Optimal Resource Allocation for Elastic and Inelastic Jobs

Benjamin Berg*
Carnegie Mellon University
Pittsburgh, PA, USA
bsberg@cs.cmu.edu

Mor Harchol-Balter†
Carnegie Mellon University
Pittsburgh, PA, USA
harchol@cs.cmu.edu

Benjamin Moseley‡
Carnegie Mellon University
Pittsburgh, PA, USA
moseleyb@andrew.cmu.edu

Weina Wang
Carnegie Mellon University
Pittsburgh, PA, USA
weinaw@cs.cmu.edu

Justin Whitehouse§
Carnegie Mellon University
Pittsburgh, PA, USA
jwhiteho@andrew.cmu.edu

ABSTRACT

Modern data centers are tasked with processing heterogeneous workloads consisting of various classes of jobs. These classes differ in their arrival rates, size distributions, and job parallelizability. With respect to parallelizability, some jobs are *elastic*, meaning they can parallelize linearly across any number of servers. Other jobs are *inelastic*, meaning they can only run on a single server. Although job classes can differ drastically, they are typically forced to share a single cluster. When sharing a cluster among heterogeneous jobs, one must decide how to allocate servers to each job at every moment in time. In this paper, we design and analyze allocation policies which aim to minimize the *mean response time* across jobs, where a job's response time is the time from when it arrives until it completes.

We model this problem in a stochastic setting where each job may be elastic or inelastic. Job sizes are drawn from exponential distributions, but are unknown to the system. We show that, in the common case where elastic jobs are larger on average than inelastic jobs, the optimal allocation policy is *Inelastic-First*, giving inelastic jobs preemptive priority over elastic jobs. We obtain this result by introducing a novel sample path argument. We also show that there exist cases where *Elastic-First* (giving priority to elastic jobs) performs better than Inelastic-First. We provide the first analysis of mean response time under both Elastic-First and Inelastic-First by leveraging techniques for solving high-dimensional Markov chains.

KEYWORDS

scheduling, parallelism, elastic jobs, stochastic modeling

*This author is supported in part by the Facebook Graduate Fellowship
†This author is supported in part by NSF-CMMI-1938909, NSF-XPS-1629444, and NSF-CSR-1763701
‡This author is supported in part by a Google Research Award, an Infor Research Award, a Carnegie Bosch Junior Faculty Chair and NSF grants CCF-1824303, CCF-1845146, CCF-1733873 and CMMI-1938909
§The author is supported in part by the National Science Foundation Graduate Research Fellowship Program under grant DGE 1745016.

SPAA '20, July 15–17, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-6935-0/20/07.
https://doi.org/10.1145/3350755.3400265

ACM Reference Format:
Benjamin Berg, Mor Harchol-Balter, Benjamin Moseley, Weina Wang, Justin Whitehouse. 2020. Optimal Resource Allocation for Elastic and Inelastic Jobs. In *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '20), July 15–17, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3350755.3400265

1 INTRODUCTION

1.1 Motivation

Modern data centers are tasked with processing astonishingly diverse workloads on a common set of shared servers [49]. These jobs differ not only in their resource requirements on a single server, but also in how effectively they scale across multiple servers [13]. For instance, a simple client query may run on a single server and complete in just milliseconds. Conversely, a data intensive job may run for hours even when parallelized across dozens of servers [41]. The challenge facing system architects is to build data centers which, in light of this heterogeneity, achieve low *response time* – the time from when a job enters the system until it completes.

The state-of-the-art in many data centers is to allow users to specify their own server requirements, and then over-provision the system. By always ensuring that idle servers are available, system designers avoid having to make tough resource allocation decisions while users always receive the resources they request. Unfortunately, these over-provisioned systems are expensive to build and waste resources [18]. Most large-scale data centers, for example, run at an average utilization of less than 20% [49].

To try and reduce this waste, many cluster scheduling systems have been proposed in the literature [13, 30, 38, 39, 41, 47]. These scheduling systems aim to maintain low response times without having to over-provision the system. One way to achieve this goal [13, 47] is to have the system scheduler determine resource allocations rather than allowing users to reserve resources. While these schedulers often work well in practice, none of them offer theoretical response time guarantees.

1.2 The Problem

We propose a simple model of heterogeneous traffic running in a multiserver data center. Our goal is to design a resource allocation policy which dynamically allocates servers to jobs in order to minimize the mean response time across jobs. We assume jobs are preemptible, and that an allocation policy can change a job's server

allocation over time. In particular, we will consider a system of k servers which processes jobs that arrive over time from a workload consisting of two distinct job classes. The first class of jobs, which we call *elastic*, consists of jobs which can run on any *set* of servers at any moment in time. We assume that elastic jobs experience a *speedup factor* proportional to the number of servers they run on. That is, an elastic job which completes in 2 seconds on a single server would complete in 1 second on 2 servers, or .5 seconds on 4 servers. The second class of jobs, which we refer to as *inelastic*, consists of jobs which are not parallelizable. While an inelastic job can run on any server, it can only run on a single server at any moment in time. A resource allocation policy must determine, at every moment in time, how to allocate servers to each job in system, both elastic and inelastic.

In practice each job also has some amount of inherent work associated with it. This inherent work, which we call a job's *size*, determines how long it takes to complete the job on a single server. We assume that job sizes are unknown to the system, but are drawn independently for each job from an exponential distribution. To further model the heterogeneity of a workload, we allow elastic and inelastic job sizes to be drawn from two different exponential distributions, with rates μ_E and μ_I respectively.

Even given the simplicity of the model above, devising an *optimal* scheduling policy is non-trivial. For instance, consider the problem of dividing k servers between one elastic job and one inelastic job which are both of size 1. On the one hand, we know that completing jobs quickly benefits mean response time, so one might think to run the elastic job on all k servers before running the inelastic job. On the other hand, this schedule leaves $k-1$ servers idle while the inelastic job completes. We could thus have created a more *efficient* schedule by running the elastic and inelastic jobs simultaneously, giving $k-1$ servers to the elastic job and 1 server to the inelastic job. It turns out that the more efficient schedule is optimal in this case, but in general, a good scheduling policy must balance the trade-off between completing elastic jobs quickly and preventing long periods of low server utilization. This question becomes even more complex if the elastic and inelastic jobs have different sizes.

1.3 Elastic and Inelastic Jobs in the Real World

It is common to find systems which use a shared set of servers to process both elastic and inelastic jobs. Typically in such settings the elastic jobs have more inherent work than the inelastic jobs. For example, consider a cluster which must process a stream of many MapReduce jobs [12]. From the cluster's point of view, this workload produces a stream of *map stages* and *reduce stages*. Map stages (elastic) are designed to be parallelized across any number of servers and do a large amount of processing. Reduce stages (inelastic) are inherently sequential and do much less total work than a map stage. As another example, modern machine learning frameworks [41] advocate the use of a single platform for both the training and serving of models. Training jobs (elastic) are large, requiring large data sets and many training epochs. Distributed training methods such as distributed stochastic gradient descent are also designed to scale out across an arbitrary number of nodes [36]. Once a model has been trained, serving the model (inelastic), which consists of feeding a computed model a single data point in order to retrieve a single prediction, is done sequentially and requires comparatively little processing power.

It is less common for elastic jobs to be smaller than inelastic jobs in practice, given the overhead involved in writing parallel code. If the amount of inherent work required for a job is small to begin with, system developers may not choose to add the additional data structures and synchronization mechanisms required to make the job elastic. One exception is HPC workloads. In this setting, there are often both malleable jobs (elastic) [20] and jobs with hard requirements (inelastic). While malleable jobs are designed to run on any number of cores, jobs with hard requirements demand a fixed number of cores. It is unclear which class of jobs we would expect to involve more inherent work.

The model presented in this paper is flexible enough to capture all of the above examples.

1.4 Why stochastic analysis?

There has been a sizable amount of work considering the problem of scheduling jobs onto k parallel servers. The vast majority of this work considers *only inelastic jobs of known sizes*, and focuses on worst-case analysis. Given the optimality of the Shortest-Remaining-Processing-Time (SRPT) policy in the degenerate case where $k = 1$ [48], one might hope that SRPT is also optimal in the multiserver case where $k \geq 2$. Specifically, one might consider a policy called SRPT-k [19] which always runs the k jobs with the shortest remaining processing times. Unfortunately, [35] shows that SRPT-k can be arbitrarily far from optimal. In fact, SRPT-k has a competitive ratio of $\Theta(\log \min(p, \frac{n}{k}))$ where n is the number of jobs and p is the ratio of the maximum job size to the minimum job size. Additionally, [35] shows that this competitive ratio is a tight lower bound – no online algorithm can do better in the worst case. Using speed augmentation, SRPT-k is known to be constant competitive with $1+\epsilon$ speed for any constant $\epsilon > 0$ [9, 17].

More recently, some work has examined the case of scheduling *parallelizable* jobs of known sizes onto k parallel servers. This work assumes that each job has an arbitrary speedup curve which dictates its running time as a function of the number of servers on which it runs. Again using worst-case analysis, [16] shows how to achieve an $O(\frac{1}{\epsilon})$-competitive ratio using $(1+\epsilon)$-speed servers. Without using resource augmentation, [31] provides an algorithm with a competitive ratio of $O(\log p)$, where again p is the ratio of the largest job size to the smallest job size. This competitive ratio essentially matches the known worst-case lower bound for the problem.

The above results suggest that, without resource augmentation, there is little room to improve the worst-case performance of scheduling policies for parallelizable jobs. This is because the aforementioned lower bounds for worst-case scheduling directly apply to the case where jobs are given speedup curves. However, from the point of view of system designers, this problem remains unsolved! In particular, a competitive ratio of $\log p$ [31] can be arbitrarily high when job sizes span a wide range, which is common in practice. Thus, a $\log p$-competitive algorithm could be impractical. Additionally, the results in [16] use an elegant algorithm that is interesting theoretically, but the algorithm is difficult to implement due to frequent context switches. The problem is that results like [16, 31] and others (see Section 3) perform badly on adversarial cases which are

uncommon in practice. We therefore propose shifting to *stochastic analysis* which discounts the impact of these adversarial cases. By considering a stochastic analysis, there is the potential to reveal new algorithmic insights into the problem. It could even be possible to find online algorithms that are optimal in expectation.

