2, \ldots, 1. \quad \square$$

Since jobs are completed in SJF order, we can also conclude that, at time t, the jobs left in the system are specifically jobs $1, 2, \ldots, m(t)$. Theorem 2 is proven in Section 4.1.

Theorem 2 does not rely on the specific form of the speedup function—it holds if $s(k)$ is any increasing, strictly concave function.

Besides completion order, the other key property of the optimal allocation that we exploit is the *scale-free property*. Our scale-free property states that for any job, i, job i's allocation relative to jobs completed after job i (jobs larger than job i) is constant throughout job i's lifetime. This property is stated formally in Theorem 3.

Theorem 3 (*Scale-free Property*). *For any job, i, there exists a constant c_i^* such that, for all $t < T_i^*$*

$$\frac{\theta_i^*(t)}{\sum_{j=1}^{i}\theta_j^*(t)} = c_i^*. \quad \square$$

The scale-free property has an intuitive interpretation. One can imagine the optimal policy as starting with some optimal allocation function $\boldsymbol{\theta}^*(0)$, which gives a $\theta_i^*(0)$ fraction of the servers to job i at time 0. When job M completes, it will leave behind a set of $\theta_M^*(0)\cdot N$ newly idle servers. The scale-free property tells us that the new value of the optimal allocation is found by reallocating these idle servers to each job $i < M$ in proportion to $\theta_i^*(0)$. That is, of the $\theta_M^*(0)\cdot N$ newly available servers, job i should receive

$$\frac{\theta_i^*(0)}{\sum_{j=1}^{M-1}\theta_j^*(0)} \cdot \theta_M^*(0)\cdot N$$

additional servers.

An example of the scale-free property in action can be seen in Fig. 3. Here, the optimal allocation policy gives all 3 jobs a non-zero allocation a time $t = 0$. When job 3 completes, its servers are reallocated to jobs 1 and 2. The allocations of job 1 and job 2 both increase, but the ratio of these jobs' allocations remains constant.

The scale-free property provides an optimal substructure that we exploit to reduce the dimensionality of our optimization problem. For example, consider the case where the optimal allocation function is known for a set of $M-1$ jobs, and we wish to additionally consider adding an Mth job. If we first decide how many servers to allocate to job M, the scale-free property tells us that stealing these servers from the other $M-1$ jobs in proportion to their existing allocations will produce the optimal policy. Hence, knowing the optimal policy for a set of $M-1$ jobs reduces the problem of finding the optimal policy for a set of M jobs to a single variable optimization problem. Theorem 3 is proven in Section 4.2.

Note that, while we state the scale-free property above for the optimal policy, the scale-free property actually holds for a more a general class of policies. For any given completion order of the jobs, the policy which minimizes weighted flow time while completing jobs in the given order will obey the scale-free property.

We are now finally ready to state Theorem 5, which provides the allocation function for the optimal allocation policy which minimizes weighted flow time when weights favors small jobs.

Theorem 5 (*Optimal Allocation Function*). *At time t, when $m(t)$ jobs remain in the system,*

$$\theta_i^*(t) = \begin{cases} \left(\frac{z(i)}{z(m(t))}\right)^{\frac{1}{1-p}} - \left(\frac{z(i-1)}{z(m(t))}\right)^{\frac{1}{1-p}} & 1 \le i \le m(t) \\ 0 & i > m(t) \end{cases}$$

where

$$z(k) = \sum_{i=1}^{k} w_i.$$

We refer to the optimal allocation policy which uses $\boldsymbol{\theta}^*(t)$ as heSRPT. $\square$

Theorem 5 is proven in Section 4.3. Given the optimal allocation function $\boldsymbol{\theta}^*(t)$, we can also explicitly compute the optimal weighted flow time for any set of M jobs. This is stated in Theorem 6.

Theorem 6 (*Optimal Weighted Flow Time*). *Given a set of M jobs of size $x_1 \ge x_2 \ge \cdots \ge x_M$, the weighted flow time, F^*, under the optimal allocation policy $\boldsymbol{\theta}^*(t)$ is given by*

$$F^* = \frac{1}{s(N)} \sum_{k=1}^{M} x_k \cdot \left[z(k)^{\frac{1}{1-p}} - z(k-1)^{\frac{1}{1-p}}\right]^{1-p}$$

where

$$z(k) = \sum_{i=1}^{k} w_i. \quad \square$$

Note that the optimal allocation policy biases its allocations towards shorter jobs, but does not give strict priority to these jobs in order to maintain the overall efficiency of the system. That is,

$$0 < \boldsymbol{\theta}_1^*(t) < \boldsymbol{\theta}_2^*(t) < \cdots < \boldsymbol{\theta}_{m(t)}^*(t).$$

This is illustrated in Fig. 3, where the smallest remaining jobs always get the largest allocations under the optimal policy. We refer to the optimal policy derived in Theorem 5 as *High Efficiency Shortest-Remaining-Processing-Time* or *heSRPT*.

Section 5.1 discusses how to apply Theorems 5 and 6 in order to optimize both the mean slowdown and mean flow time metrics. These applications are summarized in Corollary 1.

Corollary 1 (*Optimal Mean Slowdown and Mean Flow Time*)**.** *If we define*

$$w_i = \frac{1}{x_i/s(N)} \quad \text{and thus} \quad z(k) = \sum_{i=1}^{k} \frac{1}{x_i/s(N)},$$

Theorem 5 yields the optimal policy with respect to mean slowdown and Theorem 6 yields the optimal total slowdown.
If we define

$$w_i = 1 \quad \text{and thus} \quad z(k) = k,$$

Theorem 5 yields the optimal policy with respect to mean flow time and Theorem 6 yields the optimal total flow time. □

The remainder of Section 5 is devoted to the development of Adaptive-heSRPT, an online version of heSRPT.

4. Minimizing weighted flow time

4.1. The optimal completion order

To determine the optimal completion order of jobs, we first show that the optimal allocation function remains constant between job departures. This implies that the optimal allocation has M decision points where new allocations must be determined.

Theorem 1. *Consider any two times t_1 and t_2 where, WLOG, $t_1 < t_2$. Let $m^*(t)$ denote the number of jobs in the system at time t under the optimal policy. If $m^*(t_1) = m^*(t_2)$ then*

$$\boldsymbol{\theta}^*(t_1) = \boldsymbol{\theta}^*(t_2).$$

Proof. This proof is straightforward, and leverages the concavity of the speedup function, $s(k)$. See Appendix A. □

For the rest of the paper, we will therefore only consider allocation functions which change exclusively at departure times.

We can now show that the optimal policy will complete jobs in Shortest-Job-First order.

Theorem 2 (*Optimal Completion Order*)**.** *The optimal policy completes jobs in Shortest-Job-First (SJF) order:*

$$M, M-1, M-2, \ldots, 1.$$

Proof. Assume the contrary, that the optimal policy does not follow the SJF completion order. This means there exist two jobs α and β such that $x_\alpha < x_\beta$ but $T^*_\beta < T^*_\alpha$. Note that any number of completions may occur between job α and job β.

If the allocation to job α had always been at least as large as the allocation to job β, it is easy to see that job α would have finished first. Hence, we can identify some intervals of time where: (i) the allocations to job α and job β are constant because there are no departures and (ii) the allocation to job β is higher than the allocation to job α. Let I be the smallest set of disjoint intervals such that for every $[t_1, t_2) = i \in I$

$$\theta^*_\alpha(t) = \theta^*_\alpha(t') \text{ and } \theta^*_\beta(t) = \theta^*_\beta(t') \qquad \forall t, t' \in [t_1, t_2) \tag{1}$$

$$\theta^*_\beta(t_1) > \theta^*_\alpha(t_1). \tag{2}$$

Note also that on the interval $[T^*_\beta, T^*_\alpha)$, job α receives some positive allocation while job β receives no allocation because it is complete.

On some subset of these intervals we will perform an interchange which will reduce the weighted flow time of the jobs, producing a contradiction. Let P be the policy resulting from this interchange. Under P, we will fully exchange β and α's allocations on the interval $[T^*_\beta, T^*_\alpha)$. That is, for any $t \in [T^*_\beta, T^*_\alpha)$, $\theta^P_\beta(t) = \theta^*_\alpha(t)$ and $\theta^P_\alpha(t) = 0$. To offset this interchange we will reallocate servers from β to α on some subset of the intervals in I. We will argue that, after this interchange, $T^P_\beta = T^*_\alpha$ and $T^P_\alpha < T^*_\beta$, leading to a reduction in weighted flow time.

