Contents lists available at ScienceDirect

Performance Evaluation

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

Scalable load balancing in the presence of heterogeneous servers

Kristen Gardner[a,*], Jazeem Abdul Jaleel[b], Alexander Wickeham[b], Sherwin Doroudi[b]

[a] Department of Computer Science, Amherst College, Amherst, MA, USA
[b] Department of Industrial and Systems Engineering, University of Minnesota, Minneapolis, MN, USA

ARTICLE INFO

Article history:
Available online xxxx

Keywords:
Load balancing
Dispatching
Heterogeneity
Join-the-Shortest-Queue
Join-Idle-Queue
Power of *d*

ABSTRACT

Heterogeneity is becoming increasingly ubiquitous in modern large-scale computer systems. Developing good load balancing policies for systems whose resources have varying speeds is crucial in achieving low response times. Indeed, how best to dispatch jobs to servers is a classical and well-studied problem in the queueing literature. Yet the bulk of existing work on large-scale systems assumes homogeneous servers; unfortunately, policies that perform well in the homogeneous setting can cause unacceptably poor performance in heterogeneous systems.

We adapt the "power-of-*d*" versions of both the Join-the-Idle-Queue and Join-the-Shortest-Queue policies to design two corresponding families of heterogeneity-aware dispatching policies, each of which is parameterized by a pair of routing probabilities. Unlike their heterogeneity-unaware counterparts, our policies use server speed information both when choosing which servers to query and when probabilistically deciding where (among the queried servers) to dispatch jobs. Both of our policy families are analytically tractable: our mean response time and queue length distribution analyses are exact as the number of servers approaches infinity, under standard assumptions. Furthermore, our policy families achieve maximal stability and outperform well-known dispatching rules – including heterogeneity-aware policies such as Shortest-Expected-Delay – with respect to mean response time.

1. Introduction

In large-scale computer systems, deciding how to dispatch arriving jobs to servers is a primary factor affecting system performance. Consequently, there is a wealth of literature on designing, analyzing, and evaluating the performance of load balancing policies. For analytical tractability, most existing work on dispatching in large-scale systems makes a key assumption: that the servers are homogeneous, meaning that they all have the same speeds, capabilities, and available resources. But this assumption is not accurate in practice. Modern computer systems are instead heterogeneous: server farms may consist of multiple generations of hardware, servers with varied resources, or even virtual machines running in a cloud environment. Given the ubiquity of heterogeneity in today's systems, it is critically important to develop load balancing policies that perform well in heterogeneous environments. In this paper, we focus on systems in which *server speeds* are heterogeneous.

[*] Corresponding author.
E-mail address: kgardner@amherst.edu (K. Gardner).

https://doi.org/10.1016/j.peva.2020.102151
0166-5316/© 2020 Elsevier B.V. All rights reserved.

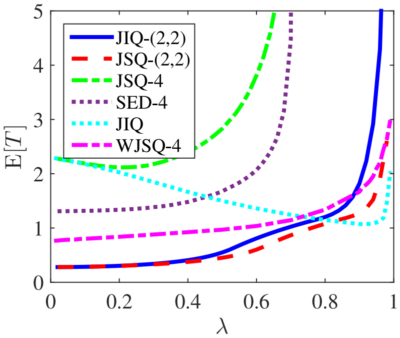


Fig. 1. Mean response time as a function of average arrival rate per server under the dispatching policies we propose as well as four standard policies. Here 20% of the servers are fast, and the fast servers have average speed 10 times that of the slow servers. The average service rate is 1, meaning that $\lambda < 1$ is the maximal stability region.

The dominant dispatching paradigm in the contemporary literature on large scale systems is the "power of d choices", wherein the dispatcher cannot use global information to make dispatching decisions, as that would require prohibitively expensive communication upon each job's arrival. Rather, a fixed number (d) of servers are queried at random, and a dispatching decision is made among these servers. Unfortunately, the "power of d" policies that have been designed to perform well in homogeneous systems can lead to unacceptably poor performance in the presence of heterogeneity. Fig. 1 illustrates this poor performance. For example, the classical Join-the-Shortest-Queue-d (JSQ-d) policy, under which, upon a job's arrival, the dispatcher queries d servers uniformly at random and sends the job to the queried server with the fewest jobs in its queue, can cause the system to become unstable if the system's capacity is concentrated among a relatively small number of fast servers. In Fig. 1, we see that JSQ-4 appears to approach instability around $\lambda = 0.7$, rather than achieving the maximal stability region of $\lambda < 1$. JSQ-d is just one example of a *heterogeneity-unaware* policy, but recent work has shown that other heterogeneity-unaware policies, including Join Idle Queue (JIQ), also can lead to high response times in heterogeneous systems. For example, in Fig. 1, we see that at low load JIQ achieves a much higher response time than many of the other policies that we consider. This is surprising in light of the fact that JIQ is known to be delay optimal, even in heterogeneous systems [1]. The seemingly contradictory observation that JIQ can lead to high response times comes from the fact that minimizing *time in queue*, i.e., delay, does not always equate to minimizing *time in system*, i.e., response time. In particular, response time might be lower when waiting in the queue at a fast server than when entering service immediately at a slow server. Clearly, when response time is the metric of interest, it is necessary to use server speed information when making dispatching decisions in heterogeneous systems.

Yet simply using heterogeneity information is not enough: it matters exactly when and how the dispatcher uses this information. Consider the Shortest-Expected-Delay-d (SED-d) policy, a natural heterogeneity-aware generalization of JSQ-d. Under SED-d, upon a job's arrival the dispatcher queries d servers uniformly at random and sends the job to the queried server at which the job's expected delay – the number of jobs in the queue scaled by the server's speed – is smallest. By allowing the dispatcher to select a fast server with a longer queue over a slow server with a shorter queue, SED-d overcomes one of the weaknesses of JSQ-d in the presence of heterogeneity. Unfortunately, this is insufficient to solve the fundamental problem faced by JSQ-d. Again, we see in Fig. 1 that SED-d, too, can cause poor performance (and apparent instability) if fast servers are queried infrequently.

While server heterogeneity poses a problem for many existing dispatching policies, it also presents an opportunity to design new policies that leverage heterogeneity to achieve good performance and maintain stability, rather than suffering in the presence of heterogeneity. Our key insight is that there are two decision points at which "power of d" policies can use server speed information. First, the dispatcher can make heterogeneity-aware decisions about which d servers to query. Second, the dispatcher can make heterogeneity-aware decisions about where among the queried servers to send an arriving job. Alone, neither decision point appears to be enough to both ensure stability and achieve good performance. In combination, they allow for the design of a new class of powerful policies that benefit from server speed heterogeneity.

We propose two new families of policies, called JIQ-(d_F, d_S) and JSQ-(d_F, d_S), that are inspired by classical "power of d" policies but use server speed information at both decision points. This enables them to outperform JSQ-d, SED-d, and other heterogeneity-aware policies in certain settings, as well as to maintain the full stability region. At the first decision point, instead of querying d servers uniformly at random from among all servers, our policies query d_F fast servers and d_S slow servers. Unlike under JSQ-d and SED-d, this guarantees that each job has the option to run on a fast server. After querying $d_F + d_S$ servers, our policies decide probabilistically based on the servers' states (idle or busy) whether to dispatch the job to a fast server or a slow server. Our policy families are analytically tractable: given the probabilistic parameter settings, we derive the mean response time and queue length distribution under each, in an asymptotic regime where the number of servers approaches infinity. While the two families are functionally similar, they require different analytical approaches. We analyze JIQ-(d_F, d_s) using a mean field approach, and JSQ-(d_F, d_S) using a system of differential equations capturing the system evolution. Our analyses of both policies are exact in the limiting regime where the number of servers approaches infinity, under standard asymptotic independence assumptions.

The remainder of this paper is organized as follows. In Section 2 we survey related work on dispatching in heterogeneous systems. Section 3 describes the system model and defines the JIQ-(d_F, d_S) and JSQ-(d_F, d_S) policy families. In Section 4 we present our analyses of both policies. We give a numerical evaluation in Section 5 and propose a heuristic for selecting policy parameters in Section 6. Finally, in Section 7, we conclude.

2. Related work

In large-scale homogeneous systems, Join-the-Shortest-Queue (JSQ) is known to minimize mean response time under first-come-first-served (FCFS) scheduling when service times are independent and identically distributed and have non-decreasing hazard rate [2,3]. While analyzing response time is challenging due to the dependencies among queue lengths, approximations exist in both the FCFS setting with exponential service times [4] and the Processor Sharing (PS) setting with general service times [5]. Because of the high communication cost required to query all servers for their queue lengths, the JSQ-d (also called SQ(d) or Power-of-d) policy was proposed and analyzed, assuming homogeneous servers and exponential service times [6,7]. This policy has been extended to variants such as JTQ(d, T), under which a job is dispatched to a server with workload less than a threshold T; when $T = 0$ this policy coincides with Join-Idle-Queue-d (JIQ-d) [8]. Other policies, such as Join-Idle-Queue (JIQ), have also been proposed as low-communication alternatives to JSQ that are able to avoid querying servers for their queue lengths [9,10].

Once the server homogeneity assumption is relaxed, the optimality and analytical tractability of state-aware dispatching policies suffers. The JSQ-2 policy has been studied in heterogeneous FCFS systems with general service times, under both light traffic [11] and heavy traffic [12] assumptions. Performance analysis also exists for JSQ-2 in heterogeneous PS systems [13]. The Shortest Expected Delay (SED) policy is a natural alternative to JSQ when server speeds are known; SED has been shown empirically to perform favorably to several other heterogeneity-aware policies [14]. However, SED is known to be suboptimal in general [15]. When service times are generally distributed, and estimating the expected delay uses information about the age of the job currently in service (as in [15]), SED requires knowledge of the full job size distribution in order to estimate the remaining service time of the job currently in service. The Generalized JSQ (GJSQ) policy has been proposed as an alternative when only the mean job size at each server, not the full job size distribution, is known [16] (note that when service times are exponentially distributed, SED and GJSQ are equivalent). The equilibrium distribution of the number of jobs in the system has been analyzed under both SED and GJSQ in a heterogeneous two-server system [16,17]. The Balanced Routing policy (which we call Weighted JSQ in Section 5) uses server speed information by querying servers probabilistically in proportion to their speeds but ignores heterogeneity information when choosing among the queried servers; this policy minimizes the system workload in heavy traffic [18], but can be suboptimal at lower load.

A common theme in much of the recent work on dispatching in heterogeneous systems is the observation that policies like JSQ-d and JIQ, which were designed for homogeneous systems, can perform poorly in heterogeneous systems. Recently several families of policies have been proposed that are throughput optimal in heterogeneous settings, including PULL [1] and Π [12]. PULL, which differs from JIQ only in some implementation details, is shown to be optimal in the sense that it stochastically minimizes the queue length distribution [1]; as we will see in Section 5, this does not mean that it is optimal with respect to other system metrics such as response time.

Another related stream of work focuses on the so-called "slow server problem", wherein the system designer must choose when to use a slow server if at all. Typically, models consist of two servers of different speeds with all jobs arriving to a single queue [19–23], with more recent work examining similar problems in settings with more than two servers [24,25]. As they examine a central queue setting rather than an immediate dispatching setting, the policies and analysis proposed in these papers are inapplicable to our setting. Closer to our setting but still within the literature on central queues is [26], which considers dispatching to one of two subsystems: a central queue for a limited number of fast servers, and a subsystem with an infinite number of slow servers.

More closely related to our work is a literature stream on dispatching in small-scale heterogeneous systems [27–31]. Such work explores policies that use information about all servers' queue lengths (or sometimes more detailed information, as in [32]) when making dispatching decisions. These are not "power of d" policies and would not typically be considered scalable; hence, our policies of interest, analytical approaches, and qualitative findings differ significantly from those in the papers above.