There has been recent work aimed at allocating servers to parallelizable jobs in a stochastic setting in order to minimize mean response time [7]. However, this line of work is in an early stage. Specifically, [7] only considers the case where all jobs are homogeneous with respect to job size and job speedup. While [7] is able to derive the optimal policy in this simpler case, they explicitly note the complexity of handling even just two different classes of jobs. In particular, the problem of allocating to servers to both elastic and inelastic jobs in a stochastic setting remains completely open. Although [7] presents some approximate numerical analysis of the case where jobs are heterogeneous, the techniques used are computationally intensive and offer no guarantees of accuracy.

1.5 Our Contributions

This paper addresses the problem of allocating servers to both elastic and inelastic jobs. Section 2 introduces our stochastic model of elastic and inelastic jobs of unknown sizes which arrive over time to a system composed of k servers. Using this model, we then present the following results:

- We propose two natural server allocation policies which aim to minimize the mean response time across jobs. First, the *Elastic-First* policy gives strict preemptive priority to elastic jobs and aims to minimize mean response time by maximizing the rate at which jobs depart the system. Second, the *Inelastic-First* policy gives strict preemptive priority to inelastic jobs. By deferring elastic work for as long as possible, Inelastic-First maximizes system efficiency. It is not immediately obvious if either of these policies is optimal, or which policy is better.
- We show in Section 4.1 that if elastic and inelastic jobs follow the same exponential size distribution, Inelastic-First is optimal with respect to mean response time. This argument uses precedence relations to show that deferring elastic work increases the long run efficiency of the system.
- Next, in Section 4.2, we show that in the case where elastic jobs are larger on average than inelastic jobs, Inelastic-First is optimal with respect to mean response time. This requires the introduction of a novel sample path argument. Our key insight is that Inelastic-First minimizes the expected amount of inelastic work in the system as well as the expected *total* work in the system. As long as elastic jobs are larger than inelastic jobs on average, this suffices for minimizing mean response time.
- In the case where elastic jobs are smaller on average than inelastic jobs, Inelastic-First is no longer optimal. We illustrate this via a counterexample in Section 4.3 which shows that Elastic-First can outperform Inelastic-First. In order to determine when Elastic-First outperforms Inelastic-First, we perform the first analysis of both the Elastic-First and Inelastic-First allocation policies in Section 5. This analysis leverages recent techniques for solving high-dimensional Markov chains. Our analytical results match simulation.
- For the sake of completeness, we also consider the case where job sizes are known and jobs arrive at time 0. Using standard dual-fitting techniques for worst-case analysis (e.g. [4, 5]), we show SRPT-k is a 4-approximation for the objective of minimizing mean response time. This demonstrates the need for stochastic modeling and analysis. Indeed, the stochastic setting yields optimality results without resorting to approximations. Due to lack of space, this final contribution is saved for the Appendix A.

2 OUR MODEL

We consider a model where jobs arrive over time to a system of k identical servers. Each job has an associated amount of inherent work which we refer to as the *job size*. We assume that each of the k servers processes jobs with a rate of 1 unit of work per second. Hence, a job's size is equal to its running time on a single server. We assume that job sizes are unknown to the system, and are drawn from exponential distributions.

Each job may be either *elastic* or *inelastic*. We assume that elastic jobs arrive according to a Poisson process with rate λ_E, and that elastic job sizes are drawn independently from an exponential distribution with rate μ_E. Similarly, inelastic jobs arrive independently according to a Poisson process with rate λ_I, and inelastic job sizes are drawn independently from an exponential distribution with rate μ_I. We let S_E and S_I be random variables representing the sizes of an elastic job or an inelastic job respectively.

Every elastic job can run on *any number* of servers at any moment in time. Because each server processes work at rate 1, n servers process work at a rate of n units of work per second. Hence,

an elastic job of size x completes in x seconds on a single server but completes in $\frac{x}{n}$ seconds on n servers.

By contrast, inelastic jobs can run on *at most one* server at any moment in time.

We note that all of the results presented in this paper hold equally if inelastic jobs can run on up to some fixed number of servers, $C < k$. We can simply renormalize our allocation policies to consider allocating in units of $\frac{k}{C}$ servers. After renormalizing, inelastic jobs can once again receive up to one unit of allocation while elastic jobs can receive any number of units of allocation. While our results do not depend on the value of C, we consider the case where $C = 1$ for the sake of simplifying our notation.

An *allocation policy*, π, must determine how many servers to allocate to each job at any moment in time t. Specifically, π can increase or decrease the allocation to a particular job as it runs. We assume that servers are capable of time sharing, and thus an allocation policy may allocate a fractional number of servers to any job. For any $n \in \mathbb{R}_{\geq 0}$, we assume that an allocation of n servers processes work at a rate of n units of work per second. At any moment in time, t, an allocation policy can allocate at most 1 server to each inelastic job, and at most k servers in total.

We can model this system under any policy π as a continuous time Markov chain where each state denotes the number of elastic and inelastic jobs currently in the system. That is, we define a continuous time Markov process $\{(N_I^\pi(t), N_E^\pi(t)) : t \geq 0\}$ where

$$(N_I^\pi(t), N_E^\pi(t)) \in \mathbb{Z}^2_{\geq 0}, \qquad \forall t \geq 0.$$

We define $N_I^\pi(t)$ to be the number of inelastic jobs in system at time t and we define $N_E^\pi(t)$ to be the number of elastic jobs in system at time t. We let the state $(N_I^\pi(t), N_E^\pi(t)) = (i, j)$ denote that there are i inelastic jobs and j elastic jobs currently in the system.

Because job sizes are exponential and arrivals occur according to a Poisson process, at any moment in time t, the distributions of remaining job sizes and the distributions of times until the next arrival for each job class can be fully specified by the numbers of inelastic jobs and elastic jobs in the system. Hence, we will only consider policies which are *stationary* and *deterministic*, meaning the policy π makes the same allocation decision at every time t, given that the system is in state (i, j). Specifically, we define $\pi_I(i, j)$ to be the number of servers allocated to inelastic jobs in state (i, j) under policy π, and we define $\pi_E(i, j)$ to be the number of servers allocated to elastic jobs in state (i, j) under policy π. Note that

$$\pi_I(i,j) \le i \qquad \forall (i,j) \in \mathbb{Z}^2_{\ge 0},$$

$$\pi_E(i,j) \le k \cdot \mathbb{1}_{\{j>0\}} \qquad \forall (i,j) \in \mathbb{Z}^2_{\ge 0},$$

and

$$\pi_I(i,j) + \pi_E(i,j) \le k \qquad \forall (i,j) \in \mathbb{Z}^2_{\ge 0}.$$

In general, $\pi_I(i, j) + \pi_E(i, j)$ could be less than k if there are not a sufficient number of jobs to use all k servers, or if π chooses to idle servers instead of allocating them to an eligible job.

We refer to a policy π as *work conserving* if and only if, in any state (i, j),

$$\pi_I(i,j) + \pi_E(i,j) \ge i,$$

and

$$\pi_I(i,j) + \pi_E(i,j) \ge k \cdot \mathbb{1}_{\{j>0\}}.$$

That is, π never leaves servers idle if there is an eligible job in the system. In Appendix B we show that there exists an optimal policy which is also work conserving. It therefore suffices to only consider work conserving policies throughout our analysis.

We define the *system load*, ρ to be

$$\rho \equiv \frac{\lambda_I}{k\mu_I} + \frac{\lambda_E}{k\mu_E}. \tag{1}$$

In Appendix C we show that for any work conserving policy, π, $(N_I^\pi(t), N_E^\pi(t))$ is an ergodic Markov chain if $\rho < 1$. Because there exists an optimal work conserving policy, (1) is necessary for stability under *any* policy π'. We therefore only consider the regime where $\rho < 1$.

We will track several stochastic quantities in our system. We define the total number of jobs in the system, $N^\pi(t)$, as

$$N^\pi(t) = N_I^\pi(t) + N_E^\pi(t).$$

We also define $W^\pi(t)$ to be the *total work* in the system under policy π at time t, where total work is the sum of the remaining sizes of all jobs in the system. Similarly, we let $W_E^\pi(t)$ and $W_I^\pi(t)$ be the *total elastic work* and the *total inelastic work* in the system under policy π at time t. These quantities are the sums of the remaining sizes of all elastic or inelastic jobs respectively. When referring to the corresponding steady-state quantities, we omit the argument t.

We define the random variable T^π to be the *response time* of a job which arrives to the system in steady-state under policy π. Here, the response time of a job is the time from when the job arrives until it is completed (i.e. its remaining size is 0). Our goal is to find the policy which minimizes the *mean response time*.

We will investigate the performance of two allocation policies, *Elastic-First* (EF) and *Inelastic-First* (IF). EF gives strict preemptive priority to elastic jobs, and processes jobs in first-come-first-serve (FCFS) order within each job class. That is, in any state (i, j) where $j > 0$, EF allocates all k servers to the elastic job with the earliest arrival time. In any state (i, j) where $j = 0$, EF allocates one server to each inelastic job, in FCFS order, until either all jobs have received a server or all k servers have been allocated. By contrast, IF gives strict preemptive priority to inelastic jobs while processing jobs in FCFS order within each job class. Under IF, in any state (i, j) where $i < k$, one server is allocated to each inelastic job and the remaining $k - i$ servers are allocated to the elastic job with the earliest arrival time if there is one. In any state (i, j) where $i \ge k$, all k servers are allocated to the inelastic jobs with the k earliest arrival times.

3 PRIOR WORK

Although many real-world systems are tasked with allocating servers to heterogeneous workloads, these systems do not allocate servers optimally in order to minimize the mean response time across jobs. Most large-scale cluster schedulers allow *users* to explicitly reserve the number of servers they want [30, 38, 39, 41, 49], only allowing the system to choose the placement of each job onto its requested number of servers. Some systems have proposed allowing the *system* to determine the number of servers allocated to each job [13, 37, 47] in order to reduce response times. However, these systems rely on heuristics and do not make theoretical guarantees.

In the theoretical literature, the closest work to the results presented in this paper come from the stochastic performance modeling community. In particular, [7] develops a model of jobs whose sizes are drawn from an exponential distribution and which receive a sublinear speedup from being allocated additional servers. However, [7] only provides optimality results when jobs are homogeneous, following a *single* speedup function and a *single* exponential size distribution. We emphasize that our paper is the first ever to consider more than one speed-up curve in the setting with stochastic arrivals over time and stochastic job sizes. Essentially all other work in the stochastic community has considered non-parallelizable inelastic jobs. Much of the prior work has been limited to scheduling jobs on a single server [11]. While there has certainly been work on scheduling in stochastic multiserver systems (e.g [1, 6, 19, 23, 24, 29]), this literature assumes that a job occupies at most one server at a time (that is, all jobs are inelastic). One notable model that considers jobs that run on multiple servers is the queueing model motivated from MapReduce [32, 42, 50]. This work assumes that each job consists of a set of pieces that can be processed on different machines at the same time. These pieces can be processed in any order and, critically, a job only completes when all of its pieces have completed. This model can only be analyzed exactly when the number of servers is $k = 2$.