Let $W^P_i(t)$ be the total amount of work done by policy P on job i by time t. Let $W^P_i(t_1, t_2)$ be the amount of work done by policy P on job i on the interval $[t_1, t_2)$. That is,

$$W^P_i(t_1, t_2) = W^P_i(t_2) - W^P_i(t_1).$$

Let $\gamma = W^*_\alpha(T^*_\beta, T^*_\alpha)$. Since job α finished at exactly time T^*_α, we know that $W^*_\alpha(T^*_\beta) = x_\alpha - \gamma$. Similarly, we know that

$$W^*_\beta(T^*_\beta) = x_\beta.$$

We can see that policy P has the capacity to do up to γ work on job β on the interval $[T^*_\beta, T^*_\alpha)$. Specifically, if

$$W^P_\beta(T^*_\beta) = x_\beta - \gamma$$

then $T^P_\beta = T^*_\alpha$.

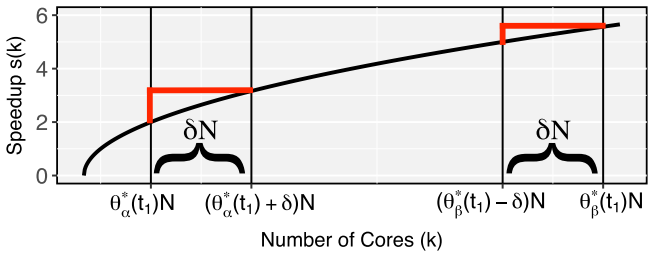

Fig. 4. An illustration of (3). Because the speedup function, s, is concave, its second derivative is negative. Hence, s increases less on the interval $[(\theta^*_\beta(t_1) - \delta)N, \theta^*_\beta(t_1)N]$ than on the interval $[\theta^*_\alpha(t_1)N, (\theta^*_\alpha(t_1) + \delta)N]$.

We will now reallocate servers from job β to job α on some of the intervals in I until $W^P_\beta(T^*_\beta) = x_\beta - \gamma$. On every such interval $i = [t_1, t_2) \in I$ we will choose

$$0 \le \delta_i \le \theta^*_\beta(t_1) - \theta^*_\alpha(t_1)$$

and let

$$\theta^P_\beta(t_1) = \theta^*_\beta(t_1) - \delta_i \qquad \text{and} \qquad \theta^P_\alpha(t_1) = \theta^*_\alpha(t_1) + \delta_i.$$

Decreasing job β's allocation on this interval has a well defined effect on $W^P_\beta(T^*_\beta)$. Namely, decreasing an allocation of a $\theta^*_\beta(t_1)$ fraction of the servers by δ_i decreases $W^P_\beta(T^*_\beta)$ by

$$(t_2 - t_1)(s(\theta^*_\beta(t_1) \cdot N) - s((\theta^*_\beta(t_1) - \delta_i) \cdot N)).$$

Note that, for any interval $i \in I$, $W^P_\beta(T^*_\beta)$ is a continuous, decreasing function of δ_i.

We will now iteratively apply the following interchange algorithm. Initialize $\delta_i = 0$, $\forall i \in I$. For each interval $i \in I$, try setting δ_i to be $\theta^*_\beta(i) - \theta^*_\alpha(i)$. If $W^P_\beta(T^*_\beta)$ is greater than $x_\beta - \gamma$ after this interchange, proceed to the next interval. If $W^P_\beta(T^*_\beta) < x_\beta - \gamma$ using this value of δ_i, there must exist some $0 < \delta_i < \theta^*_\beta(i) - \theta^*_\alpha(i)$ such that $W^P_\beta(T^*_\beta) = x_\beta - \gamma$ by the intermediate value theorem. Select this value of δ_i and terminate the algorithm.

To see this algorithm terminates, consider what would happen if

$$\delta_i = \theta^*_\beta(i) - \theta^*_\alpha(i) \qquad \forall i \in I.$$

In this case,

$$\theta^P_\beta(t) \le \theta^*_\alpha(t) \qquad \forall t \le T^*_\beta$$

which would imply that

$$W^P_\beta(T^*_\beta) \le W^*_\alpha(T^*_\beta) = x_\alpha - \gamma < x_\beta - \gamma.$$

Hence the algorithm will terminate before running out of intervals in I. By construction, we know that $T^P_\beta = T^*_\alpha$.

We now show that, after performing the interchange, $T^P_\alpha < T^*_\beta$. To see this, consider the total amount of work done on any interval in $i = [t_1, t_2) \in I$ before and after the interchange. Because only the allocations to α and β have changed, it is clear that

$$\sum_{j=1}^{m(t_1)} W^P_j(t_1, t_2) - W^*_j(t_1, t_2) = (W^P_\alpha(t_1, t_2) + W^P_\beta(t_1, t_2)) - (W^*_\alpha(t_1, t_2) + W^*_\beta(t_1, t_2)).$$

By the strict concavity of s, we know that the second derivative of s is negative. As illustrated in Fig. 4, and because $\theta^*_\beta(t_1) - \delta_i \ge \theta^*_\alpha(t_1)$ by construction, we know that

$$s((\theta^*_\alpha(t_1) + \delta_i)N) - s(\theta^*_\alpha(t_1)N) > s(\theta^*_\beta(t_1)N) - s((\theta^*_\beta(t_1) - \delta_i)N). \tag{3}$$

Factoring out N gives

$$s(\theta^*_\alpha(t_1) + \delta_i) - s(\theta^*_\alpha(t_1)) > s(\theta^*_\beta(t_1)) - s(\theta^*_\beta(t_1) - \delta_i)$$

and thus

$$\left(s(\theta^*_\alpha(t_1) + \delta_i) + s(\theta^*_\beta(t_1) - \delta_i)\right) - \left(s(\theta^*_\alpha(t_1)) + s(\theta^*_\beta(t_1))\right) > 0.$$

Multiplying both sides by $(t_2 - t_1)$, we see that

$$(W^P_\alpha(t_1, t_2) + W^P_\beta(t_1, t_2)) - (W^*_\alpha(t_1, t_2) + W^*_\beta(t_1, t_2)) > 0$$

since both allocations are constant on the interval $[t_1, t_2)$. That is, the total amount of work done on interval i has increased. Since this holds for all $i \in I$ we have that

$$\sum_{[t_1,t_2)\in I} (W_\alpha^P(t_1,t_2) + W_\beta^P(t_1,t_2)) - (W_\alpha^*(t_1,t_2) + W_\beta^*(t_1,t_2)) > 0$$

$$\sum_{[t_1,t_2)\in I} W_\alpha^P(t_1,t_2) - W_\alpha^*(t_1,t_2) - \sum_{[t_1,t_2)\in I} W_\beta^*(t_1,t_2) - W_\beta^P(t_1,t_2) > 0. \tag{4}$$

Again, by construction, we have decreased the amount of work done on job β during the intervals in I by exactly γ. That is,

$$\sum_{[t_1,t_2)\in I} W_\beta^*(t_1,t_2) - W_\beta^P(t_1,t_2) = \gamma$$

and hence by (4)

$$\sum_{[t_1,t_2)\in I} W_\alpha^P(t_1,t_2) - W_\alpha^*(t_1,t_2) > \gamma.$$

Recall that $W_\alpha^*(T_\beta^*) = x_\alpha - \gamma$. Thus

$$W_\alpha^P(T_\beta^*) - W_\alpha^*(T_\beta^*) = \sum_{[t_1,t_2)\in I} W_\alpha^P(t_1,t_2) - W_\alpha^*(t_1,t_2)$$

$$W_\alpha^P(T_\beta^*) - (x_\alpha - \gamma) > \gamma$$

$$W_\alpha^P(T_\beta^*) > x_\alpha.$$

This implies that, under policy P, job α finishes before time T_β^*.

