The Case for Phase-Aware Scheduling of Parallelizable Jobs

Parallelizable jobs typically consist of multiple phases of computation, where the job is more parallelizable in some phases and less parallelizable in others. For example, in a database, a query may consist of a highly parallelizable table scan, followed by a less parallelizable table join. In the past, this phase-varying parallelizability was summarized by a single sub-linear speedup curve which measured a job's average parallelizability over its entire lifetime. Today, however, a wide range of modern systems have fine-grained knowledge of the exact phase each job is in at every moment in time. Unfortunately, these systems do not know how to best use this real-time feedback to schedule parallelizable jobs. Current systems scheduling is largely heuristic, while theory has failed to produce practical phase-aware scheduling policies.

A phase-aware scheduling policy must decide, at every moment in time, how many servers or cores to allocate to each job in the system, given knowledge of each job's current phase. This paper provides the first stochastic model of a system processing parallelizable jobs composed of phases. Using our model, we derive the optimal phase-aware scheduling policy which minimizes the mean response time across jobs. Our provably optimal policy, Inelastic-First (IF), gives strict priority to jobs which are currently in less parallelizable phases. We validate our results using a simulation of a database running queries from the Star Schema Benchmark. We compare IF to a range of policies from both systems and theory and show that IF can reduce mean response time by a factor of 3.

ACM Reference Format:
. 2021. The Case for Phase-Aware Scheduling of Parallelizable Jobs. *Proc. ACM Meas. Anal. Comput. Syst.* 1, 1 (August 2021), 24 pages.

1 INTRODUCTION

Parallelizable workloads are ubiquitous and appear across a diverse array of modern computer systems. Data centers, supercomputers, machine learning clusters, distributed computing frameworks, and databases all process jobs designed to be parallelized across many servers or cores. Unlike the jobs in more classical models, such as the M/G/k, which each run on a single server, parallelizable jobs are capable of running on multiple servers simultaneously. A job will receive some speedup from being parallelized across additional servers or cores, allowing the job to complete more quickly.

When scheduling parallelizable jobs, a *scheduling policy* must decide *how to best allocate servers or cores among the jobs in the system at every moment in time.* This paper describes and analyzes scheduling policies for systems which process an online *stream* of incoming parallelizable jobs. Given a set of K servers, we seek scheduling policies that minimize the *mean response time* across jobs – the average time from when a job arrives to the system until it is completed.

The difficulty in scheduling parallelizable jobs arises largely from the fact that a job's parallelizability is not constant over time. Across a wide variety of systems, jobs typically consist of multiple *phases*, each of which has its own scalability characteristics.

For example, in databases, a single query typically alternates between highly parallelizable phases and non-parallelizable phases. Specifically in modern databases, queries are translated by the system

Author's address:

2476-1249/2021/8-ART $15.00
https://doi.org/

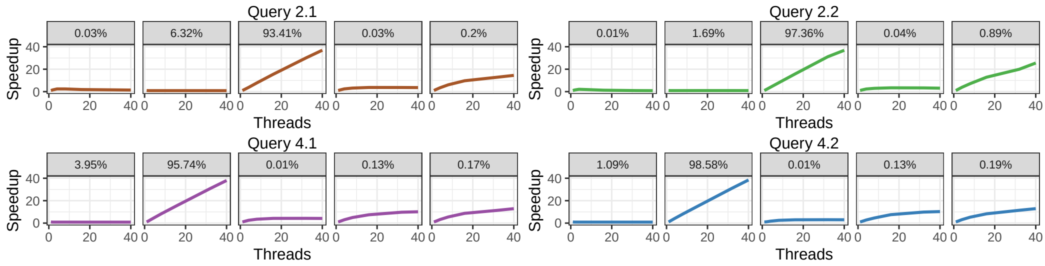


Fig. 1. Speedup functions for each phase of four queries from the Star Schema Benchmark. Queries were executed using the Noisepage database[1]. Phases are generally either elastic (highly parallelizable) or inelastic (highly sequential). The percentages denote the fraction of time spent in each phase when the query was run on a single core. Despite the queries spending most of their time in elastic phases, the overall speedup function of each query is highly sublinear due to Amdahl's law.

into a pipeline composed of multiple phases corresponding to different database operations [1]. A phase which corresponds to a sequential table scan will be *elastic*, capable of perfectly parallelizing and completing k times faster when run on k cores. On the other hand, a phase corresponding to a table join will be *inelastic*, receiving a severely limited speedup from additional cores. Figure 1 shows that this phenomenon holds for a variety of queries from the Star Schema Benchmark [27].

Our Problem

In practice, many system schedulers are aware of each job's current phase [26, 34]. The phases of a database query pipeline are invoked explicitly during query execution [38]. Cluster schedulers [10], distributed computing platforms such as Hadoop [31] and Apache Spark [9, 37], distributed machine learning frameworks[24], and supercomputing centers all process jobs composed of a mixture of highly parallelizable and highly sequential phases.

While the above systems have the capability to detect the current phase of each running job, they do not effectively leverage this information to make optimal scheduling decisions. In this paper, we address the problem of *phase-aware scheduling* — using the available phase information to allocate resources efficiently across jobs. Given a stream of parallelizable jobs composed of multiple phases, our goal is to design scheduling policies which decide, at every moment in time, how many cores or servers to allocate to each job.

Why Phase-Aware Scheduling has not Been Solved

The systems community, theoretical computer science (TCS) community, and stochastic performance modeling community have all done significant work on the problem of parallel job scheduling. However, we will see that both existing theoretical results and state-of-the-art systems schedulers can be improved by the use of phase-aware scheduling policies.

The typical approach of the systems community is to defer scheduling decisions to the user by relying on reservation-based systems [20, 30, 35]. Here, users reserve the number of cores or servers on which they want to run their jobs. Unfortunately, it is well-known that users tend to conservatively over-provision resources, leading to suboptimal resource allocations [10, 35].

While phase-aware systems schedulers do exist, they often make suboptimal scheduling decisions. For instance, database schedulers use phase knowledge to avoid over-allocating cores to queries which are in an inelastic phase, but otherwise process queries in first-come-first-served (FCFS) order [22]. We refer to this policy as *Phase-Aware FCFS* (`PA-FCFS`), to distinguish it from a naive FCFS policy that over-allocates to inelastic phases. We will see that `PA-FCFS` can be far from optimal.

The approach of the TCS community has been to analyze the problem through the lens of competitive analysis, where it is assumed that the arrival sequence of jobs is chosen adversarially. This work either assumes that jobs consist of phases with different degrees of scalability or that each job is encoded as a directed acyclic graph (DAG) [3, 11]. In these adversarial settings, strong lower bounds have been obtained on the achievable competitive ratio. In particular, no scheduling policy can perform within a constant factor of the optimal policy in the worst case [23]. The TCS community has also found policies that match these lower bounds, such as the EQUI policy [11, 21] which divides servers evenly between all jobs currently in the system. This has led the TCS community to conclude that the problem of scheduling parallelizable jobs is solved, even though these policies frequently perform worse than PA-FCFS (see Section 7).

The stochastic community has thus far largely assumed that all jobs follow the same, single speedup function that dictates how parallelizable the jobs are [4]. However, this work has not addressed how to schedule jobs whose parallelizability changes over time.

Optimal Phase-Aware Scheduling

In summary, although real-world systems process jobs composed of phases, and these systems are often aware of the current phase of each job, phase-aware scheduling remains an open problem. Hence, our first contribution is a stochastic model of jobs composed of multiple phases with different levels of parallelizability. Under this model we derive a provably optimal scheduling policy. The policy we derive, IF, is non-obvious and greatly outperforms both the PA-FCFS policy used in real systems and the EQUI policy proposed in the worst-case literature. Because our model makes some simplifying assumptions, we validate the performance of IF through a range of simulations including a simulation of a database running queries from the Star Schema Benchmark [27].

Contributions of This Paper

- In Section 3, we develop a novel model of parallelizable jobs composed of elastic and inelastic phases where the scheduler knows, at all times, what phase a job is in. Our model is far more general than prior work from the stochastic community which has assumed that all jobs follow the same, single speedup function.
- We prove that the *Inelastic First* (IF) policy, which *defers parallelizable work* by giving strict priority to jobs which are in an inelastic phase, is optimal under our model. Because the proof of optimality requires a complex coupling argument, we break this claim down by considering special cases which are easier to understand. We begin by proving the optimality of IF in simpler models in Section 5 before proving our more general claim in Section 6.
- In Section 7, we perform an extensive simulation-based performance evaluation, illustrating that IF outperforms a range of scheduling policies. Even in settings that violate the assumptions of our model, IF can perform nearly 30% better than the PA-FCFS policy used in modern databases and a factor of 3 better than the EQUI policy advocated by the TCS community.
- Finally, in Section 8, we perform a case study on scheduling in databases where queries consist of elastic and inelastic phases. In this setting, the scheduler sometimes has additional information about each query beyond just the query's current phase. We show how IF can be generalized to leverage this additional information. This generalization improves upon state-of-the-art database scheduling by roughly 50% in simulation.

2 PRIOR WORK

It is easiest to understand the prior theoretical work on scheduling parallelizable jobs in terms of the model of parallelism considered. We will therefore discuss several theoretical models of parallelism before considering prior work from the systems community on scheduling parallelizable jobs.

Jobs with Parallelizable Phases

The closest theoretical work to ours comes from the worst-case scheduling community [11–14]. This work similarly considers the problem of scheduling parallelizable jobs composed of phases of differing parallelizability. Due to the worst-case nature of the analysis, this work is forced to either consider an offline problem where all jobs arrive at time 0 [13], or to rely on resource augmentation[1] [11, 12, 14] to provide an algorithm which is within a (potentially large) constant factor of the optimal policy. This work concludes that the EQUI policy, as well as a generalization of it, is constant competitive given a small constant resource augmentation.

A related work from the SPAA community [5] recognizes that jobs have elastic and inelastic phases. However, for analytical tractability, [5] assumes that jobs consist of only a *single* phase, and are therefore either fully elastic or fully inelastic. Even in this limited setting, [5] requires that the inelastic jobs are smaller on average than the elastic jobs. By contrast, our model allows each job to have any number of phases, with different jobs having different numbers of phases. Furthermore, our model does not make any assumptions about the relative sizes of elastic and inelastic phases.

Jobs with Speedup Curves

Other theoretical work has also considered a model where, instead of consisting of phases, each job follows a single *speedup function*, $s(k)$, that describes the speedup a job receives from running on k servers. Here, $s(k)$ is some positive, concave, non-decreasing function. Work using this model from the worst-case scheduling literature finds that, when job sizes are known, a generalization of EQUI is $O(\log p)$-competitive with the optimal policy, where p is the ratio of the largest job size to the smallest job size [21]. Moreover, EQUI is again shown to be constant competitive with constant resource augmentation [12, 14]. In an analogous result using this model from the performance modeling community, [4] finds that EQUI is the optimal policy when job sizes are unknown and exponentially distributed.