In the worst case setting, the problem of scheduling jobs on identical parallel servers was introduced in [40] and has been considered extensively. However, in the classical version of the problem, all jobs are considered to be inelastic. Given inelastic jobs with known sizes and known release times, [35] shows a tight lower bound on the competitive ratio of $\Theta(\log \min(p, \frac{n}{k}))$ where n is the number of jobs and p is the ratio of the maximum job size to the minimum job

size. The policy which achieves the best competitive ratio is SRPT-k, which at every moment schedules the k jobs with the smallest remaining processing times.

Several prior works have considered scheduling parallelizable jobs in the worst-case setting. The speed-up curve model was first addressed by [14]. The best result for mean response time is [16] which gave a constant competitive algorithm with minimal speed augmentation. This paper introduced the influential LAPS scheduling algorithm that has been used in a variety of settings [15, 21]. The work of [31] considers the problem without speed augmentation and gives a $O(\log p)$ competitive algorithm with mild assumptions on the speed-up curves. Recently, there has been a line of work on the Directed-Acyclic-Graph (DAG) model for parallelism. Here a constant competitive algorithm with $1+\epsilon$ speed augmentation is known [3]. The work of [2] gave an $O(1)$-speed, $O(1)$-competitive algorithm for mean response time that is practical, using minimal preemptions. However, the best possible competitive ratio in any model with release times is still lower bounded by $\Theta(\log \min(p, \frac{n}{k}))$, since all jobs could be inelastic in the worst case.

4 OPTIMALITY RESULTS

The following sections establish two results. First, we show that if $\mu_I \geq \mu_E$, then IF is optimal for minimizing mean response time. Second, we show that if $\mu_I < \mu_E$, then IF is not necessarily optimal.

In Section 4.1, we consider the special case where $\mu_I = \mu_E$. In this case where we have homogeneous sizes, analysis is particularly easy. Unfortunately, the technique used to demonstrate optimality, which is based on the notion of precedence relations in continuous time Markov chains, does not extend to when $\mu_I \neq \mu_E$.

In Section 4.2, we consider the case where $\mu_I \geq \mu_E$. Here, we consider a novel sample path argument which allows us to demonstrate the optimality of IF.

Lastly, in section 4.3, we consider the case where $\mu_I < \mu_E$. Here, we construct a very simple example demonstrating that IF is not optimal in this environment. Furthermore, in this example, we show the policy EF actually outperforms IF. We do not know what policy is optimal in this regime.

4.1 Optimality when $\mu_I = \mu_E$

We first consider the case where $\mu_I = \mu_E$. In this case, IF is optimal with respect to minimizing mean response time. As stated in Section 1.2, the optimal policy should balance the trade-off between completing jobs quickly and preserving system efficiency. When $\mu_I = \mu_E$, IF maximizes system efficiency without reducing the overall completion rate of jobs. We argue this formally in Theorem 1 by leveraging a result from [7].

Theorem 1. IF *is optimal with respect to minimizing mean response time when* $\mu_I = \mu_E$.

Proof. Consider the server allocations made by a policy π in any state (i, j). We define the *total rate of departures* under π in the state (i, j) to be

$$d^{\pi}(i,j) = \pi_E(i,j) \cdot \mu_E + \pi_I(i,j) \cdot \mu_I.$$

Following the terminology of [7], we say that π is in the class of *GREEDY* policies if

$$d^{\pi}(i,j) = \max_{\pi'} d^{\pi'}(i,j) \qquad \forall (i,j) \in \mathbb{Z}^2_{\geq 0}.$$

That is, a policy is in GREEDY if it achieves the maximal rate of departures in every state.

Furthermore, [7] defines a class of policies called *GREEDY**. A policy is said to be in GREEDY* if, in every state (i, j), it minimizes the number of servers allocated to elastic jobs while still maximizing the total rate of departures. That is, a policy π is in GREEDY* iff

$$\pi_E(i,j) = \min_{\pi' \in GREEDY} \pi'_E(i,j) \qquad \forall (i,j) \in \mathbb{Z}^2_{\geq 0}.$$

It is shown in [7], using precedence relations, that for any policy $\pi \in$ GREEDY*

$$\mathbb{E}[T^{\pi}] = \min_{\pi' \in GREEDY} \mathbb{E}[T^{\pi'}]. \tag{2}$$

To leverage this result, we note that when $\mu_I = \mu_E$ in our model, a policy is in GREEDY if and only if it does not idle servers unnecessarily.

We now argue that IF, which is non-idling, must be in GREEDY*. In states where IF allocates zero servers to elastic jobs, $\texttt{IF}_E(i,j)$ is clearly minimal. In any state (i, j) where $\texttt{IF}_E(i,j) > 0$, servers cannot be reallocated from elastic jobs to inelastic jobs, since all i inelastic jobs must already be in service. Hence, reducing $\texttt{IF}_E(i,j)$ in this case results in a policy which is not in GREEDY. $\texttt{IF}_E(i,j)$ is therefore minimal amongst GREEDY policies in any state (i, j), and IF is in GREEDY*.

We show in Appendix B that there exists an optimal policy which is non-idling. Hence, when $\mu_I = \mu_E$, there is an optimal policy in GREEDY. This implies that there must be an optimal policy in GREEDY* as well. Because any policy in GREEDY* has the same rate of departures of elastic and inelastic jobs in every state (i, j), every policy in GREEDY* has the same mean response time. Thus, IF, which is in GREEDY*, is optimal with respect to mean response time.

□

Why the prior argument does not generalize

Unfortunately, the results of [7] do not extend to the case where $\mu_I \neq \mu_E$. In particular, the proof of (2) uses a precedence relation between any two states $(i, j-1)$ and $(i-1, j)$. This claim essentially states that a policy π in state (i, j) would perform better by transitioning to state $(i-1, j)$ than it would by transitioning to state $(i, j-1)$. In the case where $\mu_I = \mu_E$, this makes perfect intuitive sense. In this case, both states $(i-1, j)$ and $(i, j-1)$ contain the same amount of expected total work. Hence, it is better to be in state $(i-1, j)$, which benefits from having an additional elastic job. Consider how this intuition changes when $\mu_I > \mu_E$. In this case, state $(i, j-1)$ has less expected total work, but state $(i-1, j)$ has more expected elastic work. It turns out that the precedence relation shown in [7] no longer holds when $\mu_I \neq \mu_E$. Moreover, even if the precedence relations were to hold when $\mu_I > \mu_I$, [7] would yield that GREEDY* is optimal amongst GREEDY policies, not optimal amongst all policies. We must therefore devise a new argument to reason about the optimal allocation policy when elastic and inelastic jobs follow different size distributions.

4.2 Optimality when $\mu_I \geq \mu_E$

We will show IF is optimal in the more general case of $\mu_I \geq \mu_E$. While our goal is to minimize mean response time, we note that via Little's Law [25], it suffices to minimize the mean total number of jobs in the system. [1]

First, we start by defining a class of policies $\mathcal{P}$ which serve inelastic jobs on a first-come-first-serve (FCFS) basis; elastic jobs can be served in any order. In more detail, a policy π is said to be in class $\mathcal{P}$ if the following hold true:

(1) π is work-conserving.
(2) π serves inelastic jobs in FCFS order. In particular, if π allocates N servers to inelastic jobs at time t (N may be fractional, and there may be more than N inelastic jobs in the systems), the allocation must give $\lfloor N \rfloor$ servers to the $\lfloor N \rfloor$ inelastic jobs with the earliest arrival times. If there is a remaining fraction of a server, it may then be allocated to the inelastic job with the next earliest arrival time.

Clearly, IF $\in \mathcal{P}$.

Road map: Theorem 2 argues that we only need to compare IF to policies in $\mathcal{P}$. Specifically, $\mathcal{P}$ contains some optimal policy that minimizes the mean number of jobs in system and mean response time.

Next, in Theorem 3 we present a novel sample path argument which shows that IF has stochastically less work in the system than any policy in $\mathcal{P}$. We will directly leverage this fact to show that, out of all policies $\pi \in \mathcal{P}$, IF has the least expected inelastic work in system and also the least expected total work in system.

Finally, In Theorem 5 we show that, of all policies in $\mathcal{P}$, IF minimizes the expected number of jobs in system. Thus, by Little's Law, IF is optimal with respect to mean response time.

Analysis. We now present Theorem 2.

Theorem 2. *The class $\mathcal{P}$ contains a policy π which minimizes both mean response time and mean number of jobs in system. Specifically*

$$\mathbb{E}\left[N^{\pi}\right] = \min_{\pi'} \left\{\mathbb{E}\left[N^{\pi'}\right]\right\},$$

and

$$\mathbb{E}\left[T^{\pi}\right] = \min_{\pi'} \left\{\mathbb{E}\left[T^{\pi'}\right]\right\},$$

where N^{π} is the total number of jobs in the system in steady-state under policy π, and T^{π} is the response time of a job in the system under π in steady-state.

Proof. Recall that we will consider only stationary, deterministic, work-conserving policies which make allocation decisions based on state (i, j). Let π be a stationary, deterministic, work-conserving policy with the minimal mean number of jobs in system. Figure 1 shows the transition rates out of state (i, j) under π.

We see that the transition rates out of the current state (i, j) under policy π depend solely on the number of servers allocated to each type of job. Thus, neither the order in which we serve the jobs nor how many jobs of each type are running matter. In particular, we can construct a policy π' such that, for any state (i, j),

$$\pi_I(i, j) = \pi'_I(i, j) \qquad \text{and} \qquad \pi_E(i, j) = \pi'_E(i, j)$$

and π' serves inelastic jobs in FCFS order. The policy π' has the same Markov chain as π, so the expected numbers of jobs in system under π and π' are identical. Because π is work-conserving, π' is also work-conserving. Hence, π' is in $\mathcal{P}$ and achieves the minimal mean number of jobs in system. □

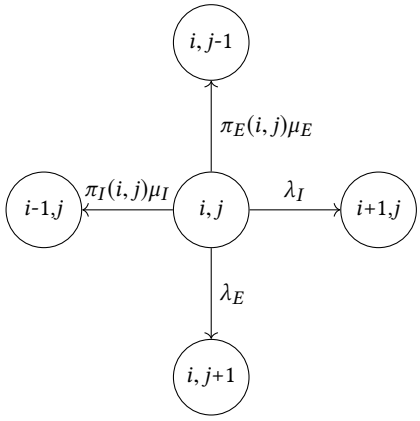


Figure 1: The Markov chain $(N_I^{\pi}(t), N_E^{\pi}(t))$ for a stationary, deterministic, work-conserving allocation policy, π.