We can thus conclude that after the interchange, policy P completes job β at exactly T_α^* and policy P completes job α at some time before T_β^*. All other jobs are unchanged by this interchange, and hence their flow times remain the same. Hence, it suffices to show that

$$w_\alpha T_\alpha^P + w_\beta T_\beta^P < w_\alpha T_\alpha^* + w_\beta T_\beta^*.$$

Because weights favor small jobs we have that $w_\beta < w_\alpha$. Furthermore, we have assumed that $T_\alpha^* > T_\beta^*$. Hence,

$$w_\beta \left(T_\alpha^* - T_\beta^*\right) < w_\alpha \left(T_\alpha^* - T_\beta^*\right)$$

$$w_\beta T_\alpha^* + w_\alpha T_\beta^* < w_\alpha T_\alpha^* + w_\beta T_\beta^*,$$

and thus by construction

$$w_\beta T_\beta^P + w_\alpha T_\alpha^P < w_\beta T_\alpha^* + w_\alpha T_\beta^* < w_\alpha T_\alpha^* + w_\beta T_\beta^*.$$

This implies that the policy P has a lower weighted flow time than the optimal policy, a contradiction. □

Definition 1. It will be useful to consider the rate at which weighted flow time is accrued in between departures. Because the optimal completion order is SJF, we know that job k will complete directly after job $k+1$, and that during the interval $[T_{k+1}^*, T_k^*)$, jobs 1 through k will be present in the system. Hence, we define $z(k)$ to be the rate at which total flow time is accrued during the interval $[T_{k+1}^*, T_k^*)$, where

$$z(k) = \sum_{i=1}^{k} w_i.$$

4.2. The scale-free property

The goal of this section is to characterize some optimal substructure of the optimal policy that will allow us to reduce the search space for the optimal policy. Hence, we now prove an interesting property of the optimal policy which we call the scale-free property. We will first need a preliminary lemma.

Lemma 1. *Consider an allocation function $\boldsymbol{\theta}(t)$ which, on some time interval $[0, T]$ leaves β fraction of the system unused. That is,*

$$\sum_{i=1}^{m(t)} \theta_i(t) = 1 - \beta \qquad \forall t \in [0, T].$$

The total work done on any job i by time T under $\boldsymbol{\theta}(t)$ is equivalent to the total work done on job i by time T under an allocation function $\boldsymbol{\theta}'(t)$ where

$$\boldsymbol{\theta}'(t) = \frac{\boldsymbol{\theta}(t)}{1-\beta} \qquad \forall t \in [0, T]$$

in a system that runs at $(1-\beta)^p$ times the speed of the original system (which runs at rate 1).

Proof. Is straightforward, see Appendix B. □

Using Lemma 1 we can characterize the optimal policy. Theorem 3 states that a job's allocation relative to the jobs larger than it will remain constant for the job's entire lifetime.

Theorem 3 (*Scale-free Property*)**.** *For any job, i, there exists a constant c_i^* such that, for all $t < T_i^*$*

$$\frac{\theta_i^*(t)}{\sum_{j=1}^{i} \theta_j^*(t)} = c_i^*.$$

Proof. We will prove this statement by induction on the overall number of jobs, M. First, note that the statement is trivially true when $M = 1$. It remains to show that if the theorem holds for $M = k$, then it also holds for $M = k + 1$.

Let $M = k + 1$ and let T_i^* denote the finishing time of job i under the optimal policy. Recall that the optimal policy finishes jobs according to the SJF completion order, so $T_{i+1}^* \leq T_i^*$. Consider a system which optimally processes k jobs, which WLOG are jobs $1, 2, \ldots, M-1$. We will now ask this system to process an additional job, job M. From the perspective of the original k jobs, there will be some constant portion of the system, θ_M^*, used to process job M on the time interval $[0, T_M^*]$. The remaining $1-\theta_M^*$ fraction of the system will be available during this time period. Just after time T_M^*, there will be at most k jobs in the system, and hence by the inductive hypothesis the optimal policy will obey the scale-free property on the interval $(T_M^*, T_1^*]$.

Consider the problem of minimizing the weighted flow time of the M jobs *given any fixed value* of θ_M^* such that the SJF completion order is obeyed. We can write the optimal weighted flow time of the M jobs, F^*, as

$$F^* = w_M T_M^* + \sum_{j=1}^{M-1} w_j T_j^*$$

where $w_M T_M^*$ is a constant. Clearly, optimizing weighted flow time in this case is equivalent to optimizing the weighted flow time for $M - 1 = k$ jobs with the added constraint that θ_M^* is unavailable (and hence "unused" from the perspective of jobs $M-1$ through 1) during the interval $[0, T_M^*]$. By Lemma 1, this is equivalent to having a system that runs at a fraction $\left(1-\theta_M^*\right)^p$ of the speed of a normal system during the interval $[0, T_M^*]$.

Thus, for some $d > 1$, we will consider the problem of optimizing weighted flow time for a set of k jobs in a system that runs at a speed $\frac{1}{d}$ times as fast during the interval $[0, T_M^*]$.

Let $F^*[x_{M-1}, \ldots, x_1]$ be the optimal weighted flow time of k jobs of size $x_{M-1} \ldots x_1$. Let $F^s[x_{M-1}, \ldots, x_1]$ be the weighted flow time of these jobs in a *slow system* which always runs $\frac{1}{d}$ times as fast as a normal system.

If we let T_i^s be the finishing time of job i in the slow system, it is easy to see that

$$T_i^* = \frac{T_i^s}{d}$$

since we can just factor out a d from the expression for the completion time of every job in the slow system. Hence, we see that

$$F^*[x_{M-1}, \ldots, x_1] = \frac{F^s[x_{M-1}, \ldots, x_1]}{d}$$

by the same reasoning. Clearly, then, the optimal allocation function in the slow system, $\boldsymbol{\theta}^s(t)$, is equal to $\boldsymbol{\theta}^*(t)$ at the respective departure times of each job. That is,

$$\boldsymbol{\theta}^*(T_j^*) = \boldsymbol{\theta}^s(T_j^s) \qquad \forall 1 \leq j \leq M-1.$$

We will now consider a *mixed* system which is "slow" for some interval $[0, T_M^*]$ that ends before T_{M-1}^s, and then runs at normal speed after time T_M^*. Let $F^Z[x_{M-1}, \ldots, x_1]$ denote the weighted flow time in this mixed system and let $\boldsymbol{\theta}^Z(t)$ denote the optimal allocation function in the mixed system. We can write

$$F^Z[x_{M-1}, \ldots, x_1] = z(k) \cdot T_M^* + F^*[x_{M-1} - \frac{s(\theta_{M-1}^Z)T_M^*}{d}, \ldots, x_1 - \frac{s(\theta_1^Z)T_M^*}{d}].$$

Similarly we can write

$$F^s[x_{M-1},\ldots,x_1] = z(k)\cdot T_M^* + F^s[x_{M-1}-\frac{s(\theta_{M-1}^s)T_M^*}{d},\ldots,x_1-\frac{s(\theta_1^s)T_M^*}{d}].$$

Let T_i^Z be the finishing time of job i in the mixed system under $\boldsymbol{\theta}^Z(t)$. Since $z(k)\cdot T_M^*$ is a constant not dependent on the allocation function, we can see that the optimal allocation function in the mixed system will make the same allocation decisions as the optimal allocation function in the slow system at the corresponding departure times in each system. That is,

$$\boldsymbol{\theta}^*(T_j^*) = \boldsymbol{\theta}^s(T_j^s) = \boldsymbol{\theta}^Z(T_j^Z) \qquad \forall 1 \le j \le M-1.$$

By the inductive hypothesis, the optimal allocation function in the slow system obeys the scale-free property. Hence, $\boldsymbol{\theta}^Z(t)$ also obeys the scale-free property for this set of k jobs given any fixed value of θ_M^*.