Overall, the general consensus from both the worst-case scheduling community and the performance modeling community is that EQUI should be used to achieve good or possibly optimal mean response time. However, as we will see, EQUI is far from optimal when jobs are composed of elastic and inelastic phases (see Figure 7). This discrepancy is largely due to the overly pessimistic nature of the prior theory work, which all assumes that the system is incapable of determining how parallelizable a job is at each moment in time. We assume that the scheduler knows whether a job is in an elastic phase or an inelastic phase, which is reasonable for a wide range of systems [9, 17, 25, 29]. As a result, we are able to provide the optimal policy with respect to mean response time in a variety of cases.

DAG Jobs

A separate branch of theoretical work on scheduling parallel jobs that developed concurrently with the above models considers every parallel job as consisting of a set of tasks with precedence constraints specified by a Directed Acyclic Graph (DAG). In this model, introduced in [7], a task can only run on a single server, but any two tasks that do not share a precedence relationship can be run in parallel. Much of the work in this area is concerned with how to efficiently schedule a *single* DAG job onto a set of servers [6–8]. When multiple DAG jobs arrive online, there are strong lower bounds on the competitive ratio of any online algorithm for mean response time [23]. Recently, [3] considered the online problem of scheduling a stream of DAG jobs to minimize the

[1] Resource augmentation analysis is a relaxation of competitive analysis that, for some $s > 1$, compares an algorithm using speed s processors against the optimal policy using speed 1 processors.

worst case mean response time. Using a resource augmentation argument, they show that EQUI and its generalization are constant competitive with constant resource augmentation.

Systems Literature

The need to schedule jobs with sublinear speedup functions has been corroborated across a wide range of systems. Perhaps most famously, the computer architecture community identified Amdahl's law [19] around the advent of multicore architectures. The problem of scheduling parallelizable jobs is similarly known in the context of data center scheduling [10], supercomputing [28, 33], distributed machine learning [24], databases [15], and distributed computing frameworks such as MapReduce [9, 36]. Existing schedulers in these contexts are highly dependent on heuristics [10, 18, 25, 29], often require significant parameter tuning, and do not provide formal guarantees about performance. Our goal is to improve upon these state-of-the-art heuristic policies by providing practical policies with provably optimal or near-optimal performance.

3 MODEL

In this section, we develop a model of jobs composed of distinct phases running in a system consisting of K homogeneous servers.

Multi-phase Jobs

We begin by noting that, in a wide range of systems applications, job phases are either highly parallelizable or highly sequential. This can be clearly seen in the case of database queries in Figure 1. A similar phenomenon applies in systems using a map-reduce paradigm [9] where parallelizable map stages are interlaced with sequential reduce stages. Machine learning training jobs also consist of highly parallelizable iterations of distributed gradient descent followed by a sequential step which coalesces the results on a central parameter server[24]. Hence, while job phases could potentially experience intermediate parallelizability, we will consider the highly practical case where job phases are either *elastic*, perfectly parallelizable, or *inelastic*, totally sequential.

To model the duration of each job phase, we define a phase's *inherent size* to be the amount of time it takes the phase to complete when run on a single server. For analytical tractability, we will assume that inelastic phase sizes are distributed as $\mathtt{Exp}(\mu_I)$ and elastic phase sizes are distributed as $\mathtt{Exp}(\mu_E)$, and that all phase sizes are independently distributed. Although the scheduler often knows the current phase of each job in the system, it is less common in real systems for the scheduler to know the full sequence of phases comprising each job, or the size of each phase. Hence, we will generally assume that the scheduler knows the current phase of each job, but that the scheduler does not know the future phases or any of the phase sizes of a job. In Section 8, we will consider the specific case of scheduling in databases, where it is common for the database to have additional information about the phases and phase sizes of each job.

Only elastic phases can be parallelized across multiple servers. An elastic phase of size x, when run on k servers, takes $\frac{x}{k}$ time to complete. Equivalently, the running time of an elastic phase on k servers can be viewed as a random variable which is distributed as $\mathtt{Exp}(k\mu_E)$. By contrast, an inelastic phase cannot be parallelized and runs on at most one server at any moment in time.

Because the sizes of a job's phases are assumed to be exponentially distributed and unknown to the system, we can model a multi-phase job via a continuous-time Markov chain, as seen in Figure 2. We will model each job via a Markov chain consisting of three states: an E state that denotes that the job is in an elastic phase, an I state that denotes that the job is in an inelastic phase, and an absorbing state, C, that denotes that the job has been completed. Each arriving job can either start in the E state or in the I state. We will assume that a job can only transition to the completion state from the inelastic state. This is realistic for a wide range of systems where the results of a parallel computation must be sequentially coalesced and returned to the user [9, 16, 37]. It also simplifies our analysis without weakening our results (see Remark 3). We define q to be the probability that a job completes after an inelastic phase; with probability $1 - q$ the job will transition to an elastic phase.

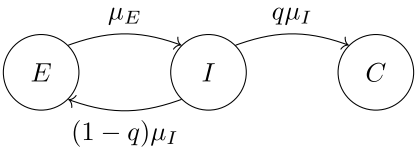


Fig. 2. The Markov chain governing the evolution of a multi-phase job when running on a single server. E refers to the elastic phase, I refers to the inelastic phase, and C is the completion state.

We assume that all jobs are modeled by the same underlying Markov chain. However the exact number of phases and the sizes of the phases belonging to each job can be different. Under this model, the expected total inherent size of a job depends on whether the job begins with an E phase or an I phase, and is given by the following expressions:

$$\begin{aligned}\mathbb{E}[\text{Job size if start in } E] &= \left(\frac{1}{\mu_E} + \frac{1}{\mu_I}\right)\frac{1}{q} \\ \mathbb{E}[\text{Job size if start in } I] &= \left(\frac{1}{\mu_E} + \frac{1}{\mu_I}\right)\frac{1}{q} - \frac{1}{\mu_E}\end{aligned}$$

We refer to the completion of a job's final inelastic phase as a *job completion*. We refer to the completion of any of the job's phases as a *transition*. An *inelastic transition* occurs when an inelastic phase is completed and an *elastic transition* occurs when an elastic phase is completed.

Scheduling Policies

A *scheduling policy*, π, determines how to allocate the K servers to the present jobs at every moment in time. While our policies are fully preemptive, we assume that policies only change their allocation at times of job arrivals, transitions, or job completions. When a job is in its inelastic phase, it can be allocated up to 1 server, i.e., fractional allocations are admitted. When a job is in its elastic phase, it can be allocated any number of servers up to K.

This paper will focus on the analysis of the *Inelastic First* (IF) policy. The key property of IF is that it *defers parallelizable work*. That is, at every moment in time, IF gives strict priority to jobs which are in inelastic phases. Specifically, if there are i jobs in the system that are in their inelastic phase, then IF will allocate $\min\{i, K\}$ servers to these jobs. Any remaining servers will be allocated to a job in an elastic phase if such a job exists, otherwise these extra servers will remain idle.

We will show that IF is optimal with respect to minimizing mean response time. Observe that IF does not require any knowledge of the job parameters (μ_I, μ_E, and q). Thus, optimally scheduling multi-phase jobs can be done regardless of whether these parameters are known to the system.

Arrival Processes and Metrics

We allow for an arbitrary arrival process. To be precise, we first define an *arrival time sequence* as two fixed, infinite sequences, $(t_n)_{n\geq 1}$ and $(\ell_n)_{n\geq 1}$, where t_n is the time at which the nth job arrives and $\ell_n \in \{E, I\}$ denotes whether the arriving job begins with either an E phase or an I phase. We define an *arrival time process* as a distribution over arrival sequences.

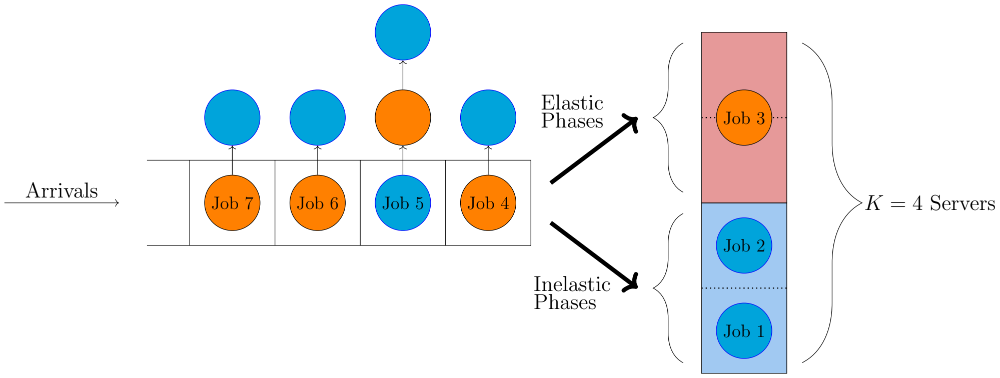


Fig. 3. The central queue and servers for our system. Jobs 1-7 are all modeled by the Markov chain presented in Figure 2. We use the color orange to illustrate the elastic phases of jobs, and blue to illustrate the inelastic phases. While we assume the number of remaining phases is unknown to the scheduler, we have drawn out the remaining phases to illustrate job structure. Here, there are $K = 4$ servers. At this moment, servers 1 and 2 are allocated to jobs in an inelastic phase (Jobs 1 and 2), and servers 3 and 4 are allocated to a single job in the elastic phase (Job 3).

We define the *response time* of the nth job under policy π to be the time from when the job arrives until it completes. We denote this quantity by the random variable $T_\pi^{(n)}$. We let T_π denote the the steady-state response time whenever this quantity exists.

As an example, consider the case where the arrival time process is a Poisson process with rate λ and each job starts with an E phase with probability r_E and with an I phase with probability r_I. Then we can define the system load as:

$$\rho = \text{System load } = \frac{\lambda \cdot \mathbb{E}[\text{Job size}]}{K},$$

where

$$\mathbb{E}[\text{Job size}] = r_E \cdot \mathbb{E}[\text{Job size if start in } E] + r_I \cdot \mathbb{E}[\text{Job size if start in } I] \ .$$

In this setting, if $\rho < 1$, the steady-state mean response time under policy π exists and is denoted by $\mathbb{E}[T_\pi]$.

Stochastically Minimizing the Number of Jobs in System

Our goal is to show that IF minimizes the steady-state mean response time across jobs. To show this, we will prove a series of claims about the number of jobs in the system at any point in time. Namely, we will argue that IF stochastically maximizes the number of jobs completed by any point in time. This is equivalent to saying IF stochastically minimizes the the number of jobs in system at any point in time.