The power of Theorem 2 is that, to show IF is optimal with respect to mean response time, it now suffices to show:

$$\mathbb{E}\left[N^{\text{IF}}\right] \leq \mathbb{E}\left[N^{\pi}\right] \qquad \forall \pi \in \mathcal{P}. \tag{3}$$

However, it is hard to directly compare the *numbers of jobs* under different policies. We get around this roadblock by instead analyzing how the *remaining work* in the system under IF relates to other policies $\pi \in \mathcal{P}$. In particular, we obtain the following strong result.

Theorem 3. *For all policies $\pi \in \mathcal{P}$, if we assume that*

$$(N_I^{\pi}(0), N_E^{\pi}(0)) = (N_I^{\text{IF}}(0), N_E^{\text{IF}}(0)),$$

then:

$$W^{\text{IF}}(t) \leq_{ST} W^{\pi}(t) \text{ and } W_I^{\text{IF}}(t) \leq_{ST} W_I^{\pi}(t) \qquad \forall t \geq 0,$$

where $W^{\pi}(t)$ is the total remaining work under policy π at time t, $W_I^{\pi}(t)$ is the remaining inelastic work under policy π at time t, and $\leq_{ST}$ denotes stochastic dominance.

Proof. Fix an arbitrary policy $\pi \in \mathcal{P}$, and let us consider a fixed *arrival sequence*, that is, a fixed sequence of arrival times and job sizes. We couple π and IF under this sequence. Here, it suffices to consider arrival sequences where the total number of job arrivals up to any time t is finite, as this occurs with probability 1.

Recall that $W_I^{\pi}(t)$ and $W_E^{\pi}(t)$ are respectively the remaining inelastic and elastic work in the system at time t under scheduling policy π. Furthermore, also recall that $W^{\pi}(t)$, the total work at time t, is given by:

$$W^{\pi}(t) = W_I^{\pi}(t) + W_E^{\pi}(t).$$

In order to show the desired stochastic dominance relations, it will suffice to show that on any such arrival sequence

$$W_I^{\text{IF}}(t) \leq W_I^{\pi}(t) \quad \text{and} \quad W^{\text{IF}}(t) \leq W^{\pi}(t) \quad \forall t \geq 0.$$

[1] Little's Law states that for any ergodic system with average total arrival rate λ, the mean response time, $\mathbb{E}[T]$ is related to the mean total number of jobs in system, $\mathbb{E}[N]$ via the formula $\mathbb{E}[T] = \frac{\mathbb{E}[N]}{\lambda}$.

First, we see it is immediate that, under our arrival sequence, $W_I^{\text{IF}}(t) \le W_I^{\pi}(t)$ for all $t \ge 0$. Since IF and π process inelastic jobs in FCFS order, each inelastic job enters service at least as early under IF as it does under π. Furthermore, IF never preempts inelastic jobs. Hence, at each time t, the remaining size of each inelastic job that has arrived by time t is no larger under IF than it is under π. Since the inelastic work in system is just the sum of the remaining sizes of inelastic jobs, the total inelastic work at time t under IF is less than the total inelastic work at time t under π.

It remains to show that

$$W^{\text{IF}}(t) \le W^{\pi}(t) \qquad \forall t \ge 0. \tag{4}$$

We prove our claim by induction. For a base case, it is clear that $W^{\text{IF}}(0) \le W^{\pi}(0)$, as the policies have the same set of jobs at time zero, and no work has been completed. For any time t, we partition the interval $[0, t)$ into subintervals $[t_i, t_{i+1})$ such that either

(1) IF allocates all k servers on $[t_i, t_{i+1})$, or
(2) IF allocates strictly less than k servers on $[t_i, t_{i+1})$.

We now induct on i, and show that $W^{\text{IF}}(t_i) \le W^{\pi}(t_i)$ implies $W^{\text{IF}}(t_{i+1}) \le W^{\pi}(t_{i+1})$.

If the interval $[t_i, t_{i+1})$ falls into case (1), IF is completing work at the maximal rate of any policy. In particular, IF completes exactly $(t_{i+1} - t_i) \cdot k$ work on $[t_i, t_{i+1}]$. Let ω denote the work completed by π on $[t_i, t_{i+1})$. Then, we must have $\omega \le (t_{i+1} - t_i) \cdot k$. Since IF and π experience the same set of arrivals on this interval, we have:

$$\begin{aligned} W^{\pi}(t_{i+1}) - W^{\text{IF}}(t_{i+1}) &= \big(W^{\pi}(t_i) - \omega\big) - \big(W^{\text{IF}}(t_i) - (t_{i+1} - t_i) \cdot k\big) \\ &= \big(W^{\pi}(t_i) - W^{\text{IF}}(t_i)\big) + ((t_{i+1} - t_i) \cdot k - \omega) \\ &\ge 0. \end{aligned}$$

Thus, we have $W^{\text{IF}}(t_{i+1}) \le W^{\pi}(t_{i+1})$, as desired.

If the interval $[t_i, t_{i+1})$ falls into case (2), IF allocates strictly less than k servers on $[t_i, t_{i+1})$. We aim to show that $W^{\text{IF}}(t_{i+1}) \le W^{\pi}(t_{i+1})$. Observe that IF can have no elastic jobs in its system on $[t_i, t_{i+1})$. This is because we have defined IF to be work-conserving. Hence, if there was an elastic job, IF would run it on all available servers.

Observe that, assuming no elastic job arrives at time t_{i+1},

$$W^{\text{IF}}(t_{i+1}) = W_I^{\text{IF}}(t_{i+1}).$$

Likewise, we know

$$W^{\pi}(t_{i+1}) = W_I^{\pi}(t_{i+1}) + W_E^{\pi}(t_{i+1}) \ge W_I^{\pi}(t_{i+1}).$$

We get the inequality above because π cannot have negative elastic work at time t_{i+1}. Finally, we have

$$\begin{aligned} W^{\pi}(t_{i+1}) - W^{\text{IF}}(t_{i+1}) &= \big(W_I^{\pi}(t_{i+1}) + W_E^{\pi}(t_{i+1})\big) - W_I^{\text{IF}}(t_{i+1}) \\ &= \big(W_I^{\pi}(t_{i+1}) - W_I^{\text{IF}}(t_{i+1})\big) + W_E^{\pi}(t_{i+1}) \\ &\ge W_I^{\pi}(t_{i+1}) - W_I^{\text{IF}}(t_{i+1}) \\ &\ge 0, \end{aligned}$$

where the last inequality follows from the fact that $W_I^{\text{IF}}(t') \le W_I^{\pi}(t')$ for all $t' \ge 0$. Thus, we have $W^{\text{IF}}(t_{i+1}) \le W^{\pi}(t_{i+1})$.

Note that some elastic work could arrive at exactly time t_{i+1}. However, this increases the total work in both systems by the same amount and thus has no effect on the ordering of these quantities.

Thus, for any interval $[t_i, t_{i+1})$, if $W^{\text{IF}}(t_i) \le W^{\pi}(t_i)$, then we have $W^{\text{IF}}(t_{i+1}) \le W^{\pi}(t_{i+1})$. Since $W^{\text{IF}}(0) \le W^{\pi}(0)$, it follows that this inequality holds at the end of the last subinterval. The end of this final subinterval is exactly time t. Thus, for any $t \ge 0$, we have $W^{\text{IF}}(t) \le W^{\pi}(t)$, as desired.

We have thus found a coupling of π and IF such that the amount of total work and the amount of inelastic work in each system is ordered at every moment in time. This implies that

$$W_I^{\text{IF}}(t) \le_{ST} W_I^{\pi}(t) \quad \forall t \ge 0$$

and

$$W^{\text{IF}}(t) \le_{ST} W^{\pi}(t) \quad \forall t \ge 0$$

as desired. □

In other words, IF is the best policy in $\mathcal{P}$ for minimizing remaining inelastic and total work in the system. One possible explanation for this is that, by deferring parallelizable work, IF ensures that all k servers are saturated with work for as long as possible.

We now understand that, out of all policies in $\mathcal{P}$, IF is optimal with respect to minimizing both expected remaining inelastic work *and* expected remaining total work at any time t. We now establish a relationship between expected remaining work and expected number of jobs in system.

Lemma 4. *For any policy π, we have:*

$$\mathbb{E}[W_I^{\pi}] = \frac{1}{\mu_I}\mathbb{E}[N_I^{\pi}] \qquad \text{and} \qquad \mathbb{E}[W_E^{\pi}] = \frac{1}{\mu_E}\mathbb{E}[N_E^{\pi}],$$

where W_I^{π} and N_I^{π} are respectively the inelastic work and number of inelastic jobs in the system in steady-state under policy π, and where the size of an inelastic job is $S_I \sim Exp(\mu_I)$. W_E^{π}, N_E^{π}, and μ_E are the analogous quantities for elastic jobs.

Proof. We do the proof for the inelastic relationship, but the proof for the elastic relationship is identical. Let the random variable $N_I^{\pi}(t)$ denote the number of inelastic jobs in the system under policy π at time t. For every $\ell \in \{1, \ldots, N_I^{\pi}(t)\}$ we define $R_{\ell,I}^{\pi}(t)$ as the *remaining size* of inelastic job ℓ under policy π at time t.