We now apply Lemma 1 again, multiplying $\boldsymbol{\theta}^Z(t)$ by $1-\theta_M^*$ on the interval $[0, T_M^*]$ to recover an allocation policy with θ_M^* unused servers on $[0, T_M^*]$. We call the resulting policy $\boldsymbol{\theta}^P(t)$. By Lemma 1, jobs $M-1,\ldots,1$ have the same residual sizes at time T_M^* in a mixed system under $\boldsymbol{\theta}^Z(t)$ as they do in a constant speed system under $\boldsymbol{\theta}^P(t)$. Hence, because the mixed system under $\boldsymbol{\theta}^Z(t)$ and the constant speed system under $\boldsymbol{\theta}^P(t)$ are identical after time T_M^*, the weighted flow time in these two systems is the same. Therefore, if $\boldsymbol{\theta}^Z(t)$ is optimal in the mixed system, $\boldsymbol{\theta}^P(t)$ minimizes the weighted flow time of jobs $M-1,\ldots,1$ for any fixed value of θ_M^* in a normal speed system. Since $\boldsymbol{\theta}^Z(t)$ obeys the scale-free property, $\boldsymbol{\theta}^P(t)$ obeys the scale-free property for jobs $1,\ldots,M-1$. Hence, for a set of $M=k+1$ jobs, given any allocation θ_M^* to job M on the interval $[0, T_M^*]$, the optimal allocations to the remaining $M-1$ jobs obey the scale-free property. Finally, the scale-free property is trivially satisfied for job M by allocating any fixed θ_M^* to job M on the interval $[0, T_M^*]$ and setting $c_i^* = \theta_M^*$. The optimal allocation function for processing the $M=k+1$ jobs therefore obeys the scale-free property. This completes the proof by induction. □

Definition 2. The scale-free property tells us that, under the optimal policy, job i's allocation relative to the jobs completed after it is a constant, c_i^*. Note that the jobs completed after job i are precisely the jobs with an initial size of at least x_i, since the optimal policy follows the SJF completion order. It will be useful to define an **optimal scale-free constant**, ω_i^*, for every job i, where for any $t < T_i^*$

$$\omega_i^* = \frac{1}{c_i^*} - 1 = \frac{\sum_{j=1}^{i-1}\theta_j^*(t)}{\theta_i^*(t)}.$$

Note that we define $\omega_1^* = 0$. Let $\boldsymbol{\omega}^* = (\omega_1^*, \omega_2^*, \ldots, \omega_M^*)$ denote the optimal scale-free constants corresponding to each job.

4.3. Finding the optimal allocation function

We will now make use of the scale-free property and our knowledge of the optimal completion order to find the optimal allocation function. We will consider the weighted flow time under any policy, P, which obeys the scale-free property and follows the SJF completion order. In Lemma 2, we derive an expression for the weighted flow time under the policy P as a function of the *scale-free constants* $\boldsymbol{\omega}^P$. In Theorem 4 we then minimize this expression to find the optimal scale-free constants and the optimal allocation function.

Lemma 2. *Consider a policy P which obeys the scale-free property and which completes jobs in shortest-job-first order. We define*

$$\omega_i^P = \frac{\sum_{j=1}^{i-1}\theta_j^P(t)}{\theta_i^P(t)} \qquad \forall 1 < i \le M,\ 0 \le t < T_i^P$$

and $\omega_1^P = 0$. We can then write the weighted flow time under policy P as a function of $\boldsymbol{\omega}^P = (\omega_1^P, \omega_2^P, \ldots, \omega_M^P)$ as follows

$$F^P(\boldsymbol{\omega}^P) = \frac{1}{s(N)}\sum_{k=1}^{M} x_k \cdot \left[z(k)s(1+\omega_k^P) - z(k-1)s(\omega_k^P)\right]$$

Proof. To analyze the total weighted flow time under P we will relate P to a simpler policy, P', which is much easier to analyze. We define P' to be

$$\theta_i^{P'} = \theta_i^{P'}(t) = \theta_i^P(0) \qquad \forall 1 \le i \le M.$$

Importantly, each job receives some initial optimal allocation at time 0 which does not change over time under P'. Since allocations under P' are constant we have that

$$T_k^{P'} = \frac{x_k}{s(\theta_k^{P'})}.$$

We can now derive equations that relate T_k^P to $T_k^{P'}$.

By Theorem 3, during the interval $[T_{k+1}^{P'}, T_k^{P'}]$,

$$\omega_i^P = \omega_i^{P'} \qquad \forall 1 \le i \le k.$$

Note that a fraction of the system $\sum_{i=k+1}^{M} \theta_i^{P'}$ is unused during this interval, and hence by Lemma 1, we have that

$$T_k^{P'} - T_{k+1}^{P'} = \frac{T_k^P - T_{k+1}^P}{s(\theta_1^{P'} + \cdots + \theta_k^{P'})}.$$

Let $\alpha_k = \frac{1}{s(\theta_1^{P'} + \cdots + \theta_k^{P'})}$ be the scaling factor during this interval.

If we define $x_{M+1} = 0$ and $T_{M+1}^P = 0$, we can express the weighted flow time under policy P, F^P, as

$$\begin{aligned} F^P &= z(M)T_M^P + z(M-1)(T_{M-1}^P - T_M^P) + \cdots + z(2)(T_2^P - T_3^P) + z(1)(T_1^P - T_2^P) \\ &= \sum_{k=1}^{M} z(k)(T_k^P - T_{k+1}^P) \\ &= \sum_{k=1}^{M} z(k)\frac{T_k^{P'} - T_{k+1}^{P'}}{\alpha_k}. \end{aligned}$$

We can now further expand this expression in terms of the job sizes, using the fact that $s(ab) = s(a) \cdot s(b)$, as follows:

$$\begin{aligned} F^P &= \sum_{k=1}^{M} z(k)\frac{\frac{x_k}{s(\theta_k^{P'}N)} - \frac{x_{k+1}}{s(\theta_{k+1}^{P'}N)}}{\alpha_k} \\ &= \frac{1}{s(N)}\sum_{k=1}^{M} z(k) \cdot \left[x_k s(1+\omega_k^{P'}) - x_{k+1}s(\omega_{k+1}^{P'})\right] \\ &= \frac{1}{s(N)}\sum_{k=1}^{M} x_k \cdot \left[z(k)s(1+\omega_k^{P}) - z(k-1)s(\omega_k^{P})\right] \end{aligned} \tag{5}$$

as desired. □

We now have an expression for the weighted flow time of any policy P which obeys the scale-free property and completes jobs in SJF order. Since the optimal policy obeys these properties, the choice of P which minimizes the above expression for weighted flow time must be the optimal policy. In Theorem 4 we find a closed form expression for the optimal scale-free constants.

Theorem 4 (*Optimal Scale-Free Constants*)**.** *The optimal scale-free constants are given by the expression*

$$\omega_k^* = \frac{1}{\left(\frac{z(k)}{z(k-1)}\right)^{\frac{1}{1-p}} - 1} \qquad \forall 1 < k \le M.$$

Proof. Consider a policy P which obeys the scale-free property and completes jobs in SJF order. Let $F^P(\boldsymbol{\omega}^P)$ be the expression for weighted flow time for P from Lemma 2. Our goal is to find a closed form expression for $\boldsymbol{\omega}^*$, the scale-free constants which minimize the above expression for weighted flow time.

A sufficient condition for finding $\boldsymbol{\omega}^*$ is that a policy completes jobs in SJF order *and* satisfies the following first-order conditions:

$$\frac{\partial F^P}{\partial \omega_k^P} = 0 \qquad \forall 1 \le k \le M$$

The second-order conditions are satisfied trivially. Note that solely satisfying the first order conditions is insufficient, because the resulting solution is not guaranteed to respect the SJF completion order. Luckily, we can show that any solution which satisfies the first-order conditions also completes jobs in SJF order. To see this, we will begin by finding the allocation function, $\boldsymbol{\Theta}^P(t)$, which satisfies the first-order conditions.

The first order conditions, obtained by differentiating (5), give

$$z(k)s'(1+\omega_k^P) - z(k-1)s'(\omega_k^P) = 0 \qquad \forall 1 \le k \le M$$

and hence

$$\omega_k^P = \frac{1}{\left(\frac{z(k)}{z(k-1)}\right)^{\frac{1}{1-p}} - 1} \qquad \forall 1 < k \le M$$

We can show that the values of $\Theta_k^P(t)$ are increasing in k (see Appendix C). That is, smaller jobs always have larger allocations than larger jobs under $\boldsymbol{\Theta}^P(t)$. This implies that $\boldsymbol{\Theta}^P(t)$ follows the SJF completion order, since a larger job cannot complete before a smaller job unless it receives a larger allocation for some period of time. $\boldsymbol{\Theta}^P(t)$ therefore satisfies the sufficient condition for optimality. We thus have found a closed form expression for the optimal scale free constants. That is,

$$\omega_k^* = \frac{1}{\left(\frac{z(k)}{z(k-1)}\right)^{\frac{1}{1-p}} - 1} \qquad \forall 1 < k \le M$$

as desired. □

We now derive the optimal allocation function in Theorem 5.