3. Model

Our system consists of k heterogeneous servers (see Fig. 2). There are two classes of servers: k_F of the servers are "fast" servers and $k_S = k - k_F$ of the servers are "slow" servers. We let $q_F = \frac{k_F}{k}$ and $q_S = \frac{k_S}{k} = 1 - q_F$ denote the fraction of servers that are fast and slow respectively. Service times are independent with rate μ_F on fast servers and rate μ_S on slow servers, where the speed ratio $r \equiv \mu_F/\mu_S > 1$. For most of the paper, unless otherwise specified, we assume that service times are exponentially distributed. For simplicity, we assume that $\mu_F q_F + \mu_S q_S = 1$, so that the system has total capacity k.

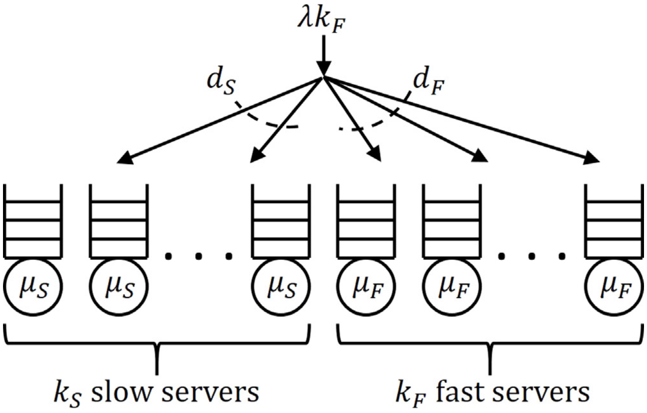


Fig. 2. The system consists of k_F fast servers, each with service rate μ_F, and k_S slow servers, each with service rate μ_S. Job arrive to the system as a Poisson process with rate λk and are dispatched immediately.

Jobs arrive to the system as a Poisson process with rate λk. Upon arrival to the system, a job is dispatched immediately to a single server according to some policy. Each server works on the jobs in its queue in first-come first-served (FCFS) order.

We consider two families of dispatching policies: JIQ-(d_F, d_S) and JSQ-(d_F, d_S). The common framework shared by both families favors idle fast servers whenever possible, and leverages the idea that slow servers are still occasionally worth utilizing (motivating probabilistic decision-making), and it is better to utilize them when idle rather than busy (motivating the use of two – rather than just one – probabilistic parameters). Both policy families are parameterized by d_F and d_S, as well as by probabilities p_F and p_S; each setting for these four parameters defines a different specific policy within the family. In the sections that follow, we will discuss how one can determine how to set these parameters.

Definition 1. Under both **JIQ-(**d_F, d_S**)** and **JSQ-(**d_F, d_S**)**, when a job arrives the dispatcher queries d_F fast servers and d_S slow servers, chosen uniformly at random without replacement. The job is then dispatched to one of the queried servers as follows:

- If any of the d_F fast servers are idle, the job begins service on one of them.
- If all d_F fast servers are busy and any of the d_S slow servers are idle:
  - With probability p_S the job begins service on an idle slow server.
  - With probability $1 - p_S$ the job is dispatched to a **chosen** fast server among the d_F queried.
- If all $d_F + d_S$ queried servers are busy:
  - With probability p_F the job is dispatched to a **chosen** fast server among the d_F queried.
  - With probability $1 - p_F$ the job is dispatched to a **chosen** slow server among the d_S queried.

The difference between the two policies lies in how a busy server (among those under consideration) is **chosen**. Under JIQ-(d_F, d_S) the server is chosen uniformly at random. Under JSQ-(d_F, d_S) the server with the shortest queue is chosen. Under both policies all ties are broken uniformly at random.

4. Analysis

In this section we analyze performance under both JIQ-(d_F, d_S) and JSQ-(d_F, d_S). One of the significant downsides to heterogeneity-unaware dispatching policies such as JSQ-d and SED-d is that they appear to become unstable under certain system parameters, including, for example, when q_F is low and the fast servers are significantly faster than the slow servers. We begin by showing that JIQ-(d_F, d_S) and JSQ-(d_F, d_S) achieve the maximal stability region (Section 4.1). We then turn to analyzing the queue length distributions and mean response times under our policies (Section 4.2).

4.1. Stability

Both JIQ-(d_F, d_S) and JSQ-(d_F, d_S) are parameterized by two probabilistic parameters, p_F and p_S, that determine whether an arriving job is routed to one of the queried fast servers or to one of the queried slow servers. In Section 4.2, after analyzing the queue length distribution and mean response time for fixed choices of p_F and p_S, we will present an optimization problem for determining the values of p_F and p_S that minimize mean response time. In Theorem 1, we assume that p_F and p_S are set optimally, and show that both policies achieve the maximum possible stability region. *Stability* refers to the property that each server experiences an average arrival rate less than its service rate; because all servers are work conserving in our model, this notion of stability is equivalent to the property that each server is idle a nonzero fraction of the time. This property is a necessary condition for achieving finite mean response time. Note that our results in this section allow for generally distributed service times.

Theorem 1. *Under both JIQ-(d_F, d_S) and JSQ-(d_F, d_S), for any values of $d_F, d_S \geq 1$, there exist choices of p_F and p_S such that the system is stable for $\lambda < \mu_F q_F + \mu_S q_S = 1$.*

Proof. We will begin by showing that the system is stable under JIQ-(d_F, d_S) when $p_S = 1$ and $p_F = \mu_F q_F$, for all $d_F, d_S \geq 1$. The system's stability cannot be affected by the arrival rates to idle servers. Hence, we consider only the arrival rates to an arbitrary tagged busy fast server and to an arbitrary tagged busy slow server, which we denote by λ_{BF} and λ_{BS} respectively. Let λ_{QF} denote the arrival rate of jobs that query a tagged fast server. We have

$$\lambda_{QF} = \lambda k \frac{\binom{k_F-1}{d_F-1}\binom{k_S}{d_S}}{\binom{k_F}{d_F}\binom{k_S}{d_S}} = \frac{\lambda d_F}{q_F}. \tag{1}$$

λ_{QS} is defined similarly.

The arrival rate λ_{BF} depends not only on the state of the tagged fast server, but also on whether the *other* servers queried by an arriving job are busy or idle. When the tagged fast server is busy, an arriving job that queries the tagged server will be dispatched to it if none of the other queried fast servers are idle, and if either (1) all queried slow servers are busy, the dispatcher chooses to send the job to a fast server (probability p_F), and the tagged fast server is chosen uniformly at random among all queried fast servers (probability $1/d_F$), or (2) the arrival queries an idle slow server, the dispatcher chooses to send the job to a fast server (probability $1 - p_S$), and the tagged fast server is chosen uniformly at random among all queried fast servers (probability $1/d_F$). We thus have:

$$\lambda_{BF} = \frac{\lambda_{QF}}{d_F}\mathbf{P}\left\{\begin{matrix}\text{all other queried}\\ \text{fast servers busy}\end{matrix}\right\} \cdot \left(\mathbf{P}\left\{\begin{matrix}\text{all queried}\\ \text{slow servers busy}\end{matrix}\right\} p_F + \mathbf{P}\left\{\begin{matrix}\text{not all queried}\\ \text{slow servers busy}\end{matrix}\right\}(1 - p_S)\right), \tag{2}$$

which is at most $\lambda p_F/q_F$ because $p_S = 1$, $\mathbf{P}\{\text{all other fast servers busy}\} \leq 1$, and $\mathbf{P}\{\text{all queried slow servers busy}\} \leq 1$. Let $p_F = \mu_F q_F$. Then we have $\lambda_{BF} \leq \frac{\lambda}{q_F} p_F = \lambda\mu_F < \mu_F$, ensuring the stability of the fast servers, if $\lambda < 1$.

We also must consider the arrival rate to a busy slow server, denoted λ_{BS}. Our approach is similar, yielding:

$$\lambda_{BS} = \frac{\lambda_{QS}}{d_S}\mathbf{P}\left\{\begin{matrix}\text{all other queried}\\ \text{slow servers busy}\end{matrix}\right\}\mathbf{P}\left\{\begin{matrix}\text{all queried}\\ \text{fast servers busy}\end{matrix}\right\}(1 - p_F), \tag{3}$$

which is at most $\lambda(1-p_F)/q_S$ because $\mathbf{P}\{\text{all other slow servers busy}\} \leq 1$ and $\mathbf{P}\{\text{all queried fast servers busy}\} \leq 1$. Again, let $p_F = \mu_F q_F$. Then we have

$$\lambda_{BS} \leq \frac{\lambda}{q_S}(1 - p_F) = \frac{\lambda}{q_S}\mu_S q_S = \lambda\mu_S,$$

which is less than μ_S, ensuring the stability of the slow servers, if $\lambda < 1$.

At this point we have shown that JIQ-(d_F, d_S) is stable for $p_S = 1$, $p_F = \mu_F q_F$. We obtain the same stability result for JSQ-(d_F, d_S) by observing that joining the shortest queue among d_F fast servers (or among d_S slow servers) instead of routing randomly to one of those d_F fast servers (d_S slow servers) cannot decrease the stability region. □

Theorem 1 tells us that there always exist settings for p_S and p_F for which the system is stable; in Theorem 2 we identify more specific necessary and sufficient conditions for stability as $\lambda \to 1$.

Theorem 2. *As $\lambda \to 1$, the system is unstable if $p_F \neq \mu_F q_F$, and the system is stable if $p_F = \mu_F q_F$ and $p_S \geq \mu_S q_S$.*

Proof. We first show that the system is stable if $p_F = \mu_F q_F$ and $p_S \geq \mu_S q_S$. We begin by considering an arbitrary tagged fast server. Note that the arrival rate to the tagged server when it is idle does not affect the stability region of that server. The arrival rate to a tagged busy fast server is given in (2). We have $\mathbf{P}\{\text{all other queried fast servers busy}\} \leq 1$, $p_F = \mu_F q_F$, and $p_S \geq \mu_S q_S$, so $1 - p_S \leq 1 - \mu_S q_S = \mu_F q_F$. Applying these bounds to (2) we obtain

$$\begin{aligned}\lambda_{BF} &\leq \frac{\lambda}{q_F}\left(\mathbf{P}\left\{\begin{matrix}\text{all queried}\\ \text{slow servers busy}\end{matrix}\right\}\mu_F q_F + \mathbf{P}\left\{\begin{matrix}\text{not all queried}\\ \text{slow servers busy}\end{matrix}\right\}\mu_F q_F\right)\\ &= \lambda\mu_F,\end{aligned}$$

which is less than μ_F, ensuring the stability of the tagged server – and hence, of all fast servers – if $\lambda < 1$.

We now establish the stability of the slow servers. Because the fast servers are stable as $\lambda \to 1$, it must also be the case that $\mathbf{P}\{\text{tagged fast server busy}\} \to 1$. Thus an arriving job is likely to query d_F busy servers: $\mathbf{P}\{\text{all queried fast servers busy}\} \to 1$. Let $\mathbf{P}\{\text{all queried fast servers busy}\} = 1 - \epsilon$ for some small $\epsilon > 0$, where $\epsilon \to 0$ as

$\lambda \to 1$. The total arrival rate of jobs that find all queried fast servers busy – and hence, that query slow servers – is then $\lambda k(1-\epsilon)$. Consider an arbitrary tagged slow server, and note that, as for the fast servers, the arrival rate to a slow server when it is idle does not affect its stability region. For a tagged busy slow server, we have

$$\lambda_{BS} = \frac{\lambda}{q_S}(1-\epsilon)\cdot \mathbf{P}\left\{\begin{array}{c}\text{all other queried}\\ \text{slow servers busy}\end{array}\right\}\cdot(1-p_F).$$

We have $\mathbf{P}\{\text{all other queried slow servers busy}\} \le 1$ and $p_F = \mu_F q_F$, so $1-p_F = 1-\mu_F q_F = \mu_S q_S$, which gives

$$\lambda_{BS} \le \lambda\mu_S(1-\epsilon).$$

This is less than μ_S, ensuring stability of the tagged slow server—and hence, of all slow servers—if $\lambda < 1$.

We now turn to the second part of the result: that the system is unstable when $p_F \ne \mu_F q_F$ (for any choice of p_S). The argument hinges on the observation that the maximum throughput of the system is $k(\mu_F q_F + \mu_S q_S) = k$ (because $\mu_F q_F + \mu_S q_S = 1$). In order for the system to be stable as $\lambda \to 1$ (i.e., the total system arrival rate approaches k), it must therefore be the case that the probability that all servers are busy approaches 1; if some servers were idle with probability $\epsilon > 0$, then the maximum possible system throughput would be less than the arrival rate and the system would be unstable.