Recall $W_I^{\pi}(t)$ is the remaining inelastic work in system at time t under policy π. We have the following equivalence:

$$W_I^{\pi}(t) = \sum_{\ell=1}^{N_I^{\pi}(t)} R_{\ell,I}^{\pi}(t).$$

By the memoryless property of the exponential distribution, the remaining size of jobs $\ell \in \{1, \ldots, N_I^{\pi}(t)\}$ are exponentially distributed. Specifically, $R_{\ell,I}^{\pi}(t) \sim Exp(\mu_I)$, for any policy π or time t. Thus, $N_I^{\pi}(t)$ and $R_{\ell,I}^{\pi}(t)$ are independent and we have that

$$\begin{aligned} \mathbb{E}[W_I^{\pi}(t)] &= \mathbb{E}[R_{\ell,I}^{\pi}(t)] \cdot \mathbb{E}[N_I^{\pi}(t)] \\ &= \mathbb{E}[S_I] \cdot \mathbb{E}[N_I^{\pi}(t)]. \end{aligned}$$

As shown in Appendix C, $\mathbb{E}[N_I^{\pi}(t)]$ converges to $\mathbb{E}[N_I^{\pi}]$ as $t \to \infty$. This implies the convergence of $\mathbb{E}[W_I^{\pi}(t)]$. Thus, taking the limit as $t \to \infty$ yields

$$\mathbb{E}[W_I^{\pi}] = \mathbb{E}[S_I] \cdot \mathbb{E}[N_I^{\pi}] = \frac{1}{\mu_I} \cdot \mathbb{E}[N_I^{\pi}],$$

as desired [2]. □

[2] Technically, we have only proven that $\mathbb{E}[W_I^{\pi}(t)]$ converges to *some value*, but not that it converges to $\mathbb{E}[W_I^{\pi}]$. This would be sufficient for our subsequent results. In fact, $\mathbb{E}[W_I^{\pi}(t)]$ converges to $\mathbb{E}[W_I^{\pi}]$ as $t \to \infty$, but we omit this proof for brevity.

We can now show that IF has the lowest expected number of jobs in system when $\mu_I \geq \mu_E$.

THEOREM 5. *For any policy π, if $\mu_I \geq \mu_E$, we have:*

$$\mathbb{E}\left[N^{\text{IF}}\right] \leq \mathbb{E}\left[N^{\pi}\right].$$

And via Little's Law, we have:

$$\mathbb{E}\left[T^{\text{IF}}\right] \leq \mathbb{E}\left[T^{\pi}\right].$$

PROOF. Because there exists an optimal work-conserving policy in $\mathcal{P}$, it suffices to consider any policy $\pi \in \mathcal{P}$. We write total work under π as $W^{\pi} = W_I^{\pi} + W_E^{\pi}$. Likewise, we have the equality $N^{\pi} = N_I^{\pi} + N_E^{\pi}$. First, from Lemma 4, we have the following equalities:

$$\mathbb{E}[W_I^{\pi}] = \frac{1}{\mu_I}\mathbb{E}[N_I^{\pi}] \qquad \text{and} \qquad \mathbb{E}[W_E^{\pi}] = \frac{1}{\mu_E}\mathbb{E}[N_E^{\pi}].$$

Furthermore, by the stochastic dominance results of Theorem 3,

$$\mathbb{E}[W_I^{\text{IF}}] \leq \mathbb{E}[W_I^{\pi}] \quad \text{and} \quad \mathbb{E}[W^{\text{IF}}] \leq \mathbb{E}[W^{\pi}]$$

Thus, we have:

$$\begin{aligned}
\mathbb{E}\left[N^{\text{IF}}\right] &= \mathbb{E}\left[N_I^{\text{IF}} + N_E^{\text{IF}}\right] \\
&= \mu_I \mathbb{E}\left[W_I^{\text{IF}}\right] + \mu_E \mathbb{E}\left[W_E^{\text{IF}}\right] \\
&= (\mu_I - \mu_E)\mathbb{E}\left[W_I^{\text{IF}}\right] + \mu_E \mathbb{E}\left[W_I^{\text{IF}} + W_E^{\text{IF}}\right] \\
&\leq (\mu_I - \mu_E)\mathbb{E}\left[W_I^{\pi}\right] + \mu_E \mathbb{E}\left[W_I^{\pi} + W_E^{\pi}\right] \qquad (5) \\
&= \mu_I \mathbb{E}\left[W_I^{\pi}\right] + \mu_E \mathbb{E}\left[W_E^{\pi}\right] \\
&= \mathbb{E}[N_I^{\pi}] + \mathbb{E}[N_E^{\pi}] = \mathbb{E}[N^{\pi}]
\end{aligned}$$

Note, we leverage the fact $\mu_I \geq \mu_E$ in (5). If $\mu_E > \mu_I$, then $\mu_I - \mu_E$ would be negative, so we would not be able establish a relationship like (5). This completes the proof. □

We have therefore established that IF is optimal with respect to mean response time when $\mu_I \geq \mu_E$.

4.3 Failure when $\mu_I < \mu_E$

Now, we consider the case when $\mu_I < \mu_E$. Here, we demonstrate that IF is not optimal in minimizing mean response time. In fact, IF is not even optimal in the simplified environment where there are only two servers and no arrivals. We construct our counterexample in Theorem 6 below.

THEOREM 6. *In general,* IF *is not optimal for minimizing mean response time when $\mu_I < \mu_E$.*

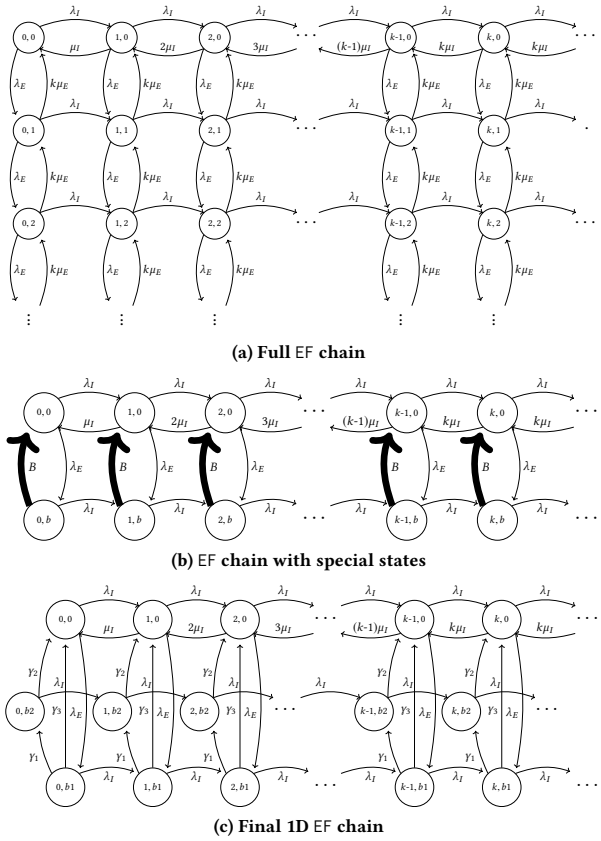
PROOF. Assume we have $k = 2$ servers, $\mu_E = 2\mu_I$, and there are no arrivals. We show that, if the system starts with two inelastic jobs and one elastic job, the policy EF outperforms IF.

We directly compute the mean response time for both policies, starting with IF. We let T^{IF} denote response time under IF, and T^{EF} denote response time under elastic first. We have:

$$\mathbb{E}\left[T^{\text{IF}}\right] = \frac{3}{2\mu_I} + \frac{2}{\mu_I + \mu_E} + \frac{\mu_I}{\mu_I + \mu_E}\left(\frac{1}{2\mu_E}\right) + \frac{\mu_E}{\mu_I + \mu_E}\left(\frac{1}{\mu_I}\right) = \frac{35}{12\mu_I}.$$

On the other hand, we see:

$$\mathbb{E}[T^{\text{EF}}] = \frac{3}{2\mu_E} + \frac{2}{2\mu_I} + \frac{1}{\mu_I} = \frac{33}{12\mu_I}.$$



(a) Full EF chain

(b) EF chain with special states

(c) Final 1D EF chain

Figure 2: The transformation of the 2D-infinite EF chain to a 1D-infinite chain via the busy period transformation. Special states representing an $M/M/1$ busy period are shown in (b), and these busy periods are approximated by a Coxian distribution in (c).

In particular, we have $\mathbb{E}[T^{\text{EF}}] < \mathbb{E}[T^{\text{IF}}]$. Thus, in general, IF is not optimal when $\mu_I < \mu_E$. In fact, in this environment, we see EF outperforms IF. □

5 RESPONSE TIME ANALYSIS RESULTS

From the results of Section 4, we know that IF is optimal with respect to mean response time when $\mu_I \geq \mu_E$. However, Section 4 also shows that EF can outperform IF when $\mu_I < \mu_E$. This begs the question of which allocation policy, IF or EF, performs better for given values of μ_I and μ_E.

In this section we derive the mean response time for EF under a range of values of μ_I, μ_E, λ_I, λ_E, and k. First, in Section 5.1, we present the Markov chains for EF. This Markov chain is 2D-infinite. Then, in Section 5.2, we present a technique from the stochastic literature called Busy Period Transitions [45, 46] which reduces the 2D-infinite chain to a 1D-infinite chain. Although Busy Period Transitions produce an approximation, it is known to be highly accurate, with errors of less than 1% [26–28, 45, 46]. Finally, in Section 5.3, we apply standard Matrix-Analytic methods to solve the 1D-infinite Markov chain, obtaining the stationary distribution

and finally the mean response time under EF. The analysis for the IF policy is similar, and thus we defer it to Appendix D.

The results of our analysis for IF and EF are shown in Figures 3, 4, and 5. We compared our analysis with simulation, and all numbers agree within 1%. We note that [7] used MDP-based techniques to analyze allocation policies in a similar model. These previous results required truncating the state space, and were computationally intensive. The techniques presented in this section do not require truncating the state space, can be tuned to arbitrary precision, and are comparatively efficient.

Figure 3 presents an overview of our results, showing only the relative performance of IF and EF as the system load, ρ, is moved from (a) low load to (b) medium load to (c) high load. In every case, IF outperforms EF when $\mu_I \geq \mu_E$, as expected from Theorem 5. When $\mu_I < \mu_E$, Figure 3 shows us that EFcan outperform IF, and that the region where EF is better grows as ρ increases.

Figure 4 shows the absolute mean response times under IF and EF as a function of μ_I. We again examine the system under various fixed values of ρ. The dotted lines at $\mu_I = 1$ denote the case where $\mu_I = \mu_E$. We therefore know that IF is optimal to the right of this line in every graph, while EF may dominate IF to the left of this line. We see that our choice of allocation policy has a major impact on mean response time.

While Figures 3 and 4 assume that $k = 4$, our analysis works equally well with any number of servers, k. Figure 5 shows how the mean response time under IF and EF changes as k increases while system load, ρ, remains constant.

5.1 Markov Chains for IF and EF

Figure 2a shows the Markov chain which exactly describes EF. The corresponding IF chain is given in Appendix D. Recall that the state (i, j) denotes having i inelastic jobs and j elastic jobs in the system. This chain is infinite in 2 dimensions – the number of inelastic jobs and the number of elastic jobs. Because there is no general method for solving 2D-infinite Markov chains, we provide a technique for converting this chain to a 1D-infinite Markov chain in Section 5.2.