To reason about the number of completions by time t, we will count the number of elastic and inelastic transitions as well as the number of job completions. We define $C_\pi(t)$ to be the number of job completions by time t under policy π. We define $I_\pi(t)$ (and $E_\pi(t)$) to be the number of inelastic (resp. elastic) transitions under policy π by time t. Finally, we define $I_\pi(s, t)$ to be the number of inelastic transitions under π on the interval $(s, t]$ and we define $E_\pi(s, t)$ and $C_\pi(s, t)$ analogously.

With respect to the number of jobs in system, let $N_\pi(t)$ denote the number of jobs present at time t, under policy π. We define $N_\pi^E(t)$ to be the number of jobs in an elastic phase at time t under π and we define $N_\pi^I(t)$ to be the number of jobs in an inelastic phase at time t under π.

4 OVERVIEW OF THEOREMS

In this section, we provide an overview of the theoretical results in Sections 5 and 6.

4.1 Main Result

We first state the main theorem in full generality. At a high level, the theorem states that IF is the most effective policy in terms of completing jobs. More specifically, we show that the number of jobs completed by any point in time under IF stochastically dominates the number of jobs completed by the same time under any other algorithm.

THEOREM 1. *Consider a K server system serving multi-phase jobs. The policy* IF *stochastically maximizes the number of jobs completed by any point in time. Specifically, for a policy A, let $C_A(t)$ denote the number of jobs completed by time t and let $N_A(t)$ denote the number of jobs in the system at t. Then under any arbitrary arrival time process, $C_{\texttt{IF}}(t) \geq_{st} C_A(t)$ for all times $t \geq 0$. Consequently, $N_{\texttt{IF}}(t) \leq_{st} N_A(t)$ for all times $t \geq 0$.*

We can leverage Theorem 1 to derive far-reaching results about job response time. In particular, if the arrival time process is a renewal process[2], we can show that IF minimizes the steady-state mean response time. We formalize this idea in the following immediate corollary of Theorem 1.

COROLLARY 2. *Suppose the same system setup as in Theorem 1. For any arbitrary policy A, let T_A be the steady-state job response time when it exists. Then, if the arrival time process is a renewal process, we have $\mathbb{E}[T_{\texttt{IF}}] \leq \mathbb{E}[T_A]$.*

PROOF. By Theorem 1, we know IF stochastically minimizes the number of jobs in the system at any point in time. Since the arrival time process is a renewal process, this implies IF minimizes the steady-state mean number of jobs in the system. By Little's law, minimizing the mean number of jobs in the system suffices for minimizing the steady-state mean response time. ■

The takeaway from Theorem 1 and its corollary is that there is a massive benefit to *deferring parallelizable work* by prioritizing inelastic phases. Specifically, while elastic phases can be completed quickly by parallelizing across all servers, there are benefits to keeping elastic phases in the system. These elastic phases are flexible and can keep the system running at high efficiency. It is also possible to allocate some servers to inelastic phases without significantly increasing the runtime of an elastic phase. For these reasons, the optimal policy, IF, defers as much parallelizable work as possible without over-allocating to inelastic phases.

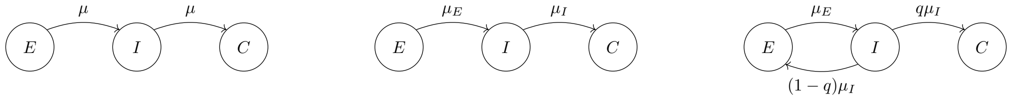
Remark 3. One might assume that that IF benefits not from deferring parallelizable work, but rather from how we have defined our model, where jobs in inelastic phases have smaller expected remaining sizes. This is a misconception. As we show in Section 8, favoring jobs with smaller remaining sizes is not nearly as important as deferring parallelizable work in real-world settings.

4.2 How We Prove Theorem 1

We now provide a road map for how we prove Theorem 1. The high-level picture is that it suffices to find a coupling between two systems, one running IF and one running an arbitrary policy A, under which $C_{\texttt{IF}}(t) \geq C_A(t), \forall t \geq 0$. However, finding such a coupling is difficult due to the complicated job structure in Figure 2. In particular, the inherent size distributions are different between the two phases and jobs are composed of an unknown number of elastic and inelastic phases.

We therefore begin by considering several simpler job structures, as seen in Figure 4. The simplest job structure has elastic and inelastic phases with the same size distribution, and no transitions from

[2] Here, by a renewal process, we mean the inter-arrival times $t_n - t_{n-1}$ are i.i.d., and that the initial phases of jobs p_n are i.i.d. as well.



(a) Two-phase job with equal rate

(b) Two-phase job with unequal rates

(c) Multi-phase job

Fig. 4. The three job structures we consider. E refers to the elastic state, I refers to the inelastic state, and C refers to the completion state. In Figure 4(a), jobs just have two phases, both with inherent size distributed as $\mathsf{Exp}(\mu)$. In Figure 4(b), jobs still have two phases. Phase I has size distributed as $\mathsf{Exp}(\mu_I)$, and phase E has size distributed as $\mathsf{Exp}(\mu_E)$. In Figure 4(c), we add in potential transitions from phase I to phase E.

the inelastic phase to the elastic phase. We then add the complexities gradually back to the model. In each case, we argue that studying the number of inelastic transitions suffices to understand the total number of job completions. Recalling that $I_A(t)$ is the number of inelastic transitions under A by time t, we prove results of the form: "For any policy A, there exists a coupling under which $I_{\mathtt{IF}}(t) \geq I_A(t)$ for all $t \geq 0$. Consequently, $I_{\mathtt{IF}}(t) \geq_{st} I_A(t)$ for all $t \geq 0$."

In Section 5.1, we start with the simplest job structure as shown in Figure 4(a). In this structure, jobs consist of a single elastic phase followed by a single inelastic phase. Moreover, we assume the inherent sizes of both the elastic and inelastic phases are identically distributed as $\mathsf{Exp}(\mu)$. We refer to such jobs as *two-phase jobs with equal rates*. We are able to couple two systems experiencing jobs of this structure by (1) having them experience the same sequence of arrivals and (2) splitting time into roughly uniform chunks of length $\mathsf{Exp}(K\mu)$. At the end of each chunk of time, the systems will both potentially experience a job transition. By splitting time into "busy" and "idle" periods under this coupling (as defined in the proof of Lemma 4), we prove the desired result.

We then consider the slightly more complicated job structure shown in Figure 4(b) in Section 5.2. In this job structure, jobs again consist of a single elastic phase followed by a single inelastic phase. However, we now assume the inherent sizes of the elastic and inelastic phases are no longer identically distributed. We refer to such jobs as *two-phase jobs with unequal rates*. While having unequal rates between phases complicates splitting time into roughly equal blocks, we work around this by leveraging a trick called *uniformization*. More specifically, we reformulate this more general job structure via a Markov chain in which the elastic and inelastic phases have the same inherent size distribution, but some additional self-loop transitions are added to the chain. We then expand our existing coupling argument by coupling the transition outcomes of the two systems.

Finally, we consider the general job structure as shown in Figure 4(c) in Section 6. In this job structure, jobs can have alternating elastic and inelastic phases, each with a different service rate. We refer to such jobs as *multi-phase jobs*. This case may seem very different from the previous settings, since now an inelastic transition can produce an elastic phase. However, we show that such a transition can be viewed as a job completion followed immediately by an arrival of a job beginning with an elastic phase. Using this argument, we show how a coupling in the general case follows from our coupling in the cases with two-phase jobs.

5 TWO-PHASE JOBS

5.1 Two-Phase Jobs with Equal Rates

We first consider two-phase jobs with equal rates. These are jobs that consist of a single elastic phase followed by a single inelastic phase where both phases have inherent size distributed as $\mathsf{Exp}(\mu)$, as illustrated in Figure 4(a).

LEMMA 4. *Consider a K server system serving two-phase jobs with equal rates. Consider any policy A and let the policies* IF *and A start from the same initial conditions and have the same arrival time process. Then there exists a coupling between* IF *and A such that $I_{\texttt{IF}}(t) \geq I_A(t)$ and $C_{\texttt{IF}}(t) \geq C_A(t)$ for all $t \geq 0$, where $I_{\texttt{IF}}(t)$ (resp. $I_A(t)$) is the number of inelastic transitions by time t under* IF *(resp. A), and $C_{\texttt{IF}}(t)$ (resp. $C_A(t)$) is the number of jobs completed by time t under* IF *(resp. A).*

The proof of Lemma 4 can be found in Appendix A. We describe the coupling used in the proof below in Section 5.1.1, as it serves as a building block for subsequent arguments.

Note that for two-phase jobs with equal rates, every inelastic job transition is also a completion, so we have $I_A(t) = C_A(t)$ for all times $t \geq 0$ under any policy A. Therefore, to prove Lemma 4, it suffices to construct a coupling under which $I_{\texttt{IF}}(t) \geq I_A(t)$ for all $t \geq 0$. Then the claim $C_{\texttt{IF}}(t) \geq C_A(t)$ follows directly.

5.1.1 *Coupling* IF *and A.*

Let $S_{\texttt{IF}}$ be the system running IF and S_A be the system running any arbitrary policy A. The high level intuition of the coupling is as follows. Since both phases, inelastic and elastic, have inherent size $\texttt{Exp}(\mu)$, we can parse time into blocks of length $\texttt{Exp}(K\mu)$. At the end of each of these blocks, both systems will potentially experience a job transition. Outside of these points of time, no job transitions can occur. This makes counting job completions/inelastic transitions much simpler. Arrivals do not directly impact the number of transitions/job completions, and hence we do not need assumptions on the arrival time process.

Job arrivals: We assume that the two systems, $S_{\texttt{IF}}$ and S_A, have the same number of jobs in each phase at time 0 (for instance, 7 jobs in an inelastic phase, and 3 jobs in an elastic phase). Formally, we assume that $N^E_{\texttt{IF}}(0) = N^E_A(0)$ and $N^I_{\texttt{IF}}(0) = N^I_A(0)$.

We fix an arrival time sequence which is shared between $S_{\texttt{IF}}$ and S_A. Recall that an arrival sequence is just a fixed sequence of arrival times $(t_n)_{n\geq1}$ and a corresponding binary sequence $(\ell_n)_{n\geq1}$, where t_n is the time the nth overall job arrival occurs in both systems and $\ell_n \in \{E, I\}$ determines which phase a job starts in.

Job transitions and departures: Suppose the current time is t. We generate a random variable $X \sim \texttt{Exp}(K\mu)$, that is shared by both systems. Suppose s is the next unrealized arrival time in the arrival sequence. If $s < t + X$, we allow the arrival to occur simultaneously into both systems. We then set $t \longleftarrow s$, and return to the beginning of this paragraph. If $s > t + X$, then we set the current time to be $t \longleftarrow t + X$, and then select one of the K servers uniformly at random[3] (we select the same server in both systems). If a system is running a job in its inelastic phase on this randomly selected server, it is assumed to depart. Likewise, if the server is running a job in its elastic phase, the system experiences an elastic transition, producing an inelastic phase. Lastly, if the server selected is idling, nothing happens. This general event (which may or may not result in a transition/departure) will be referred to as a *potential transition*. In general, a time where either an arrival or potential transition occurs will be referred to as an *event time*.