With this observation in mind, we first consider the case where $p_F > \mu_F q_F$. Recall λ_{BF} from (2), and assume that $\mathbf{P}\left\{\begin{array}{c}\text{all other queried}\\ \text{fast servers busy}\end{array}\right\} \to 1$ and $\mathbf{P}\left\{\begin{array}{c}\text{all queried}\\ \text{slow servers busy}\end{array}\right\} \to 1$ (if not, the system already is unstable). Then, as $\lambda \to 1$, we have that $\lambda_{BF} \to p_F/q_F$, which is less than μ_F if $p_F < \mu_F q_F$; this contradicts our assumption that $p_F > \mu_F q_F$, hence the system is unstable in this case. The case where $p_F < \mu_F q_F$ is similar. □

It is possible that the system also remains stable for a wider range of values for $p_S > 0$, but identifying the full stability region remains an open problem.

4.2. Queue length and response time

In this section we analyze the queue length distribution and mean response time under both JIQ-(d_F, d_S) and JSQ-(d_F, d_S). We will assume that $k \to \infty$ and that in this limiting regime the queue lengths at each of the servers become independent. This lets us treat a single queue as its own isolated system. This asymptotic independence assumption is common in analyses of large-scale systems and has been proved in a number of related systems [33–37]; one approach used to prove this assumption is the cavity queue method [38–40]. Consistent with the approach used in [8,41], we do not formally prove asymptotic independence; instead, we demonstrate via numerical studies that as k becomes large our approximation is highly accurate.

Let ρ_F and ρ_S denote respectively the fraction of time that a fast server is busy and that a slow server is busy. We begin with the observation that ρ_F and ρ_S are independent of the choice of policy between JIQ-(d_F, d_S) and JSQ-(d_F, d_S) and of the service time distribution. For both policies, and for any service time distribution such that the system is stable (i.e., $\rho_F, \rho_S < 1$), we have

$$\rho_F = \lambda k \mathbf{P}\{\text{job runs on a fast server}\}\cdot\frac{1}{\mu_F k_F} = \frac{\lambda}{\mu_F q_F}\left((1-\rho_F^{d_F}) + \rho_F^{d_F}(1-\rho_S^{d_S})(1-p_S) + \rho_F^{d_F}\rho_S^{d_S}p_F\right) \tag{4}$$

$$\rho_S = \lambda k \mathbf{P}\{\text{job runs on a slow server}\}\cdot\frac{1}{\mu_S k_S} = \frac{\lambda}{\mu_S q_S}\left(\rho_F^{d_F}(1-\rho_S^{d_S})p_S + \rho_F^{d_F}\rho_S^{d_S}(1-p_F)\right). \tag{5}$$

Note that we use our asymptotic independence assumption in these expressions. Solving this system of equations, numerically if an exact analytical solution is not possible, yields ρ_F and ρ_S. We will define $\pi_{0F} = 1-\rho_F$ (respectively, $\pi_{0S} = 1-\rho_S$) to be the probability that a fast (slow) server is idle.

4.2.1. JIQ-(d_F, d_S)

We will derive performance metrics under JIQ-(d_F, d_S) first for exponential service times, then for general service times. For both analyses, we use a mean field approach and study a tagged fast server and a tagged slow server, each in isolation. We will need the arrival rates to fast and slow servers when they are busy and when they are idle; we note that these rates are independent of the service time distribution. Let λ_{BF}, λ_{IF}, λ_{BS}, and λ_{IS} denote respectively the arrival rates to a tagged busy fast, idle fast, busy slow, and idle slow server; while we use the same notation as in Section 4.1, note that we are now assuming that $k \to \infty$ and that the asymptotic independence assumption holds.

Eqs. (2) and (3) give the arrival rates to a tagged busy fast server and a tagged busy slow server, respectively. Under our asymptotic independence assumption, all other fast (respectively, slow) servers have the same stationary distribution, π_{iF}

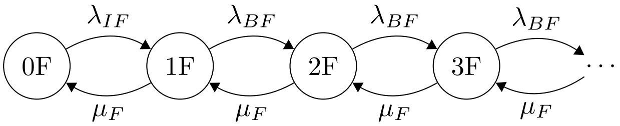


Fig. 3. The Markov chain tracking the number of jobs at a tagged fast server. State *iF* indicates that there are *i* jobs at the fast server (including the job in service, if there is one).

(π_{iS}), as the tagged fast (slow) server, where π_{iF} (π_{iS}) denotes the stationary probability that there are i jobs at a tagged fast (slow) server, $i \in \{0, 1, \ldots\}$. We thus have, for a tagged fast server:

$$\mathbf{P}\left\{\begin{matrix}\text{all other queried}\\ \text{fast servers busy}\end{matrix}\right\} = (1-\pi_{0F})^{d_F-1}$$

$$\mathbf{P}\left\{\begin{matrix}\text{not all queried}\\ \text{slow servers busy}\end{matrix}\right\} = 1-(1-\pi_{0S})^{d_S}$$

$$\mathbf{P}\left\{\begin{matrix}\text{all queried}\\ \text{slow servers busy}\end{matrix}\right\} = (1-\pi_{0S})^{d_S},$$

and, substituting into (2),

$$\lambda_{BF} = \frac{\lambda}{q_F}(1-\pi_{0F})^{d_F-1}\left(\left(1-(1-\pi_{0S})^{d_S}\right)(1-p_S)+(1-\pi_{0S})^{d_S}p_F\right). \tag{6}$$

For a tagged slow server, we have:

$$\mathbf{P}\left\{\begin{matrix}\text{all other queried}\\ \text{slow servers busy}\end{matrix}\right\} = (1-\pi_{0S})^{d_S-1}$$

$$\mathbf{P}\left\{\begin{matrix}\text{all queried}\\ \text{fast servers busy}\end{matrix}\right\} = (1-\pi_{0F})^{d_F},$$

and, substituting into (3),

$$\lambda_{BS} = \frac{\lambda}{q_S}(1-\pi_{0F})^{d_F}(1-\pi_{0S})^{d_S-1}(1-p_F). \tag{7}$$

We next find λ_{IF} and λ_{IS}, the arrival rates to a tagged idle fast server and a tagged idle slow server respectively. When the tagged fast server is idle, an arriving job that queries the tagged server will be dispatched to it if it is chosen (uniformly at random) among all idle fast servers queried by the arrival. We have:

$$\lambda_{IF} = \frac{\lambda d_F}{q_F}\left(\sum_{i=0}^{d_F-1}\binom{d_F-1}{i}\frac{\pi_{0F}^i(1-\pi_{0F})^{d_F-1-i}}{i+1}\right). \tag{8}$$

Similarly, for the tagged idle slow server, we have:

$$\lambda_{IS} = \lambda_{QS}(1-\pi_{0F})^{d_F}\left(\sum_{i=0}^{d_S-1}\binom{d_S-1}{i}\frac{\pi_{0S}^i(1-\pi_{0S})^{d_S-1-i}}{i+1}p_S\right). \tag{9}$$

We are now ready to derive mean response time under both exponential and general service times.

When service times are exponentially distributed, our approach involves setting up and solving a Markov chain for a tagged fast server and for a tagged slow server. We begin with the fast server. Recall that state *iF* denotes that there are *i* jobs at the fast server, including the job in service if there is one, and π_{iF} denotes that state's stationary probability. The number of jobs at the tagged fast server will evolve as a state-dependent birth–death process with arrival rate λ_{IF} when it is idle, arrival rate λ_{BF} when it is busy, and service rate μ_F. Fig. 3 depicts the Markov chain corresponding to this server.

The stationary probabilities for this Markov chain are:

$$\pi_{iF} = \frac{\lambda_{IF}}{\mu_F}\left(\frac{\lambda_{BF}}{\mu_F}\right)^{i-1}\pi_{0F}, \qquad i \geq 1.$$

With the normalization equation, $\sum_{i=0}^{\infty}\pi_{iF} = 1$, this yields:

$$\pi_{0F} = \frac{\mu_F-\lambda_{BF}}{\mu_F-\lambda_{BF}+\lambda_{IF}}. \tag{10}$$

Our approach for the slow server is similar, yielding:

$$\pi_{iS} = \frac{\lambda_{IS}}{\mu_S}\left(\frac{\lambda_{BS}}{\mu_S}\right)^{i-1}\pi_{0S}, \qquad i \geq 1.$$
$$\pi_{0S} = \frac{\mu_S - \lambda_{BS}}{\mu_S - \lambda_{BS} + \lambda_{IS}} \tag{11}$$

We now have six Eqs. (6), (7), (8), (9), (10), (11) to solve for six unknown variables (π_{0F}, λ_{IF}, λ_{BF}, π_{0S}, λ_{IS}, λ_{BS}), after which we will have obtained the full queue length distribution under JIQ-(d_F, d_S).

We are now ready to give an expression for mean response time as a function of the system parameters and the policy parameters p_F and p_S. Let $\mathbf{E}[N_F]$ and $\mathbf{E}[N_S]$ denote respectively the mean number of jobs at a fast server and at a slow server. We have:

$$\mathbf{E}[N_F] = \sum_{i=0}^{\infty} i\pi_{iF} = \pi_{0F}\frac{\lambda_{IF}}{\mu_F}\sum_{i=1}^{\infty} i\left(\frac{\lambda_{BF}}{\mu_F}\right)^{i-1} = \frac{\lambda_{IF}\mu_F}{(\mu_F - \lambda_{BF})(\mu_F - \lambda_{BF} + \lambda_{IF})} \tag{12}$$
$$\mathbf{E}[N_S] = \frac{\lambda_{IS}\mu_S}{(\mu_S - \lambda_{BS})(\mu_S - \lambda_{BS} + \lambda_{IS})}. \tag{13}$$

Putting this together, the mean number of jobs in the system is:

$$\mathbf{E}[N] = k_F\mathbf{E}[N_F] + k_S\mathbf{E}[N_S]. \tag{14}$$

Finally, we apply Little's Law to obtain the mean response time:

$$\mathbf{E}[T] = \frac{k_F\mathbf{E}[N_F] + k_S\mathbf{E}[N_S]}{\lambda k} = \frac{q_F\lambda_{IF}\mu_F}{\lambda(\mu_F - \lambda_{BF})(\mu_F - \lambda_{BF} + \lambda_{IF})} + \frac{q_S\lambda_{IS}\mu_S}{\lambda(\mu_S - \lambda_{BS})(\mu_S - \lambda_{BS} + \lambda_{IS})}. \tag{15}$$

For general service times, our Markov chain approach no longer applies. Now, a job's service time on a fast server (respectively, a slow server) is distributed like Y_F (Y_S). We define r in a manner that is consistent with its definition in the setting with exponential service times, i.e., $r = \mathbb{E}[Y_S]/\mathbb{E}[Y_F]$, and furthermore we assume that Y_S and Y_F are drawn from the same distribution modulo scaling (i.e., Y_S is distributed like rY_F). Note that the servers exhibit heterogeneity in speed, but (as in the case of exponential service times) the coefficient of variation associated with service times, denoted by c_v, is the same across both server speeds.