5.2 Converting From 2D-Infinite to 1D-Infinite

To reduce the dimensionality of the Markov chain for EF we follow a three step process:

Step 1: Response time of elastic jobs is trivial. Under EF, elastic jobs have preemptive priority over inelastic jobs. Thus, their behavior is independent of the state of inelastic jobs in the system. We can therefore model the response time of elastic jobs as an $M/M/1$ queueing system with arrival rate λ_E and service rate $k\mu_E$, which is well understood in the queueing literature [33]. What remains is to understand the response time of the inelastic jobs.

Step 2: The busy period transformation. Looking at Figure 2a, we notice that the chain has a repeating structure when there is at least 1 elastic job in the system ($j \geq 1$). We leverage this repeating structure to reduce the Markov chain for EF to a 1D-infinite chain. Specifically, while there are elastic jobs in the system, EF does not process any inelastic jobs. The length of time where EF is not processing any inelastic jobs can be viewed as an $M/M/1$ *busy period*. In an $M/M/1$ system, a busy period is defined to be the time between when a job arrives into an empty system until the system empties. In our case, this busy period is the time from when an elastic job arrives into a system with no elastic jobs until the system next has 0 elastic jobs. In Figure 2b, we show how to the entire portion of the Markov chain where $j \geq 0$ with a set of special states which represent the duration of an $M/M/1$ busy period for the elastic jobs.

Step 3: Creating 1D chain for inelastic jobs. Looking at Figure 2b, we note the bolded transition arrows (labeled "B") emanating from the busy period states. Because the duration of an $M/M/1$ busy period is not exponentially distributed, we must replace these special transitions with a mixture of exponential states (a Coxian distribution) which accurately approximates the duration of a busy period. A technique for matching the first three moments of the busy period with a Coxian is given in [45]. The 1D-infinite chain resulting from this technique is described in Figure 2c.

We use the same three-step technique to make an analogous simplification of the Markov chain for IF (see Appendix D).

Given these 1D-infinite chains, we now apply standard matrix analytic techniques to solve for mean response time.

5.3 Matrix Analytic Method

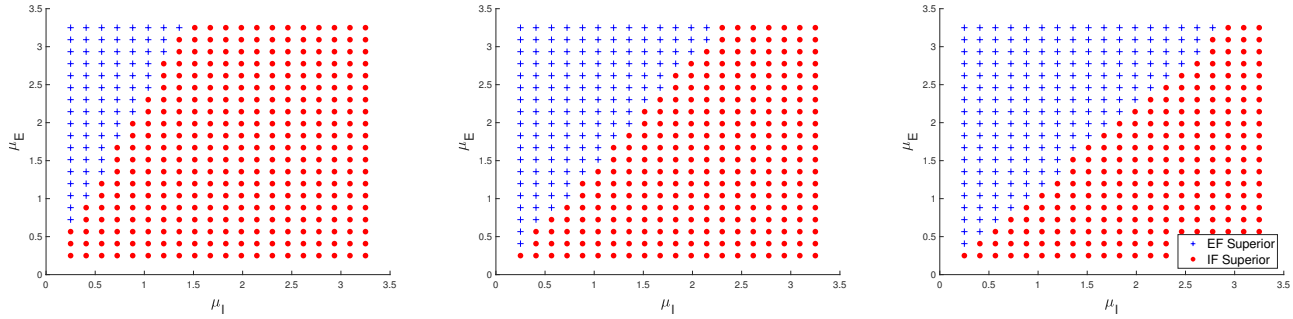
We now explain how to analyze IF and EF using the 1D-infinite Markov chains developed in the previous section. We do this by applying matrix analytic methods [34, 43, 44]. Matrix analytic methods are iterative procedures which compute the stationary distribution of a repeating, 1D-infinite Markov chain.

Consider Figure 2c, which shows the 1D-infinite chain for EF. Observe that each column of this chain, after the first column, has identical transitions. The idea of matrix analytic methods is to represent the stationary distribution of column $j + 1$ as a product of the stationary distribution of column j and some unknown matrix R. The matrix R is determined iteratively through a numeric procedure [34, 43, 44]. This procedure yields the stationary distribution of the chain. Using the stationary distribution we can easily determine the mean number of inelastic jobs, and hence the mean response time for inelastic jobs (recall that the response time for elastic jobs under EF is trivial). An analogous argument can be applied to solve the 1D-infinite chain for IF.

6 CONCLUSION

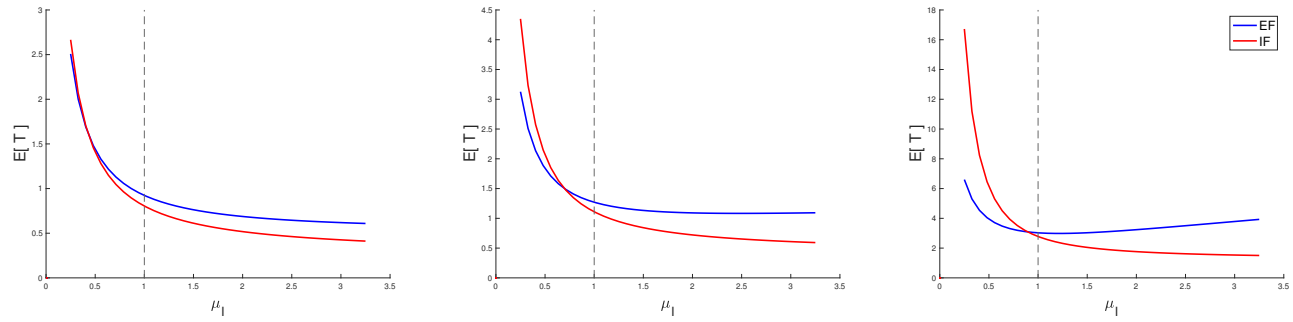
This paper establishes optimality results and provides the first analysis of policies for scheduling jobs which are heterogeneous with respect to parallelizability. Specifically, we study a model where jobs are either inelastic or elastic: inelastic jobs can parallelize across at most C servers and elastic jobs parallelize linearly across any number of servers. We prove that the policy *Inelastic-First* (IF), which gives inelastic jobs preemptive priority over elastic jobs, is optimal for minimizing the mean response time across jobs in the common case where elastic jobs are larger on average than inelastic jobs. We provide analysis of mean response times under the Elastic-First (EF) and Inelastic-First (IF) policies.

There are many open questions in scheduling jobs which are heterogeneous with respect to their parallelizability. One immediate follow-up of our work is to find optimal policies when elastic jobs are smaller on average than inelastic jobs. This paper shows that, in this setting, EF can outperform IF, but we do not show that EF



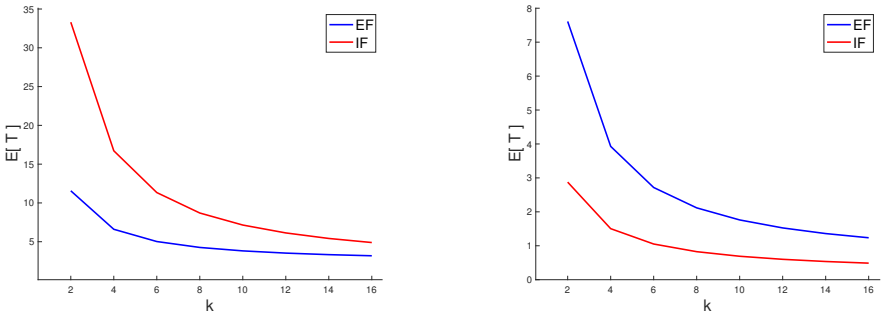
(a) Low load, $\rho = .5$ (b) Med. Load, $\rho = .7$ (c) High load, $\rho = .9$

Figure 3: Heat maps showing the relative performance of IF and EF as a function of μ_I and μ_E when $k = 4$. We fix load ρ and vary μ_I and μ_E. To offset the changes to μ_I and μ_E, we change λ_I and λ_E to keep ρ constant. In every graph, $\lambda_I = \lambda_E$. The red circles represent settings where IF dominates EF. The blue +'s represent cases where EF dominates IF. As ρ increases, the region where EF dominates IF grows. However, as expected, when $\mu_I \geq \mu_E$ IF dominates EF for all loads.



(a) Low load, $\rho = .5$ (b) Med. load, $\rho = .7$ (c) High load, $\rho = .9$

Figure 4: Graphs showing the absolute mean response times under IF and EF as a function of μ_I when $k = 4$. In each graph, we fix system load, ρ, and set $\mu_E = 1$. We then vary μ_I. To offset the changes in μ_I, we change λ_I and λ_E to keep ρ constant. In every graph, $\lambda_I = \lambda_E$. The dotted lines at $\mu_I = 1$ denote the case where $\mu_I = \mu_E$. Thus IF is optimal to the right of this line, while EF may dominate IF to the left of this line. We see that the allocation policy has a major impact on mean response time.



(a) $\mu_I = .25, \mu_E = 1$ (b) $\mu_I = 3.25, \mu_E = 1$

Figure 5: Graphs showing the mean response time under IF and EF as a function of the number of servers, k under high load ($\rho = 0.9$). The values of μ_I and μ_E are chosen to represent the extreme ends of Figure 4c (where the performance gap between the policies is the largest). Even when $k = 16$, the difference between IF and EF remains large.

is the optimal allocation policy. Furthermore, the model studied in this paper can be generalized in many ways to capture a broad range of application scenarios. For example, consider a model where elastic jobs are not fully elastic as in this paper, but are elastic up to a certain number of servers. Additionally, we might have more than two classes of jobs with different levels of parallelizability and different job size distributions. The problem of finding optimal policies and providing analysis in these models is wide open.

APPENDIX

A APPROXIMATION WHEN JOBS ARRIVE AT THE SAME TIME

In this section we use dual fitting to show that a generalization of SRPT-k is a 4-approximate algorithm for mean response time if all jobs arrive at the same time. This case is entirely deterministic. This result holds in a general parallelizability setting where every job j is parallelizable up to k_j processors. That is, if job j is given $k' \leq k$ processors, the rate it is processed is $\min\{k_j, k'\}$.

