Queueing Systems (2022) 102:431–480
https://doi.org/10.1007/s11134-022-09736-z

A general "power-of-*d*" dispatching framework for heterogeneous systems

Jazeem Abdul Jaleel[1] · **Sherwin Doroudi**[1] · **Kristen Gardner**[2] · **Alexander Wickeham**[1]

Received: 1 January 2021 / Revised: 17 December 2021 / Accepted: 13 January 2022 /
Published online: 28 April 2022

Abstract

Intelligent dispatching is crucial to obtaining low response times in large-scale systems. One common scalable dispatching paradigm is the "power-of-*d*," in which the dispatcher queries *d* servers at random and assigns the job to a server based only on the state of the queried servers. The bulk of power-of-*d* policies studied in the literature assume that the system is homogeneous, meaning that all servers have the same speed; meanwhile, real-world systems often exhibit server speed heterogeneity. This paper introduces a general framework for describing and analyzing heterogeneity-aware power-of-*d* policies. The key idea behind our framework is that dispatching policies can make use of server speed information at two decision points: when choosing which *d* servers to query and when assigning a job to one of those servers. Our framework explicitly separates the dispatching policy into a querying rule and an assignment rule; we consider general families of both rule types. While the strongest assignment rules incorporate both detailed queue-length information and server speed information, these rules typically are difficult to analyze. We overcome this difficulty by focusing on heterogeneity-aware assignment rules that ignore queue length information beyond idleness status. In this setting, we analyze mean response time and formulate novel optimization problems for the joint optimization of querying and assignment. We build upon our optimized policies to develop heuristic queue length-aware dispatching policies. Our heuristic policies perform well in simulation, relative to policies that have appeared in the literature.

Keywords Dispatching · Power-of-*d* · Mean Field Analysis · Heterogeneity

Mathematics Subject Classification 60K25 · 68M20

✉ Jazeem Abdul Jaleel
abdul314@umn.edu

[1] University of Minnesota Twin Cities: University of Minnesota, Minneapolis, USA

[2] Department of Computer Science, Amherst College, Amherst, MA, USA

1 Introduction

Large-scale systems are everywhere, and deciding how to dispatch an arriving job to one of the many available servers is crucial to obtaining low response time. One common scalable dispatching paradigm is the "power-of-d," in which the dispatcher queries d servers at random and assigns the job to a server based only on the state of the queried servers. Such policies incur a much lower communication cost than querying all servers while sacrificing little in the way of performance. However, many power-of-d policies, such as Join the Shortest Queue-d (JSQ-d)[1] [19], share a notable weakness: they do not account for the fact that, in many modern systems, the servers' speeds are *heterogeneous*. Unfortunately, such heterogeneity-unaware dispatching policies can perform quite poorly in the presence of server heterogeneity [7]. Indeed, it is not straightforward to determine how to dispatch in heterogeneous systems to achieve low mean response times. For example, it may sometimes be desirable to exclude the slowest classes of servers entirely, yet at other times even the slow servers are needed to maintain the system's stability.

Motivated by the need for dispatching policies that perform well in heterogeneous systems, researchers have designed new policies for this setting. For example, under the Shortest Expected Delay-d (SED-d) policy the dispatcher queries d servers uniformly at random and assigns the arriving 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 the smallest [26]. Under the Balanced Routing (BR) policy, the dispatcher queries d servers with probabilities proportional to the servers' speeds and assigns the arriving job to the queried server with the fewest jobs in the queue [4]. While both of these policies generally lead to better performance than the fully heterogeneity-unaware JSQ-d policy, there is still substantial room for improvement. Together, SED-d and BR illustrate a key observation about how to design heterogeneity-aware power-of-d dispatching policies. There are two decision points at which such policies can use server speed information: when choosing which d servers to query (exploited by BR), and when assigning a job to one of those servers (exploited by SED-d).

One of the primary contributions of this paper is the introduction of a general framework to describe and analyze heterogeneity-aware power-of-d policies; we discuss our framework in detail in Sect. 3. Our framework explicitly separates the dispatching policy into a *querying rule* that determines how to select d servers upon a job's arrival, and an *assignment rule* that determines where among the d queried servers to send the job. Both SED-d and BR fit within our framework, as do many other policies that have been proposed and studied in the literature. For example, recent work has proposed two families of policies that leverage heterogeneity at both decision points by querying fixed numbers of "fast" and "slow" servers, then probabilistically choosing whether to assign the job to a fast or a slow server based on the idle/busy statuses of the queried servers [7]. One can also imagine designing new policies within our framework; for example, a policy could query d servers probabilistically in proportion to their speeds—as in BR—and then assign the job to the queried server at which its

[1] Throughout, names and abbreviations of *individual* rules and policies are rendered in sans-serif font; see "Appendix A" for a list of individual rules and policies proposed, studied, and/or referenced in this paper.

expected delay is smallest—as in SED-d (for more details on how such policies fit into our framework, see Sects. 3 and 7).

Our framework is quite general in the space of querying rules it permits: we allow for any querying rule that is *static* (i.e., ignores past querying and assignment decisions) and *symmetric* (i.e., treats servers of the same speed class identically). The BR *querying rule*—viewed separately from the fact that the BR *dispatching policy* from [4] uses JSQ assignment—for example, clearly satisfies these properties. The BR querying rule is a member of what we call the **Independent and Identically Distributed Querying** (**IID**) *family* of querying rules.[2] Each specific policy within this family selects each of the d servers independently according to the same distribution over the server speed classes. That is, the **IID** family of querying rules is parameterized by a set of probabilities that determine the rates at which each server class is queried.

We consider several families of querying rules that satisfy the static and symmetric properties; as is the case for the **IID** family, each family is characterized by its own set of probabilistic parameters that determine how to select the d servers, and different settings for these parameters specify different policies within the family (e.g., one parameter setting of the **IID** querying rule family yields the BR querying rule, as alluded to above). Other examples of querying rule families in the literature include **Single Random Class** (**SRC**) [20], under which a single server class is selected probabilistically for each arriving job and all d queried servers are chosen from that class, and **Deterministic Class Mix** (**DET**) [7], under which the d queried servers always contain a fixed number of servers of each class. We also introduce several new families of querying rules that generalize those in the literature in various ways.

Our framework also permits a wide range of *assignment rules*. For example, included in our framework are assignment rules such as Shortest Expected Delay (SED) and Join the Shortest Queue (JSQ), which when paired with Uniform Querying (UNI)—the querying rule defined in Sect. 3.2 that queries each server with equal probability, regardless of its class—constitute the SED-d and JSQ-d *dispatching policies* as they are typically defined in the literature. The SED assignment rule is especially attractive as it simultaneously incorporates both detailed queue-length information and server class information when making an assignment decision among the queried servers. The potentially powerful rules that make use of both class and queue-length information, such as SED, fall within what we call the **Class and Length Differentiated** (**CLD**) family of assignment rules. Unfortunately, general **CLD** assignment rules (including SED) preclude tractable exact performance analysis. In light of this tractability barrier, we introduce the **Class and Idleness Differentiated** (**CID**) family of assignment rules, a subfamily of **CLD**. The assignment rules in the **CID** family eschew detailed queue length information and make assignment decisions based only on the idle/busy statuses and classes (speeds) of the queried servers. Even with the information limitations imposed by the **CID** family, there is a rich space of reasonable ways to assign jobs among queried servers of different speeds and idle/busy statuses. While it is natural to favor an idle fast server over a slower server—whether busy or idle—it is less obvious whether a busy fast server or an idle slow server is prefer-

[2] Throughout, names and abbreviations of parameterized *families* of rules and policies are rendered in **bold serif font**; see "Appendix A" for a list of families of rules and policies proposed, studied, and/or referenced in this paper.

able; it can even be beneficial to occasionally assign jobs to a busy slow server over a busy fast server. Following our earlier work in [7], we make decisions of this sort probabilistically. As a result, policies within the **CID** family of assignment rules are parameterized by the probabilities with which each queried server class is assigned the arriving job. Specifically, each set of parameters that specifies an assignment rule within **CID** encodes a distribution over the classes for each type of "scenario" the dispatcher may confront—in terms of the speed classes of servers queried and their idle/busy statuses. As we show, unlike the dispatching policies driven by **CLD** assignment, dispatching policies constructed from any static and symmetric querying rule and a **CID** assignment rule are amenable to exact analysis (Sect. 4).

In light of the fact that we can—and do—analyze **CID**-driven dispatching polices, the bulk of this paper (Sects. 4–6) is devoted to the study of families of *dispatching policies* that are formed by combining one of several families of querying rules (e.g., **IID**, **SRC**) with the **CID** family of assignment rules. Each resulting family constitutes (often infinitely) many possible individual dispatching policies, each of which is specified by a different choice of the probabilistic parameters governing the chosen querying and assignment rule families. In Sect. 5, we formulate optimization problems for *jointly* determining the querying and assignment rule parameterizations that yield the lowest mean response time for a given set of system parameters (e.g., arrival rate, server classes, etc.). To the best of our knowledge, this paper is the first to feature a joint-optimization of the querying and assignment decisions across continuous parameter spaces for both rule types; while our earlier work [7] features a joint optimization, that paper considers only the **DET** querying family with only two server classes, which yields at most $|\mathbf{DET}| = d + 1$ possible querying rules. In addition to our allowance for continuous spaces of querying and assignment rules, in this paper we allow for any number of server classes, yielding substantially larger and more complicated optimization problems; for details on the sizes of our optimization problems see Appendix D of [12]. Nonetheless, the problem of selecting an optimal policy from many of the families introduced in this paper is significantly less computationally intensive than the corresponding problem associated with **DET**-based policies, such as those in [7], because the continuous space of our querying rules allows for purely continuous optimization, obviating the need for combinatorial optimization. We discuss practical considerations and present a numerical study of the performance of **CID**-driven dispatching policies in Sect. 6.

Understandably, restricting ourselves to the **CID** assignment rule family leads to sacrifices in performance; one would expect **CLD** assignment rules such as SED to yield lower mean response times when paired with a judiciously chosen querying rule. At the same time, because of the difficulty of finding exact mean response times for **CLD** assignment rules—which make use of both server speed and detailed queue length information—it is also challenging to systematically identify querying rules that perform well in tandem with the **CLD** assignment rules. In Sect. 7, we offer the following heuristic remedy to the problem of finding suitable querying rules to be paired with those assignment rules that are not amenable to tractable analysis: we pair various assignment rules in **CLD** (e.g., SED) with a querying rule that was jointly optimized with a **CID** assignment rule. Simulation results demonstrate that these heuristic dispatching policies tend to perform favorably to other policies—both

those existing in the literature and the **CID**-based policies we study in this paper. Furthermore, our results yield insights about the relative importance of the querying and assignment decisions at different system loads: we observe that at light load the querying decision drives the dispatching policy's performance, whereas at heavy load the assignment decision plays the larger role.

While throughout the paper, we operate under the assumption that job sizes are exponentially distributed, many of our results hold for generally distributed job sizes (see Appendix F of [12] for details). The work presented in this paper is a starting point for the further study of the policies within our framework; to this end, we discuss ample opportunities for future work in Sect. 8.

2 Literature review

In large-scale systems, the power-of-d is the dominant dispatching paradigm; power-of-d policies operate by querying d servers uniformly at random and dispatching an arriving job to one of the queried servers. The best-known policy within this paradigm is Join the Shortest Queue-d (JSQ-d), under which a job is dispatched to the server with the shortest queue among the d queried servers. Response time under JSQ-d has been analyzed, under the assumption of homogeneous servers and exponential service times [19, 32]. JSQ-2 has also been studied in heterogeneous systems with general service times under both the FCFS [11, 40] and Processor Sharing (PS) scheduling rules [20]. Variants of JSQ-d include JSQ(d, T), under which a job is dispatched to a queried server with workload less than a threshold T, and Join the Idle Queue-d (JIQ-d), which is a special case of JSQ(d, T) with $T = 0$ [9]. While power-of-d policies typically are designed for homogeneous systems, several heterogeneity-aware policies akin to JSQ-d also have been proposed. These include Shortest Expected Delay-d (SED-d), which uses server speed information to assign a job to a queried server based on the expected waiting time rather than the number of jobs in the queue, and Balanced Routing (BR), which queries d servers with probabilities proportional to their speeds and then uses JSQ assignment [4]. Other power-of-d-like families of policies that make use of server speed information include **JIQ**-(d_F, d_S) and **JSQ**-(d_F, d_S) [7], as well as the **Hybrid SQ**(2) **Scheme**, which has been studied under the Processor Sharing (PS) scheduling discipline [20]. All of these policies fit within our framework; we will discuss many of them in more detail, in the context of our framework, in the sections that follow.

A different stream of related literature focuses on policies that use information about *all* servers' states when making dispatching decisions; because these policies do not involve querying a subset of the servers, they fall outside of our framework. The most well-known policy in this category is Join the Shortest Queue (JSQ), which is known to minimize mean response time in homogeneous systems with FCFS scheduling, assuming that service times are independent and identically distributed and have non-decreasing hazard rate [35, 38]. Mean response time under JSQ has been analyzed approximately under both FCFS scheduling, assuming exponential service times [21], and PS scheduling, assuming general service times [8]. Join the Idle Queue (JIQ) was proposed as a low-communication alternative to JSQ [16, 34]; again, the analysis

assumes homogeneous servers. More recently, several heterogeneity-aware variants on JSQ and JIQ have been proposed and studied [28, 40]. While some of these policies have been shown to stochastically minimize the queue length distribution in heterogeneous systems [28], this does not imply optimality with respect to mean response time. Indeed, policies within our framework can outperform these heterogeneity-aware policies that use state information from all servers (see, e.g., [7]).

Still other scalable heterogeneity-aware policies have been designed for systems with slightly different modeling assumptions than those we consider in this work. For example, the JFIQ and JFSQ policies were designed for systems in which jobs have locality constraints (i.e., each job is capable of running on only a subset of the servers) [36]. While the *assignment* rules used in these policies are similar to some of the assignment rules that fit within our framework, the JFIQ and JFSQ *dispatching policies* would not be considered part of our framework because they do not involve querying a subset of servers; instead, the dispatcher considers all compatible servers for each arriving job. Similarly, the **Local Shortest Queue** (**LSQ**) family of policies [31] is orthogonal to our work; these policies assume multiple dispatchers, each of which store a local—possibly out of date—view of server states. While some of the policies in the **LSQ** family are quite similar to policies in our framework, the analytical approach and key insights of [31] are fundamentally different from our work because of the use of out of date information.

Another category of heterogeneity-aware dispatching policies that fall outside our framework includes those policies that are designed specifically for small-scale systems. Policies in this category use information about all servers' queue lengths—and sometimes more detailed information—when making dispatching decisions [2, 3, 6, 10, 27, 30]. These policies typically would not be considered scalable and hence are less applicable to the setting we consider in this paper. Some policies, such as Shortest Expected Delay and Generalized Join the Shortest Queue, have well-defined power-of-d variants appropriate for large-scale systems. Thus far, analysis of these policies has focused on systems with only a small number of servers [1, 25, 26, 37]; we consider the power-of-d versions of these policies, which do fall within our framework, in later sections. Further away from our setting is work focusing on the "slow server problem," which asks whether a slow server should be used at all [13–15, 18, 22–24]. These models consider systems with a central queue, and thus, the policies proposed do not apply to our setting.

3 Model and framework

The framework introduced in this paper necessitates a large volume of notation. Throughout the paper, notation is defined when introduced. Additionally, most of the notation in the paper is summarized in "Appendix A."

3.1 Preliminaries

We consider a system with k servers. There are s classes of server speeds,

$$\mathcal{S} \equiv \{1, \ldots, s\}, \tag{1}$$

where the number of class-i servers is k_i; let $q_i \equiv k_i/k$ be the fraction of servers belonging to class i. In the interest of both clarity and tractability, we assume that the size (i.e., service requirement in terms of time) of a job running on a class-i server is exponentially distributed with rate μ_i (for a discussion of generally distributed service times, see Appendix F of [12]). Classes are indexed in decreasing order of speed, i.e., $\mu_1 > \cdots > \mu_s$. We assume that $\sum_{i=1}^{s} \mu_i q_i = 1$. Jobs arrive to the system as a Poisson process with rate λk. Except where stated otherwise, we carry out our analysis in the regime where $k \to \infty$ under the assumption of asymptotic independence (see Sect. 4 for details).

The goal is to minimize the mean response time $\mathbb{E}[T]$, i.e., the end-to-end duration of time from when a job first arrives to the dispatcher until it completes service at one of the servers. Upon a job's arrival, the dispatcher (i) queries a given number ($d \ll k$) of servers according to a *querying rule* and then (ii) sends the job to one of the queried servers according to an *assignment rule*, at which (iii) the job is queued and/or served according to a work-conserving *scheduling rule*. In this paper, we are primarily interested in elaborating on and analyzing the consequences of the first two rule types—querying and assignment; together these two rules constitute the totality of the *dispatching policy*. We denote the dispatching policy that uses querying rule QR and assignment rule AR by $\langle \mathsf{QR}, \mathsf{AR} \rangle$. Our goal is to find dispatching policies (i.e., jointly determine how to query servers and how to assign jobs) that result in low mean response times. While explicitly determining and evaluating the performance of the optimal policy will be prohibitively difficult, we propose some families of rules that are simple to implement and understand alongside techniques for identifying optimal rules within these families given a particular problem instance.

The details of how individual rules function can depend on the parameters of a particular system (i.e., on the number of server classes s, the server speeds $\mu_1, \ldots, \mu_s$, the fraction of the total server count constituting each class, the arrival rate, λ, etc.) and the query count d (which we can take as given). A *family* of (querying or assignment) rules is a collection of individual rules parameterized by a shared set of additional decision variables (e.g., probabilistic parameters indicating which server classes should be queried or which server should be assigned a job given the state of the queried servers). We are interested in rule families insofar as they allow us to optimize over their parameter spaces in order to find the specific rule that minimizes the mean response time $\mathbb{E}[T]$ within that family for a given system parameterization. We note that this optimization is performed once for a given system; the same querying rule and assignment rule are then applied throughout the system's lifetime. Even where optimization is prohibitively intractable, we can still set parameter values heuristically in the hope of finding strong policies among those available within a family.

Throughout the paper, we use the following convention: the abbreviated names of *individual* (querying, assignment, and scheduling) rules and dispatching policies are rendered in a sans-serif font (e.g., a querying rule QR, an assignment rule AR, and a dispatching policy DP), while those of entire *families* of rules and policies are rendered in a bold serif font (e.g., a querying rule family **QRF**, an assignment rule family **ARF**, and a dispatching policy family **DPF**). Often, we will also denote families of dispatching policies by extending our notation for individual dispatching rules $\langle \mathsf{QR}, \mathsf{AR} \rangle$ as follows: for an individual querying rule QR and a family of assignment rules **ARF**, let $\langle \mathsf{QR}, \mathbf{ARF} \rangle \equiv \{\langle \mathsf{QR}, \mathsf{AR} \rangle : \mathsf{AR} \in \mathbf{ARF}\}$ be the family of dispatching policies constructed from the individual querying rule QR in combination with any individual assignment rule AR belonging to the family **ARF**. By analogy, for a querying rule family **QRF** and individual assignment rule AR, let $\langle \mathbf{QRF}, \mathsf{AR} \rangle \equiv \{\langle \mathsf{QR}, \mathsf{AR} \rangle : \mathsf{QR} \in \mathbf{QRF}\}$. When discussing a family of dispatching policies where neither querying nor assignment is restricted to an individual rule, we write $\langle \mathbf{QRF}, \mathbf{ARF} \rangle \equiv \{\langle \mathsf{QR}, \mathsf{AR} \rangle : \mathsf{QR} \in \mathbf{QRF},\ \mathsf{AR} \in \mathbf{ARF}\}$.