To analyze this system, we make the observation that the dynamics of a busy fast server are identical to those of an M/G/1 system with arrival rate λ_{BF} and service time distributed like Y_F. The only difference between these two systems is that they have different arrival rates when idle; this does not affect the response time distribution. Hence we can conclude that the response time distribution at a fast server under JIQ-(d_F, d_S) is the same as that of this M/G/1 system. A similar result holds for slow servers. The Pollaczek–Khinchine formula gives us:

$$\mathbf{E}[T_F] = \frac{\lambda_{BF}(1 + c_v^2)\mathbf{E}[Y_F]^2}{2(1 - \lambda_{BF}\mathbf{E}[Y_F])} + \mathbf{E}[Y_F]$$
$$\mathbf{E}[T_S] = \frac{\lambda_{BS}(1 + c_v^2)\mathbf{E}[Y_S]^2}{2(1 - \lambda_{BS}\mathbf{E}[Y_S])} + \mathbf{E}[Y_S].$$

Conditioning on whether an arriving job is dispatched to a fast or a slow server, we then obtain the system mean response time:

$$\mathbf{E}[T] = \frac{q_F\left(\lambda_{IF}\left(1 - \rho_F\right) + \lambda_{BF}\rho_F\right)}{\lambda}\left(\frac{\lambda_{BF}(1 + c_v^2)\mathbf{E}[Y_F]^2}{2(1 - \lambda_{BF}\mathbf{E}[Y_F])} + \mathbf{E}[Y_F]\right)$$
$$+ \frac{q_S\left(\lambda_{IS}\left(1 - \rho_S\right) + \lambda_{BS}\rho_S\right)}{\lambda}\left(\frac{\lambda_{BS}(1 + c_v^2)\mathbf{E}[Y_S]^2}{2(1 - \lambda_{BS}\mathbf{E}[Y_S])} + \mathbf{E}[Y_S]\right) \tag{16}$$

which coincides with (15) when Y_F and Y_S are exponentially distributed. Note the expression $\lambda_{IF}\left(1 - \rho_F\right) + \lambda_{BF}\rho_F$ (respectively, $\lambda_{IS}\left(1 - \rho_S\right) + \lambda_{BS}\rho_S$) in (16) is the average arrival rate experienced by a fast (slow) server.

The observation that a tagged fast server essentially behaves like an M/G/1 also allows us to adapt standard techniques, such as M/G/1 transform analysis, to derive queue length distributions and other system metrics (see Chapter 26 of [42]).

Having determined $\mathbf{E}[T]$ for a fixed p_F and p_S, we can now optimize the JIQ-(d_F, d_S) policy by finding the optimal values for p_F and p_S. We will assume a fixed d_F and d_S, but note that we could also optimize over d_F and d_S; only a small set of values for d_F and d_S are likely to be practical. Eq. (16) tells us that the optimal values of p_F and p_S depend on the mean service times $\mathbf{E}[Y_F] = 1/\mu_F$ and $\mathbf{E}[Y_S] = 1/\mu_S$ and the coefficient of variation c_v.

Our optimization problem for JIQ-(d_F, d_S) for general service time distributions is as follows:

$$\begin{aligned} \underset{p_F, p_S}{\text{minimize}} \quad & \mathbf{E}[T] \\ \text{subject to} \quad & \text{Eqs. (4), (5), (6), (7), (8), (9)} \\ & 0 < \pi_{0F}, \pi_{0S} \leq 1 \\ & 0 \leq p_F, p_S < 1 \end{aligned} \tag{17}$$

where $\mathbf{E}[T]$ is given in (16). We provide an explicit formulation of this problem in Appendix A.

4.3. JSQ-(d_F, d_S)

While the difference between JIQ-(d_F, d_S) and JSQ-(d_F, d_S) may seem like only a minor policy modification, it necessitates a fundamentally different analytical approach. Imagine applying the tagged server approach used to analyze JIQ-(d_F, d_S) to JSQ-(d_F, d_S), and consider a tagged fast server under JSQ-(d_F, d_S). As under JIQ-(d_F, d_S), this server experiences a state-dependent arrival rate. Unlike under JIQ-(d_F, d_S), this arrival rate is different for *every* state, and it depends on the queue lengths of all other polled servers. Hence adopting the Markov chain-based approach we used for JIQ-(d_F, d_S) would require solving a highly complicated infinite system of equations.

Instead, our approach for analyzing JSQ-(d_F, d_S) will involve considering a tagged *arrival* to the system, again assuming that $k \to \infty$ and that in this limiting regime, all servers have independent queue lengths. We return to considering exponential service times in this section.

We condition on whether the tagged arrival runs on a fast or slow server and on whether or not it waits in the queue:

$$\begin{aligned} \mathbf{E}[T] &= \mathbf{E}[T|\text{run on idle fast}] \cdot \mathbf{P}\{\text{run on idle fast}\} + \mathbf{E}[T|\text{run on idle slow}] \cdot \mathbf{P}\{\text{run on idle slow}\} \\ &\quad + \mathbf{E}[T|\text{queue at busy fast}] \cdot \mathbf{P}\{\text{queue at busy fast}\} + \mathbf{E}[T|\text{queue at busy slow}] \cdot \mathbf{P}\{\text{queue at busy slow}\} \\ &= \frac{1}{\mu_F} \cdot (1 - \rho_F^{d_F}) + \frac{1}{\mu_S} \cdot \rho_F^{d_F}(1 - \rho_S^{d_S})p_S + \mathbf{E}[T|\text{queue at busy fast}] \cdot \rho_F^{d_F}(\rho_S^{d_S} p_F + (1 - \rho_S^{d_S})(1 - p_S)) \\ &\quad + \mathbf{E}[T|\text{queue at busy slow}] \cdot \rho_F^{d_F} \rho_S^{d_S}(1 - p_F). \end{aligned} \tag{18}$$

In line (18) we use the asymptotic independence assumption.

We next derive $\mathbf{E}[T|$ queue at busy fast]. Here, the job joins the shortest queue among the d_F polled fast servers, all of which are busy. In order to derive response time, we first need to determine the distribution of the number of jobs in a fast server's queue.

Let $f_i(t)$ be the fraction of fast servers that have at least i jobs at time t. We note that $f_0(t) = 1$ for all t.

As in [6], we consider a limiting system, where $k \to \infty$ and the system exhibits deterministic steady-state behavior where $df_i(t)/dt = 0$ for all $i \geq 0$. This setting lets us describe our system's evolution through a system of differential equations wherein all $f_i(t)$ functions are constant (henceforth we write f_i rather than $f_i(t)$).

We formulate the differential equations by considering the expected change in the fraction of fast servers' queues with at least $i > 1$ jobs over a small interval of time dt. This fraction will increase if an arriving job joins the queue at a fast server with exactly $i - 1$ jobs. The average arrival rate per fast server is λ/q_F (recall that q_F is the fraction of servers that are fast). With probability $f_{i-1}^{d_F} - f_i^{d_F}$ all d_F of the polled fast servers have at least $i - 1$ jobs, but not all d_F have at least i jobs (that is, the shortest queue among the d_F fast servers contains exactly $i - 1$ jobs). The arriving job will join the length-$(i - 1)$ queue if either (1) there is an idle slow server among the d_S polled slow servers (probability $1 - \rho_S^{d_S}$) and the job is assigned to join the queue at a fast server (probability $1 - p_S$), or (2) there are no idle slow servers among the d_S polled slow servers (probability $\rho_S^{d_S}$) and the job is assigned to join the queue at a fast server (probability p_F). The number of queues with at least $i > 1$ jobs will decrease if a job departs from a queue with exactly i jobs. This happens with rate $\mu_F k_F(f_i - f_{i+1})$. Putting this together, we have, for $i > 1$:

$$\frac{df_i}{dt} = \frac{\lambda}{q_F}\left(f_{i-1}^{d_F} - f_i^{d_F}\right)\left((1 - \rho_S^{d_S})(1 - p_S) + \rho_S^{d_S} p_F\right) - \mu_F(f_i - f_{i+1}). \tag{19}$$

The case where $i = 1$ is similar, except here an arriving job that finds a fast server with $i - 1 = 0$ jobs in the queue will simply begin service on that server with probability 1. So for $i = 1$ we have:

$$\frac{df_1}{dt} = \frac{\lambda}{q_F}\left(1 - f_1^{d_F}\right) - \mu_F(f_1 - f_2). \tag{20}$$

This gives us a system of equations for the f_i terms, recalling that $f_0 = 1$. We further note that f_1 is the fraction of fast servers that are busy; using our asymptotic independence assumption, we have $f_1 = \rho_F$. We now set $\frac{df_i}{dt} = 0$ for all i

and solve for the f_i terms as follows. Given the policy parameters p_F and p_S, we can obtain $f_1 = \rho_F$ by solving Eqs. (4) and (5). We then iteratively calculate $f_2, f_3, f_4, \ldots$numerically using Eq. (19). In Appendix B, we prove that this system of differential equations has a unique and stable fixed point.

Once we have the f_i terms, we can find $\mathbf{E}[T|\text{queue at busy fast}]$ by conditioning on the queue length seen by an arriving job:

$$\begin{aligned}\mathbf{E}[T\,|\text{queue at busy fast}] &= \sum_{i=1}^{\infty} \mathbf{P}\{\text{job joins queue with } i \text{ jobs}|\text{ queue at busy fast}\}(i+1)\frac{1}{\mu_F} \\ &= \frac{1}{\mu_F}\sum_{i=1}^{\infty}(i+1)\cdot\frac{f_i^{d_F} - f_{i+1}^{d_F}}{f_1^{d_F}}. \end{aligned} \tag{21}$$

Note that the probability that a job joins a queue with i jobs is *not* the same as the probability that a server has i jobs in its queue. In our numerical computations of $\mathbf{E}[T]$, we truncate the summation at $i = 100$; we numerically validated that this truncation point results in including all terms such that $f_i > 10^{-10}$. While we compute only a finite number of f_i terms, this has no significant effect on the resulting value of $\mathbf{E}[T]$.

Our approach to find $\mathbf{E}[T|\text{queue at busy slow}]$ is similar. Let $s_i(t)$ denote the fraction of slow servers with at least i jobs at time t (we will write s_i when the meaning is clear). We obtain the following system of differential equations for the s_i terms:

$$\frac{ds_i}{dt} = \frac{\lambda}{q_S}\left(s_{i-1}^{d_S} - s_i^{d_S}\right)\rho_F^{d_F}(1-p_F) - \mu_S(s_i - s_{i+1}). \tag{22}$$

$$\frac{ds_1}{dt} = \frac{\lambda}{q_S}\left(1 - s_1^{d_S}\right)\rho_F^{d_F}p_S - \mu_S(s_1 - s_2), \tag{23}$$

where we note that $s_0(t) = 1$ for all t. Again, setting $\frac{ds_i}{dt} = 0$ for all i allows us to solve for a fixed point for the s_i terms using the same approach as for the f_i terms, and this fixed point is unique and stable (see Appendix B).

As with the fast servers, we now find

$$\mathbf{E}[T\,|\text{queue at busy slow}] = \frac{1}{\mu_S}\sum_{i=1}^{\infty}(i+1)\cdot\frac{s_i^{d_S} - s_{i+1}^{d_S}}{s_1^{d_S}}. \tag{24}$$

In our numerical computations, we again truncate the summation at $i = 100$ and numerically validate that this includes all terms with $s_i > 10^{-10}$.

The overall system mean response time results from combining (18), (21), and (24).

As under JIQ-(d_F, d_S), we now find the values of p_F and p_S that minimize mean response time under JSQ-(d_F, d_S) (assuming d_F and d_S are fixed). Our optimization problem is as follows:

$$\begin{aligned} \underset{p_F, p_S}{\text{minimize}} \quad & \mathbf{E}[T] \\ \text{subject to} \quad & \text{Eqs. (4), (5)} \\ & \frac{df_i}{dt} = \frac{ds_i}{dt} = 0 \quad i \geq 1 \\ & f_0 = s_0 = 1 \\ & f_1 = \rho_F \\ & s_1 = \rho_S \\ & 0 \leq \rho_F, \rho_S < 1 \\ & 0 \leq p_F, p_S \leq 1 \end{aligned} \tag{25}$$

where $\mathbf{E}[T]$ is given in (18), (21), (24) and $\frac{df_i}{dt}$, $\frac{ds_i}{dt}$ are given in (19), (20), (22), (23). We provide an explicit formulation of this problem in Appendix A.

5. Numerical results

In this section we present a numerical study to evaluate performance under the JIQ-(d_F, d_S) and JSQ-(d_F, d_S) policy families. For each set of system parameters considered, we report results for the optimal policy within each family, i.e., p_F and p_S are chosen to minimize mean response time, as discussed in Sections 4.2.1 and 4.3. We consider different levels of server heterogeneity by varying two parameters: q_F (the fraction of servers that are fast) and $r \equiv \mu_F/\mu_S$ (the speed ratio). Unless otherwise specified, we set $d_F = d_S = 2$, and service times are exponentially distributed.