Additionally, if a system, at time t, is serving i inelastic jobs, we assume they are running on servers 1 through i. If an elastic job is being served, it is run on servers $i + 1$ through e, where e is some number less than or equal to K. The remaining servers are left idle.

5.2 Two-Phase Jobs with Unequal Rates

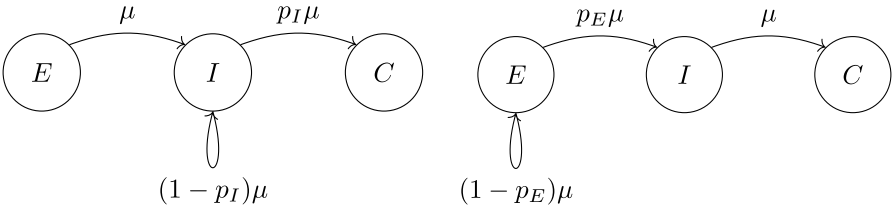
We now consider two-phase jobs with unequal rates, as illustrated in Figure 4(b). In this case, the inherent sizes of elastic phases are distributed as $\texttt{Exp}(\mu_E)$, and the inherent sizes of inelastic

[3] Here, for the sake of simplicity, we assume that jobs can only be allocated an integral number of servers. However, our result generalizes to the case where allocations are fractional. When allocations are fractional, we treat the servers as a continuous interval, $[0, K]$ and generate $U \sim \text{Unif}[0, K]$. The type of phase running at the corresponding point in the interval $[0, K]$ determines what type of transition occurs.

phases are distributed as $\mathsf{Exp}(\mu_I)$. In this section, we will show how to generalize the coupling in Section 5.1.1 to establish Lemma 5, below.

Lemma 5. *Consider a K server system serving two-phase jobs with unequal rates. Consider any policy A and let the policies* IF *and A start from the same initial conditions and have the same arrival time process. Then there exists a coupling between* IF *and A such that $I_{\mathtt{IF}}(t) \geq I_A(t)$ and $C_{\mathtt{IF}}(t) \geq C_A(t)$ for all $t \geq 0$, where $I_{\mathtt{IF}}(t)$ (resp. $I_A(t)$) is the number of inelastic transitions by time t under* IF *(resp. A), and $C_{\mathtt{IF}}(t)$ (resp. $C_A(t)$) is the number of jobs completed by time t under* IF *(resp. A).*

At first glance, it is not clear how to apply the coupling in Section 5.1.1 to the situation where different phases (elastic and inelastic) have different exponential rates (μ_E and μ_I). The key component of our coupling in Section 5.1.1 was that we could parse time into blocks of length $\mathsf{Exp}(K\mu)$ to keep both systems in sync. Now, the size of the blocks could depend on which types of phases (elastic or inelastic) are being served, and thus may be unequal between the two systems.



(a) Two-phase job, unequal rates, $\mu_E > \mu_I$ (b) Two-phase job, unequal rates, $\mu_E < \mu_I$

Fig. 5. Two cases of uniformizing two-phase jobs with unequal rates. In Figure 5(a), $\mu_E > \mu_I$, so we take our dominating rate as $\mu := \mu_E$. We then take $p_I := \frac{\mu_I}{\mu_E}$, and set the inherent size of the inelastic phase to be $\mathsf{Exp}(\mu)$. With probability $1 - p_I$, after completing the inelastic phase, we immediately start another one. With probability p_I, the job completes and exits the system. The description of Figure 5(b) is analogous.

To tackle this problem, we leverage the technique of Markov chain *uniformization*. In uniformization, we find a rate μ which is larger than the transition rates at any state of the Markov chain. For instance, if $\mu_E > \mu_I$, we take $\mu := \mu_E$. We then set the transition rates of both states to be μ. Since $\mu_I < \mu$, we add a self-loop at the inelastic state. This self-loop occurs with probability $1 - p_I$, where $p_I = \frac{\mu_I}{\mu_E}$. With complementary probability p_I, the job will complete and exit the system. Figure 5(a) shows the resulting uniformized Markov chain. It is easy to confirm that the uniformized Markov chain is equivalent to the original Markov chain (Figure 4(b)). The case where $\mu_E < \mu_I$ is exactly analogous and the uniformized Markov chain for this case is shown in Figure 5(b).

Going forward, when we refer to inelastic transitions by time t, $I_A(t)$, we refer to the number of transitions in the uniformized job model. This holds analogously for $E_A(t)$. In some cases, $I_A(t)$ can differ from the total number of job completions, $C_A(t)$. However, under the coupling we present, the number of inelastic transitions can be used to directly recover the total number of job completions.

5.2.1 ***System Coupling.***

Our goal in the coupling is twofold. Once again, we want to chop up time into blocks of length $\mathsf{Exp}(K\mu)$ to keep $S_{\mathtt{IF}}$ and S_A roughly in sync. Additionally, we want to construct a coupling where reasoning about the number of inelastic transitions, $I_A(t)$, suffices for reasoning about total job completions, $C_A(t)$. More specifically, we want to find a coupling under which $I_{\mathtt{IF}}(t) \geq I_A(t), \forall t \geq 0$ implies $C_{\mathtt{IF}}(t) \geq C_A(t), \forall t \geq 0$.

Job arrivals: $S_{\mathtt{IF}}$ and S_A share the same arrival time sequence, and start with the same initial conditions.

Job transitions and departures: Our coupling in this case closely follows the coupling in Section 5.1.1. Specifically, the current time t is updated in the same manner as in Section 5.1.1. However, we handle potential transitions slightly differently, due to uniformization. We only discuss the case $\mu_I < \mu_E$, as the reverse case can be handled symmetrically.

When $\mu_I < \mu_E$, we take our dominating rate to be $\mu := \mu_E$. We generate an infinite sequence, $(X_n)_{n\geq 1}$, of i.i.d. *Bern*(p_I) random variables, where $p_I = \frac{\mu_I}{\mu_E}$. The realizations of $(X_n)_{n\geq 1}$ are shared between the two systems. These *coin flips* will determine whether an inelastic transition results in a self-loop or in a job completion.

Throughout time, both systems keep track of the total number of inelastic transitions which have occurred. More concretely, each system starts with its own counter, n, which is initialized to 0. For each system, if we randomly select a server holding an inelastic job while experiencing a potential transition, we increment this system's counter ($n \leftarrow n + 1$). We then check position n of the shared infinite sequence of coin flips. If $X_n = 1$, the inelastic job completes and exits the system. Otherwise, we have a self-loop transition and no job exits the system.

Since $S_{\texttt{IF}}$ and S_A share a common sequence of coin flips, $I_{\texttt{IF}}(t) \geq I_A(t)$ implies $C_{\texttt{IF}}(t) \geq C_A(t)$.

5.2.2 ***Proof of Lemma 5.*** Since we just need to show $I_{\texttt{IF}}(t) \geq I_A(t), \forall t \geq 0$, we can use the proof of Lemma 4 found in Appendix A verbatim to prove Lemma 5.

6 OPTIMALITY IN THE GENERAL CASE

We now consider the fully general multi-phase job structure, as seen in Figure 4(c). In order to prove Theorem 1, it suffices to prove Lemma 6 below.

LEMMA 6. *Consider a K server system serving multi-phase jobs. Consider any policy A and let the policies* IF *and A start from the same initial conditions and have the same arrival time process. Then there exists a coupling between* IF *and A such that $I_{\texttt{IF}}(t) \geq I_A(t)$ and $C_{\texttt{IF}}(t) \geq C_A(t)$ for all $t \geq 0$, where $I_{\texttt{IF}}(t)$ (resp. $I_A(t)$) is the number of inelastic transitions by time t under* IF *(resp. A), and $C_{\texttt{IF}}(t)$ (resp. $C_A(t)$) is the number of jobs completed by time t under* IF *(resp. A).*

As in Section 5.2, we use uniformization to rewrite the job structure of Figure 4(c) so that the elastic and inelastic phase transitions have equal rates. There are two possible uniformizations here, once again depending on how μ_E and μ_I relate. Determining the dominating rate μ and transition probabilities p_I or p_E is the same as in Section 5.2, and the two possible uniformized Markov chains are shown in Figure 6. With these job structures in mind, we present the system coupling which allows us to prove the optimality of IF.

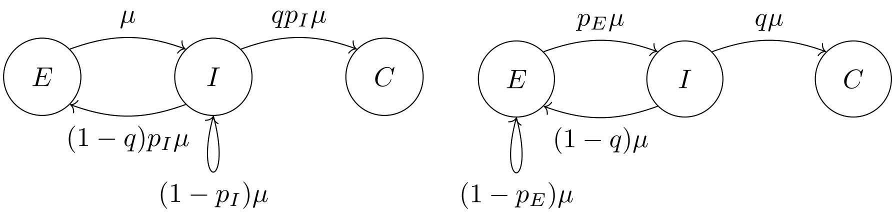
6.1 System coupling

As in Section 5.1.1, we wish to construct a coupling that keeps systems S_A and $S_{\texttt{IF}}$ in sync with respect to potential transition times and that allows us to use $I_A(t)$ to reason about $C_A(t)$.

Job arrivals: As in Sections 5.1.1 and 5.2.1, we let $S_{\texttt{IF}}$ and S_A share the same arrival time sequence and start with the same initial conditions.

Job transitions and departures: For the most part, the transition process is similar to the uniformized case presented in Section 5.2.1. However, while we previously only needed a single infinite sequence of i.i.d. Bernoulli random variables, here we will need two. We state the two cases ($\mu_E < \mu_I$ and $\mu_E > \mu_I$) separately, as they differ slightly in their construction.

First, we consider $\mu_E < \mu_I$ (Figure 6(b)). Here, instead of a single sequence of coin flips, we have two shared sequences of coin flips. The first sequence, $(X_n)_{n\geq 1}$, is an i.i.d. sequence of *Bern*(p_E)



(a) Multi-phase job, $\mu_E > \mu_I$

(b) Multi-phase job, $\mu_E < \mu_I$

Fig. 6. Two cases of uniformizing multi-phase jobs. In Figure 6(a), $\mu_E > \mu_I$, so we take our dominating rate as $\mu := \mu_E$. We then take $p_I := \frac{\mu_I}{\mu_E}$, and set the inherent size of the inelastic phase to be Exp(μ). With probability $1-p_I$, after completing the inelastic phase, we immediately start another one. With complementary probability p_I, the job does one of two things. With probability q, it completes. Otherwise, with probability $1-q$, it begins an elastic phase. Figure 6(b) can be described similarly.

random variables. If $X_n = 1$, the nth elastic transition results in an elastic phase completion, producing an inelastic phase. Otherwise, if $X_n = 0$, the elastic transition does not result in a phase completion. The second sequence, $(Y_n)_{n\geq 1}$, is a sequence of i.i.d. $Bern(q)$ random variables. Recall that q is the probability that the completion of an inelastic phase will result in a job completion. If $Y_n = 0$, the nth inelastic transition results in the creation of an elastic phase. If $Y_n = 1$, the nth inelastic transition results in a job completion.