We assume throughout that the sizes of specific jobs are unknown until they are completed, and hence, we restrict attention to querying, assignment, and scheduling rules that cannot make use of (i.e., are "blind" to) job size information. We further assume that querying and assignment decisions are made and carried out instantaneously without any overheads; consequently, jobs may not be held at the server for dispatching at some later time. Under the assumption of exponentially distributed job sizes, our analysis and results hold under all work conserving size-blind scheduling rules. Under general service time distributions, this is no longer the case; for a discussion of the interaction between service time distributions and scheduling rules, see Appendix F of [12]).

3.2 Overview of querying rules

When a job arrives, the dispatcher queries d servers at random according to a *querying rule*. Throughout this paper, in the interest of tractability, brevity, and simplicity, we restrict attention to those querying rules that are *static* and *symmetric* (properties that we define below).

Definition 1 A querying rule is *static* if each querying decision is made without reference to any kind of state information, i.e., the set of servers queried upon a job's arrival is chosen independently of all past and future querying and assignment decisions.

Insisting that our querying rules be static is motivated by simplicity and may preclude some superior querying rules: it is conceivable that there would be some benefit in weighting the likelihood that a server is queried in terms of how recently it was queried (or better yet, in terms of how recently it was assigned a job), which is not possible under static querying rules. We note in particular that restricting attention to static querying rules precludes round-robin querying (i.e., the rule where all servers would be put into an ordered list, and one would query by going down the list and querying the next d servers at each arrival, cycling back to the beginning of the list after querying the server at the end of the list). Nevertheless, this restriction comes with an important advantage: static querying rules can be uniquely and unambiguously

described in terms of a probability distribution over the set of all d-tuples of servers. By further imposing that our static querying rules also be symmetric (according to the definition that follows), we can simplify these distributions even further.

Definition 2 A static querying rule is *symmetric* if it is equally likely to query a set of d servers U_1 or U_2 whenever U_1 and U_2 contain the same number of class-i servers for all $i \in \mathcal{S}$.

Essentially, a static symmetric querying rule is one where each query is carried out independently of all others (as with all static querying rules), while no server (respectively, combination of servers) is *ex ante* treated any differently than any other server (respectively, combination of servers) of the same class (respectively, class composition). As with the restriction to static querying rules, requiring that a querying rule be symmetric may preclude superior dispatching policies.

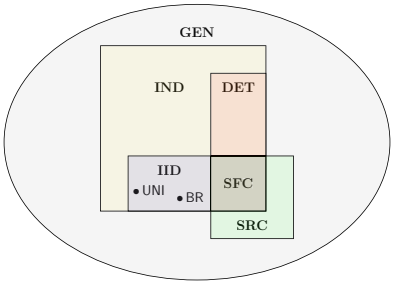
These restrictions motivate the introduction of some additional notation and terminology. Let D_i denote the number of class-i servers in a given query, let $\boldsymbol{D} \equiv (D_1, \ldots, D_s)$ denote the *class mix*, let d_i and $\boldsymbol{d} \equiv (d_1, \ldots, d_s)$ denote the realizations of the random variable D_i and the random vector $\boldsymbol{D}$, respectively, and finally let

$$\mathcal{D} \equiv \{\boldsymbol{d} : d_1 + \cdots + d_s = d\} \tag{2}$$

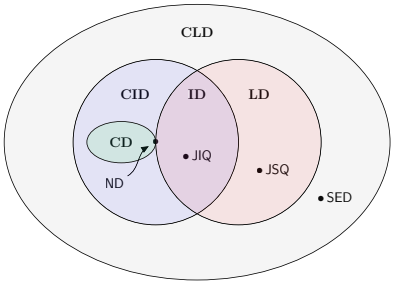
be the set of all possible class mixes $\boldsymbol{d}$ (involving exactly d servers). Observe that any static symmetric querying rule can be uniquely and unambiguously defined in terms of a distribution over the set of all possible query mixes, $\mathcal{D}$. Formally, a querying rule is given by a function $p : \mathcal{D} \to [0, 1]$ satisfying $\sum_{\boldsymbol{d} \in \mathcal{D}} p(\boldsymbol{d}) = 1$. The querying rule selects servers so that $\mathbb{P}(\boldsymbol{D} = \boldsymbol{d}) = p(\boldsymbol{d})$.

We conclude this subsection by introducing the main families of querying rules—in addition to two individual rules—studied in this paper, taking the query count d as given:

- The **General Class Mix** (**GEN**) family consists of all (and only those) querying rules that are static and symmetric. Note that such querying rules are equally likely to query any combination of d servers that constitute the same query mix $\boldsymbol{d} \in \mathcal{D}$. The following families are all subsets of **GEN**.
- The **Independent Querying** (**IND**) family consists of those querying rules in **GEN** where each of the d servers to be queried is chosen independently according to some (but not necessarily the same) probability distribution over the set of classes $\mathcal{S}$. Consider the following example of a policy in **IND** when $s = d = 3$: always query at least one class-1 server, exactly one class-2 server, and either an additional class-1 server or a class-3 server with equal probability. Note that we ignore the possibility of a single server being queried more than once, as we are primarily concerned with the setting where the number of servers in each class $k_i \to \infty$.
- The **Independent and Identically Distributed Querying** (**IID**) family consists of those querying rules in **GEN** where each of the d servers to be queried are chosen independently according to *the same* probability distribution over the set of classes $\mathcal{S}$, and hence, the random vector $\boldsymbol{D}$ is drawn from a multinomial distribution under **IID** querying. **IID** is a subfamily of **IND**.



(a) Querying rule families. Note that **SFC** is the intersection of any two of the **IID**, **DET**, and **SRC** families. Moreover, we have $\mathbf{SFC} = \mathbf{IND} \cap \mathbf{SRC}$.



(b) Assignment rule families. Note that we have $\mathbf{CD} \cap \mathbf{ID} = \{\mathsf{ND}\}$.

Fig. 1 Set inclusion diagrams for the (a) querying rule families and (b) assignment rule families discussed in this paper. In both diagrams, rule families are shown as regions and individual rules are shown as points

- The **Deterministic Class Mix** (**DET**) family consists of those querying rules in **GEN** that always query the same class mix for some fixed class mix $\boldsymbol{d} \in \mathcal{D}$. **DET** is a subfamily of **IND**.
- The **Single Random Class** (**SRC**) family consists of those querying rules in **GEN** that select one of the s server types according to some probability distribution over the set of classes $\mathcal{S}$ and then queries d servers all of that class.
- The **Single Fixed Class** (**SFC**) family consists of those querying rules that always query d class-i servers for some fixed class $i \in \mathcal{S}$. Such rules essentially discard all servers except those of the chosen class, rendering the system homogeneous. The **SFC** family consists of only s querying rules and is precisely the intersection of the **IID** and **DET** families as well as the intersection of **SRC** and any (nonzero) number of the **IND**, **IID** and **DET** families.
- The Uniform Querying (UNI) rule is equally likely to query any combination of d servers. To elaborate, the UNI querying rule is a member of the **IID** family where each of the d servers queried is a class-i server with a probability equal to the fraction of servers that belong to class i (i.e., with probability q_i).
- The Balanced Routing (BR) rule queries d servers independently, with the probability that any given server is queried being proportional to its speed. To elaborate, the BR querying rule is a member of the **IID** family where each of the d servers queried is a class-i server with a probability equal to the fraction of the total system-wide service capacity provided by class-i servers (i.e., with probability $\mu_i q_i$).

Remark 1 In [4], Balanced Routing referred to what would be understood in our framework as the *dispatching policy* constructed from (i) what we call the Balanced Routing *querying* rule and (ii) the Join the Shortest Queue *assignment* rule. From this point forward, in our paper we use the acronym BR to refer to the Balanced Routing querying rule and not the dispatching policy.

Figure 1a depicts the set inclusion relationships between querying rule families and individual querying rules described above.

3.3 Overview of assignment rules

Once a set of servers has been queried, the job is assigned to one of these servers according to an *assignment rule*, which specifies a distribution over the servers queried. Our assignment rules are allowed to depend on *state* information, consisting of knowledge of each queried server's class (and hence, their associated μ_i and q_i values) and knowledge of the queue length—including the job or jobs in service, if any—at each queried server. We restrict attention to assignment rules that satisfy restrictions analogous to those adopted for our querying rules.

Definition 3 An assignment rule is *static* if each assignment decision is made without *direct* regard to past querying or assignment decisions (although such decisions can impact the state at a server, which assignment rules may use).

Remark 2 More formally, let $\boldsymbol{X}_t$ denote the state of the *entire* system at the time of the t-th assignment (including the queue length at and class of each of the servers in the system) and let $\vec{\boldsymbol{A}}_t$ denote the result of the t-th query (by analogy with the notation $\vec{\boldsymbol{A}}$, which we introduce in Sect. 7.1). Let $\mathscr{F}_t$ denote the natural filtration of $\{\boldsymbol{X}_t, \vec{\boldsymbol{A}}_t\}$. An assignment policy is *static* if the (potentially random) assignment choice given $\vec{\boldsymbol{A}}_t$ is the same as the assignment choice given $\mathscr{F}_t$.

Definition 4 A static assignment rule is *symmetric* if it does not use information about the specific *identities* of the queried servers and can only use their *state* information. That is, given a set of queried servers with identical states, the job is equally likely to be assigned to any one of those servers and the probability with which that job is assigned to one of those servers depends only on the state (and not the identities) of those servers and the states (and not the identities) of the other queried servers.

We consider six families of static and symmetric assignment rules. We proceed to describe these families, which differ from one another in the ways they can differentiate the states of the queried servers for the purpose of making assignment decisions:

- The Non-Differentiated (ND) assignment rule cannot differentiate between server states. This is equivalent to uniform assignment among the servers in the query. We note that using the ND assignment rule is antithetical to the purpose of the power-of-d paradigm, as an equivalent dispatching policy can always be implemented with $d = 1$.
- Assignment rules in the **Class Differentiated** (**CD**) family may differentiate between server states only on the basis of class information.
- Assignment rules in the **Idleness Differentiated** (**ID**) family may differentiate between server states only on the basis of idleness information, e.g., Join the Idle Queue (JIQ).
- Assignment rules in the **Length Differentiated** (**LD**) family may differentiate between server states only on the basis of queue-length information, e.g., Join the Shortest Queue (JSQ).
- Assignment rules in the **Class and Idleness Differentiated** (**CID**) family may differentiate between server states only on the basis of class and idleness information.

– Assignment rules in the **Class and Length Differentiated** (**CLD**) family may differentiate between server states on the basis of both class and queue-length information, e.g., Shortest Expected Delay (SED).

As shown in Fig. 1b, the **CLD** family includes all of the other assignment rule families under consideration. Naturally, among the dispatching policies that we consider, those that achieve the best performance (i.e., the lowest mean response time) necessarily make use of the querying rules in the **CLD** family. Specific policies that belong to only the **CLD** family (among the six mentioned above) may be amenable to numerical response time approximation. However, the curse of dimensionality frequently obstructs the use of optimization techniques for the systematic discovery of strong-performing policies within this family. Meanwhile, the study of the **LD** family can exhibit complications similar to those exhibited by **CLD**, while lacking the advantage of exploiting heterogeneity to obtain low response times. Therefore, **CID**—which subsumes **CD** and **ID**—emerges as the richest family under consideration that is amenable to analysis, so we devote Sects. 4–6 to exploring this family of assignment rules (in conjunction with the various families of querying rules introduced in Sect. 3.2). We explore the wider **CLD** family of assignment rules in Sect. 7, where we leverage our extensive study of **CID**-driven dispatching policies (presented in the aforementioned sections) to find superior policies with assignment rules in **CLD**.

4 Analysis of Class and Idleness Differentiated assignment rules

In this section, we examine the **CID** family of assignment rules in detail. We provide a formal presentation of this family (Sect. 4.1), prove stability results (Sect. 4.2), and present an analysis of the mean response time of the ⟨**GEN**, **CID**⟩ dispatching policies (Sect. 4.3).

4.1 Formal presentation of the Class and Idleness Differentiated family of assignment rules

Assignment rules in the **CID** family are—as the family's name clearly suggests—length-blind but idle-aware, i.e., such an assignment rule can observe and make assignment decisions based on the idle/busy status of each of the queried servers, but it cannot observe the queue length at each busy server (of course, the queue length at each idle server must be 0). By eschewing examining detailed queue length information, we facilitate tractable analysis. Meanwhile, idle-awareness motivates the introduction of some new notation: we encode the idle/busy statuses of the queried servers by $\boldsymbol{a} \equiv (a_1, \ldots, a_s)$, where a_i is the number of *idle* class-i servers among the d_i queried. The set of all possible $\boldsymbol{a}$ vectors is given by $\mathcal{A} \equiv \{\boldsymbol{a} \colon a_1 + \cdots + a_s \leq d\}$. Note that a_i and $\boldsymbol{a}$ are realizations of the random variable A_i and the random vector $\boldsymbol{A}$ (which are defined analogously to D_i and $\boldsymbol{D}$), respectively.

Formally, an assignment rule is given by a family of functions $\alpha_i : \mathcal{A} \times \mathcal{D} \to [0, 1]$ parameterized by $i \in \mathcal{S}$. For all $\boldsymbol{a} \in \mathcal{A}$ and $\boldsymbol{d} \in \mathcal{D}$ such that $\boldsymbol{a} \leq \boldsymbol{d}$ (element-wise) these families must satisfy $\sum_{i \in \mathcal{S}} \alpha_i(\boldsymbol{a}, \boldsymbol{d}) = 1$ and $\alpha_i(\boldsymbol{a}, \boldsymbol{d}) = 0$ if $d_i = 0$. Given such

a family of functions (together with a query resulting in vectors $\boldsymbol{a} \in \mathcal{A}$ and $\boldsymbol{d} \in \mathcal{D}$) the dispatcher sends the job to a class-i server with probability $\alpha_i(\boldsymbol{a}, \boldsymbol{d})$. At this point, we assign to an idle class-i server (if possible) or a busy class-i server (otherwise), chosen uniformly at random.

We prune the set of assignment rules by avoiding rules that allow assignment to a slower server when a *faster idle* server has been queried. That is, $\alpha_i(\boldsymbol{a}, \boldsymbol{d}) = 0$ whenever there is a class $j < i$ such that $a_j \geq 1$. Moreover, whenever $\boldsymbol{a} \neq \boldsymbol{0}$, the value of $\alpha_i(\boldsymbol{a}, \boldsymbol{d})$ depends only on the realized value of the random variable $J \equiv \min\{j \in \mathcal{S} \colon A_j > 0\}$—the class of the *fastest idle queried server*—and on $\boldsymbol{d}$ (specifically, on the realization of the random set $\{j < J \colon d_j > 0\}$). For notational convenience, we take $\min \emptyset \equiv s+1$, so that J is defined on $\bar{\mathcal{S}} \equiv \mathcal{S} \cup \{s+1\} = \{0, 1, 2, \ldots, s, s+1\}$ and $J = s+1$ when all queried servers are busy, in which case there is no idle server and we can consider the (nonexistent) fastest idle queried server as belonging to (the nonexistent) class $s+1$. This structure allows us to introduce the following abuse of notation that will facilitate the discussion of our analysis: $\alpha_i(j, \boldsymbol{d}) \equiv \alpha_i(\boldsymbol{a}, \boldsymbol{d})$ for all $j \in \bar{\mathcal{S}}$ and $\boldsymbol{a} \in \mathcal{A}$ such that $J = j$ whenever $\boldsymbol{A} = \boldsymbol{a}$, i.e., such that $j = \min\{j' \in \mathcal{S} \colon a_{j'} > 0\}$. Note that as a consequence of this notation, we have $\alpha_i(s+1, \boldsymbol{d}) = \alpha_i(\boldsymbol{0}, \boldsymbol{d})$. Further note that we must have $\alpha_i(j, \boldsymbol{d}) = 0$ whenever $d_i = 0$ (we cannot send the job to a server that was not queried) and moreover we set $\alpha_i(j, \boldsymbol{d}) = 0$ whenever $d_j = 0$ and $j \neq s+1$ (the fastest queried idle server must of course be queried).

4.2 Stability

In this section, we identify necessary and sufficient conditions for the existence of a stable dispatching policy within the $\langle \mathbf{QRF}, \mathbf{CID} \rangle$ family for the various families of querying rules, **QRF**, presented in Sect. 3.2. We say that the system is stable if the underlying Markov chain is positive recurrent. This is a necessary condition for achieving finite mean response time. In order to establish stability, it is sufficient to show that, when busy, each server experiences an average arrival rate that is less than its service rate. This implies that the mean time between visits to the idle state is finite, and hence that the underlying Markov chain is positive recurrent as required. Let λ_i^{B} denote the average arrival rate to a busy class-i server.

Definition 5 The system is stable if, for all server classes $i \in \mathcal{S}$, we have $\lambda_i^{\mathrm{B}} < \mu_i$.

Proposition 1 *Recalling that λ is the average arrival rate per server (i.e., λk is the total arrival rate to the system), the following necessary and sufficient conditions for stability hold:*

1. *There exists a policy in the $\langle \mathbf{SRC}, \mathbf{CID} \rangle$ family such that the system is stable if and only if $\lambda < 1$.*
2. *There exists a policy in the $\langle \mathbf{SFC}, \mathbf{CID} \rangle$ family such that the system is stable if and only if $\lambda < \max_j \mu_j q_j$.*
3. *Consider a dispatching policy in the $\langle \mathbf{DET}, \mathbf{CID} \rangle$ family, where the query mix is always $\boldsymbol{d}$ (note that each individual policy within $\langle \mathbf{DET}, \mathbf{CID} \rangle$ has only one query mix). The system is stable if and only if $\lambda < \sum_{i \colon d_i > 0} \mu_i q_i$.*

4. *Under* $\langle BR, \textbf{CID}\rangle$*, the system is stable if and only if* $\lambda < 1$.
5. *There exists a policy in* $\langle \textbf{IID}, \textbf{CID}\rangle$ *such that the system is stable if and only if* $\lambda < 1$.
6. *There exists a policy in each of* $\langle \textbf{IND}, \textbf{CID}\rangle$ *and* $\langle \textbf{GEN}, \textbf{CID}\rangle$ *such that the system is stable if and only if* $\lambda < 1$.