Theorem 5 (*Optimal Allocation Function*)**.** *At time t, when $m(t)$ jobs remain in the system,*

$$\theta_i^*(t) = \begin{cases} \left(\frac{z(i)}{z(m(t))}\right)^{\frac{1}{1-p}} - \left(\frac{z(i-1)}{z(m(t))}\right)^{\frac{1}{1-p}} & 1 \le i \le m(t) \\ 0 & i > m(t) \end{cases}$$

where

$$z(k) = \sum_{i=1}^{k} w_i.$$

We refer to the optimal allocation policy which uses $\boldsymbol{\theta}^*(t)$ as heSRPT.

Proof. We can now solve a system of equations to derive the optimal allocation function. Consider a time, t, when there are $m(t)$ jobs in the system. Since the optimal completion order is SJF, we know that the jobs in the system are specifically jobs $1, 2, \ldots, m(t)$. We know that the allocation to jobs $m(t)+1, \ldots, M$ is 0, since these jobs have been completed. Hence, we have that

$$\theta_1^* + \theta_2^* + \cdots + \theta_{m(t)}^* = 1.$$

Furthermore we have $m(t) - 1$ constraints provided by the expressions for $\omega_2^*, \omega_3^*, \ldots, \omega_{m(t)}^*$.

$$\omega_2^* = \frac{\theta_1^*(t)}{\theta_2^*(t)}, \omega_3^* = \frac{\theta_2^*(t) + \theta_1^*(t)}{\theta_3^*(t)}, \cdots, \omega_{m(t)}^* = \frac{\sum_{i=1}^{m(t)-1} \theta_i^*(t)}{\theta_{m(t)}^*(t)}.$$

These can be written as

$$\begin{aligned} \theta_1^*(t) &= \omega_2^* \theta_2^*(t) \\ \theta_1^*(t) + \theta_2^*(t) &= \omega_3^* \theta_3^*(t) \\ &\cdots \\ \theta_1^*(t) + \theta_2^*(t) + \cdots + \theta_{m(t)-1}^*(t) &= \omega_{m(t)}^* \theta_{m(t)} \\ \theta_1^* + \theta_2^* + \cdots + \theta_{m(t)}^* &= 1 \end{aligned}$$

and then rearranged as

$$\begin{aligned} \theta_{m(t)}^*(t) &= \frac{1}{1+\omega_{m(t)}^*} \\ \theta_{m(t)-1}^*(t) &= \frac{\theta_{m(t)}^*(t)\omega_{m(t)}^*}{1+\omega_{m(t)-1}^*} = \frac{\omega_{m(t)}^*}{(1+\omega_{m(t)-1}^*)(1+\omega_{m(t)}^*)} \\ &\cdots \\ \theta_2^*(t) &= \frac{\theta_3^*(t)\omega_3^*}{1+\omega_2^*} = \frac{\omega_3^* \cdots \omega_{m(t)}^*}{(1+\omega_2^*)\cdots(1+\omega_{m(t)}^*)} \end{aligned}$$

$$\theta_1^*(t) = \frac{\theta_2^*(t)\omega_2^*}{1+\omega_1^*} = \frac{\omega_2^* \cdots \omega_{m(t)}^*}{(1+\omega_1^*)\cdots(1+\omega_{m(t)}^*)}.$$

We can now plug in the known values of ω_i^* and find that

$$\boldsymbol{\theta}_i^*(t) = \left(\frac{z(i)}{z(m(t))}\right)^{\frac{1}{1-p}} - \left(\frac{z(i-1)}{z(m(t))}\right)^{\frac{1}{1-p}} \quad \forall 1 \le i \le m(t)$$

This argument holds for any $m(t) \ge 1$, and hence we have fully specified the optimal allocation function. □

Taken together, the results of this section yield an expression for weighted flow time under the optimal allocation policy. This expression is stated in Theorem 6.

Theorem 6 (*Optimal Weighted Flow Time*)**.** *Given a set of M jobs of size $x_1 \ge x_2 \ge \cdots \ge x_M$, the weighted flow time, F^*, under the optimal allocation policy $\boldsymbol{\theta}^*(t)$ is given by*

$$F^* = \frac{1}{s(N)} \sum_{k=1}^{M} x_k \cdot \left[z(k)^{\frac{1}{1-p}} - z(k-1)^{\frac{1}{1-p}}\right]^{1-p}$$

where

$$z(k) = \sum_{i=1}^{k} w_i.$$

5. Discussion and evaluation

We now examine the impact of the results shown in Section 4.

Section 5.1 shows how the results of Section 4 can be applied to provide the optimal policy with respect to mean slowdown and mean flow time.

We perform a numerical evaluation of heSRPT in Section 5.2. We compare heSRPT to several competitor policies from the literature and show that heSRPT not only outperforms these policies as expected, but often reduces slowdown by over 30%.

The above results apply in the common case where all jobs are present at time 0, but it is also interesting to consider the online case where jobs arrive over time. To address the online case, we use heSRPT as the basis of a heuristic policy. Section 5.3 compares our heuristic policy, Adaptive-heSRPT, to other heuristic allocation policies from the literature. Adaptive-heSRPT vastly outperforms other heuristic policies in this online case.

5.1. Applying heSRPT

The optimality results of Section 4.3 were stated in terms of any weighted flow time metric where weights favor small jobs. Specifically, this class of metrics can be used to find the optimal policy with respect to several popular metrics including mean slowdown and mean flow time. This is summarized in the following corollary.

Corollary 1 (*Optimal Mean Slowdown and Mean Flow Time*)**.** *If we define*

$$w_i = \frac{1}{x_i/s(N)} \quad \textit{and thus} \quad z(k) = \sum_{i=1}^{k} \frac{1}{x_i/s(N)},$$

Theorem 5 yields the optimal policy with respect to mean slowdown and Theorem 6 yields the optimal total slowdown.

If we define

$$w_i = 1 \quad \textit{and thus} \quad z(k) = k,$$

Theorem 5 yields the optimal policy with respect to mean flow time and Theorem 6 yields the optimal total flow time.

This corollary follows directly from the definitions of mean slowdown and mean flow time, respectively. Specifically, our results hold in these cases because the weights used in both cases favor small jobs.

One might ask which, if any, of our results hold in the case where weights do not favor small jobs. If weights do not favor small jobs it is easy to construct an example where one should not complete jobs in SJF order. Giving a job of any size a sufficiently high weight can cause it to complete first under the optimal policy. In addition to obviously contradicting Theorem 2, the SJF completion order was exploited in the proof of Theorem 4. It is worth noting, however, that regardless of the weights used, the optimal policy will obey the scale free property of Theorem 3.

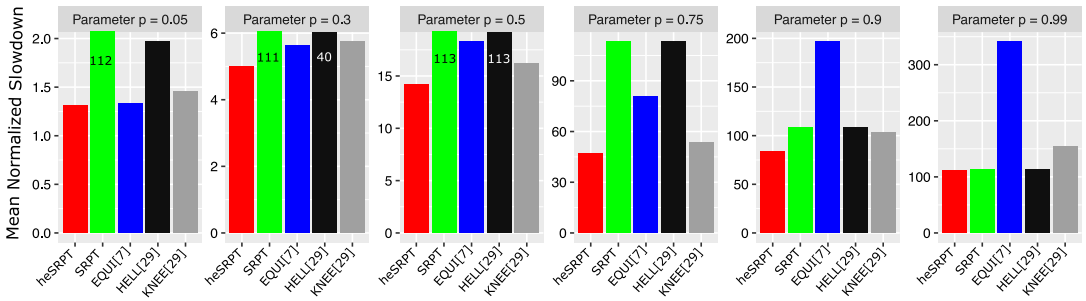

Fig. 5. Mean slowdown of heSRPT and allocation policies from the literature in the offline setting with all jobs present at time 0. Evaluation assumes $N = 1{,}000{,}000$ servers and $M = 500$ jobs whose sizes are Pareto($\alpha = .8$) distributed. Each graph shows mean slowdown for one value of the speedup parameter, p, where $s(k) = k^p$. heSRPT often dominates by over 30%.