The case when $\mu_E > \mu_I$ (Figure 6(a)) is slightly more complex. Here, we do not have any self-loops for elastic phases. However, there are three possible outcomes for inelastic phases. We therefore keep track of two sequences of i.i.d. Bernoulli random variables, $(X_n)_{n\geq 1}$ and $(Y_n)_{n\geq 1}$. In the first sequence, X_n is distributed as $Bern(p_I)$. In the second sequence, Y_n is distributed as $Bern(q)$.

If $X_n = 0$, the nth inelastic transition does not result in the completion of an inelastic phase. If $X_n = 1$, the nth inelastic transition results in a phase completion, and we then examine the sequence (Y_n). If the nth inelastic transition results in the mth overall inelastic phase completion, we check Y_m. If $Y_m = 1$, the job completes, and if $Y_m = 0$, the job transitions to an elastic phase.

Because S_{IF} and S_A share the same sequence of coin flips, comparing the number of inelastic transitions between systems is equivalent to comparing the number of job completions. That is, if $I_{\text{IF}}(t) \geq I_A(t), \forall t \geq 0$, then $C_{\text{IF}}(t) \geq C_A(t)$.

6.2 Proof of Lemma 6

Multi-phase jobs add an extra layer of complexity which prevents us from directly leveraging the arguments used in Lemmas 4 and 5. When an inelastic phase completes, there are two possible outcomes: either an elastic phase will be produced or a job will complete. Our insight is that we can view the creation of an elastic phase as a job completion immediately followed by an arrival of a job in an elastic phase. This reduction puts us back in the case of two-phase jobs with unequal rates, allowing us to invoke Lemma 5. We formalize this argument in the proof of Lemma 6 below.

Proof of Lemma 6. First, we replace each inelastic transition that produces an elastic phase with a different type of transition. Namely, we replace these transitions with a job completion followed immediately by an arrival of a job in an elastic phase. We will refer to this replacement as our *re-framing* of the problem. Observe that the schedules produced in S_{IF} and S_A remain the same under the re-framing. While the number of job completions by any point t, $C_{\text{IF}}(t)$ and $C_A(t)$, may change under this re-framing, the key insight is that the number of inelastic transitions, $I_{\text{IF}}(t)$ and $I_A(t)$

respectively, remains identical. Thus, if we can argue that IF maximizes the number of inelastic transitions by any point in time under the re-framing, it does so in the original environment as well. This is sufficient for proving that $C_{\text{IF}}(t) \geq C_A(t), \forall t \geq 0$ in the original system.

Second, observe that our proof of Lemma 5 still holds if we allow additional arrivals at potential transition times, so long as these arrivals occur simultaneously in both systems. However, under our re-framing, the arrivals we add may not occur simultaneously in S_{IF} and S_A since they are generated by inelastic transitions to elastic phases. We address this issue by establishing the following claim.

Claim. *Let the sequence of additional arrival times under the re-framing be (t_n) in S_{IF} and (s_n) in S_A. For any $n \geq 1$, we have $t_n \leq s_n$, i.e. the nth additional arrival occurs in S_{IF} before it occurs in S_A.*

We will prove this claim below, allowing us complete the proof of Lemma 6. Specifically, for any time t, let n be the index such that $t_n \leq t < t_{n+1}$. The claim tells us that S_A experiences additional arrivals at $s_1 \geq t_1, s_2 \geq t_2, \ldots, s_{n+1} \geq t_{n+1}$. However, we can view S_A as a system that has additional arrivals at $t_1, t_2, \ldots, t_{n+1}$, but chooses to not schedule these additional arrivals until after $s_1, s_2, \ldots, s_{n+1}$. Then by Lemma 5, we have $I_{\text{IF}}(t) \geq I_A(t)$, which completes the proof of Lemma 6.

The only thing left is to prove the claim above. We show inductively that the nth of these additional arrivals occurs in S_{IF} before it does is S_A. We first argue that $t_1 \leq s_1$. Observe that, on the time interval $[0, t_1 \wedge s_1]$, S_{IF} and S_A experience precisely the same sequence of arrivals. Hence, IF maximizes the number of inelastic transitions by any time $t \in [0, t_1 \wedge s_1]$. In particular, it maximizes the number inelastic transitions by time $t_1 \wedge s_1$. Since S_{IF} and S_A share the same sequences of Bernoulli random variables, it must be that the system with more inelastic transitions experiences the first inelastic to elastic transition, and hence $t_1 \leq s_1$. Now note that the schedule produced by S_A is identical to that produced by a policy which receives the additional arrival at time t_1 instead of time s_1, but just chooses to ignore its existence until later on. This allows us to assume the extra arrival into S_A occurs at t_1 instead of s_1. We then observe that, on the interval $[0, s_2 \wedge t_2]$, systems S_{IF} and S_A experience the same sequence of arrivals. Hence, by Lemma 5, we have that $t_2 \leq s_2$.

Using the same iterative argument, it follows that S_A and S_{IF} experience the same sequence of arrivals up to time $s_n \wedge t_n$, and thus by Lemma 5 we have that $t_n \leq s_n$. ■

Having proven Lemma 6, we can now prove Theorem 1.

Theorem 1. *Consider a K server system serving multi-phase jobs. The policy* IF *stochastically maximizes the number of jobs completed by any point in time. Specifically, for a policy A, let $C_A(t)$ denote the number of jobs completed by time t and let $N_A(t)$ denote the number of jobs in the system at t. Then under any arbitrary arrival time process, $C_{\text{IF}}(t) \geq_{st} C_A(t)$ for all times $t \geq 0$. Consequently, $N_{\text{IF}}(t) \leq_{st} N_A(t)$ for all times $t \geq 0$.*

Proof. Lemma 6 implies the existence of a coupling such that $C_{\text{IF}}(t) \geq C_A(t), \forall t \geq 0$. Consequently, $C_{\text{IF}}(t) \geq_{st} C_A(t), \forall t \geq 0$. Since the number of jobs in the system at time t is just the total number of arrivals by time t minus the total number of completions by time t, the claim $N_{\text{IF}}(t) \leq_{st} N_A(t), \forall t \geq 0$ also readily follows from Lemma 6. ■

7 EVALUATION

The analysis of Section 6 has shown that, when jobs have the structure presented in Figure 2, IF is optimal. In particular, IF minimizes the steady-state mean response time for any settings of μ_E, μ_I, q, and any arrival time process such that the system is stable.

The purpose of this section is two-fold. First, we examine the benefit of doing IF as opposed to other scheduling policies used in real-world systems or proposed in the literature. Second, we will relax the assumption that the phases are exponentially distributed and consider a range of

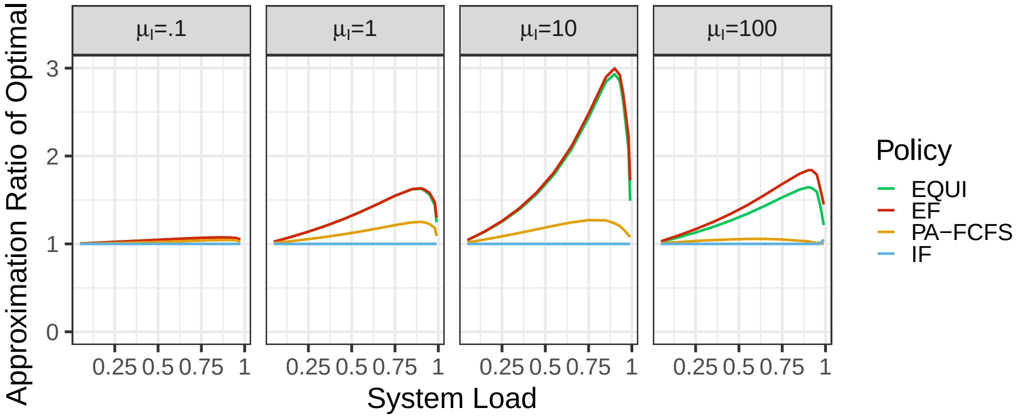


Fig. 7. The approximation ratio of mean response time under EQUI, EF, PA-FCFS, and IF when compared with the optimal mean response time. Phases have exponentially distributed inherent sizes. IF is optimal (see Section 6) and thus has an approximation ratio of 1. In each case, $K = 100$, $\mu_E = 1$, $q = 0.2$, and jobs arrive according to a Poisson process. All jobs begin with an elastic phase. Results are shown as the duration of the inelastic phase varies from $\mu_I = 0.1$ (the rare case where inelastic phases are long compared to elastic phases) to $\mu_I = 100$ (the more common case where inelastic phases are short compared to elastic phases).

phase size distributions from low-variability to high-variability. We find that, even with this relaxed assumption, IF is almost always a great choice compared to all other competitor policies.

We begin by describing the competitor policies in Section 7.1. Then we show the comparisons to IF via simulation in Sections 7.2 and 7.3.

7.1 Competitor Scheduling Policies

We compare IF to three competitor policies.

EQUI is a policy for scheduling parallelizable jobs that has been widely advocated for in both the worst-case [11–13] and stochastic [4] theoretical literature. EQUI divides severs equally among all jobs in the system. If the number of jobs in the system exceeds the number of servers, K, EQUI allocates 1 server to each of the K earliest arriving jobs.

Phase-Aware First-Come-First-Served (PA-FCFS) is a popular policy in systems applications because it is easy to implement with little space or time overhead. PA-FCFS proceeds by iteratively looking at the next earliest arriving job in the system and allocating as many servers as possible to this job until all servers have been allocated. (A job in an inelastic phase is obviously allocated only 1 server.)

Elastic-First (EF) gives strict priority to the earliest arriving job in an elastic phase. If no jobs are in an elastic phase, servers are allocated to any jobs in an inelastic phase in FCFS order. Intuitively, EF seems like it might perform well in cases where elastic phases are smaller than inelastic phases on average. In this case, EF can be thought of as a greedy policy which continuously minimizes the expected time until the next phase completion. This is analogous to the GREEDY* policy proposed in [4]. However, we will see that this intuition is wrong.

7.2 Evaluation of Policies Under Our Job Model

Figure 7 shows the results of simulations comparing the performance of IF, EQUI, PA-FCFS, and EF under the model defined in Section 3 as we vary μ_I. Each simulation consists of 100 million job completions. Although we have already proven the optimality of IF in these cases, Figure 7 shows that the improvement of IF over the competitor policies is significant. In this small sample of the parameter space, IF outperforms PA-FCFS by 25%, and outperforms EF and EQUI by a factor of 3. It is interesting to note that IF outperforms EF even when $\mu_E > \mu_I$. Even in this case, EF suffers from its failure to defer parallelizable work.