Consider the following LP relaxation of the problem. We use x_j to denote the inherent size of job j. The variable $y_{j,t}$ is how much job j is processed at time t.

$$\min_{\{y_{jt}\}} \quad \sum_j \sum_{t \geq 0} \left(\frac{t}{x_j} + \frac{1}{2k_j} \right) \cdot y_{jt} \qquad \mathsf{LP}_{\mathsf{primal}}$$

$$\begin{aligned} \sum_{t \geq 0} y_{jt} &\geq x_j && \forall j \\ \sum_{j \,:\, t \geq 0} y_{jt} &\leq k && \forall t \\ y_{jt} &\geq 0 && \forall j, t \,:\, t \geq 0 \end{aligned}$$

It is easy to show that the above LP lower bounds the optimal flow time of a feasible schedule. This is essentially an LP for a k speed single machine plus the standard corrective term in the objective. See [10] for similar relaxations. The dual of $\mathsf{LP}_{\mathsf{primal}}$ is:

$$\max_{\{\alpha_j\},\{\beta_t\}} \quad \sum_j \alpha_j - \sum_t \beta_t \qquad \mathsf{LP}_{\mathsf{dual}}$$

$$\begin{aligned} \frac{\alpha_j}{x_j} - \frac{\beta_t}{k} &\leq \frac{t}{x_j} + \frac{1}{2k_j} && \forall j, t \,:\, t \geq 0 \\ \alpha_j &\geq 0 && \forall j \\ \beta_t &\geq 0 && \forall t \end{aligned}$$

The algorithm we will use is a natural generalization of SRPT-k to the case of parallelizable jobs. The algorithm sorts jobs according to their inherent size in increasing order. We will thus assume that the jobs are in this order such that $x_1 \leq x_2 \leq \ldots x_n$, where n is the total number of jobs . At any point in time, the algorithm gives the cores to the jobs in this priority order. Each job j is assigned up to k_j processors and then the algorithm considers the next job in the list with the remaining processors. We let $U_j = \sum_{i=1}^{j-1} x_i$ be the total amount of work strictly ahead of job j.

We will assume that the processors the algorithm has are of speed $s \geq 1$. That is, each processor completes s units of work each timestep. Later we will set $s = 2$. We compare to an optimal solution with one speed processors. The following theorem allows us to do this comparison with minimal loss in the approximation ratio.

Lemma 7 ([22]). *Let OPT_s denote the value of the total response time of the optimal algorithm where the optimal algorithm has processors of speed s. Then for any $s \geq 1$,*

$$OPT_1 \leq sOPT_s.$$

We now define the dual variables. Let $Q(t)$ denote the set of jobs released and unsatisfied at time t in the algorithm's schedule. Let $\alpha_j = \frac{U_j}{ks} + \frac{x_j}{sk_j}$ and let $\beta_t = \frac{1}{s}|Q(t)|$. Our main claim is the following.

Lemma 8. *Let C denote the algorithm's total completion time. It is the case that $\sum_j \alpha_j - \sum_t \beta_t \geq \left(1 - \frac{1}{s}\right) C$. Moreover, α, β correspond to a feasible dual solution when $s = 2$.*

Lemma 8 is sufficient to prove our theorem.

Theorem 9. *The SRPT-k algorithm is a 4-approximation for mean response time when all jobs arrive at time 0.*

Proof. Set $s = 2$. Lemma 8 ensures that C is at most a factor 2 larger than the optimal solution using 1 speed. Lemma 7 ensures that the 1 speed optimal is within a factor 2 of the 2 speed optimal. Together this shows the algorithm is a 4 approximation. □

To prove Lemma 8 we first establish the value of the objective.

Lemma 10. $\sum_j \alpha_j - \sum_t \beta_t = \left(1 - \frac{1}{s}\right) C.$

Proof. Notice that $\sum_t \beta_t = \sum_t \frac{1}{s}|Q(t)|$. This is precisely $\frac{1}{s}C$. Thus, it suffices to prove $\frac{U_j}{ks} + \frac{x_j}{k_j} \geq C$. To do so, we show that $\frac{U_j}{ks} + \frac{x_j}{sk_j}$ is an upper bound on job j's response time. Indeed, we know that either all k processors are working on work in $U_j + x_j$ with speed s if j is unsatisfied or job j is being worked on with k_j processors with speed s. □

We now show that this setting of the dual variables corresponds to a feasible dual solution.

Lemma 11. *The dual solution α, β is feasible when $s = 2$.*

Proof. We must show that for all jobs j and times $t \geq 0$:

$$\frac{\alpha_j}{x_j} - \frac{\beta_t}{k} \leq \frac{t}{x_j} + \frac{1}{2k_j}.$$

Consider the left hand side for a fixed job j and time t. Let $x^r_{j'}(t)$ be the remaining work left on job j' at time t and $x^p_{j'}(t) = x_{j'} - x^r_{j'}(t)$ be the amount of job j' that has been processed up to time t. This is equivalent to the following given the definitions of α and β:

$$\begin{aligned} & \frac{1}{x_j}\left(\frac{U_j}{ks} + \frac{x_j}{sk_j}\right) - \frac{1}{sk}|Q(t)| \\ = \quad & \frac{1}{x_j}\left(\frac{1}{ks}\sum_{j' \in [n], x_{j'} < x_j} \left(x^p_{j'}(t) + x^r_{j'}(t)\right) + \frac{x_j}{sk_j}\right) - \frac{1}{sk}|Q(t)|. \end{aligned}$$

Now consider any job that is in complete at time t. That is, those in $Q(t)$. We can remove these from the first term by combining terms with the $-\frac{1}{sk}|Q(t)|$ term. The prior expression is only less than the following:

$$\frac{1}{x_j}\left(\frac{1}{ks}\sum_{j'\in[n]\setminus Q(t),\, x_{j'}<x_j}\left(x^p_{j'}(t)+x^r_{j'}(t)\right)+\frac{x_j}{sk_j}\right)$$

$$\leq \quad \frac{1}{x_j}\left(\frac{1}{ks}\sum_{j'\in[n]\setminus Q(t),\, x_{j'}<x_j}\left(x^p_{j'}(t)\right)+\frac{x_j}{sk_j}\right).$$

$$[x^r_{j'}(t)=0 \text{ for jobs completed at } t]$$

Notice that $\sum_{j'\in[n]\setminus Q(t),\, x_{j'}<x_j}\left(x^p_{j'}(t)\right)$ is less than kst. This is because the summation is counting work that the algorithm has processed by time t. The algorithm has k processors of speed s. Thus, the prior term is less than the following:

$$\frac{t}{x_j}+\frac{1}{sk_j}.$$

Given the $s=2$, we get that the dual solution is feasible. □

Together Lemmas 10 and 11 complete the proof.

B IDLING POLICIES

We define a policy to be *idling* if it leaves one or more servers idle rather than allocating them to some eligible jobs.

Theorem 12. *For any policy π which unnecessarily idles servers there exists a non-idling policy π' such that*

$$\mathbb{E}[T^{\pi'}] \leq \mathbb{E}[T^{\pi}].$$

Hence, there exists an optimal policy which is non-idling.

Proof. The proof of this claim appears in [8]. □

C LYAPUNOV STABILITY OF WORK CONSERVING POLICIES

Theorem 13. *For any work-conserving policy π, the associated Markov chain $\{(N_I^\pi(t), N_E^\pi(t)) : t \geq 0\}$ has a stationary distribution. If we define (N_I^π, N_E^π) to be a random element that follows this stationary distribution, then*

$$\lim_{t\to\infty}(N_I^\pi(t), N_E^\pi(t)) \overset{d}{=} (N_I^\pi, N_E^\pi). \tag{6}$$

Furthermore,

$$\lim_{t\to\infty}\mathbb{E}[N_I^\pi(t)] = \mathbb{E}[N_I^\pi], \tag{7}$$

and

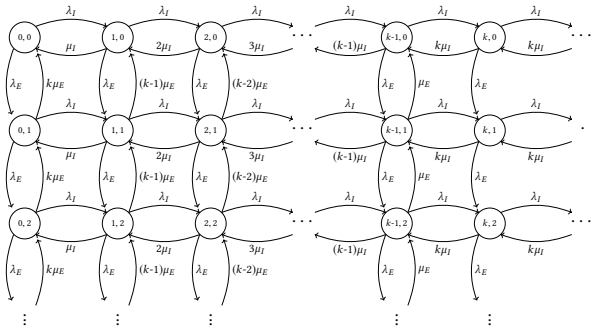
$$\lim_{t\to\infty}\mathbb{E}[N_E^\pi(t)] = \mathbb{E}[N_E^\pi]. \tag{8}$$

Proof. The proof of this claim appears in [8]. □

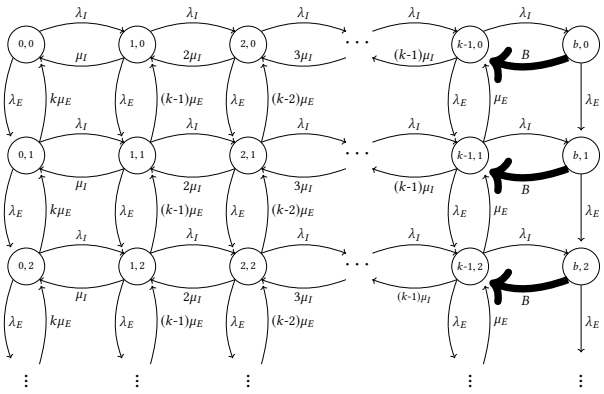
D MARKOV CHAINS FOR IF

Figure 6a shows the Markov chain for IF. To analyze this chain we will use a busy period transformation as described in Section 5.2.

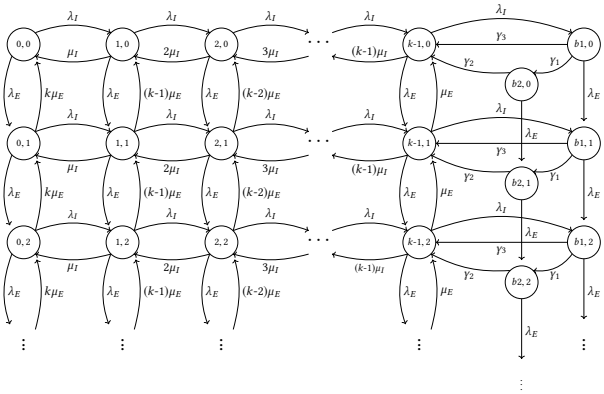
We note that the inelastic jobs under IF see an $M/M/k$ queueing system, and hence their mean response time is known. We therefore only need to consider the mean response time of elastic jobs under IF. When there are more than k inelastic jobs in the system under IF, elastic jobs receive no service. The time from when there are first k inelastic jobs in the system until there are $k-1$ inelastic jobs is exactly an $M/M/1$ busy period. Hence, we perform the same busy period transformation described in Section 5.2 to the Markov chain for IF. This results in a 1D-infinite Markov chain which we can analyze using matrix analytic methods. We depict the busy period transformation for IF in Figure 6



(a) Full IF chain



(b) IF chain with special states



(c) Final 1D IF chain

Figure 6: The transformation of the 2D-infinite IF chain to a 1D-infinite chain via the busy period transformation. Special states representing an $M/M/1$ busy period are shown in (b), and these busy periods are approximated by a Coxian distribution in (c).