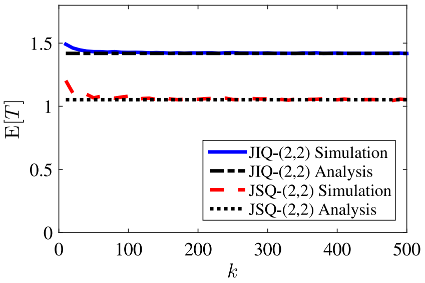


Fig. 4. Analytical and simulated mean response time as a function of k under both JIQ-(d_F, d_S) and JSQ-(d_F, d_S). Here $q_F = 0.5$, $r = 10$, $d_F = d_S = 2$, and p_F and p_S are optimized separately for each policy family.

5.1. Convergence in k

Our analyses for both JIQ-(d_F, d_S) (Section 4.2.1) and JSQ-(d_F, d_S) (Section 4.3) are approximate because they assume that the server states are independent as the number of servers $k \to \infty$. We evaluate the accuracy of our approximations by comparing our analytical results to simulation (see Fig. 4). As k increases our analytical results for mean response time under both policies become increasingly accurate. By $k = 500$, the analytical and simulation results are indistinguishable. We obtained similar results for other system parameter settings. In the remainder of this section, we set $k = 1000$ for all simulated results.

5.2. Mean response time

Fig. 5 compares mean response time under JIQ-(d_F, d_S) and JSQ-(d_F, d_S) to that under four other policies (results for our policies are analytical, while results for the following policies are simulated):

- Under **JSQ-*d***, the dispatcher queries d servers uniformly at random and sends the job to the server among those d with the shortest queue.
- Under **SED-*d***, the dispatcher queries d servers uniformly at random and sends the job to the server among those d at which it has the shortest expected delay.
- Under **WJSQ-*d*** (the W stands for "Weighted"), the dispatcher queries d servers, where the probability that a server is queried is proportional to that server's speed, and sends the job to the server among those d with the shortest queue.
- Under **JIQ**, the dispatcher sends the job to an idle server if there is one, and to a busy server chosen uniformly at random otherwise.

We note that JSQ-*d* and JIQ are heterogeneity-unaware, SED-*d* only uses heterogeneity information when dispatching, and WJSQ-*d* only uses heterogeneity information when querying. Unlike the other five polices that we consider, JIQ is not a "power of *d*" policy; we include it here as a point of comparison because it is known to minimize the probability that an arriving job waits in the queue [1].

When there is little difference in speed between fast and slow servers ($r = 1.1$, top row of Fig. 5), JSQ-*d* and SED-*d* perform similarly to each other, and both outperform our policies at high load. This is because when all servers are similar in speed, providing more flexibility when selecting among queried servers offers a greater advantage than ensuring that some fast servers are queried. But in systems with more pronounced heterogeneity, JSQ-*d* and SED-*d* cannot maintain their good performance. As r increases, JSQ-*d* suffers significantly: here it is a serious shortcoming to make dispatching decisions based only on queue lengths. SED-*d* corrects for this problem by scaling queue lengths in proportion to server speeds. Yet when r is high and q_F is low, both JSQ-*d* and SED-*d* can lead to apparent instability. In this regime, much of the system's capacity belongs to the fast servers, but an arriving job may not query any fast servers because JSQ-*d* and SED-*d* use uniform querying (e.g., when $q_F = 0.2$, only about 40% of jobs query a fast server). This causes the slow servers to become overloaded. WJSQ-*d* avoids instability in this regime by ensuring that faster servers are more likely to be queried and thus sent a job. However, performance under WJSQ-*d* still suffers at low load; here all queue lengths are relatively short, so WJSQ-*d* effectively ignores server speeds when dispatching.

Our policies always remain stable, and in some cases achieve better performance, by differentiating between fast and slow servers both when querying and when choosing where to dispatch among the queried servers. At low load, JIQ-(d_F, d_S) and JSQ-(d_F, d_S) perform similarly to each other, and both outperform SED-*d*, JSQ-*d*, and WJSQ-*d*. As r increases, the gap between our policies and JSQ-*d* becomes particularly pronounced: JSQ-*d* frequently sends jobs to slow servers even when there are idle fast servers, whereas our policies are more likely to find and select an idle fast server. Indeed, our policies effectively throw out the slow servers when load is sufficiently low or r is sufficiently high. At high load, too, JSQ-(d_F, d_S) tends to perform competitively with or better than JSQ-*d*, SED-*d*, and WJSQ-*d*. Most notably, while JSQ-*d* and SED-*d*

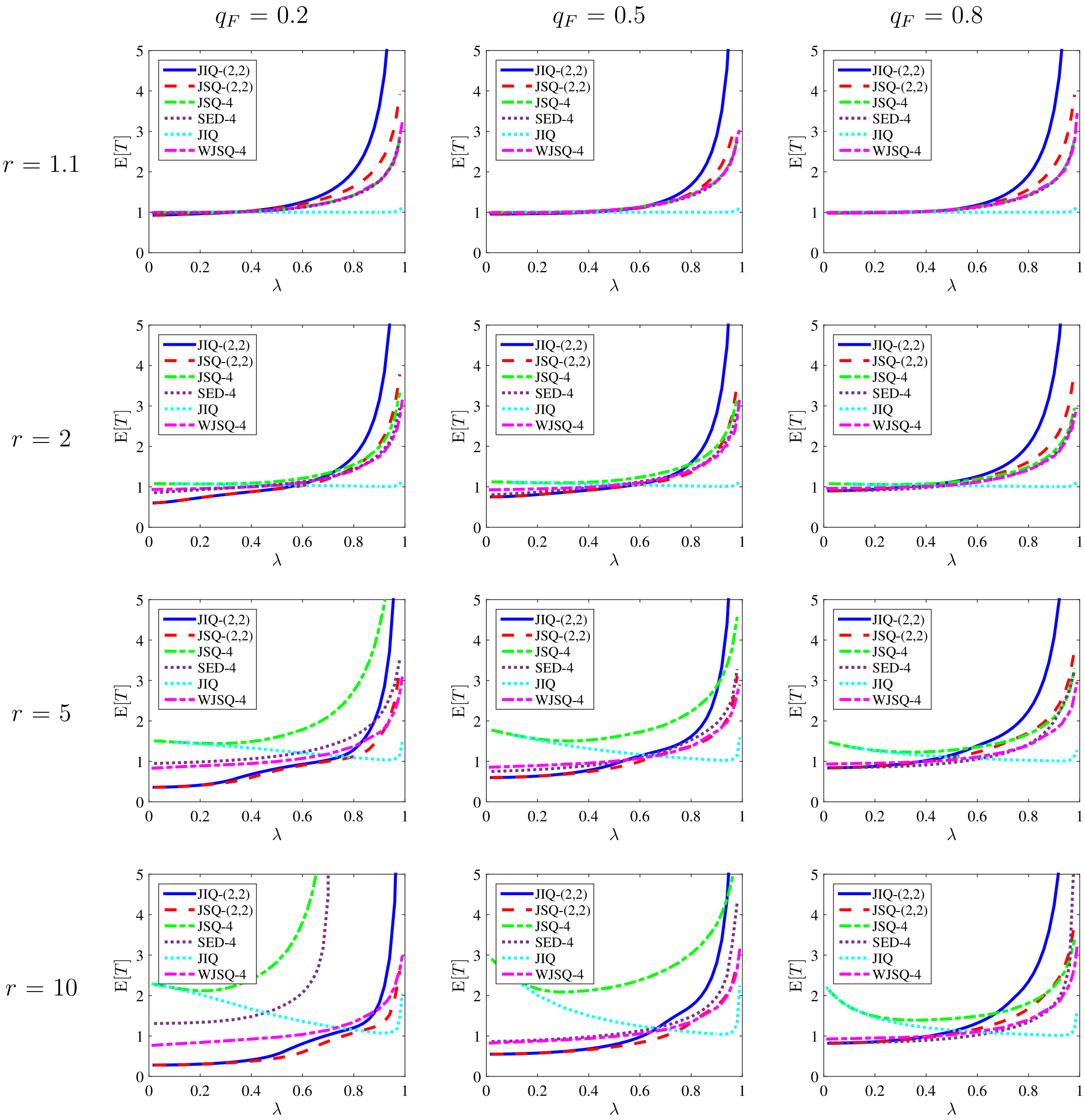


Fig. 5. Mean response time as a function of λ under JIQ-(2,2), JSQ-(2,2), JSQ-4, SED-4, and JIQ. Left to right: $q_F = 0.2$, $q_F = 0.5$, $q_F = 0.8$. Top to bottom: $r = 1.1$, $r = 2$, $r = 5$, $r = 10$.

appear to have a reduced stability region when q_F is low and r is high, both JIQ-(d_F, d_S) and JSQ-(d_F, d_S) are guaranteed to be stable provided $\lambda < \mu_F q_F + \mu_S q_S$, as shown in Theorem 1.

Unsurprisingly, JSQ-(d_F, d_S) always outperforms JIQ-(d_F, d_S). This makes sense: when using the same p_F and p_S values, the only difference between the two policies is that the JSQ version makes a better dispatching decision when choosing among busy servers. Note that the results in Fig. 5 do *not* necessarily have the same values of p_F and p_S for JSQ-(d_F, d_S) and JIQ-(d_F, d_S) because both policy families are optimized over the parameters. Even though JSQ-(d_F, d_S) is guaranteed to achieve lower mean response time than JIQ-(d_F, d_S), the two policies perform similarly until λ becomes high. At this point the advantage of JSQ-(d_F, d_S) becomes more apparent, as this is when queues actually build up. Under both JIQ-(d_F, d_S) and JSQ-(d_F, d_S), mean response time appears to be non-convex in λ. This surprising result is due to our optimization over p_F and p_S. For any fixed p_F and p_S, mean response time is convex in λ, and indeed the convex regions in the plots in Fig. 5 occur when p_F and p_S do not change (for example, when λ is relatively low it is optimal to set $p_S = 0$, i.e., to never use the slow servers). The non-convex regions appear when either p_F or p_S is varying between 0 and 1.

We also compare our policies to JIQ, which uses queue length information from all servers, not just a subset of d servers. At high load, JIQ outperforms all of the "power of d" policies; this is unsurprising given that JIQ will always find an idle server if there is one. But at low load and high r, JIQ yields a substantially higher mean response time than our

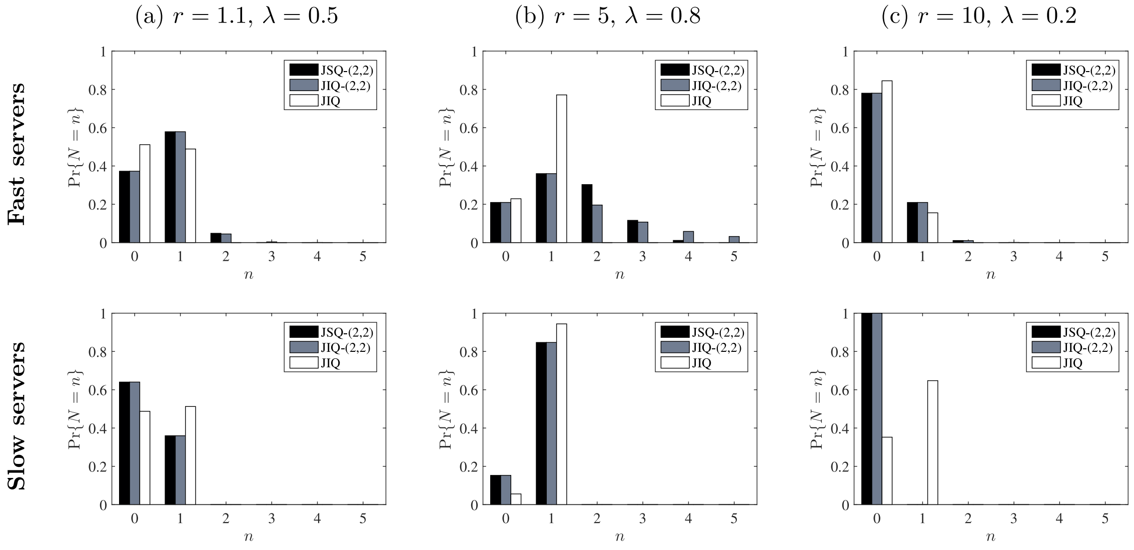


Fig. 6. Comparing the queue length distribution under JSQ-(2,2), JIQ-(2,2), and JIQ for fast servers (top row) and slow servers (bottom row) when $q_F = 0.5$. (a) $r = 1.1$, $\lambda = 0.5$, (b) $r = 5$, $\lambda = 0.8$, (c) $r = 10$, $\lambda = 0.2$.

policies. This is because, like JSQ-d and WJSQ-d, JIQ does not use server speed information to break ties between idle servers. That our policies outperform JIQ may seem surprising in light of the fact that JIQ is delay optimal [1]; we explore this result further in Section 5.3.