7.3 Sensitivity Analysis

Although we have shown that IF is optimal when phase sizes are exponentially distributed, we wish to further show that IF outperforms other policies under a range of phase size distributions. To examine the sensitivity of IF's performance to the underlying phase size distributions, we examine different distributions with a range of variances. Specifically, we consider the case where phases are Weibull distributed and the *squared coefficient of variation*, C^2, of the phase size distribution is both higher and lower than that of an exponential distribution.[4]

Figure 8 shows the performance of each competitor policy in simulation relative to the performance of IF (hence, the performance of IF is always normalized to 1). In most cases, IF is still the best of the four policies by a wide margin.

When $C^2 = 50$ we do find examples where EQUI outperforms IF. Here, when job sizes are highly variable, EQUI benefits from its insensitivity to the variance of job size distribution [4]. Specifically, because phase sizes have decreasing failure rates, working on phases with the *least attained service* will generally result in completing smaller phases before larger phases [2]. EQUI biases in this direction by dividing servers equally amongst all jobs in the system.

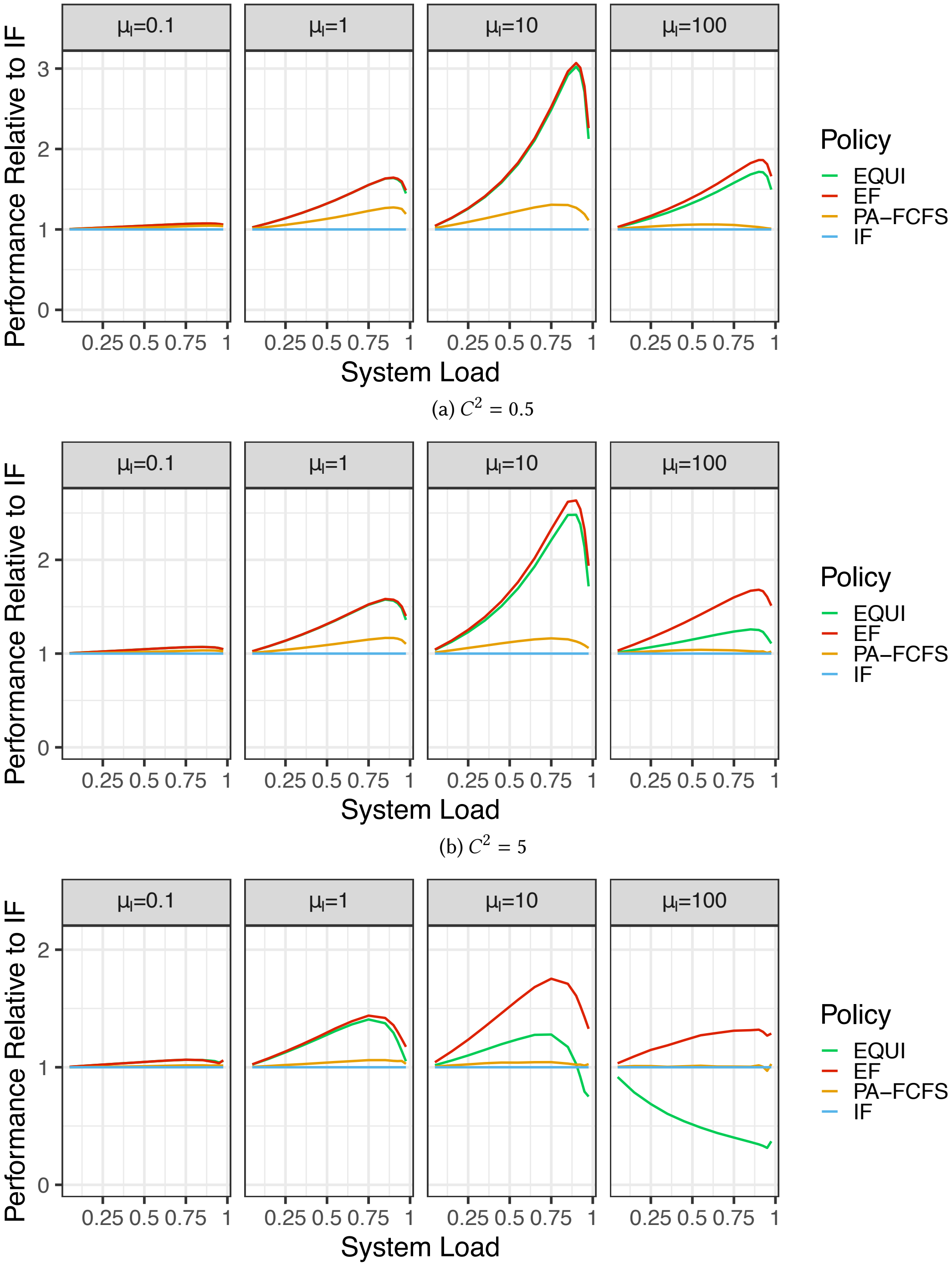
The relative performance of the competitor policies compared to IF also depends on the distribution of the number of phases comprising each job. For instance, when $q = 1$ and all jobs begin with an elastic phase, IF and PA-FCFS are equivalent policies. However, as q decreases, for a given system load, the gap between IF and PA-FCFS widens, since it becomes increasingly likely that PA-FCFS will make a mistake and give priority to an elastic phase. Similarly, when considering Weibull distributed phases, Figure 8 shows that EQUI can outperform IF when $q = .2$. However, if we instead consider the case where $q = .025$, IF again outperforms EQUI at all loads.

8 CASE STUDY: SCHEDULING IN DATABASES

Throughout this paper, we have drawn inspiration for our model from a range of systems including modern databases. In this section, we consider whether the scheduling policies that we have proposed work well for real database workloads. Because this section specifically considers scheduling in databases, we will refer to a scheduling policy as allocating *cores* to *queries* rather than allocating servers to jobs. Real database workloads differ from our modeling assumptions in two ways. First, phase sizes are not exponentially distributed. Second, the sequence of phases for each query is not determined by an underlying Markov chain. In this case study we ask whether IF will still perform well under these real-world conditions.

To answer this question, we perform simulations using a workload consisting of a mixture of 5 queries from the Star Schema Benchmark. The ordering of phases and the phase durations of each query in our simulations are based on timings of the actual queries running in the Noisepage database. Figure 9 shows the results of these simulations. The ordering of the policies with respect to mean response time is the same as what we observed in Section 7. In particular, IF is again consistently the best of the policies we consider, and IF outperforms the PA-FCFS policy used in the current version of Noisepage by up to 30%.

[4] A Weibull distribution with shape parameter $k = 1$ collapses to an exponential distribution ($C^2 = 1$). Adjusting k changes the distribution to have either higher C^2 ($k < 1$) or lower C^2 ($k > 1$) than an exponential distribution.



(a) $C^2 = 0.5$

(b) $C^2 = 5$

(c) $C^2 = 50$

Fig. 8. The approximation ratio of mean response time under EQUI, EF, PA-FCFS, and IF, all compared with IF, when phases follow a Weibull distribution. In each case, $K = 100$, $\mu_E = 1$, $q = 0.2$, and jobs arrive according to a Poisson process. IF typically still outperforms the competitor policies. When jobs are highly variable ($C^2 = 50$) EQUI outperforms IF due to its insensitivity to job size variance.

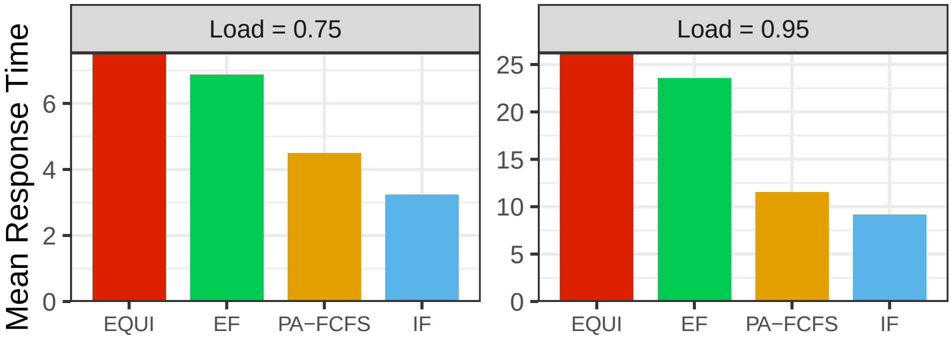


Fig. 9. The mean response time of EQUI, EF, PA-FCFS, and IF processing a workload consisting of a mixture of 5 queries from the Star Schema Benchmark. We assume Poisson arrivals. Although this workload violates our modeling assumptions, IF is still the best policy by a wide margin. IF improves upon the next best policy, the PA-FCFS policy used in the Noisepage database, by up to 30%.

8.1 Scheduling with known sizes

Although the focus of this paper has been the setting where job sizes are unknown to the scheduler, we recognize that schedulers in real-world databases often have knowledge of the size of each query phase and the number of phases comprising each query. Specifically, the query planner in the Noisepage database on which we have based our simulations can provide the scheduler with information about the sequence of phases for each query and an estimate of phase sizes in addition to information about the current phase.

Historically, when job sizes are known, the performance modeling community has advocated for reducing mean response time by trying to complete smaller jobs before larger jobs [32]. This begs the question of whether the phase-aware policies developed in this paper can be improved by adapting them to favor short jobs. Notably, Noisepage and many other databases use a PA-FCFS policy, and do not leverage the available information about query sizes to make better scheduling decisions. Would favoring short queries improve response times in these systems?

Our theorems in Sections 5 and 6 have shown the importance of *deferring parallelizable work* by giving priority to inelastic phases in order to maintain the overall efficiency of the system. In the case where phase sizes are known, it is not immediately clear how to balance the objectives of favoring shorter queries and deferring parallelizable work.

8.2 Size-Aware Scheduling Policies

We now consider two size-aware scheduling policies that favor queries with smaller *remaining total size*, the sum of the remaining sizes of all of a query's remaining phases. As we will see, one of the scheduling policies performs well because it manages to both favor short queries and grant strict priority to inelastic phases. However, the other policy, which favors the shortest queries in the system but does not otherwise defer parallelizable work, does even worse than PA-FCFS.

The first policy we consider is an adaptation of IF to the case where query sizes are known to the scheduler. We call this new policy *Inelastic-First-Shortest-Remaining-Processing-Time* (IF-SRPT) because it combines IF with the ubiquitous SRPT scheduling policy. IF-SRPT gives strict priority to inelastic phases over elastic phases in the same manner as IF. However, among inelastic phases, IF-SRPT gives priority to the phases belonging to the queries with the smallest remaining total sizes. Likewise, when choosing to run an elastic phase, IF-SRPT will choose the elastic phase belonging to the query with the smallest remaining total size.