Proof We prove each case separately:

1. Consider a querying rule in **SRC** where the probability that all queried servers are of class i is given by $\mu_i q_i$. Then, by Poisson splitting, the class-i servers act like a homogeneous system, independent of all other server classes, with a total arrival rate $\lambda k \mu_i q_i = \lambda k_i \mu_i$. Given that only class-$i$ servers are present in the query, the **CID** assignment rule will assign the arriving job to an idle server, if one is present in the query, and a busy server chosen uniformly at random (among the servers in the query) if not. This assignment rule is symmetric among class-i servers, and so the arrival rate to an individual class-i server is $\lambda \mu_i$, which is less than μ_i, ensuring the stability of the system, provided that $\lambda < 1$.
2. **SFC** effectively throws out all server classes except one, which we will call class i; by a similar argument as in the proof for **SRC**, the class-i subsystem will remain stable provided that $\lambda < \mu_i q_i$. Then, the largest stability region is achieved by selecting the server class with the largest total capacity.
3. Given that we always query according to some fixed query mix $\boldsymbol{d} \in \mathcal{D}$, construct an assignment rule in **CID** (yielding a dispatching policy in $\langle \textbf{DET}, \textbf{CID}\rangle$) under which, for all $i \in \mathcal{S}$ such that $d_i > 0$, the job is dispatched to a queried class-i server (chosen uniformly at random without considering any idle/busy statuses) with probability $\mu_i q_i \Big/ \sum_{j:d_j>0} \mu_j q_j$ (note that this assignment rule is a member of $\textbf{CD} \subseteq \textbf{CID}$ as it ignores idle/busy statuses, and therefore does not adhere to our pruning of the space of assignment rules). Then, the total arrival rate to class-i servers is

$$\lambda k \cdot \frac{\mu_i q_i}{\sum_{j:d_j>0} \mu_j q_j} = \mu_i k_i \cdot \frac{\lambda}{\sum_{j:d_j>0} \mu_j q_j},$$

   which is less than $\mu_i k_i$, ensuring stability of the class-i servers, provided that $\lambda < \sum_{j:d_j>0} \mu_j q_j$.
4. From [4], we have that $\langle \mathsf{BR}, \mathsf{JSQ}\rangle$ is stable if and only if $\lambda < 1$. For all $(i, \boldsymbol{d}) \in \mathcal{S} \times \mathcal{D}$ let $\beta_i(\boldsymbol{d})$ denote the probability that an arriving job is sent to a class-i server under $\langle \mathsf{BR}, \mathsf{JSQ}\rangle$, given that the query mix is $\boldsymbol{d}$ (i.e., $\beta_i(\boldsymbol{d})$ is the probability that the shortest queue is at a class-i server, given query mix $\boldsymbol{d}$). Now form a policy in the family $\langle \mathsf{BR}, \textbf{CID}\rangle$ by sending the job to a queried class-i server (chosen uniformly at random without considering any idle/busy statuses) with probability $\beta_i(\boldsymbol{d})$ for all $(i, \boldsymbol{d}) \in \mathcal{S} \times \mathcal{D}$ (note that this assignment rule is a member of $\textbf{CD} \subseteq \textbf{CID}$ as it ignores idle/busy statuses, and therefore does not adhere to our pruning of the space of assignment rules). The probability that an arriving job is dispatched to a

class-i server is the same under this newly defined policy in $\langle \mathsf{BR}, \mathbf{CID} \rangle$ as under $\langle \mathsf{BR}, \mathsf{JSQ} \rangle$; the only difference is that now all jobs can be viewed as being routed entirely probabilistically. This will not change the stability region, as λ_i^{B} remains unchanged for all $i \in \mathcal{S}$.

5. This follows from the stability condition for $\langle \mathsf{BR}, \mathbf{CID} \rangle$, which is a member of $\langle \mathbf{IID}, \mathbf{CID} \rangle$.
6. This follows from item 5 above and the fact that $\langle \mathbf{IID}, \mathbf{CID} \rangle \subseteq \langle \mathbf{IND}, \mathbf{CID} \rangle \subseteq \langle \mathbf{GEN}, \mathbf{CID} \rangle$. □

Note that in proving the existence of a stable dispatching policy in the $\langle \mathbf{DET}, \mathbf{CID} \rangle$ and $\langle \mathsf{BR}, \mathbf{CID} \rangle$ families (items 3 and 4 of Proposition 1, respectively), we constructed stable dispatching policies where the assignment rules were members of $\mathbf{CD} \subseteq \mathbf{CID}$, and hence, did not adhere to our pruning rules. It is not hard to modify these policies to also prove the existence of stable dispatching policies within these families that make use of idle/busy statuses and adhere to our pruning rules. Consider the simple modification where whenever the original policy would assign the job to a server that is slower than the fastest idle server (or to a busy server of the same speed), instead assign the job to the fastest idle server (if there is more than one fastest idle server, assign the job to one of them chosen uniformly at random). This modification decreases the arrival rate to busy servers and increases the arrival rate to idle servers, which cannot destabilize the system.

We also present the following result, which amounts to a necessary condition for stability under the UNI querying rule and any assignment rule:

Proposition 2 *For any dispatching policy using the UNI querying rule, the system is unstable if there exists a server class $i \in \mathcal{S}$ such that $\lambda > \mu_i / q_i^{d-1}$.*

Proof Under UNI, a query mix consists of only class-i servers—and hence, the arriving job *must* be dispatched to a class-i server under any assignment rule—with probability q_i^d. The total arrival rate to the class-i subsystem is then greater than or equal to $\lambda k q_i^d$. The system is unstable if this total arrival rate is greater than the capacity of the class-i subsystem, i.e., if $\lambda k q_i^d > \mu_i k_i$, or, equivalently, if $\lambda > \mu_i / q_i^{d-1}$. □

4.3 Mean response time analysis

We proceed to present a procedure for determining the mean response time $\mathbb{E}[T]$ under $\langle \mathsf{QR}, \mathsf{AR} \rangle$ for any static symmetric querying rule QR (i.e., any $\mathsf{QR} \in \mathbf{GEN}$) and any $\mathsf{AR} \in \mathbf{CID}$ that yield a stable system.

We carry out all analysis in steady-state and rely on mean-field theory. We let $k \to \infty$, holding q_i fixed for all $i \in \mathcal{S}$; consequently, we also have $k_i \to \infty$ for all $i \in \mathcal{S}$. We further assume that *asymptotic independence* holds in this limiting regime, meaning that (i) the states of (i.e., the number of jobs at) all servers are independent, and (ii) all servers of the same class behave stochastically identically (see Appendix B of [12] for simulation evidence in support of this assumption). With the asymptotic independence assumption in place, we now find the overall mean response time as follows:

Proposition 3 *Let λ_i^{I} and λ_i^{B} denote, respectively, the arrival rates to idle and busy class-i servers. Then, the overall system mean response time is*

$$\mathbb{E}[T] = \frac{1}{\lambda} \sum_{i=1}^{s} q_i \left(\frac{(1-\rho_i)\lambda_i^{\mathrm{I}} + \rho_i \lambda_i^{\mathrm{B}}}{\mu_i - \lambda_i^{\mathrm{B}}} \right), \tag{3}$$

where ρ_i is the fraction of time that a class-i server is busy, given by

$$\rho_i \equiv \frac{\lambda_i^{\mathrm{I}}}{\mu_i - \lambda_i^{\mathrm{B}} + \lambda_i^{\mathrm{I}}}. \tag{4}$$

Proof First observe that under our querying and assignment rules, servers of the same class are equally likely to be queried and, within a class, servers with the same idle/busy status are equally likely to be assigned a job. Hence, by Poisson splitting, it follows that (for any $i \in \mathcal{S}$) each class-i server experiences status-dependent Poisson arrivals with rate λ_i^{I} when idle and rate λ_i^{B} when busy.

Now observe that each class-i server, when busy, operates exactly like a standard M/M/1 system (under the chosen work-conserving scheduling rule) with arrival rate λ_i^{B} and service rate μ_i. Since, by virtue of their own presence, jobs experience only busy systems, the mean response time experienced by jobs at a class-i server—which we denote by $\mathbb{E}[T_i]$—is $1/(\mu_i - \lambda_i^{\mathrm{B}})$. Furthermore, standard M/M/1 busy period analysis gives the expected time of the busy period duration at a class-i server as $\mathbb{E}[B_i] \equiv 1/(\mu_i - \lambda_i^{\mathrm{B}})$; we note that the standard analysis of the M/M/1 queueing system also tells us that while $\mathbb{E}[B_i] = \mathbb{E}[T_i]$, B_i and T_i are *not* identically distributed.

Applying the renewal reward theorem immediately yields that ρ_i (the fraction of time that a class-i server is busy) is as given in Eq. 5 as claimed:

$$\rho_i = \frac{\mathbb{E}[B_i]}{1/\lambda_i^{\mathrm{I}} + \mathbb{E}[B_i]} = \frac{\lambda_i^{\mathrm{I}}}{\mu_i - \lambda_i^{\mathrm{B}} + \lambda_i^{\mathrm{I}}}. \tag{5}$$

Finally, we find the system's overall mean response time by taking a weighted average of the mean response times at each server class. Let $\lambda_i \equiv (1-\rho_i)\lambda_i^{\mathrm{I}} + \rho_i \lambda_i^{\mathrm{B}}$ denote the average arrival rate experienced by a class-i server. Recalling that $q_i = k_i/k$, it follows that the proportion of jobs that are sent to a class-i server is $k_i \lambda_i/(k\lambda) = q_i \lambda_i/\lambda$, and hence

$$\mathbb{E}[T] = \sum_{i=1}^{s} \left(\frac{q_i \lambda_i}{\lambda} \right) \mathbb{E}[T_i] = \frac{1}{\lambda} \sum_{i=1}^{s} q_i \left(\frac{(1-\rho_i)\lambda_i^{\mathrm{I}} + \rho_i \lambda_i^{\mathrm{B}}}{\mu_i - \lambda_i^{\mathrm{B}}} \right), \tag{6}$$

which completes the proof.

□

Remark 3 Note that while *mean* response times are insensitive to the choice of (size-blind) scheduling rule, the distribution (and higher moments) of response time do not exhibit this insensitivity. The same method presented in this section can also allow one

to readily obtain the Laplace transform of response time under many work-conserving scheduling rules. For example, under first come first served (FCFS) scheduling one could use the result that $\widetilde{T}_i(w) = (\mu_i - \lambda_i)/(\mu_i - \lambda_i + w)$ for an M/M/1/FCFS with arrival and service rates λ_i and μ_i, respectively, to obtain the overall transform of response time $\widetilde{T}(w)$.

In order to use Proposition 3 to determine $\mathbb{E}[T]$ values, we must be able to compute the arrival rates λ_i^{I} and λ_i^{B} for each $i \in \mathcal{S}$. The following notation will prove useful in expressing these rates: for all $i \in \bar{\mathcal{S}} \equiv \{1, 2, \ldots, s+1\}$ and $\boldsymbol{d} \in \mathcal{D}$, we let $b_i(\boldsymbol{d})$ denote the probability that all queried servers that are faster than those in class i are busy (i.e., all queried servers with classes in $\{1, 2, \ldots, i-1\}$ are busy). It immediately follows that

$$b_i(\boldsymbol{d}) \equiv \mathbb{P}(A_1 = \cdots = A_{i-1} = 0 | \boldsymbol{D} = \boldsymbol{d}) = \prod_{\ell=1}^{i-1} \rho_\ell^{d_\ell}. \tag{7}$$

Remark 4 Note that for all $\boldsymbol{d} \in \mathcal{D}$, we have $b_1(\boldsymbol{d}) = 1$ as it is vacuously true that all queried servers faster than server 1 are busy as no such servers exist. Moreover, $b_{s+1}(\boldsymbol{d})$ denotes the probability that all queried servers are busy given that $\boldsymbol{D} = \boldsymbol{d}$.

In the following theorem, we present a pair of equations (parameterized by $i \in \mathcal{S}$) for λ_i^{I} and λ_i^{B}.

Theorem 1 *For all $i \in \mathcal{S}$, the arrival rates to idle and busy class-i servers (i.e., λ_i^{I} and λ_i^{B}, respectively), satisfy*

$$\lambda_i^{\mathrm{I}} = \frac{\lambda}{q_i} \sum_{\boldsymbol{d} \in \mathcal{D}} \left\{ d_i b_i(\boldsymbol{d}) p(\boldsymbol{d}) \alpha_i(i, \boldsymbol{d}) \sum_{a_i=1}^{d_i} \binom{d_i - 1}{a_i - 1} \frac{(1-\rho_i)^{a_i - 1} \rho_i^{d_i - a_i}}{a_i} \right\} \tag{8}$$

$$\lambda_i^{\mathrm{B}} = \frac{\lambda}{q_i \rho_i} \sum_{\boldsymbol{d} \in \mathcal{D}} \left\{ p(\boldsymbol{d}) \sum_{j=i+1}^{s+1} b_j(\boldsymbol{d}) \left(1 - \rho_j^{d_j}\right) \alpha_i(j, \boldsymbol{d}) \right\}, \tag{9}$$

where we use the abuse of notation $\rho_{s+1}^{d_{s+1}} \equiv 0$.

Theorem 1 yields $2s$ equations, which we can solve as a system for the $2s$ unknowns $\{\lambda_i^{\mathrm{I}}\}_{i \in \mathcal{S}}$ and $\{\lambda_i^{\mathrm{B}}\}_{i \in \mathcal{S}}$, where we take $\{\rho_i\}_{i=1}^{s}$ and $\{b_i(\boldsymbol{d})\}_{i=1}^{s+1}$ to be as defined by Equations (5) and (7), respectively. With the λ_i^{I} and λ_i^{B} (and consequently, the ρ_i) values determined for all $i \in \mathcal{S}$, we can then compute $\mathbb{E}[T]$ directly from Eq. (6), completing our analysis.

The rest of this section is devoted to proving Theorem 1, by way of three lemmas. These lemmas will be concerned with the quantities $r_i^{\mathrm{I}}(\boldsymbol{d})$ and $r_i^{\mathrm{B}}(\boldsymbol{d})$, defined for all $i \in \mathcal{S}$ as follows: for all $\boldsymbol{d} \in \mathcal{D}$ for which $d_i > 0$, $r_i^{\mathrm{I}}(\boldsymbol{d})$ (respectively, $r_i^{\mathrm{B}}(\boldsymbol{d})$) denotes the probability that the job is assigned to the tagged (class-i) server under query mix $\boldsymbol{d}$ given that the tagged server is queried and idle (respectively, busy). Meanwhile, for all $\boldsymbol{d} \in \mathcal{D}$ for which $d_i = 0$, we adopt the convention where $r_i^{\mathrm{I}}(\boldsymbol{d}) \equiv 0$ and $r_i^{\mathrm{B}}(\boldsymbol{d}) \equiv 0$.

Lemma 1 *The arrival rates λ_i^{I} and λ_i^{B} are given by:*

$$\lambda_i^{\mathrm{I}} = \frac{\lambda}{q_i} \sum_{\boldsymbol{d} \in \mathcal{D}} d_i \, p(\boldsymbol{d}) r_i^{\mathrm{I}}(\boldsymbol{d}) \tag{10}$$

$$\lambda_i^{\mathrm{B}} = \frac{\lambda}{q_i} \sum_{\boldsymbol{d} \in \mathcal{D}} d_i \, p(\boldsymbol{d}) r_i^{\mathrm{B}}(\boldsymbol{d}). \tag{11}$$

Proof Recall that the rate at which the tagged server is queried does not depend on its idle/busy status. Given query mix $\boldsymbol{d}$, the probability that the query includes the tagged server is d_i/k_i (by symmetry). Because a query is of mix $\boldsymbol{d}$ with probability $p(\boldsymbol{d}) = \mathbb{P}(\boldsymbol{D} = \boldsymbol{d})$, the tagged server is queried at rate

$$\lambda k \sum_{\boldsymbol{d} \in \mathcal{D}} \left(\frac{d_i}{k_i} \right) p(\boldsymbol{d}) = \frac{\lambda}{q_i} \sum_{\boldsymbol{d} \in \mathcal{D}} d_i \, p(\boldsymbol{d}).$$

Of course, the tagged server's presence in the query does not guarantee that the job will be assigned to it. The arrival rate *from queries with mix* $\boldsymbol{d}$ observed by the tagged server when it is idle is

$$\lambda k \left(\frac{d_i}{k_i} \right) p(\boldsymbol{d}) r_i^{\mathrm{I}}(\boldsymbol{d}) = \left(\frac{\lambda}{q_i} \right) d_i \, p(\boldsymbol{d}) r_i^{\mathrm{I}}(\boldsymbol{d}),$$

with the analogous expression holding when the tagged server is busy. It follows that the overall arrival rates to an idle and busy class-i server (i.e., λ_i^{I} and λ_i^{B}, respectively) are as claimed. □

Lemma 2 *For all $i \in \mathcal{S}$ and all $\boldsymbol{d} \in \mathcal{D}$ such that $d_i > 0$, the probability that the job is assigned to the tagged class-i server under query mix $\boldsymbol{d}$ given that the tagged server is queried and idle is*

$$r_i^{\mathrm{I}}(\boldsymbol{d}) = b_i(\boldsymbol{d}) \alpha_i(i, \boldsymbol{d}) \sum_{a_i=1}^{d_i} \binom{d_i - 1}{a_i - 1} \frac{(1-\rho_i)^{a_i - 1} \rho_i^{d_i - a_i}}{a_i}. \tag{12}$$

Proof Observe that since we are assuming that the assignment policy $\mathsf{AR} \in \mathbf{CID}$, the job can be assigned to the tagged server only if all faster servers in the query are busy, which occurs with probability $b_i(\boldsymbol{d})$ (see Equation 4.3 for details) for a given query mix $\boldsymbol{d} \in \mathcal{D}$. If this is the case, then with probability $\alpha_i(i, \boldsymbol{d})$ the job is assigned to an idle class-i server chosen uniformly at random; hence, the tagged server is selected among the a_i idle class-i servers with probability $1/a_i$. Enumerating over all possible cases of $A_i = a_i$ when the tagged class-i server is idle, we find the probability that the tagged server is assigned the job when queried with mix $\boldsymbol{d}$:

$$r_i^{\mathrm{I}}(\boldsymbol{d}) = b_i(\boldsymbol{d}) \alpha_i(i, \boldsymbol{d}) \sum_{a_i=1}^{d_i} \frac{\mathbb{P}(A_i = a_i | \boldsymbol{D} = \boldsymbol{d}, \text{tagged class-}i \text{ server is idle})}{a_i}$$

$$= b_i(\boldsymbol{d})\alpha_i(i,\boldsymbol{d}) \sum_{a_i=1}^{d_i} \binom{d_i-1}{a_i-1} \frac{(1-\rho_i)^{a_i-1}\rho_i^{d_i-a_i}}{a_i}, \tag{13}$$

where the latter equality follows from the fact that $A_i \geq 1$ when the tagged class-i server is idle, and so

$$\begin{aligned}(A_i|\boldsymbol{D}=\boldsymbol{d}&, \text{tagged class-}i\text{ server is idle})\\ &\sim (A_i|\boldsymbol{D}=\boldsymbol{d}, A_i \geq 1) \sim \text{Binomial}(d_i-1, 1-\rho_i)+1,\end{aligned}$$

which is in turn a consequence of our asymptotic independence assumption. □

Lemma 3 *For all $i \in \mathcal{S}$ and all $\boldsymbol{d} \in \mathcal{D}$ such that $d_i > 0$, the probability that the job is assigned to the tagged class-i server under query mix $\boldsymbol{d}$ given that the tagged server is queried and busy is*

$$r_i^{\mathrm{B}}(\boldsymbol{d}) = \frac{1}{d_i\rho_i} \sum_{j=i+1}^{s+1} b_j(\boldsymbol{d}) \left(1-\rho_j^{d_j}\right) \alpha_i(j,\boldsymbol{d}), \tag{14}$$

where (as in the statement of Theorem 1) we use the abuse of notation $\rho_{s+1}^{d_{s+1}} \equiv 0$.