5.3. Queue length distribution

In this section we look at the queue length distributions under JIQ-(d_F, d_S), JSQ-(d_F, d_S), and JIQ in more detail to gain insight as to why our policies can outperform JIQ in terms of response time, even though they lack JIQ's queue length optimality property.

Fig. 6 shows the queue length distribution under JIQ-(d_F, d_S), JSQ-(d_F, d_S), and JIQ for both fast servers (top row) and slow servers (bottom row) in three settings selected from those featured in Fig. 5. At left, we show a case in which all three policies have similar mean response times; in this case the queue length distributions are also similar. The center column shows a case in which JIQ yields lower mean response time than our policies: in this case $r = 5$ and $\lambda = 0.8$. Because λ is high, few slow servers are idle, but both our policies and JIQ prevent queues from building up at the slow servers. The key difference between the policies lies in what happens at the fast servers. Under our policies, the optimal value of p_F in this setting is 1, meaning that a job will never choose to wait in the queue at a slow server. This means that many jobs are deferred back to the (busy) fast servers, causing the queue lengths to increase. JIQ prevents the queue lengths at the fast servers from growing. A slightly greater proportion of jobs run on slow servers under JIQ, but the jobs that run on fast servers do not have to wait in the queue. When λ is high, this tradeoff favors JIQ.

In contrast, when λ is low the same tradeoff favors JIQ-(d_F, d_S) and JSQ-(d_F, d_S), as shown in the right column of Fig. 6, where $r = 10$ and $\lambda = 0.2$. Again, under JIQ a higher proportion of slow servers are busy because JIQ does not differentiate between fast and slow idle servers. Indeed, there are no busy slow servers under JIQ-(d_F, d_S) and JSQ-(d_F, d_S) because the combination of high r and low λ means that the optimal value of p_S is 0: it is best not to use any of the slow servers at all. As a result, the fast servers have a slightly lower probability of being idle under our policies than under JIQ. However, because λ is low the queue lengths under JIQ-(d_F, d_S) and JSQ-(d_F, d_S) remain short. In this case, JIQ's decision to prioritize server idleness over server speed works against it, and our policies achieve lower mean response time.

5.4. Sensitivity to d

One of the primary selling points of policies like JSQ-d, SED-d, and WJSQ-d is the "power of two choices": often, there is a large benefit in going from $d = 1$ (i.e., random routing) to $d = 2$, but a much smaller marginal benefit in further increasing d. Consequently, JSQ-2 is the most commonly considered variant of JSQ-d. Our JIQ-(d_F, d_S) and JSQ-(d_F, d_S) policies query fast and slow servers separately; while setting $d_F = d_S = 1$ offers two choices in total, it does not offer a choice within each speed. Therefore, JIQ-(1,1) and JSQ-(1,1) are equivalent: once the dispatcher has chosen to send the job to a fast (or slow) server there is only one choice for which server to use. Henceforth, we will refer to both policies as JIQ-(1,1).

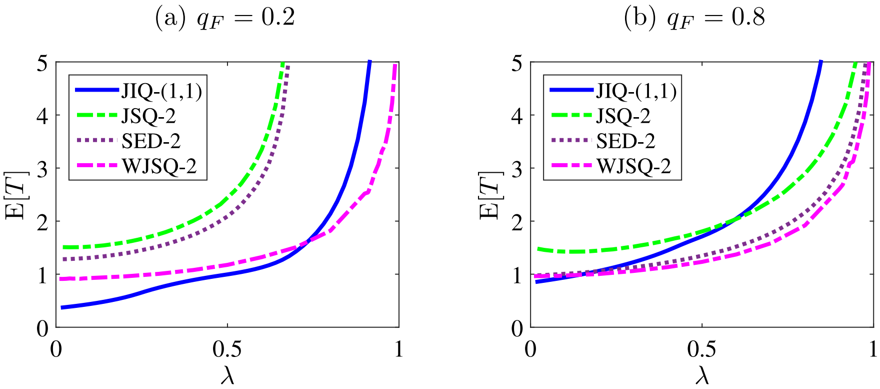


Fig. 7. Mean response time as a function of λ under JIQ-(1,1), JSQ-2, SED-2, and WJSQ-2 when $r = 5$. (a) $q_F = 0.2$, (b) $q_F = 0.8$.

Unlike JSQ-2 and SED-2, JIQ-(1,1) uses queue length information only when deciding between an idle slow server and a busy fast server; all other decisions are made probabilistically. This makes JIQ-(1,1) much closer to random routing than either JSQ-2 or SED-2, and one might think that consequently JIQ-(1,1) would generally exhibit poor performance. However, our results indicate the opposite: JIQ-(1,1) often substantially outperforms JSQ-2 and SED-2, especially when q_F is low (see Fig. 7). As previously noted, both JSQ-2 and SED-2 appear unstable when q_F is low and r is high, whereas JIQ-(1,1) guarantees that the system will remain stable.

In Fig. 8 we consider the effect of varying $d = d_F + d_S$ on the performance of JIQ-(d_F, d_S) and JSQ-(d_F, d_S): does the marginal benefit of increasing d decrease as d gets larger? When $d = 1$, we interpret our policies to collapse the querying and dispatching decision points into a single probabilistic choice: we dispatch to a random fast server with probability p_F and to a slow server otherwise. For all other values of d, we choose the optimal combination of d_F, d_S, p_F, and p_S such that $d_F + d_S = d$. As under JSQ-d and SED-d, the steepest drop in mean response time comes from going from $d = 1$ to $d = 2$, and mean response time is convex in d. When the fast and slow servers are similar in speed (Fig. 8 (a)), JSQ-d and SED-d perform slightly better at low d, and all policies have similar performance at high d. When the r is high and q_F is low (Fig. 8 (b)), JIQ-(d_F, d_S) and JSQ-(d_F, d_S) are stable at all values of d, and outperform JSQ-d and SED-d even when d is high enough for the latter two policies to be stable.

6. A heuristic for p_F and p_S

A key part of defining the JIQ-(d_F, d_S) and JSQ-(d_F, d_S) policies involves choosing values for p_F and p_S; in Section 4 we do this by finding the values of p_F and p_S that minimize mean response time. Fig. 9 shows mean response time under JSQ-(d_F, d_S) as a function of p_F and p_S for two different parameter settings (results for JIQ-(d_F, d_S) are similar). When λ is low to moderate (Fig. 9(a)), mean response time is relatively insensitive to the particular parameter choices, provided that p_S is high enough to ensure stability. When λ is high (Fig. 9(b)), it becomes more important to choose the right p_F and p_S: even small variations in p_F and p_S can lead to substantial changes in response time, and there is a smaller set of p_F and p_S values for which the system is stable.

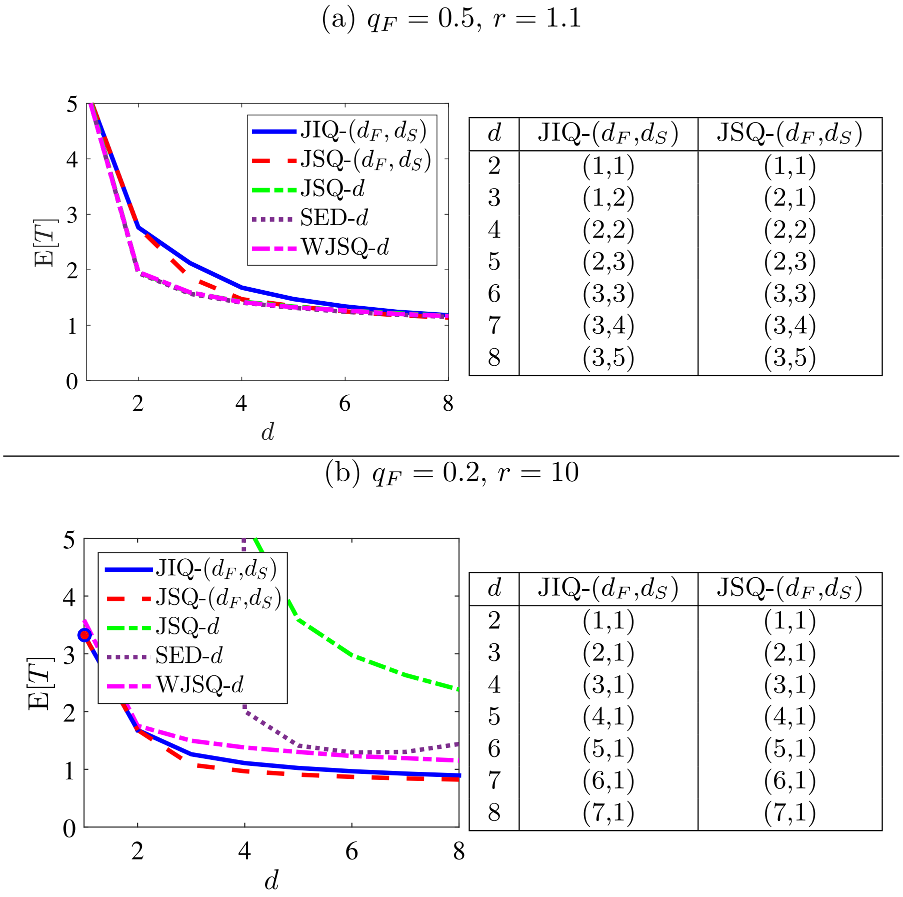
The extreme sensitivity to p_F and p_S occurs only at very high λ; at most parameter settings the optimal values of p_F and p_S fall into one of a few cases. If the fast servers comprise a sufficiently high fraction of the total system capacity or if the system load is very low, it is best to set $p_S = 0$. If the fast and slow servers are relatively similar in speed or if the system load is sufficiently high, it is best to set $p_S = 1$. As we showed in Theorem 2, as $\lambda \to 1$, $p_F = \mu_F q_F$ is the only value of p_F for which the system is stable.

Motivated by these observations, we propose a heuristic for choosing appropriate values of p_F and p_S. Instead of optimizing over the entire parameter space for p_F and p_S, which can be computationally expensive, we consider the following parameter settings:

- $p_S = 0$. Note that in this case the slow servers are never used, so the choice of p_F does not matter.
- All combinations of $p_S \in \{\mu_S q_S, 1\}$ and $p_F \in \{0, \mu_F q_F, 1\}$.

For each setting of λ, q_F, and r, this gives us only seven policies to compare; we select the p_F and p_S that yields the best performance among these seven alternatives.

Table 1 shows our results for JIQ-(d_F, d_S) and JSQ-(d_F, d_S); each row shows a different value of λ, for a system with $q_F = 0.2$ and $r = 10$. Under both policies, when λ is low it is optimal to set $p_S = 0$, and our heuristic correctly selects this policy. As λ starts to increase, it becomes optimal to increase p_S continuously and set $p_F = 1$. Our heuristic sets $p_F = 1$ and changes p_S in discrete steps from 0 to $\mu_S q_S$ to 1; because λ is still relatively low, mean response time is relatively insensitive to selecting a slightly suboptimal value of p_S and our heuristic has low error. When λ becomes high,



(a) $q_F = 0.5$, $r = 1.1$

d	JIQ-(d_F,d_S)	JSQ-(d_F,d_S)
2	(1,1)	(1,1)
3	(1,2)	(2,1)
4	(2,2)	(2,2)
5	(2,3)	(2,3)
6	(3,3)	(3,3)
7	(3,4)	(3,4)
8	(3,5)	(3,5)

(b) $q_F = 0.2$, $r = 10$

d	JIQ-(d_F,d_S)	JSQ-(d_F,d_S)
2	(1,1)	(1,1)
3	(2,1)	(2,1)
4	(3,1)	(3,1)
5	(4,1)	(4,1)
6	(5,1)	(5,1)
7	(6,1)	(6,1)
8	(7,1)	(7,1)

Fig. 8. Effect of varying d on mean response time under JIQ-(d_F, d_S), JSQ-(d_F, d_S), JSQ-d, SED-d, and WJSQ-d when $\lambda = 0.8$. (a) $q_F = 0.5$, $r = 1.1$. (b) $q_F = 0.2$, $r = 10$. The tables at right show the optimal choices of (d_F, d_S) for each d.