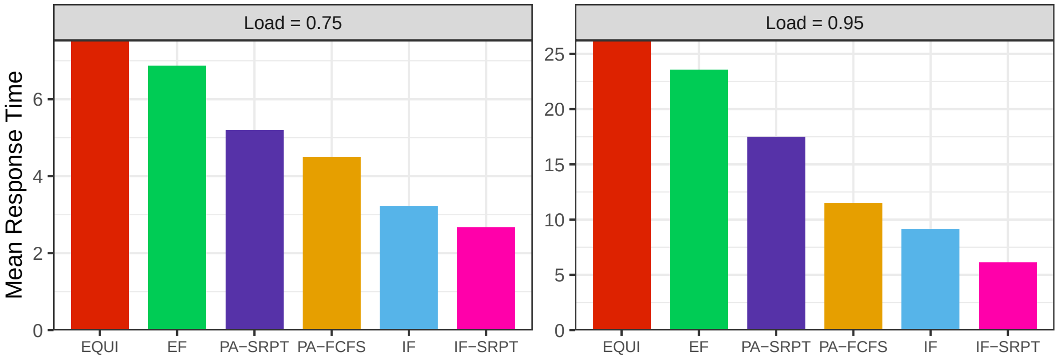


Fig. 10. The mean response time of EQUI, EF, PA-SRPT, PA-FCFS, IF and IF-SRPT processing a workload consisting of a mixture of 5 queries from the Star Schema Benchmark. We assume Poisson arrivals. IF-SRPT can improve upon IF by 33% by leveraging query size information. Notably, PA-SRPT performs worse than PA-FCFS despite attempting to leverage size information.

Our second policy is a *Phase-Aware SRPT* policy, which we refer to as PA-SRPT. PA-SRPT gives strict priority to the phases belonging to the queries with the smallest remaining total sizes, regardless of whether a phase is elastic or inelastic. However, PA-SRPT is phase-aware in that it avoids allocating too many cores to inelastic phases. Hence, if the query with the smallest remaining total size is in an inelastic phase, PA-SRPT will allocate one core to this query. If the next smallest query is in an elastic phase, PA-SRPT will allocate the remaining $K-1$ cores to this second smallest query. Although PA-SRPT does not explicitly defer parallelizable work, it biases more strongly towards the shortest queries in the system than IF-SRPT does.

We again evaluate these policies using a workload based on the Star Schema Benchmark, and the results are shown in Figure 10. Unsurprisingly, IF-SRPT is the best performer. It benefits from biasing its allocations towards shorter queries while still deferring parallelizable work. This leads IF-SRPT to achieve a mean response time which can be 33% lower than that of IF, and 47% lower than that of the PA-FCFS policy used in Noisepage. What is more counter-intuitive is that PA-SRPT performs quite poorly. In fact, PA-SRPT is worse than PA-FCFS in both of the cases shown in Figure 10, and IF-SRPT outperforms PA-SRPT by up to a factor of 3.

8.3 Why PA-SRPT is worse than PA-FCFS

As seen in this paper, deferring parallelizable work is vital to reducing mean response time. PA-SRPT suffers from its failure to defer parallelizable work. It is not immediately clear, however, why PA-SRPT is even worse than PA-FCFS, given that neither policy explicitly defers parallelizable work.

Although neither PA-SRPT nor PA-FCFS explicitly defers parallelizable work, we can see that PA-SRPT suffers because it inadvertently defers far less parallelizable work than PA-FCFS. We define the *percentage of deferred parallelizable work* under a given policy at time t to be the number of cores allocated to inelastic phases divided by the number of cores that IF would allocate to inelastic phases. We can then consider the long-run time-average percentage of deferred parallelizable work under various policies. We normalize this quantity using the allocations under IF because IF allocates as many cores to inelastic phases as possible without being wasteful. As a result, IF defers 100% of parallelizable work by definition. Phase-unaware policies, such as EQUI, can defer more than 100% of parallelizable work by wastefully allocating too many cores to inelastic phases.

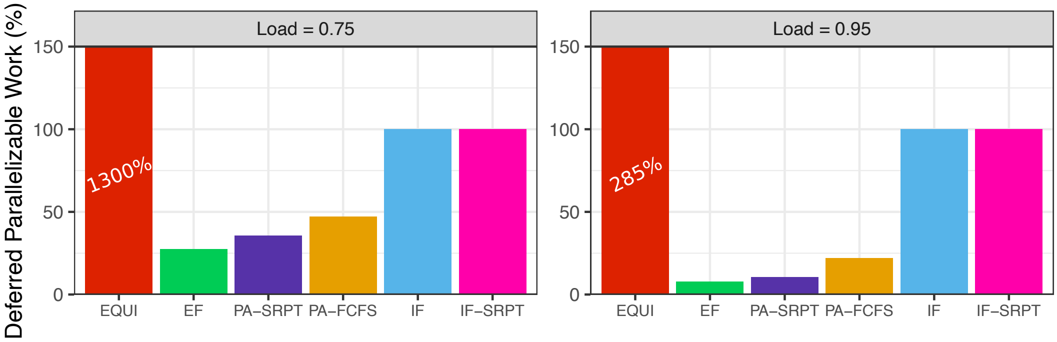


Fig. 11. The percentage of deferred parallelizable work under EQUI, EF, PA-SRPT, PA-FCFS, IF and IF-SRPT given a workload consisting of a mixture of 5 queries from the Star Schema Benchmark. IF-SRPT defers 100% of parallelizable work, but PA-SRPT defers even less parallelizable work than PA-FCFS.

Figure 11 shows that PA-SRPT defers far less parallelizable work than PA-FCFS, leading PA-SRPT to perform poorly. Figure 11 also shows how IF-SRPT avoids this pitfall. IF-SRPT is able to defer 100% of parallelizable work *and* prioritize shorter queries, leading to lower mean response time.

9 CONCLUSION

This paper addresses the optimal scheduling of parallelizable jobs, specifically jobs that consist of different numbers of elastic and inelastic phases. While optimality results in the literature often involve asymptotic approximations such as scaling of system size or heavy traffic assumptions, the results in this paper make no such assumptions. We prove that the IF policy, which defers parallelizable work, is optimal in a strong sense: for any number of servers, K, for any system load, ρ, for any arrival process (including adversarial arrivals), and when jobs can each consist of an arbitrary number of phases. While our proofs do require that the phases have exponentially distributed sizes, experimental evaluation shows that the dominance of IF typically extends to cases when the phase sizes are not exponentially distributed as well. Furthermore, IF does not need knowledge of the job structure (other than knowing the current phase), i.e., IF does not require knowledge of the job parameters, μ_I, μ_E, and q.

We also show that IF performs well in simulation under database workloads. To show how our theoretical results can be further adapted to scheduling in databases, we consider the case where the scheduler not only knows the phase of each job but also has knowledge of the job's size. When job sizes are known, a natural policy is PA-SRPT, which is phase-aware and allocates servers to the jobs with the shortest remaining total sizes. However, we find that PA-SRPT performs poorly because it does not defer parallelizable work. By contrast, IF-SRPT defers parallelizable work *and* favors short jobs, performing even better than IF. This somewhat counter-intuitive result underscores the importance of deferring parallelizable work when scheduling parallelizable jobs composed of phases.

REFERENCES

[1] NoisePage - The Self-Driving Database Management System. https://noise.page.

[2] Samuli Aalto, Urtzi Ayesta, and Rhonda Righter. On the Gittins index in the M/G/1 queue. *Queueing Systems*, 63(1):437–458, 2009.

[3] Kunal Agrawal, Jing Li, Kefu Lu, and Benjamin Moseley. Scheduling parallel DAG jobs online to minimize average flow time. In *SIAM Symposium on Discrete Algorithms*, pages 176–189. SIAM, 2016.

[4] B. Berg, J.P. Dorsman, and M. Harchol-Balter. Towards optimality in parallel scheduling. *ACM POMACS*, 1(2), 2018.

[5] Benjamin Berg, Mor Harchol-Balter, Benjamin Moseley, Weina Wang, and Justin Whitehouse. Optimal resource allocation for elastic and inelastic jobs. In *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures*, pages 75–87, 2020.

[6] Guy E Blelloch, Phillip B Gibbons, and Yossi Matias. Provably efficient scheduling for languages with fine-grained parallelism. *Journal of the ACM (JACM)*, 46(2):281–321, 1999.

[7] Robert D Blumofe and Charles E Leiserson. Space-efficient scheduling of multithreaded computations. *SIAM Journal on Computing*, 27(1):202–229, 1998.

[8] Robert D Blumofe and Charles E Leiserson. Scheduling multithreaded computations by work stealing. *Journal of the ACM (JACM)*, 46(5):720–748, 1999.

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

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

[11] J. Edmonds. Scheduling in the dark. *Theoretical Computer Science*, 1999.

[12] J. Edmonds and K. Pruhs. Scalably scheduling processes with arbitrary speedup curves. SODA '09, pages 685–692. ACM, 2009.

[13] Jeff Edmonds, Donald D Chinn, Tim Brecht, and Xiaotie Deng. Non-clairvoyant multiprocessor scheduling of jobs with changing execution characteristics. *Journal of Scheduling*, 6(3):231–250, 2003.

[14] Jeff Edmonds, Sungjin Im, and Benjamin Moseley. Online scalable scheduling for the l_k-norms of flow time without conservation of work. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 109–119. SIAM, 2011.

[15] Nikos Hardavellas, Ippokratis Pandis, Ryan Johnson, Naju Mancheril, Anastassia Ailamaki, and Babak Falsafi. Database servers on chip multiprocessors: Limitations and opportunities. In *Proceedings of the Biennial Conference on Innovative Data Systems Research*, 2007.

[16] Stavros Harizopoulos and Anastassia Ailamaki. A case for staged database systems. In *CIDR*, 2003.

[17] Stavros Harizopoulos, Vladislav Shkapenyuk, and Anastassia Ailamaki. Qpipe: A simultaneously pipelined relational query engine. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 383–394, 2005.

[18] Chen He, Ying Lu, and David Swanson. Matchmaking: A new mapreduce scheduling technique. In *2011 IEEE Third International Conference on Cloud Computing Technology and Science*, pages 40–47. IEEE, 2011.

[19] John L Hennessy and David A Patterson. *Computer architecture: a quantitative approach.* Elsevier, 2011.

[20] 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.

[21] Sungjin Im, Benjamin Moseley, Kirk Pruhs, and Eric Torng. Competitively scheduling tasks with intermediate parallelizability. *TOPC*, 3(1):4, 2016.

[22] Viktor Leis, Peter Boncz, Alfons Kemper, and Thomas Neumann. Morsel-driven parallelism: A numa-aware query evaluation framework for the many-core age. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, pages 743—-754, 2014.

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

[24] S. Lin, M. Paolieri, C. Chou, and L. Golubchik. A model-based approach to streamlining distributed training for asynchronous SGD. In *MASCOTS*. IEEE, 2018.