Proof We determine $r_i^{\mathrm{B}}(\boldsymbol{d})$ by conditioning on the random variable J, denoting the class of the fastest idle queried server (see Sect. 4.1 for details). Recall that $J \equiv \min\{j \in \mathcal{S}\colon A_j > 0\}$, where we take $\min\emptyset \equiv s+1$, so that $J = s+1$ whenever all servers are busy. Letting $r_i^{\mathrm{B}}(\boldsymbol{d}|J=j)$ denote the probability that the job is assigned to the tagged (class-i) server under query mix $\boldsymbol{d}$ given that $J=j$ and the tagged server is queried and busy, the law of total probability yields

$$r_i^{\mathrm{B}}(\boldsymbol{d}) = \sum_{j=1}^{s+1} r_i^{\mathrm{B}}(\boldsymbol{d}|J=j) \cdot \mathbb{P}(J=j|\boldsymbol{D}=\boldsymbol{d}, \text{tagged class-}i\text{ server is busy}). \tag{15}$$

In order to compute $r_i^{\mathrm{B}}(\boldsymbol{d})$, we first observe that, for all $j \in \bar{\mathcal{S}}$, the job is assigned to some class-i server with probability $\alpha_i(j,\boldsymbol{d})$ (recall that $\alpha_i(s+1,\boldsymbol{d}) \equiv \alpha_i(\boldsymbol{0},\boldsymbol{d})$ in our abuse of notation), and hence the probability that the job is assigned to *some* class-i server given that $J=j$ is

$$r_i^{\mathrm{B}}(\boldsymbol{d}|J=j) = \frac{\alpha_i(j,\boldsymbol{d})}{d_i}. \tag{16}$$

It now remains to determine $\mathbb{P}(J=j|\boldsymbol{D}=\boldsymbol{d}, \text{tagged class-}i\text{ server is busy})$. First, we address the case where $J=j$ for some $j \in \mathcal{S}$. Since $\mathsf{AR} \in \mathbf{CID}$, whenever $j \in \{1,2,\ldots,i\}$, we must have $\alpha_i(j,\boldsymbol{d}) = 0$ as the query contains an idle server at least as fast as the tagged (class-i) server (which happens to be busy). Hence, we may restrict attention to $j > i$, in which case—recalling that $b_j(\boldsymbol{d})$ denotes the probability

that all queried servers faster than the tagged (class-i server) are busy, as given by Eq. (7)—we have

$$\begin{aligned}\mathbb{P}(J = j|\boldsymbol{D} = \boldsymbol{d}, \text{tagged class-}i \text{ server is busy}) &= \mathbb{P}(J = j|\boldsymbol{D} = \boldsymbol{d}, A_i < D_i) \\ &= \frac{b_j(\boldsymbol{d})\left(1 - \rho_j^{d_j}\right)}{\rho_i},\end{aligned} \tag{17}$$

where we recall that $\rho_{s+1}^{d_{s+1}} \equiv 0$ and note that the $1/\rho_i$ factor is introduced due to the fact that $A_i < D_i$ (because the server is known to be busy).

We can now combine Eqs. (15), (16), and (17) together with the fact that $\alpha_i(j, \boldsymbol{d}) = 0$ whenever $j \in \{1, 2, \ldots, i\}$ in order to obtain the claimed formula for $r_i^{\mathrm{B}}(\boldsymbol{d})$. □

The proof of Theorem 1 follows from Lemmas 1, 2, and 3, together with the convention that $r_i^{\mathrm{I}}(\boldsymbol{d}) \equiv 0$ and $r_i^{\mathrm{B}}(\boldsymbol{d}) \equiv 0$ whenever $d_i = 0$.

5 Finding optimal dispatching policies under Class and Idleness Differentiated assignment

Based on the analysis in the previous section, we can now write a nonlinear program for determining optimal dispatching policies in the ⟨**QRF**, **ARF**⟩ family for various choices of **QRF**. This amounts to jointly determining an optimal probability distribution p over query mixes and an optimal family of functions constituting the assignment rule α_i (for $i \in \mathcal{S}$).

Each choice of querying rule family **QRF** yields a different optimization problem. All of these optimization problems can be formulated to share a common objective function. Meanwhile, the set of permissible querying rules (i.e., the chosen querying rule family **QRF**) restricts the set of feasible decision variables. Naturally, formulating problems in this way, if $\mathbf{QRF}' \subseteq \mathbf{QRF}$, then the feasibility region of the optimization problem associated with $\langle \mathbf{QRF}', \mathbf{CID} \rangle$ is contained within that associated with ⟨**QRF**, **CID**⟩, and hence, all such optimization problems have feasibility regions contained within that of ⟨**GEN**, **CID**⟩. Consequently, if we can solve the problem associated with ⟨**GEN**, **CID**⟩, then solving a problem associated with ⟨**QRF**, **CID**⟩ for another querying rule family **QRF** will never yield a policy that results in a strictly lower mean response time than the one we have already found. In fact, the problem associated with ⟨**GEN**, **CID**⟩ can be viewed as a "relaxation" of the others.

While the above discussion seems to suggest that one need only study the optimization problem associated with ⟨**GEN**, **CID**⟩, there are several reasons for studying problems associated with ⟨**QRF**, **CID**⟩ for other querying rule families, $\mathbf{QRF} \subset \mathbf{GEN}$. First, as discussed in Appendix D of [12], many of the feasibility regions associated with the other optimization problems can be expressed as polytopes in a space with far fewer dimensions than those studied under **GEN**, suggesting that these other problems might be solved more efficiently. Numerical evidence that we will present in Sect. 6.3 corroborates this suggestion. Second, as we shall discuss in detail throughout Sect. 6, these problems are often prohibitively difficult to solve, so we rely on heuristics to

find strong performing (although not necessarily optimal) solutions within each family of dispatching policies. Therefore, it will sometimes be the case that even though $\mathbf{QRF}' \subset \mathbf{QRF}$, a heuristic (rather than truly optimal) "solution" to a problem associated with $\langle \mathbf{QRF}', \mathbf{CID} \rangle$ may outperform those obtained from $\langle \mathbf{QRF}, \mathbf{CID} \rangle$. Finally, some families of rules with simpler structures may be more desirable for practical implementation purposes.

Before presenting our optimization problems, we note that we have not consistently formulated each problem as a restriction on the problem associated with $\langle \mathbf{GEN}, \mathbf{CID} \rangle$. While for any $\langle \mathbf{QRF}, \mathbf{CID} \rangle$ there exists at least one formulation of the optimization problem that resembles that of $\langle \mathbf{GEN}, \mathbf{CID} \rangle$ with additional constraints, we have opted for a more "natural" approach where we tailor the optimization problem for each dispatching policy $\langle \mathbf{QRF}, \mathbf{CID} \rangle$ to the structure of the choice of querying rule family **QRF**.

Remark 5 The optimization problems that we study are of the form where we minimize $f\colon \mathcal{X} \to \mathbb{R}$ on the feasible set $\mathcal{X}$ such that for each $x \in \mathcal{X}$, x corresponds to a dispatching policy that yields an overall mean response time $\mathbb{E}[T] = f(x)$. We say that two optimization problems with feasible regions $\mathcal{X}_1$ and $\mathcal{X}_2$, respectively, are *equivalent formulations* of one another if both (i) for each $x_1 \in \mathcal{X}_1$, there exists an $x_2 \in \mathcal{X}_2$ such that the policies corresponding to x_1 in the first problem and x_2 in the second yield stochastically identical systems, and (ii) the analogous statement holds for each $x_2 \in \mathcal{X}_2$. While all formulations of a given problem have solutions that yield identical system behavior, some formulations may be more tractable (or more amenable to heuristic analysis) than others.

5.1 Optimizing over the General Class Mix family

We begin by considering the case where $\mathbf{QRF} = \mathbf{GEN}$, i.e., the case where we allow for all possible (static symmetric) querying rules, where all functions $p\colon \mathcal{D} \to [0, 1]$ are valid so long as $\sum_{\boldsymbol{d} \in \mathcal{D}} p(\boldsymbol{d}) = 1$.

Since both p and all of the α_i functions take arguments from a domain with finitely many elements, we would like to treat each *evaluation* of these functions as a decision variable, i.e., we would like to treat $p(\boldsymbol{d})$ for each $\boldsymbol{d} \in \mathcal{D}$ and $\alpha_i(\boldsymbol{a}, \boldsymbol{d})$ for each triple $(i, \boldsymbol{a}, \boldsymbol{d}) \in \mathcal{S} \times \mathcal{A} \times \mathcal{D}$ (or $\alpha_i(j, \boldsymbol{d})$ for each triple $(i, j, \boldsymbol{d}) \in \mathcal{S} \times \bar{\mathcal{S}} \times \mathcal{D}$ when using our abuse of notation) as decision variables, with appropriate constraints. However, as we have discussed earlier, we have pruned the decision space so that $\alpha_i(\boldsymbol{a}, \boldsymbol{d})$ depends only on the class of the fastest idle queried server $J \equiv \min\{j \in \mathcal{S}\colon A_j > 0\}$ realized under the event $(\boldsymbol{A}, \boldsymbol{D}) = (\boldsymbol{a}, \boldsymbol{d})$ and on the (realized value of the) set of classes of queried servers that are faster than class-J servers $\{j < J\colon D_j > 0\}$ under the same event. For example, consider a setting where $s = 4$ and $d = 6$, where $\boldsymbol{d}_1 = (4, 0, 1, 1)$, $\boldsymbol{a}_1 = (0, 0, 1, 1)$, $\boldsymbol{d}_2 = (2, 0, 3, 1)$, and $\boldsymbol{a}_2 = (0, 0, 3, 0)$. Under both the events $(\boldsymbol{A}, \boldsymbol{D}) = (\boldsymbol{a}_1, \boldsymbol{d}_1)$ and $(\boldsymbol{A}, \boldsymbol{D}) = (\boldsymbol{a}_2, \boldsymbol{d}_2)$, we have $J = 3$, while $\{j < J\colon D_j > 0\} = \{1\}$, and so we must have $\alpha_i(\boldsymbol{a}_1, \boldsymbol{d}_1) = \alpha_i(\boldsymbol{a}_2, \boldsymbol{d}_2)$—and equivalently, using our abuse of notation, we must have $\alpha_i(3, \boldsymbol{d}_1) = \alpha_i(3, \boldsymbol{d}_2)$—for all $i \in \{1, 2, 3, 4\}$.

The pruning described above could be enforced in our optimization problem through the introduction of constraints, but we may also approach pruning more directly by reducing the set of decision variables. We opt for the latter, to which end we introduce the map $\gamma : \bar{\mathcal{S}} \times \mathcal{D} \to \mathcal{D}$. In order to define γ, let $I\{\cdot\}$ denote the indicator function, let $\boldsymbol{e}_i$ denote the i-th s-dimensional unit vector (so that, e.g., when $s = 4$, we have $\boldsymbol{e}_3 \equiv (0, 0, 1, 0)$) and let $h(\boldsymbol{d}) \equiv \min\{\ell \in \mathcal{S} : d_\ell > 0\}$ denote the class of the fastest queried server (regardless of whether this server is idle or busy). The map γ is defined as follows:

$$\gamma(j, \boldsymbol{d}) \mapsto \sum_{i=1}^{s} I\{d_i > 0 \text{ and } i \leq j\}\boldsymbol{e}_i + \left(d - \sum_{i=1}^{s} I\{d_i > 0 \text{ and } i \leq j\}\right)\boldsymbol{e}_{h(\boldsymbol{d})},$$

(so that, e.g., when $s = d = 8$, we have $\gamma(5, (0, 2, 1, 0, 3, 0, 2, 0)) = (0, 6, 1, 0, 1, 0, 0, 0)$)). Given some $j \in \bar{\mathcal{S}}$ and $\boldsymbol{d} \in \mathcal{D}$, $\gamma(j, \boldsymbol{d})$ is the unique query mix with the maximum possible number of queried class-$h(\boldsymbol{d})$ servers such that the realized value of the set $\{j < J : D_j > 0\}$ is the same under events $(J, \boldsymbol{D}) = (j, \boldsymbol{d})$ and $(J, \boldsymbol{D}) = (j, \gamma(j, \boldsymbol{d}))$—thus guaranteeing that $\alpha_i(j, \boldsymbol{d})$ and $\alpha_i(j, \gamma(j, \boldsymbol{d}))$ are identical due to the pruning. We note that the fact that the number of class-$h(\boldsymbol{d})$ queried servers is maximized is not of any particular significance; rather, the map γ allows us to specify a unique query mix to act as a "representative" for all query mixes that would be treated in the same way by the assignment rule under a given realization of the random variable J. Returning to our optimization problem, observe that we can reduce the dimensionality of the feasible region by assigning values only to those $\alpha_i(j, \boldsymbol{d})$ when $(i, j, \boldsymbol{d}) \in \mathcal{T}$ where the set $\mathcal{T}$ represents a pruned set of triples $(i, j, \boldsymbol{d})$, for which each $\alpha_i(j, \boldsymbol{d})$ can be assigned a distinct nonzero value in formulating an assignment rule:

$$\mathcal{T} \equiv \left\{(i, j, \boldsymbol{d}) \in \mathcal{S} \times \bar{\mathcal{S}} \times \mathcal{D} : i \leq j,\ d_i > 0,\ (j \leq s) \implies d_j > 0,\ \gamma(j, \boldsymbol{d}) = \boldsymbol{d}\right\}. \tag{18}$$

Meanwhile, wherever the optimization problem would make reference to $\alpha_i(j, \boldsymbol{d})$, we instead write the decision variable $\alpha_i(j, \gamma(j, \boldsymbol{d}))$ as both values are the same. Furthermore, as defined in Eq. (18), $\mathcal{T}$ excludes triples $(i, j, \boldsymbol{d}) \in \mathcal{S} \times \bar{\mathcal{S}} \times \mathcal{D}$ where (i) $j < i$, (ii) $d_i = 0$, or (iii) $d_j = 0$ and $j \neq s + 1$. Defining $\mathcal{T}$ in such a way allows us to omit $\alpha_i(j, \boldsymbol{d})$ for such triples, as all of these values must be 0 (see Sect. 3.3 for details). In order to write the $\sum_{i=1}^{s} \alpha_i(j, \boldsymbol{d}) = 1$ constraints concisely, without reference to $\alpha_i(j, \boldsymbol{d})$ for triples $(i, j, \boldsymbol{d}) \notin \mathcal{T}$, we need a way to specify those $(j, \boldsymbol{d})$ pairs that can form a triple $(i, j, \boldsymbol{d}) \in \mathcal{T}$ with one or more classes $i \in \mathcal{S}$, so we also introduce the notation $\mathcal{P}$ to denote such pairs:

$$\mathcal{P} \equiv \left\{(j, \boldsymbol{d}) \in \bar{\mathcal{S}} \times \mathcal{D} : (\exists i \in \mathcal{S} : (i, j, \boldsymbol{d}) \in \mathcal{T})\right\}. \tag{19}$$

Similarly, in expressing the inner sum in Eq. (9), we avoid making reference to the same forbidden triples and ensure that $j \geq i + 1$, by introducing the following notation

for any fixed $(i, \boldsymbol{d}) \in \mathcal{S} \times \mathcal{D}$:

$$\mathcal{J}_i(\boldsymbol{d}) \equiv \{j \in \{i+1, i+2, \ldots, s+1\} \colon (i, j, \gamma(j, \boldsymbol{d})) \in \mathcal{T}\}. \tag{20}$$

Finally, building upon our analysis in Sect. 4 (including requiring $\lambda_i^{\mathrm{B}} < \mu_i$ for all $i \in \mathcal{S}$ in order to guarantee stability), we have the following optimization problem:

Optimization Problem for the ⟨GEN, CID⟩ family
Given values of s, d, λ, and μ_i and q_i (both given for all $i \in \mathcal{S}$), determine **nonnegative** values of the decision variables λ_i^{I} and λ_i^{B} (both for all $i \in \mathcal{S}$), $p(\boldsymbol{d})$ (for $\boldsymbol{d} \in \mathcal{D}$), and $\alpha_i(j, \boldsymbol{d})$ (for all $(i, j, \boldsymbol{d}) \in \mathcal{T}$) that solve the following nonlinear program:

$$\min \quad \frac{1}{\lambda} \sum_{i=1}^{s} q_i \left(\frac{(1-\rho_i)\lambda_i^{\mathrm{I}} + \rho_i \lambda_i^{\mathrm{B}}}{\mu_i - \lambda_i^{\mathrm{B}}} \right)$$

$$\text{s.t.} \quad \lambda_i^{\mathrm{I}} = \frac{\lambda}{q_i} \sum_{\boldsymbol{d} \in \mathcal{D}} \left\{ d_i b_i(\boldsymbol{d}) p(\boldsymbol{d}) \alpha_i(i, \gamma(i, \boldsymbol{d})) \sum_{a_i=1}^{d_i} \binom{d_i - 1}{a_i - 1} \frac{(1-\rho_i)^{a_i - 1} \rho_i^{d_i - a_i}}{a_i} \right\} \qquad (\forall i \in \mathcal{S})$$

$$\lambda_i^{\mathrm{B}} = \frac{\lambda}{q_i \rho_i} \sum_{\boldsymbol{d} \in \mathcal{D}} \left\{ p(\boldsymbol{d}) \sum_{j \in \mathcal{J}_i(\boldsymbol{d})} b_j(\boldsymbol{d}) \left(1 - \rho_j^{d_j}\right) \alpha_i(j, \gamma(i, \boldsymbol{d})) \right\} \qquad (\forall i \in \mathcal{S})$$

$$\lambda_i^{\mathrm{B}} < \mu_i \qquad (\forall i \in \mathcal{S})$$

$$\sum_{\boldsymbol{d} \in \mathcal{D}} p(\boldsymbol{d}) = 1$$

$$\sum_{\substack{i \in \mathcal{S}: \\ (i, j, \boldsymbol{d}) \in \mathcal{T}}} \alpha_i(j, \boldsymbol{d}) = 1 \qquad (\forall (j, \boldsymbol{d}) \in \mathcal{P})$$

where for all $i \in \mathcal{S}$ in writing ρ_i we are denoting the expression $\lambda_i^{\mathrm{I}} / \left(\mu_i - \lambda_i^{\mathrm{B}} + \lambda_i^{\mathrm{I}}\right)$ with $\rho_{s+1}^{d_{s+1}} \equiv 1$ for all $\boldsymbol{d} \in \boldsymbol{D}$, and for all $j \in \bar{\mathcal{S}}$ in writing $b_j(\boldsymbol{d})$ we are denoting the expression $\prod_{\ell=1}^{j-1} \left(\lambda_\ell^{\mathrm{I}} / \left(\mu_\ell - \lambda_\ell^{\mathrm{B}} + \lambda_\ell^{\mathrm{I}}\right)\right)^{d_\ell}$.
The $p(\boldsymbol{d})$ values (for all $\boldsymbol{d} \in \mathcal{D}$) and the $\alpha_i(j, \boldsymbol{d})$ values (for all $(i, j, \boldsymbol{d}) \in \mathcal{T}$ together with $\alpha_i(j, \boldsymbol{d}) = 0$ for all $(i, j, \boldsymbol{d}) \notin \mathcal{T}$) from an optimal solution specify the querying and assignment rules associated with an optimal ⟨**GEN**, **CID**⟩ dispatching policy, respectively.