5.2. Numerical evaluation: Offline setting

We now compare the optimal mean slowdown under heSRPT with the mean slowdown under policies from the literature.

Our comparison continues to assume that all jobs are present at time 0. While we have a closed form expression for the optimal mean slowdown under heSRPT, we wish to compare heSRPT to policies for which there is no closed form analysis. Hence, we perform a numerical analysis of heSRPT and several policies from the literature.

We compare heSRPT to the following list of competitor policies:

SRPT allocates the entire system to the single job with shortest remaining processing time. While SRPT is optimal when $p = 1$, we expect SRPT to perform poorly when jobs make inefficient use of servers. When all jobs are present at time 0, SRPT is equivalent to the RS policy [22] which minimizes mean slowdown in an M/G/1.

EQUI allocates an equal fraction of the servers to each job at every moment in time. EQUI has been analyzed using competitive analysis [27,28] in similar models of parallelizable jobs, and was shown to be optimal in expectation when job sizes are unknown and exponentially distributed [16]. Other policies such as *Intermediate-SRPT* [26] reduce to EQUI in our model where the number of jobs, M, is assumed to be less than the number of servers, N.

HELL is a heuristic policy proposed in [2] which, similarly to heSRPT, tries to balance system efficiency with biasing towards short jobs. HELL defines a job's efficiency to be the function $\frac{s(k)}{k}$. HELL then iteratively allocates servers. In each iteration, HELL identifies the job which can achieve highest ratio of efficiency to remaining processing time, and allocates to this job the servers required to achieve this maximal ratio. This process is repeated until all servers are allocated. While HELL is consistent with the goal of heSRPT, the specific ratio that HELL uses is just a heuristic.

KNEE is the other heuristic policy proposed in [2]. KNEE defines a job's *knee allocation* to be the number of servers for which the job's marginal reduction in run-time falls below some threshold, α. KNEE then iteratively allocates servers. In each iteration, KNEE identifies the job with the lowest knee allocation and gives this job its knee allocation. This process repeats until all servers are allocated. Because there is no principled way to choose this α, we perform a brute-force search of the parameter space and present the results given the best α parameter we found. Hence, results for KNEE are an optimistic prediction of KNEE's performance.

We evaluate heSRPT and the competitor policies in a system of $N = 1{,}000{,}000$ servers and $M = 500$ jobs whose sizes are Pareto($\alpha = .8$) distributed. The speedup parameter p is set to values between 0.05 and 0.99. Fig. 5 shows the results of this analysis.

heSRPT outperforms every competitor policy in every case as expected. When p is low, EQUI is within 1% of heSRPT, but EQUI is over 3× worse than heSRPT when $p = 0.99$. Conversely, when $p = 0.99$, SRPT is nearly optimal. However, SRPT is an order of magnitude worse than heSRPT when $p = 0.05$. While HELL performs similarly to SRPT in most cases, it is 50% worse than optimal when $p = 0.05$. The KNEE policy is the best of the competitor policies that we consider. In the worst case, when $p = 0.99$, KNEE is roughly 50% worse than heSRPT. However, these results for KNEE required brute-force tuning of the allocation policy. We examined several other job size distributions and in all cases we saw similar trends.

In Appendix D we compare these competitor policies to the optimal policy with respect to mean flow time.

5.3. Numerical evaluation: Online setting

While we have shown that heSRPT provides the optimal mean slowdown in the case where all jobs are present at time 0, minimizing the slowdown of a stream of arriving parallelizable jobs remains an open theoretical problem.

To understand the added complexity of the online setting, consider the problem of allocating servers to two jobs at time 0, while knowing that an arrival will occur at time 1. The optimal policy might try to complete one job quickly and allow the arrival to compete for servers with the second job, or it might try and complete both of the original jobs before time 1 in order to reduce the flow time of the arriving job. In general, it is no longer sufficient to compare two jobs and

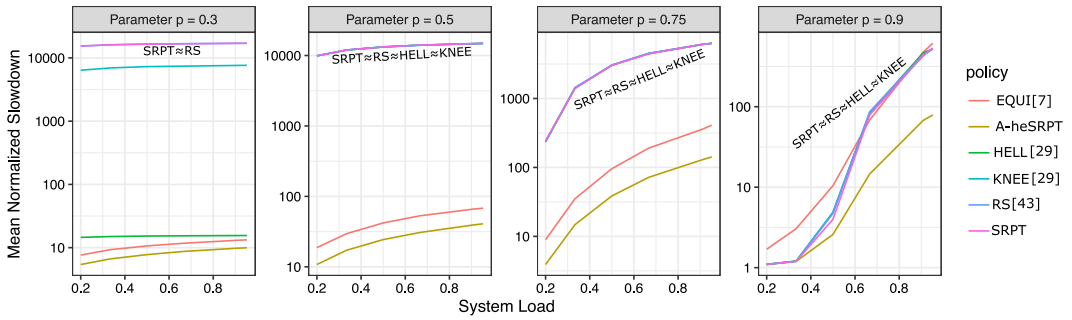

Fig. 6. Mean slowdown of A-heSRPT and policies from the literature in the online setting. Simulations assume $N = 10{,}000$ servers and a Poisson arrival process. Job sizes are Pareto($\alpha = 1.5$) distributed. Each graph shows mean slowdown for one value of the speedup parameter, p, where $s(k) = k^p$. A-heSRPT often dominates by an order of magnitude.

decide which to complete first. One must also decide how many jobs to complete before the next arrival. This prevents the results in this paper from generalizing cleanly to the online case.

We therefore use simulation to compare the performance of heSRPT to the competitor policies described above in an online setting where jobs arrive over time. For this comparison, we define a heuristic online policy, inspired by heSRPT, which we refer to as Adaptive-heSRPT (A-heSRPT). A-heSRPT recalculates the allocation to each job every time a job departs from *or arrives to* the system, using the heSRPT allocations as defined in Theorem 5. All of the other competitor policies we examine generalize to online setting in a similar way, reevaluating their allocation decisions on each arrival or departure.

For the sake of completeness, we also examine an additional competitor policy in the online case, the **RS** policy [22] which is known to minimize mean slowdown online for non-parallelizable jobs in an M/G/1 system [23]. The RS policy allocates all servers to the job with the lowest product of remaining size (R) times initial size (S). RS is identical to SRPT if all jobs are present at time 0, but must be considered separately in the online case.

Fig. 6 shows that A-heSRPT outperforms all competitor policies in every case, *often by several orders of magnitude*. When jobs are highly parallelizable, policies such as SRPT and RS which aggressively favor small jobs can occasionally compete with A-heSRPT. However, as load increases and the allocation decision becomes more complex, A-heSRPT vastly outperforms the competition. Conversely, when jobs have a low level of parallelizability (low p), SRPT, RS, HELL and KNEE fail to maintain the efficiency of the system and perform poorly. EQUI, which maximizes efficiency but does not favor small jobs, follows the opposite trend—EQUI performs decently when p is low, but it performs poorly when p is high. A-heSRPT is the only policy which is able to handle high and low values of p over a range of system loads.