[25] 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.

[26] Thu D Nguyen, Raj Vaswani, and John Zahorjan. Using runtime measured workload characteristics in parallel processor scheduling. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 155–174. Springer, 1996.

[27] Patrick O'Neil, Elizabeth O'Neil, Xuedong Chen, and Stephen Revilak. *The Star Schema Benchmark and Augmented Fact Table Indexing*, pages 237—-252. 2009.

[28] Gerald Sabin, Matthew Lang, and P Sadayappan. Moldable parallel job scheduling using job efficiency: An iterative approach. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 94–114. Springer, 2006.

[29] SchedMD. SLURM workload manager. 2021. https://slurm.schedmd.com/heterogeneous_jobs.html.

[30] Malte Schwarzkopf, Andy Konwinski, Michael Abd-El-Malek, and John Wilkes. Omega: flexible, scalable schedulers for large compute clusters. In *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 351–364, 2013.

[31] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)*, pages 1–10, 2010.

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

[33] S. Srinivasan, S. Krishnamoorthy, and P. Sadayappan. A robust scheduling strategy for moldable scheduling of parallel jobs. In *Proceedings of the IEEE International Conference on Cluster Computing*, CLUSTER '03, pages 92–99, 2003.

[34] Nathan R Tallent and John M Mellor-Crummey. Effective performance measurement and analysis of multithreaded applications. In *Proceedings of the 14th ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 229–240, 2009.

[35] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes. Large-scale cluster management at Google with Borg. In *EUROSYS*. ACM, 2015.

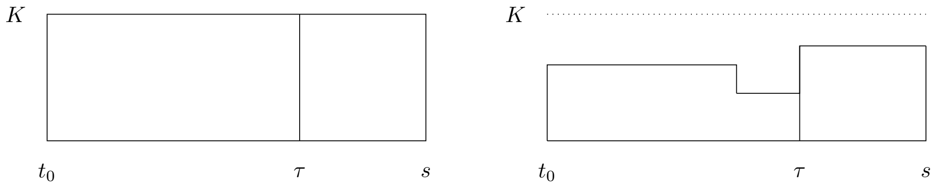
[36] Rares Vernica, Michael J Carey, and Chen Li. Efficient parallel set-similarity joins using mapreduce. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 495–506, 2010.

[37] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauly, Michael J Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 15–28, 2012.

[38] Jingren Zhou, John Cieslewicz, Kenneth A Ross, and Mihir Shah. Improving database performance on simultaneous multithreading processors. In *Proceedings of the 31st Very Large Data Bases Conference (VLDB)*, 2005.

Appendices

A PROOF OF LEMMA 4



(a) Busy period

(b) Idle period

Fig. 12. Examples of busy and idle periods used in the proof of Lemma 4. The figures both show the relative positions of the times t_0, τ, and s. The value K refers to the number of servers, and the heights of the boxes indicate how many servers $S_{\texttt{IF}}$ allocates to jobs.

Lemma 4. *Consider a K server system serving two-phase jobs with equal rates. Consider any policy A and let the policies* IF *and A start from the same initial conditions and have the same arrival time process. Then there exists a coupling between* IF *and A such that $I_{\texttt{IF}}(t) \geq I_A(t)$ and $C_{\texttt{IF}}(t) \geq C_A(t)$ for all $t \geq 0$, where $I_{\texttt{IF}}(t)$ (resp. $I_A(t)$) is the number of inelastic transitions by time t under* IF *(resp. A), and $C_{\texttt{IF}}(t)$ (resp. $C_A(t)$) is the number of jobs completed by time t under* IF *(resp. A).*

Proof. We proceed by induction on event times, as defined in Section 5.1.1. We parse time into two types of periods: *busy periods* where $S_{\texttt{IF}}$ utilizes all K of its servers, and *idle periods* where $S_{\texttt{IF}}$ idles at least one of its servers at any point in time. Let t_0 be the start of a period (either busy or idle). We show below that, if $I_{\texttt{IF}}(t_0) \geq I_A(t_0)$, then, at any point of time t during the current period, $I_{\texttt{IF}}(t) \geq I_A(t)$. This claim is sufficient for proving Lemma 4 since time can be partitioned into disjoint alternating busy and idle periods. To get a sense of how time is partitioned, refer to Figure 12.

As a base case for our induction, observe that $I_{\mathtt{IF}}(0) = I_A(0)$. Depending on the initial conditions, time $t = 0$ will serve as either the start of the first busy period or the first idle period.

Busy Periods: We first consider the case where time t_0 marks the start of a busy period, and assume inductively that $I_{\mathtt{IF}}(t_0) \geq I_A(t_0)$. We show that $I_{\mathtt{IF}}(t) \geq I_A(t)$ for all times t in the busy period by contradiction. Assume for contradiction that there is some earliest time s in the busy period such that $I_{\mathtt{IF}}(s) < I_A(s)$.

First, we argue that

$$N^E_{\mathtt{IF}}(t_0) \leq N^E_A(t_0). \tag{1}$$

If $t_0 = 0$, this follows directly from the shared initial conditions of $S_{\mathtt{IF}}$ and S_A. Now suppose $t_0 > 0$. Observe that, since t_0 marks the beginning of a busy period, immediately before time t_0, all of the jobs in $S_{\mathtt{IF}}$ must be in the inelastic phase. That is, $N^E_{\mathtt{IF}}(t_0-) = 0$, and thus $N^E_{\mathtt{IF}}(t_0-) \leq N^E_A(t_0-)$. Lastly, the event that happens at t_0 can only be job arriving since t_0 is the start of a busy period. Since the arrival occurs simultaneously in both systems, it follows that $N^E_{\mathtt{IF}}(t_0) \leq N^E_A(t_0)$, as desired.

Next, let τ be the time for the event preceding the event at s. We claim that

$$I_{\mathtt{IF}}(\tau) = I_A(\tau), \text{ and } N_{\mathtt{IF}}(\tau) = N_A(\tau), \tag{2}$$

where $N_{\mathtt{IF}}(\tau) = N_A(\tau)$ follows from $I_{\mathtt{IF}}(\tau) = I_A(\tau)$. This is because the number of inelastic transitions determines the number of job completions and both systems experience the same arrival sequence. The claim $I_{\mathtt{IF}}(\tau) = I_A(\tau)$ is true since $I_{\mathtt{IF}}(t)$ is non-decreasing, $I_A(\tau)$ can increase by at most 1 at time s, and s is the earliest time during the busy period for which $I_{\mathtt{IF}}(s) < I_A(s)$. More specifically, we can conclude that at time s, $S_{\mathtt{IF}}$ experiences an elastic transition, whereas S_A experiences an inelastic transition. This holds because $I_{\mathtt{IF}}(s) < I_A(s)$ and $I_{\mathtt{IF}}(\tau) = I_A(\tau)$ if and only if $I_{\mathtt{IF}}(s) = I_{\mathtt{IF}}(\tau)$ and $I_A(s) = I_A(\tau) + 1$. This implies that S_A experiences an inelastic transition at time s. Furthermore, $S_{\mathtt{IF}}$ experiences an elastic transition at time s since `IF` does not idle servers during a busy period.

Now, we can claim that

$$E_{\mathtt{IF}}(t_0, s) \leq E_A(t_0, s). \tag{3}$$

Since we have shown that $N^E_{\mathtt{IF}}(t_0) \leq N^E_A(t_0)$ in (1), it suffices to show that $N^E_{\mathtt{IF}}(s) \geq N^E_A(s)$. Per our coupling, the previous paragraph implies that $S_{\mathtt{IF}}$ is running fewer inelastic jobs on the interval $[\tau, s)$ than S_A. Since $S_{\mathtt{IF}}$ always runs the maximal number of inelastic jobs, we have that $N^I_{\mathtt{IF}}(\tau) < N^I_A(\tau)$. Moreover, since $N_{\mathtt{IF}}(\tau) = N_A(\tau)$ by (2), we know that $N^E_{\mathtt{IF}}(\tau) > N^E_A(\tau)$, and thus $N^E_{\mathtt{IF}}(s) \geq N^E_A(s)$.

Finally, let M denote the number of potential transitions during $(t_0, s]$. Since $S_{\mathtt{IF}}$ is never idling servers between times t_0 and s, we have the identities:

$$M = E_{\mathtt{IF}}(t_0, s) + I_{\mathtt{IF}}(t_0, s), \text{ and } M \geq E_A(t_0, s) + I_A(t_0, s).$$

Consequently, utilizing (3) and rearranging, we have that:

$$I_{\mathtt{IF}}(t_0, s) \geq I_A(t_0, s). \tag{4}$$

Moreover, recall that by definition, $I_{\mathtt{IF}}(s) = I_{\mathtt{IF}}(t_0) + I_{\mathtt{IF}}(t_0, s)$ and $I_A(s) = I_A(t_0) + I_A(t_0, s)$. Since we assumed $I_{\mathtt{IF}}(t_0) \geq I_{\mathtt{IF}}(t_0)$, and we know that $I_{\mathtt{IF}}(t_0, s) \geq I_A(t_0, s)$ by (4), we have $I_{\mathtt{IF}}(s) \geq I_A(s)$, a contradiction. This completes the induction step for busy periods.

Idle periods: Next, we consider the case that time t_0 marks the beginning of an idle period, and again inductively assume that $I_{\mathtt{IF}}(t_0) \geq I_A(t_0)$. To show $I_{\mathtt{IF}}(t) \geq I_A(t)$ for all times t in the idle period, we once again proceed by contradiction. That is, suppose there is some earliest time s in the period such that $I_{\mathtt{IF}}(s) < I_A(s)$.

First, observe that, since $S_{\mathtt{IF}}$ always chooses to idle at least one server during the idle period, there cannot be any elastic phase jobs in the system. That is, $N^E_{\mathtt{IF}}(t) = 0$ for all times t in the idle period.

Letting τ be defined again as the time for the event preceding the event at s, by a similar reasoning to before, we must have that

$$I_{\mathtt{IF}}(\tau) = I_A(\tau), \tag{5}$$

and that, at time s, $S_{\mathtt{IF}}$ does not have a transition, whereas S_A experiences an inelastic transition.

Now we show that we have a contradiction. First note that the equality $I_{\mathtt{IF}}(\tau) = I_A(\tau)$ in (5) implies that $N_{\mathtt{IF}}(\tau) = N_A(\tau)$. Next, since at time s, $S_{\mathtt{IF}}$ does not have a transition but S_A experiences an inelastic transition, per our coupling, $S_{\mathtt{IF}}$ is running strictly fewer jobs in the inelastic phase than S_A. Since $S_{\mathtt{IF}}$ has no elastic jobs, this implies that $N_{\mathtt{IF}}(\tau) < N_A(\tau)$, leading to a contradiction. This completes the induction step for idle periods.

■