5.2 Optimizing over the Independent and Identically Distributed Querying family

We now turn our attention to the case where **QRF** = **IID** as it is simpler to address **IID** before the more general (but less general than **GEN**) **IND** family. Under the **IID** querying rule, the d servers are queried independently according to an identical probability distribution over the set of server classes $\mathcal{S}$. For any querying rule $\mathsf{QR} \in$ **IID**, we can express the probability distribution p over query mixes $\boldsymbol{D}$ in terms of an auxiliary distribution $\widetilde{p}$ over the *set of classes* $\mathcal{S}$. Specifically, we express $\widetilde{p}$ as a function $\widetilde{p} \colon \mathcal{S} \to [0, 1]$ that is subject to the constraint $\sum_{i=1}^{s} \widetilde{p}(i) = 1$, where $\widetilde{p}(i)$ is the probability that an arbitrary queried server is of class i. In particular, due to the structure of querying rules in **IID**, we query d servers independently according to **IID**

according to $\widetilde{p}$ upon each arrival, yielding

$$p(\boldsymbol{d}) = \binom{d}{d_1, d_2, \ldots, d_s} \prod_{i=1}^{s} \widetilde{p}(i)^{d_i} = d! \prod_{i=1}^{s} \frac{\widetilde{p}(i)^{d_i}}{d_i!},$$

that is, any $\mathsf{QR} \in \mathbf{IID}$ makes independent queries with querying mixes $\boldsymbol{D}$ that are *multinomially distributed* random vectors. Moreover, such a querying rule QR is uniquely identified by $\widetilde{p}$. The above observations allow us to express the optimization problem for $\langle \mathbf{IID}, \mathbf{CID} \rangle$ as a modification of the optimization problem for $\langle \mathbf{GEN}, \mathbf{CID} \rangle$ (see Appendix C of [12]).

5.3 Optimizing over the Independent Querying family

When $\mathsf{QR} \in \mathbf{IID}$, the function $\widetilde{p}\colon \mathcal{S} \to [0, 1]$ governed the probability distribution by which servers were queried: specifically, $\widetilde{p}(i)$ denoted the probability that any individual queried server is of class i. We extend this notion to the case where we can query d servers according to potentially different distributions (i.e., when $\mathsf{QR} \in \mathbf{IND}$) as follows: let $\widetilde{p}_1, \widetilde{p}_2, \ldots, \widetilde{p}_d\colon \mathcal{S} \to [0, 1]$ denote a family of functions such that $\widetilde{p}_\ell(i)$ denotes the probability that upon any job's arrival, the ℓ-th server queried is of class i.

Remark 6 All servers are queried simultaneously (under all querying rules, including those contained **IND** in particular), and so the order of the queries is irrelevant (i.e., a querying rule specified by $\widetilde{p}_1, \widetilde{p}_2, \ldots, \widetilde{p}_d$ performs indistinguishably from one specified by $\widetilde{p}'_1 = \widetilde{p}_{\sigma(1)}, \widetilde{p}'_2 = \widetilde{p}_{\sigma(2)}, \ldots, \widetilde{p}'_d = \widetilde{p}_{\sigma(d)}$, for any permutation $\sigma\colon \{1, 2, \ldots, d\} \to \{1, 2, \ldots, d\}$).

We would like to define $p(\boldsymbol{d})$ in terms of $\widetilde{p}_1, \widetilde{p}_2, \ldots, \widetilde{p}_d$. With this end in mind, we introduce some additional notation: let

$$\mathcal{Q} \equiv \{1, 2, \ldots, d\} \qquad \text{and} \qquad \vec{\mathcal{Q}} \equiv (\mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_s), \tag{21}$$

where each $\mathcal{Q}_\ell$ is a subset of $\mathcal{Q}$ (i.e., $\vec{\mathcal{Q}}$ is an s-tuple of subsets of $\mathcal{Q}$), and let $\mathcal{B}(\boldsymbol{d})$ denote the set of all s-tuples $\vec{\mathcal{Q}}$ that form a partition of $\mathcal{Q}$ such that the ℓ-th entry of $\vec{\mathcal{Q}}$ contains exactly ℓ_i elements. That is,

$$\mathcal{B}(\boldsymbol{d}) = \left\{ \vec{\mathcal{Q}}\colon (\forall \ell \in \mathcal{S}\colon \mathcal{Q}_\ell \subseteq \mathcal{Q},\ |\mathcal{Q}_\ell| = d_\ell), \bigcup_{\ell=1}^{s} \mathcal{Q}_\ell = \mathcal{Q} \right\}.$$

Crucially, $\mathcal{B}(\boldsymbol{d})$ corresponds to the ways that d queries can result in the query mix $\boldsymbol{d}$. With the above notation defined, we can now write the following:

$$p(\boldsymbol{d}) = \sum_{\vec{\mathcal{Q}} \in \mathcal{B}(\boldsymbol{d})} \prod_{i=1}^{s} \prod_{u \in \mathcal{Q}_i} \widetilde{p}_u(i).$$

The optimization problem for the ⟨**IND**, **CID**⟩ family of dispatching policies—presented in Appendix C of [12]—then follows readily from that of ⟨**GEN**, **CID**⟩ family.

5.4 Optimizing over the Deterministic Class Mix family

We now address the case where **QRF** = **DET**. For any QR ∈ **DET**, p is such that there exists some specifically designated $\boldsymbol{d} \in \mathcal{D}$ where $p(\boldsymbol{d}) = 1$ and $p(\boldsymbol{d}') = 0$ for all $\boldsymbol{d}' \in \mathcal{D}\backslash\{\boldsymbol{d}\}$. That is, we always query deterministically, so that $\boldsymbol{D} = \boldsymbol{d}$ upon every job arrival. When attempting to find optimal ⟨**DET**, **CID**⟩ dispatching policies, we need to evaluate the mean response time under each ⟨$\mathsf{DQR}_{\boldsymbol{d}}$, **CID**⟩ dispatching policy, where $\mathsf{DQR}_{\boldsymbol{d}}$ is an *individual querying rule* in **DET** that always queries a set of servers with query mix $\boldsymbol{d}$ (further note that all querying rules in **DET** are of this form; hence, we have $\mathbf{DET} = \{\mathsf{DQR}_{\boldsymbol{d}} : \boldsymbol{d} \in \mathcal{D}\}$). Then, we choose the value of $\boldsymbol{d}$ (and the corresponding policy AR ∈ **CID**) that yields the best mean response time. Hence, the optimization problem for the ⟨**DET**, **CID**⟩ family of dispatching policies consists of solving $|\mathcal{D}|$ optimization *subproblems* and then comparing the objective values of these subproblems. While our approach can be seen as a disjunctive nonlinear program composed of a single objective function on the union of several nonlinear feasibility regions, one could also approach this optimization problem as a single mixed integer nonlinear program (MINLP).

Since we need only consider $\boldsymbol{D} = \boldsymbol{d}$ in each subproblem, we can dispense with the need for the map γ in this setting, however it will be useful to introduce the following analogues of $\mathcal{S}$, $\mathcal{T}$, $\mathcal{P}$, and $\mathcal{J}_i(\boldsymbol{d})$:

$$\mathcal{S}(\boldsymbol{d}) \equiv \{i \in \mathcal{S} : d_i > 0\} \tag{22}$$

$$\mathcal{T}(\boldsymbol{d}) \equiv \left\{(i, j) \in \mathcal{S} \times \bar{\mathcal{S}} : i \leq j,\ d_i > 0,\ (j \leq s) \implies d_j > 0\right\} \tag{23}$$

$$\mathcal{P}(\boldsymbol{d}) \equiv \left\{j \in \bar{\mathcal{S}} : (\exists i \in \mathcal{S} : (i, j) \in \mathcal{T}(\boldsymbol{d}))\right\} = \{j \in \mathcal{S} : d_j > 0\} \cup \{s + 1\} \tag{24}$$

$$\mathcal{J}_i(\boldsymbol{d}) \equiv \{j \in \{i + 1, i + 2, \ldots, s + 1\} : (i, j) \in \mathcal{T}(\boldsymbol{d})\}. \tag{25}$$

While Eq. (25) may appear to be a *redefinition* of $\mathcal{J}_i(\boldsymbol{d})$ it is actually consistent with the earlier definition provided in Eq. (20).

With the above notation defined, we can formulate the optimization problem for the ⟨**DET**, **CID**⟩ family of dispatching policies, which we present in Appendix C of [12].

5.5 Optimizing over the Single Random Class family

We first observe that when a dispatching policy's querying rule QR ∈ **SRC**, then assignment decisions under that policy are always among servers of the same class, so if we further impose that the dispatching policy's assignment rule AR ∈ **CID**, then the assignment decision amounts to sending the job to an idle queried server (chosen uniformly at random) whenever the query includes such a server and to any (busy) server (chosen uniformly at random) otherwise. Just as the querying rules in

the **IID** family were uniquely specified by some probability distribution over the server classes, $\widetilde{p}$, querying rules in the **SRC** family are also specified by such a probability distribution, which we denote by $\widehat{p}$. Specifically, let $\widehat{p}\colon \mathcal{D} \to [0, 1]$ be a distribution that satisfies $\mathbb{P}(\boldsymbol{D} = d\boldsymbol{e}_i) = \widehat{p}(i)$ for all $i \in \mathcal{S}$ and $\mathbb{P}(\boldsymbol{D} = \boldsymbol{d}) = 0$ for all $\boldsymbol{d} \in \mathcal{D}\backslash\{d\boldsymbol{e}_1, d\boldsymbol{e}_2, \ldots, d\boldsymbol{e}_s\}$, where $d\boldsymbol{e}_i$ corresponds to the query mix where d class-i servers have been queried (i.e., it denotes the vector of length s with all zero entries, except for an entry of d at position i). In particular, the $\alpha_i(j, \boldsymbol{d})$ values are immaterial, and we do not need to optimize over them. This yields the associated optimization problem provided in Appendix C of [12].

5.6 Optimizing over the Single Fixed Class family

As we have remarked earlier, **SFC** contains exactly s querying rules. Specifically, we always query d class-i servers for some fixed $i \in \mathcal{S}$. As the querying rule is specified by the choice of this fixed value of i alone, we can disregard querying probabilities. Moreover, as all queried servers are of the same class, we can also disregard assignment probabilities. Hence, optimization amounts to choosing the fixed value of $i \in \mathcal{S}$ that minimizes the mean response time. To this end, as in the case where **QRF** = **DET**, we make use of subproblems (and alternatively could have made uses of integer variables), although this time we only have s such subproblems. The associated optimization problem is presented in Appendix C of [12].

We note that the solution to each subproblem does not depend on the objective function (although objective function values must be computed to find i^*, the index of the subproblem with the lowest objective function value). Moreover, each subproblem will have at most one feasible solution. Essentially, solving each subproblem merely requires one to solve a system of two nonlinear equations in two constrained unknowns: $\lambda_i^{\mathrm{I}} \in [0, \infty)$ and $\lambda_i^{\mathrm{B}} \in [0, \mu_i)$.

5.7 Optimization subject to a fixed individual querying rule

We also formulate an optimization rule in order to determine optimal assignment rules $\mathsf{AR} \in$ **CID** given any individual querying rule $\mathsf{QR} \in$ **QRF** as specified by some $p\colon \mathcal{D} \to [0, 1]$, as long as that querying rule yields a stable system under some assignment rule. Since the probability distribution over $\mathcal{D}$ is specified, we need only determine the assignment probabilities. This optimization problem (i.e., the one associated with $\langle \mathsf{QR}, \textbf{CID} \rangle$) is presented in Appendix C of [12].

We are particularly interested in the UNI and BR rules, defined by

$$p(\boldsymbol{d}) = \binom{d}{d_1, d_2, \ldots, d_s} \left(\frac{1}{s^d}\right) = d! \Big/ \left(s^d \prod_{i=1}^{s} d_i!\right)$$

and

$$p(\boldsymbol{d}) = \binom{d}{d_1, d_2, \ldots, d_s} \prod_{i=1}^{s} (\mu_i q_i)^{d_i} = d! \prod_{i=1}^{s} \frac{(\mu_i q_i)^{d_i}}{d_i!},$$

respectively, in accordance with the fact that both are members of the **IID** family (see Sect. 5.2). Note that some other specific querying rules—especially those that never query servers of one or more classes—allow for significant pruning of the space of assignment rules.

6 Numerical results under Class and Idleness Differentiated assignment

In this section, we compare the performance of dispatching policies found by numerically solving our optimization problems associated with ⟨**QRF**, **CID**⟩ for various querying rule families **QRF** (including the single-rule family **QRF** = {BR}). In these comparisons, we will examine both performance (i.e., mean response time) and the computation time associated with determining the optimal parameters for the querying and/or assignment rules.

6.1 Parameter settings

We provide numerical results for a variety of parameter settings (i.e., problem instances), where each parameter setting consists of a choice of s, d, λ, $\mu_1, \ldots, \mu_s$, and $q_1, \ldots, q_s$. Our choices of λ, s, and d resemble a full factorial design, while our choices of $\mu_1, \ldots, \mu_s$ and $q_1, \ldots, q_s$ clearly depend on s and are subject to the normalization constraint $\sum_{i=1}^{s} \mu_i q_i = 1$.

Specifically, we examine all combinations of s, d, and λ, where $s, d \in \{2, 3, 4\}$ and $\lambda \in \{0.05, 0.10, \ldots, 0.95\}$ (although when plotting curves, we instead consider $\lambda \in \{0.02, 0.04, \ldots, 0.98\}$). For each (s, d, λ) setting, we then consider one set of $\mu_1, \ldots, \mu_s$ corresponding to each subset of $s-1$ elements of $\{1.25, 1.50, 2, 3, 5\}$: for each $\{R_1, R_2, \ldots, R_{s-1}\} \subseteq \{1.25, 1.50, 2, 3, 5\}$, ordered so that $R_1 > R_2 > \ldots > R_{s-1}$, we let $\mu_i = R_i \mu_s$ for each $i \in \mathcal{S}\backslash\{s\}$ (i.e., $R_i \equiv \mu_i/\mu_s$).. That is, in each parameter setting each server that does not belong to the slowest class runs at a speed that is 25%, 50%, 100%, 200%, or 400% faster than the speed of the slowest server, with each parameter setting accommodating $s-1$ such speedup factors. The speed of the slowest server depends on the values of $q_1, q_2, \ldots, q_s$ (see below), as follows:

$$\mu_s = \left(q_s + \sum_{i=1}^{s-1} R_i q_i \right)^{-1}.$$

Meanwhile, for each $(s, d, \lambda, R_1, R_2, \ldots, R_{s-1})$ setting, we consider the following $(q_1, q_2, \ldots, q_s)$ combinations:

$$\left\{(q_1, q_2, \ldots, q_s) \in \mathbb{Q}^s \colon (\forall i \in \mathcal{S} \colon 6q_i \in \mathbb{Z},\ q_i > 0),\ \sum_{i=1}^{s} q_i = 1\right\}.$$

That is, we consider all (and only those) combinations $(q_1, q_2, \ldots, q_s)$ where we can view each server class as holding a (nonzero integer) number of "shares"—out of a total of 6 such shares—with each class being allocated a number of servers proportional to the number of shares it holds. This methodology for selecting μ_i and q_i values was chosen to allow for a wide variety of parameter settings while ensuring that in each setting no class is particularly under- or over-represented nor so much slower or faster than others. In this way, we avoid extreme parameter settings that render certain classes (and hence, certain aspects of querying and assignment rules) inconsequential.

Note that there are 3 choices for s, 3 choices for d, 19 choices for λ, $\binom{5}{s}$ choices of speed configurations for each choice of s, and also $\binom{5}{s}$ "share" configurations for each choice of s (that is, 5 choices of each configuration when $s = 4$ and 10 choices of each configuration when $s = 2$ or $s = 3$). Hence, we consider a total of $(3)(19)\left(5^2 + 10^2 + 10^2\right) = 12\,825$ parameter settings. These parameter settings can be broken down into 1 875 settings for each of the 19 λ values. Alternatively, they can broken down by the choice of s: 1 425 settings where $s = 2$, and 5 700 settings each when $s = 3$ and $s = 4$.

6.2 Numerical optimization methodology and notation

All of the numerical results we present throughout this section were obtained using code written in the programming language Julia. We used the JuMP package [5] in Julia to define our optimization models (see Sect. 5), and we solved these problems using the interior point optimizer (IPOPT) optimization package [17, 33]. Note that due to the presence of non-convexity in our optimization problems, IPOPT does not consistently yield globally optimal solutions. Hence, for each policy family, we should view the associated "optimal" solution yielded by IPOPT as being the parameters of a heuristically chosen policy belonging to that family. For further implementation details and small caveats to the results presented in this section, see Appendix E of [12].

Now consider an arbitrary querying rule family **QRF** and an arbitrary assignment rule family **ARF**. Let the (admittedly cumbersome) notation

$$\mathsf{IPOptD}_{\langle \mathbf{QRF}, \mathbf{ARF} \rangle} \equiv \left\langle \mathsf{IPOptQ}_{\langle \mathbf{QRF}, \mathbf{ARF} \rangle}, \mathsf{IPOptA}_{\langle \mathbf{QRF}, \mathbf{ARF} \rangle} \right\rangle$$

denote the dispatching policy specified by the IPOPT solution to the optimization problem associated with the $\langle \mathbf{QRF}, \mathbf{ARF} \rangle$ family of dispatching policies (assuming such an optimization problem exists, has been identified, and can be implemented and given to IPOPT). That is, for a querying rule family **QRF** (e.g., **GEN**), we use IPOPT

to "solve" an optimization problem that involves jointly selecting querying and assignment probabilities, resulting in a querying rule (belonging to **QRF**), which we denote by $\mathsf{IPOptQ}_{\langle \mathbf{QRF}, \mathbf{ARF} \rangle}$, and an assignment rule (belong to **ARF**), which we denote by $\mathsf{IPOptA}_{\langle \mathbf{QRF}, \mathbf{ARF} \rangle}$. Recall that in all of the optimization problems that we have proposed in Sect. 5, we have always considered $\mathbf{ARF} = \mathbf{CID}$, so to alleviate the burden imposed by this cumbersome notation, we can omit the reference to the assignment rule family whenever we take it to be **CID**. That is, we take **CID** as the "default" assignment rule family and use the notation $\mathsf{IPOptD}_{\mathbf{QRF}} \equiv \langle \mathsf{IPOptQ}_{\mathbf{QRF}}, \mathsf{IPOptA}_{\mathbf{QRF}} \rangle$, where we let $\mathsf{IPOptQ}_{\mathbf{QRF}} \equiv \mathsf{IPOptQ}_{\langle \mathbf{QRF}, \mathbf{CID} \rangle} \in \mathbf{QRF}$ and $\mathsf{IPOptA}_{\mathbf{QRF}} \equiv \mathsf{IPOptA}_{\langle \mathbf{QRF}, \mathbf{CID} \rangle} \in \mathbf{CID}$, from which it follows that $\mathsf{IPOptD}_{\mathbf{QRF}} = \mathsf{IPOptD}_{\langle \mathbf{QRF}, \mathbf{CID} \rangle}$.