REFERENCES

[1] I.J.B.F. Adan, G.J. van Houtum, and J. van der Wal. Upper and lower bounds for the waiting time in the symmetric shortest queue system. *Annals of Operations Research*, 48:197–217, 1994.

[2] Kunal Agrawal, I-Ting Angelina Lee, Jing Li, Kefu Lu, and Benjamin Moseley. Practically efficient scheduler for minimizing average flow time of parallel jobs. In *2019 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2019, Rio de Janeiro, Brazil, May 20-24, 2019*, pages 134–144. IEEE, 2019.

[3] Kunal Agrawal, Jing Li, Kefu Lu, and Benjamin Moseley. Scheduling parallel DAG jobs online to minimize average flow time. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 176–189. SIAM, 2016.

[4] S. Anand, Naveen Garg, and Amit Kumar. Resource augmentation for weighted flow-time explained by dual fitting. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 1228–1241, 2012.

[5] Spyros Angelopoulos, Giorgio Lucarelli, and Nguyen Kim Thang. Primal-dual and dual-fitting analysis of online scheduling algorithms for generalized flow-time problems. *Algorithmica*, 81(9):3391–3421, 2019.

[6] Eitan Bachmat and Hagit Sarfati. Analysis of size interval task assigment policies. *Performance Evaluation Review*, 36(2):107–109, 2008.

[7] Benjamin Berg, Jan-Pieter Dorsman, and Mor Harchol-Balter. Towards optimality in parallel scheduling. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 1(2):1–30, 2018.

[8] Benjamin Berg, Mor Harchol-Balter, Benjamin Moseley, Weina Wang, and Justin Whitehouse. Optimal resource allocation for elastic and inelastic jobs. *https://arxiv.org/abs/2005.09745*.

[9] Carl Bussema and Eric Torng. Greedy multiprocessor server scheduling. *Oper. Res. Lett.*, 34(4):451–458, 2006.

[10] Jivitej S. Chadha, Naveen Garg, Amit Kumar, and V. N. Muralidhara. A competitive algorithm for minimizing weighted flow time on unrelatedmachines with speed augmentation. In Michael Mitzenmacher, editor, *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 679–684. ACM, 2009.

[11] Richard W Conway, Louis W Miller, and William L Maxwell. *Theory of scheduling*. Dover, 2003.

[12] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[13] Christina Delimitrou and Christos Kozyrakis. Quasar: resource-efficient and qos-aware cluster management. *ACM SIGPLAN Notices*, 49(4):127–144, 2014.

[14] Jeff Edmonds. Scheduling in the dark. *Theor. Comput. Sci.*, 235(1):109–141, 2000.

[15] Jeff Edmonds, Sungjin Im, and Benjamin Moseley. Online scalable scheduling for the k-norms of flow time without conservation of work. In Dana Randall, editor, *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 109–119. SIAM, 2011.

[16] Jeff Edmonds and Kirk Pruhs. Scalably scheduling processes with arbitrary speedup curves. In *Proceedings of the twentieth annual ACM-SIAM symposium on Discrete algorithms*, pages 685–692. SIAM, 2009.

[17] Kyle Fox and Benjamin Moseley. Online scheduling on identical machines using SRPT. In Dana Randall, editor, *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 120–128. SIAM, 2011.

[18] Anshul Gandhi and Mor Harchol-Balter. How data center size impacts the effectiveness of dynamic power management. In *2011 49th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 1164–1169. IEEE, 2011.

[19] Isaac Grosof, Ziv Scully, and Mor Harchol-Balter. Srpt for multiserver systems. *Performance Evaluation*, 127:154–175, 2018.

[20] Abhishek Gupta, Bilge Acun, Osman Sarood, and Laxmikant V Kalé. Towards realizing the potential of malleable jobs. In *2014 21st International Conference on High Performance Computing (HiPC)*, pages 1–10. IEEE, 2014.

[21] Anupam Gupta, Sungjin Im, Ravishankar Krishnaswamy, Benjamin Moseley, and Kirk Pruhs. Scheduling heterogeneous processors isn't as easy as you think. In Yuval Rabani, editor, *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 1242–1253. SIAM, 2012.

[22] Varun Gupta, Benjamin Moseley, Marc Uetz, and Qiaomin Xie. Stochastic online scheduling on unrelated machines. In *Integer Programming and Combinatorial Optimization - 19th International Conference, IPCO 2017, Waterloo, ON, Canada, June 26-28, 2017, Proceedings*, pages 228–240, 2017.

[23] Varun Gupta, Karl Sigman, Mor Harchol-Balter, and Ward Whitt. Insensitivity for ps server farms with jsq routing. *ACM SIGMETRICS Performance Evaluation Review*, 35(2):24–26, 2007.

[24] Mor Harchol-Balter. Task assignment with unknown duration. *Journal of the ACM*, 49(2):260–288, March 2002.

[25] Mor Harchol-Balter. *Performance modeling and design of computer systems: queueing theory in action*. Cambridge University Press, 2013.

[26] Mor Harchol-Balter, Cuihong Li, Takayuki Osogami, Alan Scheller-Wolf, and Mark Squillante. Cycle stealing under immediate dispatch task assignment. In *Proceedings of the 15th ACM Symposium on Parallel Algorithms and Architectures*, pages 274–285, San Diego, CA, June 2003.

[27] Mor Harchol-Balter, Cuihong Li, Takayuki Osogami, Alan Scheller-Wolf, and Mark Squillante. Task assignment with cycle stealing under central queue. In *Proceedings of the 23rd International Conference on Distributed Computing Systems*, pages 628–637, Providence, RI, May 2003.

[28] Mor Harchol-Balter, Takayuki Osogami, Alan Scheller-Wolf, and Adam Wierman. Multi-server queueing systems with multiple priority classes. *Queueing Systems: Theory and Applications*, 51(3–4):331–360, 2005.

[29] Mor Harchol-Balter, Alan Scheller-Wolf, and Andrew Young. Surprising results on task assignment in server farms with high-variability workloads. In *ACM Sigmetrics 2009 Conference on Measurement and Modeling of Computer Systems*, pages 287–298, 2009.

[30] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D Joseph, Randy H Katz, Scott Shenker, and Ion Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *NSDI*, volume 11, pages 22–22, 2011.

[31] Sungjin Im, Benjamin Moseley, Kirk Pruhs, and Eric Torng. Competitively scheduling tasks with intermediate parallelizability. *ACM Transactions on Parallel Computing (TOPC)*, 3(1):1–19, 2016.

[32] Cheeha Kim and Ashok K Agrawala. Analysis of the fork-join queue. *IEEE Transactions on computers*, 38(2):250–255, 1989.

[33] Leonard Kleinrock. *Queueing systems, volume 2: Computer applications*, volume 66. Wiley New York, 1976.

[34] Guy Latouche and V. Ramaswami. *Introduction to Matrix Analytic Methods in Stochastic Modeling*. ASA-SIAM, Philadelphia, 1999.

[35] Stefano Leonardi and Danny Raz. Approximating total flow time on parallel machines. *Journal of Computer and System Sciences*, 73(6):875–891, 2007.

[36] Xiangru Lian, Ce Zhang, Huan Zhang, Cho-Jui Hsieh, Wei Zhang, and Ji Liu. Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pages 5330–5340, 2017.

[37] Richard Liaw, Romil Bhardwaj, Lisa Dunlap, Yitian Zou, Joseph E Gonzalez, Ion Stoica, and Alexey Tumanov. Hypersched: Dynamic resource reallocation for model development on a deadline. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 61–73, 2019.

[38] David Lo, Liqun Cheng, Rama Govindaraju, Parthasarathy Ranganathan, and Christos Kozyrakis. Heracles: Improving resource efficiency at scale. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, pages 450–462, 2015.

[39] Jason Mars, Lingjia Tang, Robert Hundt, Kevin Skadron, and Mary Lou Soffa. Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations. In *Proceedings of the 44th annual IEEE/ACM International Symposium on Microarchitecture*, pages 248–259, 2011.

[40] Robert McNaughton. Scheduling with deadlines and loss functions. *Management Science*, 6(1):1–12, 1959.

[41] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I Jordan, et al. Ray: A distributed framework for emerging {AI} applications. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, pages 561–577, 2018.

[42] Randolph Nelson and Asser Tantawi. Approximate analysis of fork/join synchronization in parallel queues. *Transactions on Computers*, 37(6):739–743, 1988.

[43] Marcel F. Neuts. *Matrix-Geometric Solutions in Stochastic Models*. Johns Hopkins University Press, 1981.

[44] Marcel F. Neuts. *Structured Stochastic Matrices of M/G/1 Type and Their Applications*. Marcel Dekker, 1989.

[45] Takayuki Osogami and Mor Harchol-Balter. Closed form solutions for mapping general distributions to quasi-minimal PH distributions. *Performance Evaluation*, 63(6):524–552, 2006.

[46] Takayuki Osogami, Mor Harchol-Balter, and Alan Scheller-Wolf. Analysis of cycle stealing with switching times and thresholds. In *Proceedings of ACM Sigmetrics*, pages 184–195, San Diego, CA, June 2003.

[47] Yanghua Peng, Yixin Bao, Yangrui Chen, Chuan Wu, and Chuanxiong Guo. Optimus: an efficient dynamic resource scheduler for deep learning clusters. In *Proceedings of the Thirteenth EuroSys Conference*, pages 1–14, 2018.

[48] Donald R Smith. A new proof of the optimality of the shortest remaining processing time discipline. *Operations Research*, 26(1):197–199, 1978.

[49] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. Large-scale cluster management at google with borg. In *Proceedings of the European Conference on Computer Systems*, pages 1–17, 2015.

[50] Weina Wang, Mor Harchol-Balter, Haotian Jiang, Alan Scheller-Wolf, and Rayadurgam Srikant. Delay asymptotics and bounds for multitask parallel jobs. *Queueing Systems*, 91(3-4):207–239, 2019.