6. Conclusion

Modern data centers largely rely on users to decide how many servers to use to run their jobs. When jobs are parallelizable, but follow a sublinear speedup function, allowing users to make allocation decisions can lead to a highly inefficient use of resources. We propose to instead have the system control server allocations, and we derive the first optimal allocation policy which minimizes mean slowdown for a set of M parallelizable jobs. Our optimal allocation policy, heSRPT, leads to significant improvement over existing allocation policies suggested in the literature. The key to heSRPT is that it finds the correct balance between overall system efficiency and favoring short jobs. We derive an expression for the optimal allocation, at each moment in time, in closed form.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Mor Harchol-Balter is supported by: NSF-CMMI-1938909, NSF-CSR-1763701, NSF-XPS-1629444, and a Google 2020 Faculty Research Award. Benjamin Berg is supported by a Facebook Graduate Fellowship

Appendix A. Proof of Theorem 1

Theorem 1. *Consider any two times t_1 and t_2 where, WLOG, $t_1 < t_2$. Let $m^*(t)$ denote the number of jobs in the system at time t under the optimal policy. If $m^*(t_1) = m^*(t_2)$ then*

$$\boldsymbol{\theta}^*(t_1) = \boldsymbol{\theta}^*(t_2).$$

Proof. Consider any time interval $[t_1, t_2]$ during which no job departs the system, and hence $m^*(t_1) = m^*(t_2)$. Assume for contradiction that under the optimal policy $\boldsymbol{\theta}^*(t_1) \neq \boldsymbol{\theta}^*(t_2)$. To produce a contradiction, we will show that the weighted flow time under the optimal policy can be improved by using a *constant* allocation during the time interval $[t_1, t_2]$. The constant allocation we use is equal to the average value of $\boldsymbol{\theta}^*(t)$ during the interval $[t_1, t_2]$.

Specifically, consider the allocation function $\overline{\boldsymbol{\theta}^*(t)}$ where

$$\overline{\boldsymbol{\theta}^*(t)} = \begin{cases} \frac{1}{t_2 - t_1} \int_{t_1}^{t_2} \boldsymbol{\theta}^*(T) dT & \forall t_1 \leq t \leq t_2 \\ \boldsymbol{\theta}^*(t) & \text{otherwise.} \end{cases}$$

Note that $\overline{\boldsymbol{\theta}^*(t)}$ is constant during the interval $[t_1, t_2]$. Furthermore, because $\sum_{i=1}^{m(t)} \theta_i^*(t) \leq 1$ for any time t, $\sum_{i=1}^{m(t)} \overline{\theta_i^*(t)} \leq 1$ at every time t as well, and $\overline{\boldsymbol{\theta}^*(t)}$ is therefore a feasible allocation function. Because the speedup function, s, is a strictly concave function, Jensen's inequality gives

$$\int_{t_1}^{t_2} s\left(\overline{\boldsymbol{\theta}^*(t)}\right) dt \geq \int_{t_1}^{t_2} s\left(\boldsymbol{\theta}^*(t)\right) dt$$

and for at least one job, i, the above inequality will be strict. Hence, the residual size of each job under the allocation function $\overline{\boldsymbol{\theta}^*(t)}$ at time t_2 is at most the residual size of that job under $\boldsymbol{\theta}^*(t)$ at time t_2, and the residual time of job i is strictly less under the new policy at time t_2. This guarantees that no jobs will have its flow time increased by this transformation and at least one job will have its flow time decreased by this transformation. We have thus constructed a policy with a lower weighted flow time than the optimal policy, a contradiction. □

Appendix B. Proof of Lemma 1

Lemma 1. *Consider an allocation function $\boldsymbol{\theta}(t)$ which, on some time interval $[0, T]$ leaves β fraction of the system unused. That is,*

$$\sum_{i=1}^{m(t)} \theta_i(t) = 1 - \beta \qquad \forall t \in [0, T].$$

The total work done on any job i by time T under $\boldsymbol{\theta}(t)$ is equivalent to the total work done on job i by time T under an allocation function $\boldsymbol{\theta}'(t)$ where

$$\boldsymbol{\theta}'(t) = \frac{\boldsymbol{\theta}(t)}{1 - \beta} \qquad \forall t \in [0, T]$$

in a system that runs at $(1 - \beta)^p$ times the speed of the original system (which runs at rate 1).

Proof. Consider the policy $\boldsymbol{\theta}'(t)$ in a system which runs at $(1 - \beta)^p = s(1 - \beta)$ times the speed of the original system on $[0, T]$. The service rate of any job in this system on $[0, T]$ is given by

$$s(1 - \beta) \cdot s(\boldsymbol{\theta}'(t) \cdot N) = s((1 - \beta) \cdot \boldsymbol{\theta}'(t) \cdot N) = s(\boldsymbol{\theta}(t) \cdot N).$$

Since, at any time $t' \in [0, T]$, the service rate for any job i is the same under $\boldsymbol{\theta}(t)$ as it is under $\boldsymbol{\theta}'(t)$ in a system which is $(1 - \beta)^p$ times as fast, the same amount of work is done on job i by time T in both systems. □

Appendix C. Proof of Lemma 3

Lemma 3. *Let $\boldsymbol{\Theta}^P(t)$ be the allocation function which satisfies the following first-order conditions:*

$$\frac{\partial F^P}{\partial \omega_k^P} = 0 \qquad \forall 1 \leq k \leq M.$$

Then for any t, $\Theta_k^P(t)$ is increasing in k for $1 \leq k \leq m(t)$.

Proof. Following the same argument as Theorem 5, we can see that the expression for $\boldsymbol{\Theta}^P(t)$ is

$$\Theta_i^P(t) = \left(\frac{z(i)}{z(m(t))}\right)^{\frac{1}{1-p}} - \left(\frac{z(i-1)}{z(m(t))}\right)^{\frac{1}{1-p}} \qquad \forall 1 \leq i \leq m(t).$$

Note that $\frac{1}{1-p} > 1$, so $i^{\frac{1}{1-p}}$ is convex in i. We know that

$$z(i) - z(i-1) = w_i$$

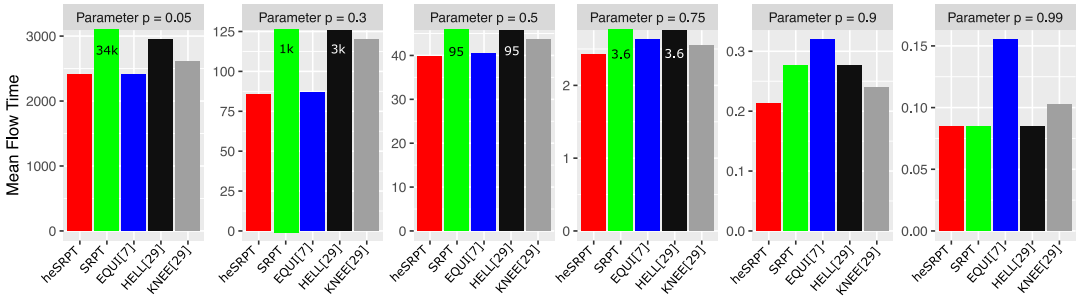

Fig. C.7. A comparison of the optimal mean flow time under heSRPT to other allocation policies found in the literature. Each policy is evaluated on a system of $N = 1{,}000{,}000$ servers and a set of $M = 500$ jobs whose sizes are drawn from a Pareto distribution with shape parameter .8. Each graph shows the mean flow time under each policy with various values of the speedup parameter, p, where the speedup function is given by $s(k) = k^p$. heSRPT outperforms every competitor by at least 30% in at least one case.

and

$$z(i+1) - z(i) = w_{i+1}.$$

Furthermore, $w_{i+1} \geq w_i$. Hence, by convexity,

$$\begin{aligned} z(i)^{\frac{1}{1-p}} - z(i-1)^{\frac{1}{1-p}} &< (z(i) + w_i)^{\frac{1}{1-p}} - (z(i-1) + w_i)^{\frac{1}{1-p}} \\ &< (z(i) + w_{i+1})^{\frac{1}{1-p}} - z(i)^{\frac{1}{1-p}} \\ &< z(i+1)^{\frac{1}{1-p}} - z(i)^{\frac{1}{1-p}} \end{aligned}$$

This implies that $\Theta_i^p(t)$ is increasing in i for $1 \leq i \leq m(t)$. □

Appendix D. Numerical evaluation for mean flow time

Section 5.1 describes how to use heSRPT to obtain the optimal policy with respect to mean flow time. Because the competitor policies in Section 5.2 were designed to minimize mean flow time, we now compare the optimal mean flow time under heSRPT to the mean flow time of these competitor policies. The competitor policies remain unchanged from their descriptions in Section 5, and the conditions of our analysis (numbers of servers, job size distribution, speedup function) remain the same as is Section 5 as well.