Remark 7 We abuse this notation by adapting it for use with specific policies, rather than only families, so that, e.g., $\mathsf{IPOptD}_{\mathsf{BR}} \equiv \mathsf{IPOptD}_{\{\mathsf{BR}\}}$, and $\mathsf{IPOptD}_{\langle \mathbf{SRC}, \mathsf{JSQ} \rangle} \equiv \mathsf{IPOptD}_{\langle \mathbf{SRC}, \{\mathsf{JSQ}\} \rangle}$.

6.3 Comparison of querying rule families with respect to $\mathbb{E}[T]$ and optimization runtime

We proceed to evaluate the performance of the $\mathsf{IPOptD}_{\mathbf{GEN}}$, $\mathsf{IPOptD}_{\mathbf{IND}}$, $\mathsf{IPOptD}_{\mathbf{IID}}$, $\mathsf{IPOptD}_{\mathbf{SRC}}$, and $\mathsf{IPOptD}_{\mathsf{BR}}$ dispatching policies. We omit examination of the **DET** and **SFC** querying rule families as well as the UNI querying rule, as under many of our parameter settings, any dispatching policy constructed from such querying rules yields an unstable system (see Sect. 4.2 on stability and Sect. 6.1 on our parameter settings). We examine the performance of $\mathsf{IPOptD}_{\mathbf{DET}}$ across a small set of parameters (taken from our earlier work in [7]) at the end of this section.

We evaluate the $\mathbb{E}[T]$ values yielded by each of the dispatching policies under consideration, for each of the 12 825 parameter settings described in Sect. 6.1. For each policy, we then compute the mean and median value of both $\mathbb{E}[T]$ and the optimization runtime (measured in seconds) across all of our parameter settings. Figure 2 illustrates the tradeoff between $\mathbb{E}[T]$ and optimization runtime as aggregated across our parameter settings. In Fig. 2 (left), we plot the (mean $\mathbb{E}[T]$, mean runtime) pairs associated with each policy, while in Fig. 2 (right) we plot the analogous pairs for median values. Before describing Fig. 2 in detail, we introduce one additional policy, motivated by the surprising observation that both $\mathsf{IPOptD}_{\mathbf{IND}}$ and $\mathsf{IPOptD}_{\mathbf{IID}}$ outperform $\mathsf{IPOptD}_{\mathbf{GEN}}$ with respect to the mean value of $\mathbb{E}[T]$ across the parameter set, with $\mathsf{IPOptD}_{\mathbf{IND}}$ outperforming $\mathsf{IPOptD}_{\mathbf{GEN}}$ with respect to the analogous median value as well. We observe this despite the fact that $\mathbf{IID} \subseteq \mathbf{IND} \subseteq \mathbf{GEN}$, which means that the best **GEN**-driven dispatching policy *must* perform at least as well as the best **IND**- and **IID**-driven policies; unfortunately, as IPOPT does not consistently find true optimal solutions, the solution found by IPOPT for a particular family can occasionally outperform the solution it finds for a more general family. We can construct a new policy to remedy the somewhat lackluster performance of $\mathsf{IPOptD}_{\mathbf{GEN}}$ by exploiting the fact that we can seed IPOPT with a feasible solution before running it to solve an optimization problem. Thus far, we have only discussed results which were obtained by running IPOPT without seeding it with an initial solution, however, IPOPT frequently yields noticeably better solutions to the optimization problem associated with the $\langle \mathbf{GEN}, \mathbf{CID} \rangle$ family

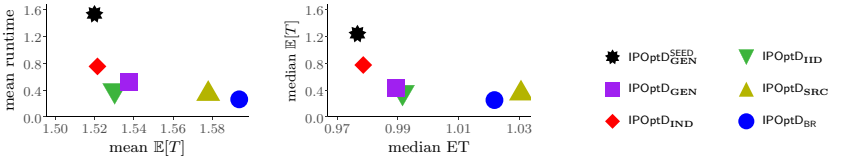


Fig. 2 Plots of the (mean $\mathbb{E}[T]$, mean runtime) pairs (left) and (median $\mathbb{E}[T]$, median runtime) pairs (right) calculated across all parameter settings defined in Sect. 6.1 for six dispatching policies

when seeded with the IPOPT solution associated with the ⟨**IND**, **CID**⟩ (as compared to the solution yielded by the "unseeded" problem associated with the ⟨**GEN**, **CID**⟩). We use the notation $\mathsf{IPOptD}^{\mathsf{SEED}}_{\mathbf{GEN}} \equiv \left\langle \mathsf{IPOptQ}^{\mathsf{SEED}}_{\mathbf{GEN}}, \mathsf{IPOptA}^{\mathsf{SEED}}_{\mathbf{GEN}} \right\rangle$ to refer to this new heuristic dispatching policy (see Appendix E.3 of [12] for details). One can similarly construct other heuristics for choosing initial values (e.g., seeding the **IID** optimization problem with the solution IPOPT found for the BR optimization problem, and even using the solution to the aforementioned seeded problem as a seed for the **IND** optimization problem, etc.); we extensively explored different heuristics for choosing initial values (e.g., seeding the **IID** optimization problem with the solution IPOPT found for the BR optimization problem); we found that—unlike $\mathsf{IPOptA}^{\mathsf{SEED}}_{\mathbf{GEN}}$—other alternative heuristics yielded negligible benefits in comparison to their "unseeded' counterparts.

Both the mean and the median results indicate that there is a tradeoff between $\mathbb{E}[T]$ and runtime: families that require a longer runtime to solve the optimization problem tend to yield lower $\mathbb{E}[T]$ values. Note, however, that the trends exhibited in Fig. 2 do not imply that the families have the same ordering with respect to $\mathbb{E}[T]$ and runtime for any *specific* parameter setting; indeed, some trends suggested in Fig. 2(left) are reversed in Fig. 2(right). For example, while $\mathsf{IPOptD}_{\mathbf{IID}}$ appears to dominate $\mathsf{IPOptD}_{\mathbf{GEN}}$ with respect to both mean measures, $\mathsf{IPOptD}_{\mathbf{IID}}$ has a higher (i.e., worse) median $\mathbb{E}[T]$ value than $\mathsf{IPOptD}_{\mathbf{GEN}}$.

Overall $\mathsf{IPOptD}_{\mathsf{BR}}$ and $\mathsf{IPOptD}_{\mathbf{SRC}}$ feature the lowest runtimes but at the expense of the worst performance (i.e., they have the highest $\mathbb{E}[T]$ values). The fast runtime of $\mathsf{IPOptD}_{\mathsf{BR}}$ can be attributed to the fact that it need only optimize over assignment; despite arising from the "smallest" of the optimization problems in many respects (see Table 2), $\mathsf{IPOptD}_{\mathbf{SRC}}$ features a higher runtime than $\mathsf{IPOptD}_{\mathsf{BR}}$. Meanwhile, $\mathsf{IPOptD}_{\mathbf{IID}}$ and $\mathsf{IPOptD}_{\mathbf{GEN}}$ offer an improvement in performance at the cost of additional runtime. Surprisingly, $\mathsf{IPOptD}_{\mathbf{IND}}$ has a longer runtime (and as previously discussed, better performance than) $\mathsf{IPOptD}_{\mathbf{GEN}}$ despite arising from solving a problem of a smaller (nominal) size. An examination of the optimization problem associated with ⟨**IND**, **CID**⟩, as presented in Appendix C of [12], provides a potential explanation for the exceptionally long runtimes associated with $\mathsf{IPOptD}_{\mathbf{IND}}$: the constraints in this optimization problem with λ_i^{B} on the left-hand side are very complicated. As previously noted, $\mathsf{IPOptD}^{\mathsf{SEED}}_{\mathbf{GEN}}$ achieves the best $\mathbb{E}[T]$ by building off of the strong performance of $\mathsf{IPOptD}_{\mathbf{IND}}$. Of course, this comes at a significant runtime expense, as one must now solve two optimization problems.

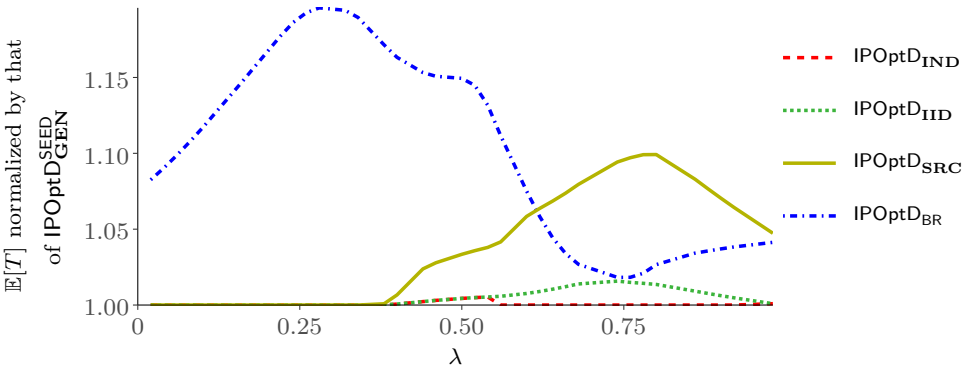


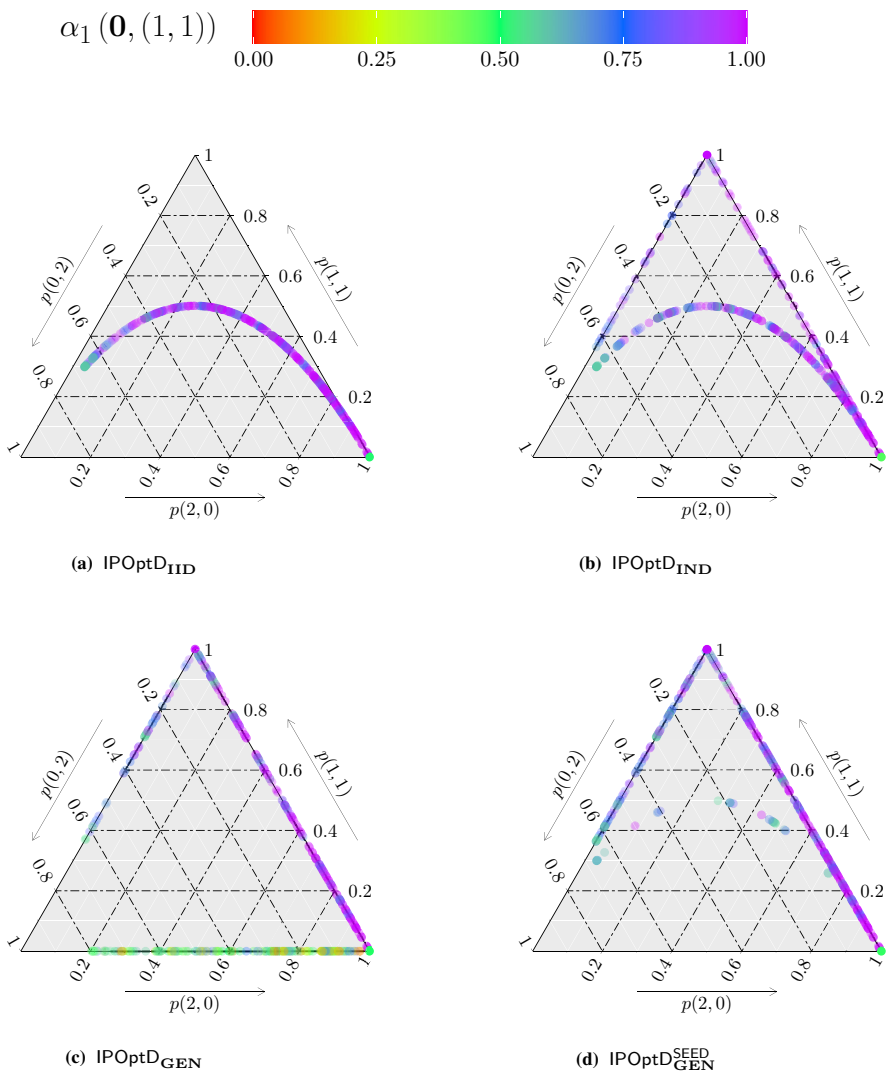
Fig. 3 $\mathbb{E}[T]$ relative to that of IPOptD$^{\text{SEED}}_{\text{GEN}}$ (i.e., $\mathbb{E}[T]^{\text{DP}}/\mathbb{E}[T]^{\text{IPOptD}^{\text{SEED}}_{\text{GEN}}}$) as a function of λ for the parameter settings where $s = d = 3$, λ varies over $\{0.02, 0.04, \ldots, 0.98\}$, $(q_1, q_2, q_3) = (1/3, 1/6, 1/2)$ and $(R_1, R_2) = (5, 2)$, yielding $(\mu_1, \mu_2, \mu_3) = (2, 4/5, 2/5)$, for the dispatching policies DP $\in$ {IPOptD$_{\text{IND}}$, IPOptD$_{\text{IID}}$, IPOptD$_{\text{SRC}}$, IPOptD$_{\text{BR}}$}

Throughout the entire parameter set, all of the optimization problems ran in well under one minute; the mean and median runtimes associated with each of the dispatching policies examined were under 2 seconds. In practice, these differences in runtimes are small enough that they would likely not be a significant factor, as this optimization would only need to be performed once to configure the system. Thus, while the tradeoff between $\mathbb{E}[T]$ and runtime is of theoretical interest, IPOptD$^{\text{SEED}}_{\text{GEN}}$ represents the best practical choice of dispatching policy among those studied here when achieving low $\mathbb{E}[T]$ is the foremost goal. On the other hand, if the simplicity or interpretability of the policy is of value to the system designer, IPOptD$_{\text{IID}}$ provides a reasonable alternative to IPOptD$^{\text{SEED}}_{\text{GEN}}$.

The results in Fig. 2 were aggregated across the entire space of parameter settings; in Fig. 3, we instead present results for a collection of settings where all parameters are fixed except for λ. This allows us to provide a *direct* comparison of how our dispatching policies perform with respect to their $\mathbb{E}[T]$ values across the spectrum of arrival rates. We first observe that IPOptD$_{\text{SRC}}$ and IPOptD$_{\text{BR}}$ each exhibit their best performance in different ranges of λ values, which provides some insight into the reversed relationship these policies exhibit when comparing their mean and median $\mathbb{E}[T]$ values over the entire parameter set. That said, these two policies perform considerably worse than the other policies examined. Both IPOptD$_{\text{IID}}$ and IPOptD$_{\text{IND}}$ achieve performance comparable to IPOptD$^{\text{SEED}}_{\text{GEN}}$, with IPOptD$_{\text{IND}}$ indistinguishable from IPOptD$^{\text{SEED}}_{\text{GEN}}$ at all but a small range of λ values.

6.4 The "optimal" dispatching policies found by IPOPT

In the previous subsection, we have evaluated the *performance* of the dispatching policies resulting from the optimal solutions found by IPOPT when given the optimization problems associated with the ⟨**QRF**, **CID**⟩ family of dispatching rules under various querying rule families **QRF**. In this subsection, we turn to numerically studying the



(a) IPOptD$_{\mathbf{IID}}$ **(b)** IPOptD$_{\mathbf{IND}}$

(c) IPOptD$_{\mathbf{GEN}}$ **(d)** IPOptD$^{\mathsf{SEED}}_{\mathbf{GEN}}$

Fig. 4 IPOPT optimal dispatching policies $s = d = 2$. The parameter $\alpha_1\,(2,\,(1,\,1)) = 0$ in all cases shown

solutions (as found by IPOPT) themselves; that is, we study the best policies found by IPOPT across our parameter settings, although to facilitate comprehensible visualizations, we restrict attention to the setting where $s = d = 2$. We also restrict attention to those (IPOPT-determined) dispatching policy families that performed best based on the study from the previous subsection: IPOptD$_{\mathbf{IID}}$, IPOptD$_{\mathbf{IND}}$, IPOptD$_{\mathbf{GEN}}$, and IPOptD$^{\mathsf{SEED}}_{\mathbf{GEN}}$. We caution that the results presented here may reveal more about the idiosyncrasies of IPOPT than they do about the "true optimal" policies belong to the dispatching policy families of interest.

Figure 4 shows four plots—one for each of the aforementioned families of dispatching policies. In each plot the optimal policy associated with each parameter setting is denoted by a single point. Each point's position in the ternary plot gives the values of $p(2,0)$, $p(1,1)$, and $p(0,2)$, which collectively describe the policy's querying rule; note that $p(2,0)+p(1,1)+p(0,2)=1$. Meanwhile, the color or shading of each point denotes the $\alpha_1(\mathbf{0},(1,1))$ parameter associated with the policy—in all cases plotted, this single parameter uniquely identifies the assignment rule, as in all such cases IPOPT reported $\alpha_1(2,(1,1))=0$, and all other assignment rule parameters can be computed from these two. That is, we find that in all of the optimal policies reported by IPOPT, jobs are never assigned to busy class-1 servers when an idle class-2 server has been queried.

Remark 8 In Fig. 4, we see that the lowest values of $\alpha_1(\mathbf{0},(1,1))$ are associated with those policies where $p(1,1)=0$; however, such policies are precisely those where the choice of $\alpha_1(\mathbf{0},(1,1))$ is immaterial; When we fix $p(1,1)=0$, $\mathbb{E}[T]$ is entirely insensitive to $\alpha_1(\mathbf{0},(1,1))$, as it does not matter how we assign jobs under the query mix $(1,1)$ if the probability of querying according to such a mix is set to zero.

Upon looking at Fig. 4 we immediately observe the following: (i) all $\mathsf{IPOptD}_{\mathbf{IID}}$ policies lie on a "curve" on the ternary plot, (ii) all $\mathsf{IPOptD}_{\mathbf{IND}}$ policies lie on either a curve or satisfy at least one of the $p(2,0)=0$ or $p(0,2)=0$ line segments, (iii) all $\mathsf{IPOptD}_{\mathbf{GEN}}$ policies line on at least one of the $p(2,0)=0$, $p(1,1)=0$, or $p(0,2)=0$ line segments, and (iv) the $\mathsf{IPOptQ}^{\mathsf{SEED}}_{\mathbf{GEN}}$ policies exhibit qualitatively similar behavior to that associated with $\mathsf{IPOptD}_{\mathbf{IND}}$, although far fewer of the points lie on a curve.

Closer inspection reveals that all of the curves alluded to above are indeed the same. In fact, this curve is defined by

$$\left\{(p(2,0),p(1,1),p(0,2))=\left(x^2,2x(1-x),(1-x)^2\right):x\in[0,1]\right\},$$

which is precisely the set of querying rules comprising **IID** when $s=d=2$ (i.e., this is the feasible set of querying rules for the optimization problem associated with $\langle\mathbf{IID},\mathbf{CID}\rangle$ when $s=d=2$). Meanwhile, with some work, one can show that the set of querying rules comprised by **IND** when $s=d=2$ corresponds to the region bounded by the "**IID** curve" and the lines $p(2,0)=0$ and $p(0,2)=0$ (inclusive). We find that *every* querying rule reported as optimal by IPOPT that is contained within **IND**—which includes all of the querying rules of the $\mathsf{IPOptQ}^{\mathsf{SEED}}_{\mathbf{GEN}}$ policies—is specifically contained within the *boundary* of **IND**.