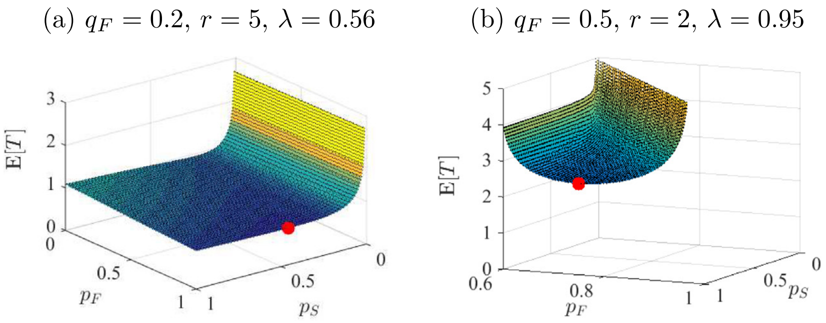


Fig. 9. Mean response time as a function of p_F and p_S. (a) $q_F = 0.2$, $r = 5$, $\lambda = 0.56$, (b) $q_F = 0.5$, $r = 2$, $\lambda = 0.95$. The red circle indicates the optimal $\mathbf{E}[T]$.

the performance of our heuristic can suffer. In this region it becomes optimal to set $p_S = 1$ and decrease p_F continuously, while our heuristic must choose either $p_F = 1$ or $p_F = \mu_F q_F$. Because λ is high, a small change in p_F (which corresponds to a small change in the arrival rate to any individual busy server), can have a big affect on mean response time, and the error of our heuristic can reach as high as 25%. However, as $\lambda \to 1$, the heuristic, which sets $p_S = 1$ and $p_F = \mu_F q_F$, again approaches perfect accuracy because $p_F = \mu_F q_F$ is the only value of p_F that maintains stability, and as $\lambda \to 1$ the queue lengths build up so using an idle slow server when one is available ($p_S = 1$) also should be optimal.

Table 1
Comparison of optimal p_F and p_S to best heuristic under JIQ-(2,2) (left) and JSQ-(2,2) (right). Here $q_F = 0.2$ and $r = 5$. The columns p_F^* and p_S^* give the optimal values of p_F and p_S, while p_F^{heur} and p_S^{heur} are the values chosen by the heuristic.

λ	JIQ-(2,2)							JSQ-(2,2)						
	p_F^*	p_S^*	$\mathbf{E}\left[T_{\text{opt}}\right]$	p_F^{heur}	p_S^{heur}	$\mathbf{E}\left[T_{\text{heur}}\right]$	% error	p_F^*	p_S^*	$\mathbf{E}\left[T_{\text{opt}}\right]$	p_F^{heur}	p_S^{heur}	$\mathbf{E}\left[T_{\text{heur}}\right]$	% error
0.14	any	0	0.384	any	0	0.384	0	any	0	0.383	any	0	0.383	0
0.24	any	0	0.443	any	0	0.443	0	any	0	0.429	any	0	0.429	0
0.34	0.999	0.018	0.575	any	0	0.576	0.023	any	0	0.514	any	0	0.514	0
0.44	1	0.426	0.742	1	0.444	0.743	0.014	1	0.103	0.677	any	0	0.689	1.693
0.54	1	0.723	0.868	1	1	0.879	1.196	1	0.405	0.832	1	0.444	0.833	0.066
0.64	1	1	0.967	1	1	0.967	0	1	0.722	0.946	1	1	0.954	0.762
0.74	1	1	1.101	1	1	1.101	0	1	1	1.039	1	1	1.039	0
0.84	0.877	1	1.547	1	1	1.605	3.732	1	1	1.217	1	1	1.217	0
0.90	0.714	1	2.331	0.555	1	2.908	24.754	0.839	1	1.595	1	1	1.957	22.697
0.98	0.579	1	10.677	0.555	1	12.837	20.231	0.597	1	3.243	0.555	1	3.659	12.804

7. Conclusion

This paper addresses the problem of dispatching in large-scale, heterogeneous systems. We design two new heterogeneity-aware families of policies, JIQ-(d_F, d_S) and JSQ-(d_F, d_S). Our policies are simple, analytically tractable, and outperform classical policies in many settings.

Our results yield several insights about how to design "power of d" policies that perform well in heterogeneous settings. In order to maintain the maximum stability region, the dispatcher must ensure that fast servers are queried sufficiently often. Alone, neither uniform sampling nor weighting querying in favor of fast servers is enough to ensure good performance. Our work establishes that, instead, dispatching policies should use heterogeneity information at two decision points: (1) when choosing which servers to query, and (2) when choosing where among the queried servers to dispatch a job. Ultimately, how best to distribute jobs among fast and slow servers depends jointly on the system load, the fraction of servers that are fast, and the relative speeds of the servers. It may be best to use only fast servers, to use slow servers only when they are idle, or to balance jobs among fast and slow servers in some other way. Because there is no single right answer, policies designed for heterogeneous systems must be able to adapt to the system parameters. JIQ-(d_F, d_S) and JSQ-(d_F, d_S) do this by optimizing over the probabilistic parameters to choose the best allocation of jobs to fast and slow servers. Moreover, as we show in Theorem 1, the optimal policy in each family is guaranteed to be stable.

We focus specifically on policies that query fixed numbers of fast and slow servers and then make probabilistic decisions about how to route among the queried servers based on idleness and queue length information. The space of policies that use heterogeneity information at both decision points is much larger than the policies we propose here. For example, our policies can be generalized at the first decision point by choosing d_F and d_S probabilistically for each query; this also allows us to adapt our policies for systems with more than two server speeds (see our work in [43] for preliminary results on this topic). At the second decision point, one could combine (d_F, d_S)-style querying with a heterogeneity-aware dispatching policy, such as SED. While optimizing over such a large policy space is likely to be challenging, we are optimistic that substantial advances could be made in future work toward understanding a wider scope of policies and settings.

Differing server speeds is just one way in which server farms may exhibit heterogeneity. Systems may also consist of servers that are heterogeneous in their memory, network bandwidth, or any other resource availability. Some jobs may be able to run on certain servers but not on others, for example due to data locality. Jobs may be capable of running on any server, but may have a preference for or run faster on certain servers. The policies we present in this paper are designed to perform well specifically for the case of heterogeneous server speeds, but we believe the insights gained will aid the design of effective load balancing policies for the broad range of heterogeneity that exists in today's systems.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The authors thank Brandon Lieberthal for assisting with the proof in Appendix B, Maaike Verloop for shepherding the paper, and the anonymous reviewers for their valuable feedback.

Appendix A

Here we give the complete expanded form of the optimization formulations given in (17), (25).

For JIQ-(d_F, d_S), for general service distribution and the coefficient of variation associated with service times the same across both server speeds (equal to c_v), our optimization formulation (17) is as follows:

$$\begin{aligned}
\underset{p_F, p_S}{\text{minimize}} \quad & \frac{q_F\left(\lambda_{IF}\left(1-\rho_F\right)+\lambda_{BF}\rho_F\right)}{\lambda}\left(\frac{\lambda_{BF}(1+c_v^2)\mathbf{E}\left[Y_F\right]^2}{2(1-\lambda_{BF}\mathbf{E}\left[Y_F\right])}+\mathbf{E}\left[Y_F\right]\right) \\
& +\frac{q_S\left(\lambda_{IS}\left(1-\rho_S\right)+\lambda_{BS}\rho_S\right)}{\lambda}\left(\frac{\lambda_{BS}(1+c_v^2)\mathbf{E}\left[Y_S\right]^2}{2(1-\lambda_{BS}\mathbf{E}\left[Y_S\right])}+\mathbf{E}\left[Y_S\right]\right) \\
\text{subject to} \quad & \rho_F=\frac{\lambda}{\mu_F q_F}\left((1-\rho_F^{d_F})+\rho_F^{d_F}(1-\rho_S^{d_S})(1-p_S)+\rho_F^{d_F}\rho_S^{d_S}p_F\right) \\
& \rho_S=\frac{\lambda}{\mu_S q_S}\left(\rho_F^{d_F}(1-\rho_S^{d_S})p_S+\rho_F^{d_F}\rho_S^{d_S}(1-p_F)\right) \\
& \lambda_{BF}=\frac{\lambda}{q_F}\rho_F^{d_F-1}\left(\left(1-\rho_S^{d_S}\right)(1-p_S)+\rho_S^{d_S}p_F\right) \\
& \lambda_{BS}=\frac{\lambda}{q_S}\rho_F^{d_F}\rho_{0S}^{d_S-1}(1-p_F) \\
& \lambda_{IF}=\frac{\lambda d_F}{q_F}\left(\sum_{i=0}^{d_F-1}\binom{d_F-1}{i}\frac{(1-\rho_F)^i\rho_F^{d_F-1-i}}{i+1}\right) \\
& \lambda_{IS}=\frac{\lambda d_S}{q_S}\rho_F^{d_F}\left(\sum_{i=0}^{d_S-1}\binom{d_S-1}{i}\frac{(1-\rho_S)^i\rho_S^{d_S-1-i}}{i+1}p_S\right) \\
& 0<\pi_{0F},\pi_{0S}\leq 1 \\
& 0\leq p_F,p_S\leq 1
\end{aligned}$$

For JSQ-(d_F, d_S) our optimization formulation (25) is as follows:

$$\begin{aligned}
\underset{p_F, p_S}{\text{minimize}} \quad & \frac{1}{\mu_F}\cdot\left(1-\rho_F^{d_F}\right)+\frac{1}{\mu_S}\cdot\rho_F^{d_F}\left(1-\rho_S^{d_S}\right)p_S \\
& +\frac{1}{\mu_F}\sum_{i=1}^{\infty}(i+1)\cdot\frac{f_i^{d_F}-f_{i+1}^{d_F}}{f_1^{d_F}}\cdot\rho_F^{d_F}\left(\rho_S^{d_S}p_F+\left(1-\rho_S^{d_S}\right)(1-p_S)\right) \\
& +\frac{1}{\mu_S}\sum_{i=1}^{\infty}(i+1)\cdot\frac{s_i^{d_S}-s_{i+1}^{d_S}}{s_1^{d_S}}\cdot\rho_F^{d_F}\rho_S^{d_S}(1-p_F) \\
\text{subject to} \quad & \rho_F=\frac{\lambda}{\mu_F q_F}\left(\rho_F^{d_F}\left(1-\rho_S^{d_S}\right)(1-p_S)\right)+\frac{\lambda}{\mu_F q_F}\left(\left(1-\rho_F^{d_F}\right)+\rho_F^{d_F}\rho_S^{d_S}p_F\right) \\
& \rho_S=\frac{\lambda}{\mu_S q_S}\left(\rho_F^{d_F}\left(1-\rho_S^{d_S}\right)p_S+\rho_F^{d_F}\rho_S^{d_S}(1-p_F)\right) \\
& \frac{\lambda}{q_F}\left(1-f_1^{d_F}\right)=\mu_F(f_1-f_2) \\
& \frac{\lambda}{q_F}\left(f_{i-1}^{d_F}-f_i^{d_F}\right)\left(\left(1-\rho_S^{d_S}\right)(1-p_S)+\rho_S^{d_S}p_F\right)=\mu_F\left(f_i-f_{i+1}\right) \qquad i\geq 2 \\
& \frac{\lambda}{q_S}\left(s_{i-1}^{d_S}-s_i^{d_S}\right)\rho_F^{d_F}p_S=\mu_S\left(s_i-s_{i+1}\right) \\
& \frac{\lambda}{q_S}\left(s_{i-1}^{d_S}-s_i^{d_S}\right)\rho_F^{d_F}(1-p_F)=\mu_S\left(s_i-s_{i+1}\right) \quad i\geq 2 \\
& f_0=s_0=1 \\
& f_1=\rho_F \\
& s_1=\rho_S \\
& 0\leq\rho_F,\rho_S<1 \\
& 0\leq p_F,p_S\leq 1
\end{aligned}$$

Appendix B

In this appendix we show that there exists a unique and stable stationary point of the system given by (19), (20), (22), (23). We will prove this is true for Eqs. (19), (20), and the proof for Eqs. (22), (23) proceeds in the same way.