Fig. C.7 shows the results of our analysis. We see that the results with respect to mean flow time are generally similar to the results of Section 5. heSRPT once again outperforms every competitor policy by at least 30% in at least one case. Hence, the results shown in Section 5 are not only caused by the fact that the competitor policies optimize for a different metric. Even when comparing policies with respect to mean flow time, each of the competitor policies can be far from optimal.

References

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al., Tensorflow: a system for large-scale machine learning, in: OSDI, Vol. 16, 2016, pp. 265–283.
[2] S. Lin, M. Paolieri, C. Chou, L. Golubchik, A model-based approach to streamlining distributed training for asynchronous SGD, in: MASCOTS, IEEE, 2018.
[3] S. Muthukrishnan, R. Rajaraman, A. Shaheen, J. Gehrke, Online scheduling to minimize average stretch, in: FOCS, IEEE, 1999.
[4] David B. Jackson, Heather L. Jackson, Quinn O. Snell, Simulation based HPC workload analysis, in: IPDPS 2001, IEEE, 2000, pp. 8–pp.
[5] Donald R. Smith, A new proof of the optimality of the shortest remaining processing time discipline, Oper. Res. 26 (1) (1978) 197–199.
[6] Nikhil Bansal, Kirk Pruhs, Server scheduling in the Lp norm: a rising tide lifts all boat, in: STOC, 2003, pp. 242–250.
[7] X. Zhan, Y. Bao, C. Bienia, K. Li, PARSEC3.0: A multicore benchmark suite with network stacks and SPLASH-2X, SIGARCH 44 (2017) 1–16.
[8] M.D. Hill, M.R. Marty, Amdahl's law in the multicore era, Computer 41 (2008) 33–38.
[9] David D. Yao, Dynamic scheduling via polymatroid optimization, in: IFIP International Symposium on Computer Performance Modeling, Measurement and Evaluation, Springer, 2002, pp. 89–113.
[10] Dimitris Bertsimas, José Niño-Mora, Conservation laws, extended polymatroids and multiarmed bandit problems; a polyhedral approach to indexable systems, Math. Oper. Res. 21 (2) (1996) 257–306.
[11] Bilal Sadiq, Gustavo De Veciana, Balancing SRPT prioritization vs opportunistic gain in wireless systems with flow dynamics, in: International Teletraffic Congress, IEEE, 2010.
[12] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, J. Wilkes, Large-scale cluster management at Google with Borg, in: EUROSYS, ACM, 2015.
[13] Runtian Ren, Xueyan Tang, Clairvoyant dynamic bin packing for job scheduling with minimum server usage time, in: SPAA, ACM, 2016, pp. 227–237.

[14] A. Gupta, B. Acun, O. Sarood, L. Kalé, Towards realizing the potential of malleable jobs, in: HiPC, IEEE, 2014.
[15] M.C Cera, Y. Georgiou, O. Richard, N. Maillard, P. Navaux, Supporting malleability in parallel architectures with dynamic CPUSETs mapping and dynamic MPI, in: ICDCN, Springer, 2010, pp. 242–257.
[16] B. Berg, J.P. Dorsman, M. Harchol-Balter, Towards optimality in parallel scheduling, ACM POMACS 1 (2) (2018).
[17] M. Harchol-Balter, A. Scheller-Wolf, A.R. Young, Surprising results on task assignment in server farms with high-variability workloads, SIGMETRICS 37 (2009) 287–298.
[18] R. Nelson, T. Philips, An approximation for the mean response time for shortest queue routing with general interarrival and service times, Perform. Eval. (1993).
[19] Y. Lu, Q. Xie, G. Kliot, A. Geller, J.R. Larus, A. Greenberg, Join-Idle-Queue: A novel load balancing algorithm for dynamically scalable web services, Perform. Eval. 68 (2011) 1056–1071.
[20] V. Gupta, M. Harchol-Balter, K. Sigman, W. Whitt, Analysis of join-the-shortest-queue routing for web server farms, Perform. Eval. (2007).
[21] J.N. Tsitsiklis, K. Xu, On the power of (even a little) centralization in distributed processing, SIGMETRICS 39 (2011) 121–132.
[22] Adam Wierman, Mor Harchol-Balter, Takayuki Osogami, Nearly insensitive bounds on SMART scheduling, SIGMETRICS 33 (1) (2005) 205–216.
[23] Esa Hyytiä, Samuli Aalto, Aleksi Penttinen, Minimizing slowdown in heterogeneous size-aware dispatching systems, SIGMETRICS 40 (1) (2012) 29–40.
[24] S.-S. Ko, R.F. Serfozo, Response times in M/M/s fork-join networks, Adv. Appl. Probab. 36 (2004) 854–871.
[25] W. Wang, M. Harchol-Balter, H. Jiang, A. Scheller-Wolf, R. Srikant, Delay asymptotics and bounds for multitask parallel jobs, Queueing Syst. (2019).
[26] Sungjin Im, Benjamin Moseley, Kirk Pruhs, Eric Torng, Competitively scheduling tasks with intermediate parallelizability, TOPC 3 (1) (2016) 4.
[27] J. Edmonds, Scheduling in the dark, Theoret. Comput. Sci. (1999).
[28] J. Edmonds, K. Pruhs, Scalably scheduling processes with arbitrary speedup curves, in: SODA '09, ACM, 2009, pp. 685–692.
[29] K. Agrawal, J. Li, K. Lu, B. Moseley, Scheduling parallelizable jobs online to minimize the maximum flow time, SPAA '16, ACM, 2016, pp. 195–205.
[30] S. Anand, Naveen Garg, Amit Kumar, Resource augmentation for weighted flow-time explained by dual fitting, in: SODA, SIAM, 2012, pp. 1228–1241.
[31] Benjamin Moseley, Kirk Pruhs, Cliff Stein, The complexity of scheduling for p-norms of flow and stretch, in: IPCO, Springer, 2013.
[32] Carl Bussema, Eric Torng, Greedy multiprocessor server scheduling, Oper. Res. Lett. 34 (4) (2006) 451–458.
[33] Robert D. Blumofe, Charles E. Leiserson, Scheduling multithreaded computations by work stealing, J. ACM 46 (5) (1999) 720–748.
[34] E. Bampis, D. Letsios, G. Lucarelli, A note on multiprocessor speed scaling with precedence constraints, in: SPAA, ACM, 2014, pp. 138–142.
[35] P. Bodík, I. Menache, J. Naor, J. Yaniv, Brief announcement: deadline-aware scheduling of big-data processing jobs, in: SPAA, ACM, 2014.
[36] Girija J. Narlikar, Guy E. Blelloch, Space-efficient scheduling of nested parallelism, TOPLAS 21 (1) (1999) 138–173.
[37] R. Chowdhury, V. Ramachandran, F. Silvestri, B. Blakeley, Oblivious algorithms for multicores and networks of processors, J. Parallel Distrib. Comput. 73 (7) (2013) 911–925.
[38] Jianzhong Du, Joseph Y.-T. Leung, Complexity of scheduling parallel task systems, SIAM J. Discrete Math. 2 (4) (1989) 473–487.
[39] Chi-Yeh Chen, An improved approximation for scheduling malleable tasks with precedence constraints via iterative method, TPDS (2018).
[40] H. Jahanjou, E. Kantor, R. Rajaraman, Asymptotically optimal approximation algorithms for coflow scheduling, in: SPAA, ACM, 2017, pp. 45–54.
[41] M. Chowdhury, Y. Zhong, I. Stoica, Efficient coflow scheduling with varys, in: SIGCOMM, Vol. 44, No. 4, ACM, 2014, pp. 443–454.
[42] Zhen Qiu, Cliff Stein, Yuan Zhong, Minimizing the total weighted completion time of coflows in datacenter networks, in: SPAA, ACM, 2015, pp. 294–303.
[43] Mehrnoosh Shafiee, Javad Ghaderi, Brief announcement: a new improved bound for coflow scheduling, in: SPAA, ACM, 2017, pp. 91–93.
[44] Samir Khuller, Manish Purohit, Brief announcement: Improved approximation algorithms for scheduling co-flows, in: SPAA, ACM, 2016, pp. 239–240.
[45] S. Aalto, A. Penttinen, P. Lassila, P. Osti, On the optimal trade-off between SRPT and opportunistic scheduling, in: SIGMETRICS, ACM, 2011.