The only dispatching policies reported by IPOPT that do not use **IND** querying are a subset of the $\mathsf{IPOptQ}_{\mathbf{GEN}}$ policies where $p(1,1)=0$; in fact, such policies use **SRC** querying. Meanwhile, the $\mathsf{IPOptQ}^{\mathsf{SEED}}_{\mathbf{GEN}}$ policies are always within **IND**: it appears that seeding IPOPT with the "**IND** solution" when giving it a "**GEN** problem" allows IPOPT to avoid finding dispatching policies using **SRC** querying in favor of those using **IND** querying, ultimately yielding better performance. Moreover, we observe that very few $\mathsf{IPOptD}^{\mathsf{SEED}}_{\mathbf{GEN}}$ policies actually lie on the curve (i.e., very few such policies are in **IID**); the optimal policies tend to be those where either $p(2,0)=0$ or $p(0,2)=0$. Such querying rules are precisely those that are either deterministic,

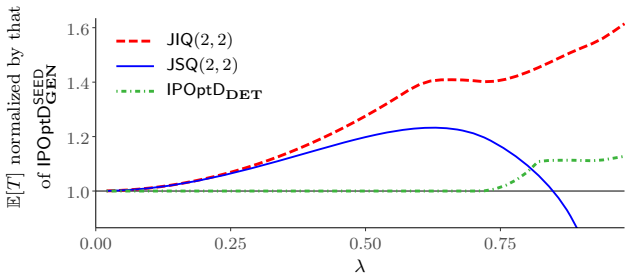


Fig. 5 $\mathbb{E}[T]$ relative to that of $\mathsf{IPOptD}^{\mathsf{SEED}}_{\mathbf{GEN}}$ (i.e., $\mathbb{E}[T]^{\mathsf{DP}}/\mathbb{E}[T]^{\mathsf{IPOptD}^{\mathsf{SEED}}_{\mathbf{GEN}}}$) as a function of λ for the parameter settings where $s = 2$ and $d = 4$, λ varies over $\{0.02, 0.04, \ldots, 0.98\}$, $(q_1, q_2) = (4/5, 1/5)$, and $R_1 = 5$, yielding $(\mu_1, \mu_2) = (25/21, 5/21)$, for the dispatching policies $\mathsf{DP} \in \{\mathsf{JIQ}(2, 2), \mathsf{JSQ}(2, 2), \mathsf{IPOptD}_{\mathbf{DET}}\}$

or "semi-deterministic" in the sense that at least one server of a chosen (fixed) class $i \in \{1, 2\}$ and determines the class for the remaining query randomly.

We leave it to future work to determine whether, how, and to what extent these observations can (i) yield results concerning "true optimal" policies and (ii) be generalized to the cases where $s > 2$ and/or $d > 2$.

Remark 9 While we have avoided plotting results for $\mathsf{IPOptD}_{\mathsf{BR}}$ in the interest of brevity, we note here that we find two kinds of solutions associated with the optimization problem for $\mathsf{IPOptD}_{\mathsf{BR}}$: those where $\alpha_1(2, (1, 1)) = 0$ as in the case of the policies discussed above, and those where $\alpha_1(\mathbf{0}, (1, 1)) = 1$, while $\alpha_1(2, (1, 1)) > 0$. The latter policies are precisely those where jobs are never assigned to a class-2 server when a class-1 server has been queried except—and only sometimes—when said class-2 server is idle and the class-1 server is busy. We conjecture that the need to consider such policies under BR is a result of the fact that BR prohibits any optimization associated with the querying rule.

6.5 Performance under the Deterministic Class Mix querying rule family

Finally, we study **DET** in the case where $s = 2$ and $d = 4$ by comparing the performance of $\mathsf{IPOptD}_{\mathbf{DET}}$ to that of the $\mathsf{JIQ}(2, 2)$ and $\mathsf{JSQ}(2, 2)$ dispatching policies studied in [7]. We consider $\mathbb{E}[T]$ under these three policies, normalized to that under $\mathsf{IPOptD}^{\mathsf{SEED}}_{\mathbf{GEN}}$, for all 12 combinations of R_1 and (q_1, q_2) that are studied in Section 5.2 of [7]. All of these 12 combinations yield similar insights; Fig. 5 shows results for the particular setting in which $R_1 = 5$ and $(q_1, q_2) = (4/5, 1/5)$ (note that this parameterization is not a member of the space discussed in Sect. 6.1, rather it is taken from [7]). The $\mathsf{JIQ}(2, 2)$ dispatching policy often performs considerably worse than $\mathsf{IPOptD}_{\mathbf{DET}}$, the performance of which is indistinguishable from $\mathsf{IPOptD}^{\mathsf{SEED}}_{\mathbf{GEN}}$ except at the highest load values. However, despite its generally strong performance, there is no advantage to using the $\mathsf{IPOptD}_{\mathbf{DET}}$ policy instead of $\mathsf{IPOptD}^{\mathsf{SEED}}_{\mathbf{GEN}}$, as $\mathsf{IPOptD}_{\mathbf{DET}}$ features a substantially higher mean runtime: across the parameter settings shown in Fig. 5 (respectively, all of the parameter settings considered in Section 5.2 of [7]), the mean runtime of $\mathsf{IPOptD}_{\mathbf{DET}}$ is more than 95% higher (respectively, more than 20% higher) than that of $\mathsf{IPOptD}^{\mathsf{SEED}}_{\mathbf{GEN}}$. This high runtime is likely due to the fact that

$\mathsf{IPOptD}_{\mathbf{DET}}$ must solve six smaller subproblems, whereas $\mathsf{IPOptD}^{\mathsf{SEED}}_{\mathbf{GEN}}$ solves only two (larger) optimization problems. Meanwhile, the queue length-aware $\mathsf{JSQ}(2, 2)$ policy tracks $\mathsf{JIQ}(2, 2)$ at low load, but considerably outperforms the other policies, including $\mathsf{IPOptD}^{\mathsf{SEED}}_{\mathbf{GEN}}$, as load approaches 1. This suggests that there is considerable value in investigating **CLD**-driven dispatching policies, which we explore in the next section.

Remark 10 The $\mathsf{JIQ}(2, 2)$ dispatching policy from [7] would be denoted in our framework as $\mathsf{IPOptD}_{\mathsf{DQR}_{(2,2)}} \in \langle \mathbf{DET}, \mathbf{CID} \rangle$, recalling that $\mathsf{DQR}_{\boldsymbol{d}}$ denotes the querying rule that always queries so that $\boldsymbol{D} = \boldsymbol{d}$. Note that while $\mathsf{JSQ}(2, 2) \in \langle \mathsf{DQR}_{(2,2)}, \mathbf{CLD} \backslash (\mathbf{CID} \cup \mathbf{LD}) \rangle$ fits within our framework, $\mathsf{JSQ}(2, 2)$ does *not* use what our framework would describe as JSQ assignment, i.e., $\mathsf{JSQ}(2, 2) \neq \langle \mathsf{DQR}_{(2,2)}, \mathsf{JSQ} \rangle$. Specifically, $\mathsf{JSQ}(2, 2)$ is a variant of $\mathsf{JIQ}(2, 2)$ that, given a set of queried servers, assigns an incoming job to the same *class* that the job would be assigned to under $\mathsf{JIQ}(2, 2)$. However, while $\mathsf{JIQ}(2, 2)$ ultimately assigns the job to a server chosen uniformly at random among those queried from the selected class, $\mathsf{JSQ}(2, 2)$ assigns the incoming job to a server chosen uniformly at random among those queried servers *with the shortest queue(s)* from the selected class.

7 The Class and Length Differentiated family of assignment rules

In this section, we discuss assignment rules in **CLD** beyond those in the **CID** family. After presenting a general structure for **CLD** assignment rules (Sect. 7.1) and discussing the difficulty of analyzing dispatching policies using such assignment rules, we turn our attention to the development of heuristic **CLD**-driven dispatching policies (Sect. 7.3). Simulations suggest that our heuristic policies perform favorably relative to existing dispatching policies presented in the literature (Sect. 7.4). These heuristic policies allow for length-aware assignment while leveraging our analysis of querying rules under **CID** (length-blind) assignment, as presented in the preceding sections.

7.1 Formal presentation of the Class and Length Differentiated family of assignment rules

We proceed to present a generalization of the **CID** family of assignment rules to account for queue lengths (rather than just idle/busy statuses), resulting in the **CLD** family of assignment rules. This family encompasses all static and symmetric assignment rules that can observe both the class (i.e., speed) of, and queue length at, each of the queried servers in assigning a newly arrived job. Throughout this subsection, we introduce a variety of new notation. To aid the reader, this new notation is summarized in Table 1.

Remark 11 When we refer to the "queue length" of/at a server, we mean the number of jobs occupying that server's subsystem: this includes all jobs currently being served by the server and all those waiting for service.

Recall that our study of the **CID** family of assignment rules motivated us to encode the idle/busy statuses of the queried servers by the random vector $\boldsymbol{A}$, which takes on

Table 1 Table of notation for Sect. 7.1 ("r.b." stands for "realized by")

$A_i^{(n)}$ r.b. $a_i^{(n)}$	$\equiv$	Number of queried class-i servers with a queue length of n
$\boldsymbol{A}^{(n)}$ r.b. $\boldsymbol{a}^{(n)}$	$\equiv$	$\{A_0^{(n)}, A_1^{(n)}, \ldots, A_s^{(n)}\}$ r.b. $\{a_0^{(n)}, a_1^{(n)}, \ldots, a_s^{(n)}\}$
$\vec{\boldsymbol{A}}$ r.b. $\vec{\boldsymbol{a}}$	$\equiv$	$\{\boldsymbol{A}^{(0)}, \boldsymbol{A}^{(1)}, \ldots\}$ r.b. $\{\boldsymbol{a}^{(0)}, \boldsymbol{a}^{(1)}, \ldots\}$
$\vec{\mathcal{A}}$	$\equiv$	$\left\{\vec{\boldsymbol{a}}\colon \sum_{i=1}^{s}\sum_{n=0}^{\infty} a_i^{(n)} = d\right\}$; Set of all possible realizations $\vec{\boldsymbol{A}}$
$\alpha_i^{(n)}(\vec{\boldsymbol{a}})$	$\equiv$	Probability that the job is assigned to a class-i server with n jobs when $\vec{\boldsymbol{A}} \equiv \vec{\boldsymbol{a}}$
$\alpha_i(\vec{\boldsymbol{a}})$	$\equiv$	Probability that the job is assigned to a class-i server with n_i^* jobs when $\vec{\boldsymbol{A}} \equiv \vec{\boldsymbol{a}}$
n_i^*	$\equiv$	$\min\left\{m \in \mathbb{N}\colon a_i^{(m)} > 0\right\}$; Queue length of the shortest queue among queried class-i servers

realizations of the form $\boldsymbol{a} \equiv (a_1, \ldots, a_s) \in \mathcal{A} \equiv \{\boldsymbol{a}\colon a_1 + \cdots + a_s \leq d\}$, where a_i is the number of *idle* class-i servers among the d_i queried. Analogously, in studying the **CLD** family it will be helpful to encode the number of class-i servers of *each possible queue length* among the d_i queried. To this end, for each $n \in \mathbb{N} \equiv \{0, 1, \ldots\}$, let $\boldsymbol{A}^{(n)} \equiv \left(A_1^{(n)}, A_2^{(n)}, \ldots, A_s^{(n)}\right)$ be a random vector taking on realizations of the form $\boldsymbol{a}^{(n)} \equiv \left(a_1^{(n)}, a_2^{(n)}, \ldots, a_s^{(n)}\right) \in \mathcal{A}$ where $A_i^{(n)}$ (respectively, $a_i^{(n)}$) is the random variable (respectively, the realization of the random variable) giving the number of queried class-i servers with a queue length of n. Three observations follow immediately from these definitions: (i) $\boldsymbol{A}^{(0)} = \boldsymbol{A}$, (ii) $\boldsymbol{A}^{(n)} \leq \boldsymbol{D}$ (element-wise) for all $n \in \mathbb{N}$, and (iii) $\sum_{n=0}^{\infty} \boldsymbol{A}^{(n)} = \boldsymbol{D}$.

Now let $\vec{\boldsymbol{A}} \equiv \{\boldsymbol{A}^{(0)}, \boldsymbol{A}^{(1)}, \ldots\}$, denote the realizations of this random object by $\vec{\boldsymbol{a}} \equiv \{\boldsymbol{a}^{(0)}, \boldsymbol{a}^{(1)}, \ldots\}$, and denote the set of all such realizations by $\vec{\mathcal{A}} \equiv \left\{\vec{\boldsymbol{a}}\colon \sum_{i=1}^{s}\sum_{n=0}^{\infty} a_i^{(n)} = d\right\}$. Each realized aggregate query state can now be fully described by some $\vec{\boldsymbol{a}} \in \vec{\mathcal{A}}$, allowing us to treat $\boldsymbol{d}$ as a derived quantity: $\boldsymbol{d} = \sum_{n=0}^{\infty} \boldsymbol{a}^{(n)}$.

Formally, a **CLD** assignment rule is uniquely specified by a family of functions $\alpha_i^{(n)}\colon \vec{\mathcal{A}} \to [0, 1]$ parameterized by $(i, n) \in \mathcal{S} \times \mathbb{N}$, where $\alpha_i^{(n)}(\vec{\boldsymbol{a}})$ denotes the probability that a job seeing a query with aggregate state $\vec{\boldsymbol{A}} = \vec{\boldsymbol{a}}$ is assigned to a class-i server with a queue length of n. Clearly, we must have $\alpha_i^{(n)}(\vec{\boldsymbol{a}}) = 0$ whenever $a_i^{(n)} = 0$ and $\sum_{i=1}^{s}\sum_{n=0}^{\infty} \alpha_i^{(n)}(\vec{\boldsymbol{a}}) = 1$ for all $\vec{\boldsymbol{a}} \in \vec{\mathcal{A}}$.

As an illustrative example, let us see how an assignment rule that opts to *ignore* queue length information (apart from idleness information) can be implemented via such a family of functions. Specifically, let us consider an assignment rule from the **CID** family (noting that such an assignment rule is also a member of the **CLD** family, as **CID** $\subseteq$ **CLD**), defined (as in Sect. 4.1) by some family of functions $\alpha_i\colon \bar{\mathcal{S}} \times \mathcal{D} \to [0, 1]$ parameterized by $i \in \mathcal{S}$, where $\alpha_i(j, \boldsymbol{d})$ denotes the probability that a job is assigned to a queried class-i server, given that it sees $J = j$ as the class of the fastest idle server

and a query mix $\boldsymbol{D} = \boldsymbol{d}$. In this case, letting $j \equiv \min\{\ell \colon a_\ell^{(0)} > 0\}$ (where we again use the convention that $\min \emptyset \equiv s + 1$) we can define $\alpha_i^{(n)}(\vec{\boldsymbol{a}})$ in terms of $\alpha_i(j, \boldsymbol{d})$ as follows:

$$\alpha_i^{(0)}(\vec{\boldsymbol{a}}) = \begin{cases} \alpha_i(j, \boldsymbol{d}) & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \tag{26}$$

$$\alpha_i^{(n)}(\vec{\boldsymbol{a}}) = \begin{cases} \dfrac{a_i^{(n)} \alpha_i(j, d)}{\sum_{m=1}^{\infty} a_i^{(m)}} & \text{if } i \leq j \\ 0 & \text{otherwise} \end{cases} \qquad (\forall n \geq 1). \tag{27}$$

Equation (26) gives the probability of assigning the job to an idle class-i server, and Eq. (27) gives the probability of assigning the job to a busy class-i server with n jobs in its queue. While the first equation is straightforward, the latter becomes clear by making the following observation: if we ignore queue lengths beyond idle/busy statuses, once we have chosen to assign the job to a busy class-i server, we choose one such server at random, and hence, the job is sent to a class-i server with a queue length of n with probability $a_i^{(n)} \big/ \sum_{m=1}^{\infty} a_i^{(m)}$.

Now observe that if one opts to make full use of queue length information, then whenever one assigns a job to a class-i server it is naturally favorable to assign the job to the class-i server with the shortest queue among those queried. If we prune the space of **CLD** assignment rules in this fashion (which would leave out the **CID** assignment rules), then we can instead uniquely specify assignment rules by a family of functions $\alpha_i \colon \vec{\mathcal{A}} \to [0, 1]$ that are parameterized only by $i \in \mathcal{S}$ (rather than also being parameterized by $n \in \mathbb{N}$). In this case, letting $n_i^* \equiv \min\left\{m \in \mathbb{N} \colon a_i^{(m)} > 0\right\}$ (i.e., letting n_i^* be the queue length of the shortest queue among queried class-i servers) for each $i \in \mathcal{S}$ (with $n_i^* \equiv \infty$ whenever $d_i = 0$), we can express (the original) $\alpha_i^{(n)}(\vec{\boldsymbol{a}})$ in terms of (the new) $\alpha_i(\vec{\boldsymbol{a}})$ as follows:

$$\alpha_i^{(n)}(\vec{\boldsymbol{a}}) = \begin{cases} \alpha_i(\vec{\boldsymbol{a}}) & \text{if } n = n_i^* \\ 0 & \text{otherwise} \end{cases}.$$

7.2 Examples of Class and Length Differentiated assignment rules

Two examples of assignment rules in **CLD\CID** that we can specify using families of functions $\alpha_i \colon \vec{\mathcal{A}} \to [0, 1]$ (where we again use n_i^* to denote the queue length of the shortest queue among queried class-i servers) include JSQ and SED.

Remark 12 When we refer to JSQ and SED, we are referring to just the *assignment rules*, rather than the traditional JSQ and SED *dispatching policies* studied in the literature in small scale settings, or the JSQ-d and SED-d *dispatching policies*, which in our framework are referred to as the ⟨UNI, JSQ⟩ and ⟨UNI, SED⟩ dispatching policies, respectively. Moreover, note that JSQ is actually a member of the **LD** family—a sub-

family of **CLD** that allows for leveraging queue length information, but is blind to server classes (i.e., speeds).