Proof. First, to show that (19), (20) have a unique stationary point $f^\star = (f_1^\star, f_s^\star, \dots)$, recall that $f_1 = \rho_F$ so Eq. (20) is

$$\frac{df_1}{dt} = \frac{\lambda}{q_F}\left(1 - \rho_F^{d_F} - \mu_F(\rho_F - f_2)\right).$$

Since the right hand side is a first order polynomial in f_2, there is only one possible value $f_2^\star$ can take. Similarly, is we assume by induction that we know $f_1^\star, \dots, f_{k-1}^\star$ then the right hand side of (19) is a first order polynomial and thus there is only one possible value for $f_k^\star$. Thus there exists a unique stationary point to (19), (20).

To show that the stationary point $f^\star$ is stable, it will suffice to show that the Jacobian of this system at $j^\star$ is negative definite. For simplicity let $\Lambda_F = \frac{\lambda}{q_F}\left(\left(1 - \rho_S^{d_S}\right)(1 - p_S) + \rho_S^{d_S} p_F\right)$ represent the average rate at which jobs are routed to a tagged fast server. Then the Jacobian is given as follows

$$\begin{aligned}
J_{1,1} &= -\frac{\lambda}{q_F} d_F f_1^{d_F - 1} - \mu_F & & \\
J_{i,i-1} &= \Lambda_F d_F f_{i-1}^{d_F - 1} & & i \geq 2 \\
J_{i,i} &= -\Lambda_F d_F f_i^{d_f - 1} - \mu_F & & i \geq 2 \\
J_{i,i+1} &= \mu_F & & \forall\, i
\end{aligned}$$

or equivalently,

$$J = \begin{bmatrix}
-\frac{\lambda}{q_F} d_F f_1^{d_F-1} - \mu_F & \mu_F & 0 & & \\
\Lambda_F d_F f_1^{d_F-1} & -\Lambda_F d_F f_2^{d_f-1} - \mu_F & \mu_F & 0 & \\
0 & \Lambda_F d_F f_2^{d_F-1} & -\Lambda_F d_F f_3^{d_f-1} - \mu_F & \mu_F & \dots \\
 & \vdots & \vdots & \vdots & \ddots
\end{bmatrix}$$

Notice that J^T is weakly diagonally dominant since for $j \geq 2$, $J_{j,j} = -\sum_{i \neq j} |J_{i,j}|$ and

$$\begin{aligned}
|J_{1,1}| &= \frac{\lambda}{q_F} d_F f_1^{d_F - 1} + \mu_F \\
&> \frac{\lambda}{q_F} d_F f_1^{d_F - 1} \\
&\geq \Lambda_F d_F f_1^{d_F - 1} \\
&= \sum_{i \neq 1} |J_{i,1}|
\end{aligned}$$

provided $\mu_F > 0$. Moreover, J^T is weakly chained diagonally dominant since its associated directed graph is strongly connected [44,45]. Hence, J^T, and thus J, is non-singular. Together with the fact that all of its diagonal entries are negative, this means J must be negative definite. □

References

[1] A. Stolyar, Pull-based load distribution in large-scale heterogeneous service systems, Queueing Syst. 80 (4) (2015) 341–361.

[2] R.R. Weber, On the optimal assignment of customers to parallel servers, J. Appl. Probab. 15 (2) (1978) 406–413.

[3] W. Winston, Optimality of the shortest line discipline, J. Appl. Probab. 14 (1) (1977) 181–189.

[4] R.D. Nelson, T.K. Philips, An Approximation to the Response Time for Shortest Queue Routing, Vol. 17, No. 1, ACM, 1989.

[5] V. Gupta, M. Harchol-Balter, K. Sigman, W. Whitt, Analysis of join-the-shortest-queue routing for web server farms, Perform. Eval. 64 (9–12) (2007) 1062–1081.

[6] M. Mitzenmacher, The power of two choices in randomized load balancing, IEEE Trans. Parallel Distrib. Syst. 12 (10) (2001) 1094–1104.

[7] N. Vvedenskaya, R. Dobrushin, F. Karpelevich, Queueing system with selection of the shortest of two queues: An asymptotic approach, Probl. Pereda. Inf. 32 (1) (1996) 20–34.

[8] T. Hellemans, T. Bodas, B. Van Houdt, Performance Analysis of Workload Dependent Load Balancing Policies, in: Proceedings of the ACM on Measurement and Analysis of Computing Systems, 2019.

[9] Y. Lu, Q. Xie, G. Kliot, A. Geller, J. Larus, A. Greenberg, Join-Idle-Queue: A novel load balancing algorithm for dynamically scalable web services, Perform. Eval. 68 (11) (2011) 1056–1071.

[10] C. Wang, C. Feng, J. Cheng, Distributed join-the-idle-queue for low latency cloud services, IEEE/ACM Trans. Netw. 26 (5) (2018) 2309–2319.

[11] A. Izagirre, A. Makowski, Light traffic performance under the power of two load balancing strategy: the case of server heterogeneity., SIGMETRICS Perform. Eval. Rev. 42 (2) (2014) 18–20.

[12] X. Zhou, F. Wu, J. Tan, Y. Sun, N. Shroff, Designing low-complexity heavy-traffic delay-optimal load balancing schemes: Theory to algorithms, Proc. ACM Meas. Anal. Comput. Syst. 1 (2) (2017) 39.

[13] A. Mukhopadhyay, R. Mazumdar, Analysis of randomized join-the-shortest-queue (JSQ) schemes in large heterogeneous processor-sharing systems, IEEE Trans. Control Netw. Syst. 3 (2) (2016) 116–126.

[14] S. Banawan, N. Zeidat, A comparative study of load sharing in heterogeneous multicomputer systems, in: Proceedings. 25th Annual Simulation Symposium, IEEE, 1992, pp. 22–31.

[15] W. Whitt, Deciding which queue to join: Some counterexamples, Oper. Res. 34 (1) (1986) 55–62.

[16] J. Selen, I. Adan, S. Kapodistria, Approximate performance analysis of generalized join the shortest queue routing, in: Proceedings of the 9th EAI International Conference on Performance Evaluation Methodologies and Tools, ICST (Institute for Computer Sciences, Social-Informatics and …, 2016, pp. 103–110.

[17] J. Selen, I. Adan, S. Kapodistria, J. van Leeuwaarden, Steady-state analysis of shortest expected delay routing, Queueing Syst. 84 (3–4) (2016) 309–354.

[18] H. Chen, H.-Q. Ye, Asymptotic optimality of balanced routing, Oper. Res. 60 (1) (2012) 163–179.

[19] G. Koole, A simple proof of the optimality of a threshold policy in a two-server queueing system, Systems Control Lett. 26 (5) (1995) 301–303.

[20] R.L. Larsen, Control of Multiple Exponential Servers with Application to Computer Systems (PhD thesis), University of Maryland at College Park, College Park, MD, USA, 1981.

[21] W. Lin, P.R. Kumar, Optimal control of a queueing system with two heterogeneous servers, IEEE Trans. Automat. Control 29 (8) (1984) 696–703.

[22] M. Rubinovitch, The slow server problem, J. Appl. Probab. 22 (1) (1985) 205–213.

[23] M. Rubinovitch, The slow server problem: A queue with stalling, J. Appl. Probab. 22 (4) (1985) 879–892.

[24] H.P. Luh, I. Viniotis, Threshold control policies for heterogeneous server systems, Math. Methods Oper. Res. 55 (1) (2002) 121–142.

[25] V.V. Rykov, D.V. Efrosinin, On the slow server problem, Autom. Remote Control 70 (12) (2009) 2013–2023.

[26] S. Shenker, A. Weinrib, The optimal control of heterogeneous queueing systems: a paradigm for load-sharing and routing, IEEE Trans. Comput. 38 (12) (1989) 1724–1735.

[27] F. Bonomi, On job assignment for a parallel system of processor sharing queues, IEEE Trans. Comput. 39 (7) (1990) 858–869.

[28] S.A. Banawan, J. Zahorjan, Load sharing in heterogeneous queueing systems, in: Proc. of IEEE INFOCOM'89, 1989, pp. 731–739.

[29] H. Feng, V. Misra, D. Rubenstein, Optimal state-free, size-aware dispatching for heterogeneous M/G/-type systems, Perform. Eval. 62 (1) (2005) 475–492, Performance 2005.

[30] J. Sethuraman, M.S. Squillante, Optimal stochastic scheduling in multiclass parallel queues, SIGMETRICS Perform. Eval. Rev. 27 (1) (1999) 93–102.

[31] A.N. Tantawi, D. Towsley, Optimal static load balancing in distributed computer systems, J. ACM 32 (2) (1985) 445–465.

[32] E. Hyytiä, Optimal routing of fixed size jobs to two parallel servers, INFOR: Inf. Syst. Oper. Res. 51 (4) (2013) 215–224.

[33] Q. Bu, L. Liu, Y.Q. Zhao, Mean Field Approximations to a Queueing System with Threshold-Based Workload Control Scheme, Technical Report, 2018.

[34] A. Karthik, A. Mukhopadhyay, R. Mazumdar, Choosing among heterogeneous server clouds, Queueing Syst. 85 (1) (2017) 1–29.

[35] Q.-L. Li, C. Chen, R.-N. Fan, L. Xu, J.-Y. Ma, Queueing analysis of a large-scale bike sharing system through mean-field theory, 2016.

[36] Q.-L. Li, G. Dai, J.C.S. Lui, Y. Wang, The mean-field computation in a supermarket model with server multiple vacations, Discrete Event Dyn. Syst. 24 (4) (2014) 473–522.

[37] Q.-L. Li, F. Yang, Mean-field analysis for heterogeneous work stealing models, in: Communications in Computer and Information Science, Vol. 564, Springer Verlag, 2015, pp. 28–40.

[38] M. Bramson, Y. Lu, B. Prabhakar, Randomized load balancing with general service time distributions, ACM SIGMETRICS Perform. Eval. Rev. 38 (1) (2010) 275, http://dx.doi.org/10.1145/1811099.1811071.

[39] M. Bramson, Y. Lu, B. Prabhakar, Asymptotic independence of queues under randomized load balancing, Queueing Syst. (2012).

[40] T. Hellemans, B. Van Houdt, On the power-of-d-choices with least loaded server selection, ACM SIGMETRICS Perform. Eval. Rev. (2019).

[41] K. Gardner, M. Harchol-Balter, A. Scheller-Wolf, M. Velednitsky, S. Zbarsky, Redundancy-d: The power of d choices for redundancy, Oper. Res. 65 (4) (2017) 1078–1094.

[42] M. Harchol-Balter, Performance Modeling and Design of Computer Systems: Queueing Theory in Action, Cambridge University Press, 2013.

[43] J. Abdul Jaleel, A. Wickeham, S. Doroudi, K. Gardner, A General"Power-of-d" Dispatching Framework for Heterogeneous Systems, in: Workshop on Mathematical Performance Modeling and Analysis, MAMA, 2020.

[44] P. Azimzadeh, P.A. Forsyth, Weakly chained matrices, policy iteration, and impulse control, SIAM J. Numer. Anal. (2016).

[45] P.N. Shivakumar, K.H. Chew, A sufficient condition for nonvanishing of determinants, Proc. Amer. Math. Soc. (1974).