We discuss these two rules (i.e., JSQ and SED) in greater detail—together with a third assignment rule in **CLD****CID**, Shortest Expected Wait (SEW), which we introduce here—below:

– Join the Shortest Queue (JSQ) is an individual assignment rule that is a member of the **LD** family (and therefore also the **CLD** family) that assigns the job to a queried server (chosen uniformly at random) among those with the shortest queue (regardless of their class). It is specified by

$$\alpha_i(\vec{\boldsymbol{a}}) = \frac{a_i^{\left(n_i^*\right)} \prod_{\ell=1}^{s} I\{n_i^* \le n_\ell^*\}}{\sum_{i'=1}^{s} a_{i'}^{\left(n_{i'}^*\right)} \prod_{\ell=1}^{s} I\left\{n_{i'}^* \le n_\ell^*\right\}}.$$

– Shortest Expected Delay (SED) is an individual assignment rule that is a member of the **CLD** family that assigns the job to a queried server (chosen uniformly at random) among those on which the job would complete soonest in expectation *under the assumption of* First Come First Serve (FCFS) *scheduling* (regardless of the actual scheduling rule being implemented). By observing that the expected delay experienced by a job (under FCFS scheduling) that is assigned to a class-i server with n other jobs already in its queue is $(n+1)/\mu_i$, we find that the SED assignment rule is specified by

$$\alpha_i(\vec{\boldsymbol{a}}) = \frac{a_i^{\left(n_i^*\right)} \prod_{\ell=1}^{s} I\left\{\frac{n_i^*+1}{\mu_i} \le \frac{n_\ell^*+1}{\mu_\ell}\right\}}{\sum_{i'=1}^{s} a_{i'}^{\left(n_{i'}^*\right)} \prod_{\ell=1}^{s} I\left\{\frac{n_{i'}^*+1}{\mu_{i'}} \le \frac{n_\ell^*+1}{\mu_\ell}\right\}}.$$

– Shortest Expected Wait (SEW) is an individual assignment rule that is a member of the **CLD** family that assigns the job to a queried server (chosen uniformly at random) among those on which the job would *enter service* soonest in expectation *under the assumption of* First Come First Serve (FCFS) *scheduling* (regardless of the actual scheduling rule being implemented). Unlike SED, SEW does not account for the expected size of the arriving job, $1/\mu_i$. By observing that the expected waiting time until entering service experienced by a job (under FCFS scheduling) that is assigned to a class-i server with n other jobs already in its queue is n/μ_i, we find that the SEW assignment rule is specified by

$$\alpha_i(\vec{\boldsymbol{a}}) = \frac{a_i^{\left(n_i^*\right)} \prod_{\ell=1}^{s} I\left\{\frac{n_i^*}{\mu_i} \le \frac{n_\ell^*}{\mu_\ell}\right\}}{\sum_{i'=1}^{s} a_{i'}^{\left(n_{i'}^*\right)} \prod_{\ell=1}^{s} I\left\{\frac{n_{i'}^*}{\mu_{i'}} \le \frac{n_\ell^*}{\mu_\ell}\right\}}.$$

Remark 13 Our nomenclature is perhaps imperfect, as *delay* is sometimes used to refer to time in queue, but here we are using *delay* (in the name of SED—which we inherit

from the literature) to refer to the *time in system* and *wait* (in the name of SEW) to refer to the *time in queue*.

We also introduce a variant of each of the above policy that breaks "ties" in favor of faster servers: (i) JSQ*, (ii) SED*, and (iii) SEW* act like the (i) JSQ, (ii) SED, and (iii) SEW assignment rules, except jobs are always assigned to one the *fastest* servers (chosen uniformly at random) among those queried servers that have the (i) shortest queue, (ii) the shortest expected delay (i.e., at which the job would experience the shortest expected response time), and (iii) the shortest wait (i.e., at which the job would experience the shortest time in queue), respectively. Note that while JSQ $\in$ **LD**, JSQ* $\in$ **CLD****LD**, as JSQ* makes use of class information in breaking ties between queried servers with the same queue length. These rules are specified by the following:

$$\alpha_i(\vec{\boldsymbol{a}}) = \prod_{\ell=1}^{i-1} I\{n_i^* < n_\ell^*\} \prod_{\ell=i+1}^{s} I\{n_i^* \le n_\ell\}. \tag{JSQ*}$$

$$\alpha_i(\vec{\boldsymbol{a}}) = \prod_{\ell=1}^{i-1} I\left\{\frac{n_i^*+1}{\mu_i} < \frac{n_\ell^*+1}{\mu_\ell}\right\} \prod_{\ell=i+1}^{s} I\left\{\frac{n_i^*+1}{\mu_i} \le \frac{n_\ell^*+1}{\mu_\ell}\right\} \tag{SED*}$$

$$\alpha_i(\vec{\boldsymbol{a}}) = \prod_{\ell=1}^{i-1} I\left\{\frac{n_i^*}{\mu_i} < \frac{n_\ell^*}{\mu_\ell}\right\} \prod_{\ell=i+1}^{s} I\left\{\frac{n_i^*}{\mu_i} \le \frac{n_\ell^*}{\mu_\ell}\right\} \tag{SEW*}.$$

7.3 A heuristic for finding strong dispatching policies

The analysis of general assignment rules in the **CLD** family introduces intractability issues that we were able to avoid in our analysis of the **CID** family of assignment rules. There are two key challenges for identifying strong dispatching policies with assignment rules in **CLD****CID**. First, while the α_i functions designating the **CID** policies had a finite domain ($\mathcal{A} \times \mathcal{D}$, and after subsequent pruning $\bar{\mathcal{S}} \times \mathcal{D}$), those functions specifying assignment rules for **CLD** policies—even with the pruning introduced in Sect. 7.1—have an infinite domain ($\bar{\mathcal{A}}$). Hence, the **CLD** assignment rules span an infinite dimensional space, unlike the finite-dimensional polytopes spanned by their **CID** counterparts (see Appendix D of [12] for details); the former generally precludes straightforward optimization, while the latter facilitates it.

The second challenge associated with identifying strong dispatching policies with assignment rules that take queue lengths into account is the lack of exact performance analysis for most dispatching policies in **CLD****CID**. Thus, even if we could solve an infinite-dimensional optimization problem (i.e., even if we could overcome the first challenge), it is challenging to formulate the objective function for such an optimization problem.

We attempt to jointly overcome these challenges by populating a roster of heuristic dispatching policies designed based on the **CID**-driven policies of Sects. 3–6. In the next subsection, we show (via simulation) that many of these policies perform well relative to the aforementioned **CID**-driven policies.

We address the first challenge (i.e., the infinite dimensional space spanned by the **CLD** assignment rules) by limiting ourselves to the example assignment rules discussed in Sect. 7.2: JSQ, SED, SEW, JSQ*, SED*, and SEW*. Note that these are *individual* assignment rules, rather than assignment rule *families*, which obviates the need for optimizing continuous probabilistic parameters.

One hopes that even without sophisticated fine-tuned probabilistic parameters, the greedy SED and SEW assignment rules (with or without class-based tie-breaking) still manage to yield stronger performance than the length-blind **CID**-driven dispatching policies—at least when paired with a judiciously chosen querying rule. Meanwhile, by studying JSQ, we can assess the extent to which queue-length information can lead to strong performance even in the absence of heterogeneity-awareness in the assignment decision.

The second challenge (i.e., the lack of performance analysis as a basis for optimization), then, reduces to the problem of choosing a querying rule to use in conjunction with our six chosen assignment rules. We propose three ideas for choosing an appropriate querying rule—ultimately, each approach will add additional dispatching policies to our roster.

The first idea for choosing a querying rule is to use the same approach that we are taking on the assignment side. That is, we can limit ourselves to one (or some small number of) individual querying rule(s). Of the two specific individual querying rules discussed in this paper, UNI does not guarantee stability, while BR does (see Sect. 4.2 for details). For this reason, we add the following six dispatching policies to our roster: $\langle$BR, JSQ$\rangle$, $\langle$BR, SED$\rangle$, $\langle$BR, SEW$\rangle$, $\langle$BR, JSQ$^\star\rangle$, $\langle$BR, SED$^\star\rangle$, and $\langle$BR, SEW$^\star\rangle$.

The remaining two ideas involve leveraging the diversity of querying rules available within the families studied throughout this paper, as it is unnecessarily restrictive to only consider dispatching policies that involve no optimization (i.e., that involve combining a specific individual querying rule with a specific individual assignment rule, as above). The broadest querying rule family that we have studied is, of course, **GEN**; unfortunately, analyzing exact mean response times under, e.g., $\langle$**GEN**, JSQ$\rangle$ and $\langle$**GEN**, SED$\rangle$ appears to be intractable.

Our second idea presents one way to overcome this tractability limitation: we restrict attention to the **SRC** family of querying rules, where one selects a class at random (according to some fixed distribution) upon the arrival of each job and then queries d servers of that class. **SRC** querying eliminates the possibility of needing to make assignment decisions between servers running at different speeds, meaning that pairing **SRC** with any of our six individual assignment rules yields the same dispatching policy; we will refer to this single policy as $\langle$**SRC**, JSQ$\rangle$. Furthermore, because assignment decisions are always made among servers of the same speed, the analysis of $\langle$**SRC**, JSQ$\rangle$ reduces to that of s independent homogeneous systems under JSQ. This exact analysis allows us to use IPOPT to find the IPOptD$_{\langle\mathbf{SRC},\mathsf{JSQ}\rangle}$ dispatching policy, which we add to our roster of dispatching policies.

Remark 14 As noted above, the analysis of $\langle$**SRC**, JSQ$\rangle$ reduces to that of s independent homogeneous systems under the $\langle$UNI, JSQ$\rangle$ dispatching policy (referred to in the literature as JSQ-d). The mean response time in such homogeneous systems was analyzed exactly in [19, 32]. We then rely on IPOPT to determine the "optimal" $\hat{p}(i)$

parameters for $i \in \mathcal{S}$ (i.e., the probability of querying each single class i; see Sect. 5.5). We further note that $\langle \mathbf{SRC}, \mathsf{JSQ} \rangle$ was previously studied in the case of $s = 2$, under the Processor Sharing (PS) scheduling discipline, in [20].

Our third idea is to use a novel heuristic that leverages our previous study of $\langle \mathbf{QRF}, \mathbf{CID} \rangle$ dispatching policies from Sects. 3–6. Our heuristic constructs a dispatching policy by combining an individual querying rule found by IPOPT and any one of our six individual assignment rules. Specifically, the heuristic uses the $\mathsf{IPOptQ}^{\mathsf{SEED}}_{\mathbf{GEN}}$ querying rule (i.e., the querying rule yielded by the IPOPT solution to the optimization problem associated with $\langle \mathbf{GEN}, \mathbf{CID} \rangle$, seeded with the IPOPT solution for $\langle \mathbf{IND}, \mathbf{CID} \rangle$). Note that our choice of querying rule (i.e., $\mathsf{IPOptQ}^{\mathsf{SEED}}_{\mathbf{GEN}}$) is not contingent on the choice of assignment rule, as tractability necessitates foregoing any kind of joint optimization. To this end, we complete our roster with the following six policies: $\left\langle \mathsf{IPOptQ}^{\mathsf{SEED}}_{\mathbf{GEN}}, \mathsf{JSQ} \right\rangle$, $\left\langle \mathsf{IPOptQ}^{\mathsf{SEED}}_{\mathbf{GEN}}, \mathsf{SED} \right\rangle$, $\left\langle \mathsf{IPOptQ}^{\mathsf{SEED}}_{\mathbf{GEN}}, \mathsf{SEW} \right\rangle$, $\left\langle \mathsf{IPOptQ}^{\mathsf{SEED}}_{\mathbf{GEN}}, \mathsf{JSQ}^{\star} \right\rangle$, $\left\langle \mathsf{IPOptQ}^{\mathsf{SEED}}_{\mathbf{GEN}}, \mathsf{SED}^{\star} \right\rangle$, and $\left\langle \mathsf{IPOptQ}^{\mathsf{SEED}}_{\mathbf{GEN}}, \mathsf{SEW}^{\star} \right\rangle$.

7.4 Simulation-driven performance evaluation

We simulate the $\langle \mathsf{BR}, \mathsf{JSQ} \rangle$, $\langle \mathsf{BR}, \mathsf{SED} \rangle$, $\left\langle \mathsf{IPOptQ}^{\mathsf{SEED}}_{\mathbf{GEN}}, \mathsf{JSQ} \right\rangle$, and $\left\langle \mathsf{IPOptQ}^{\mathsf{SEED}}_{\mathbf{GEN}}, \mathsf{SED} \right\rangle$ dispatching policies in a system with $k = 3000$ servers under the same collection of parameter settings studied in Fig. 3 in Sect. 6.3. We simulate 10 000 000 arrivals to the system and record the observed response time for each. We then average these values (discarding the first 1 000 000 to allow the system to "reach a steady state" where the running average response time was observed to stabilize) to obtain a $\mathbb{E}[T]$ value under each policy at each value of λ. We omit results for $\lambda \in \{0.92, 0.94, 0.96, 0.98\}$ as the observed variance of response times across successive runs exceeded 1% of the mean in these cases. Running longer simulations with more arrivals could reduce the variance in these cases, but doing so would have been prohibitively expensive in terms of the simulation runtime.

In Fig. 6, we plot the simulated $\mathbb{E}[T]$ of each of the above dispatching policies—as well as the computational (non-simulated, based on the assumption where $k \to \infty$) results for $\mathsf{IPOptD}_{\mathbf{IID}}$ and $\mathsf{IPOptD}_{\langle \mathbf{SRC}, \mathsf{JSQ} \rangle}$—normalized by the $\mathbb{E}[T]$ value $\mathsf{IPOptD}^{\mathsf{SEED}}_{\mathbf{GEN}} \equiv \left\langle \mathsf{IPOptQ}^{\mathsf{SEED}}_{\mathbf{GEN}}, \mathsf{IPOptA}^{\mathsf{SEED}}_{\mathbf{GEN}} \right\rangle$ as a function of λ. We examined a number of other parameter settings and chose this parameter setting in order to make the trends more salient, although qualitatively similar results are exhibited across most of the parameter settings observed.

We observe that at low values of λ, the BR-driven policies perform poorly, because they occasionally query no servers of the fastest class, even though under such light traffic one would like to discard all but the fastest servers. These policies continue to be the worst performers—as, in addition to using slow servers, queues begin to build up at these servers—until a certain point where the gap between these policies and the others begins to close. Meanwhile, in this low-λ regime, all of the other policies (including the normalizing policy, $\mathsf{IPOptD}^{\mathsf{SEED}}_{\mathbf{GEN}}$, which does not make use of queue length information) perform near-identically, because all of them query essentially

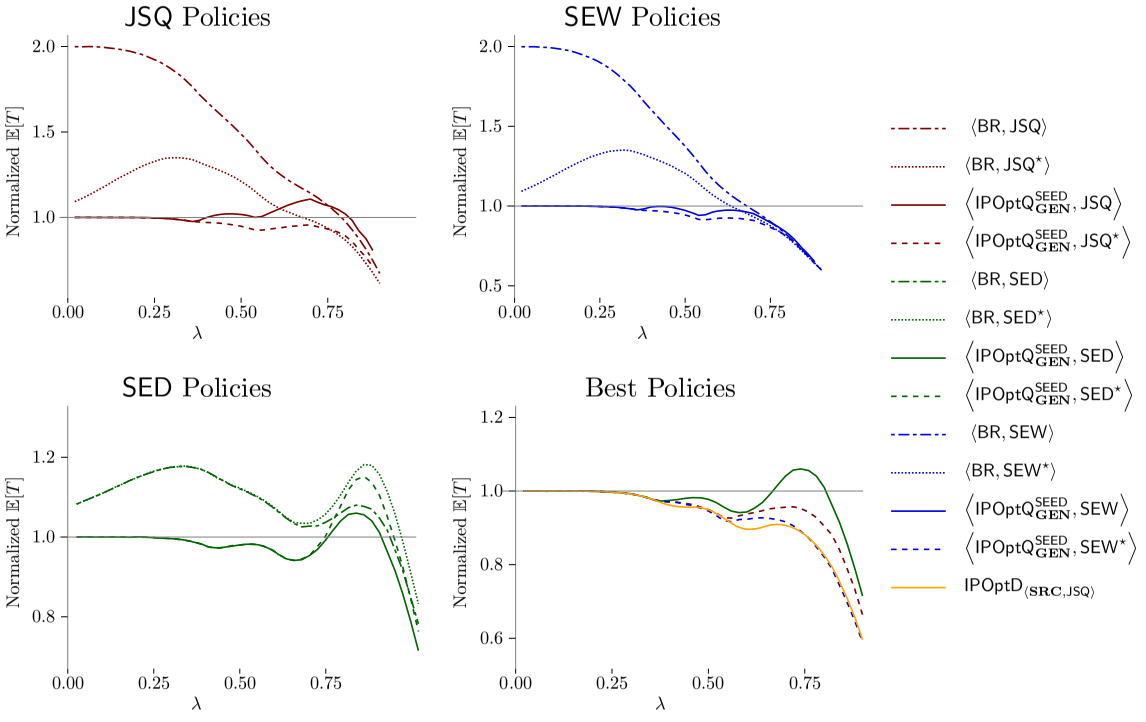


Fig. 6 $\mathbb{E}[T]$ relative to that of IPOptD$^{\text{SEED}}_{\text{GEN}}$ (i.e., $\mathbb{E}[T]^{\text{DP}}/\mathbb{E}[T]^{\text{IPOptD}^{\text{SEED}}_{\text{GEN}}}$) as a function of λ for the parameter settings where $s = d = 3$, with λ varying over $\{0.02, 0.04, \ldots, 0.90\}$, $(q_1, q_2, q_3) = (1/3, 1/6, 1/2)$ and $(R_1, R_2) = (5, 2)$, yielding $(\mu_1, \mu_2, \mu_3) = (2, 4/5, 2/5)$, for the dispatching policies DP constructed from the BR and IPOptQ$^{\text{SEED}}_{\text{GEN}}$ querying rules when paired with the JSQ and JSQ* assignment rules (upper left), the SED and SED* assignment rules (lower left), and the SEW and SEW* assignment rules (upper right). In the lower right, we compare the "best" policy (obtaining the lowest $\mathbb{E}[T]$ value on the majority of λ values) from each of the other three subfigures: $\left\langle \text{IPOptQ}^{\text{SEED}}_{\text{GEN}}, \text{JSQ}^\star \right\rangle$, $\left\langle \text{IPOptQ}^{\text{SEED}}_{\text{GEN}}, \text{SEW}^\star \right\rangle$, and $\left\langle \text{IPOptQ}^{\text{SEED}}_{\text{GEN}}, \text{SED} \right\rangle$. We also include the IPOptD$_{\langle \mathbf{SRC}, \text{JSQ}\rangle}$ dispatching policy in the lower right subfigure. All $\mathbb{E}[T]$ values were obtained through simulation except for those associated with IPOptD$_{\langle \mathbf{SRC}, \text{JSQ}\rangle}$ and the normalizing policy, IPOptQ$^{\text{SEED}}_{\text{GEN}}$. Note that not all subfigures use the same scale for the vertical axis

only the fastest servers, and most of these servers are idle, rendering the assignment rule immaterial.

At higher λ, we enter an "assignment-driven regime," where all of our **CLD**-based policies outperform the **CID**-based $\mathsf{IPOptD}^{\mathsf{SEED}}_{\mathbf{GEN}}$ policy. We call this an "assignment-driven regime" because the performance of the policies become differentiated from one another primarily on the basis of their assignment rules. That is, even though, e.g., $\mathsf{IPOptD}^{\mathsf{SEED}}_{\mathbf{GEN}} \equiv \left\langle \mathsf{IPOptQ}^{\mathsf{SEED}}_{\mathbf{GEN}}, \mathsf{IPOptA}^{\mathsf{SEED}}_{\mathbf{GEN}} \right\rangle$ and $\left\langle \mathsf{IPOptQ}^{\mathsf{SEED}}_{\mathbf{GEN}}, \mathsf{SED} \right\rangle$ use the same querying rule—which is optimized for use with **CID** assignment—the latter achieves better performance because the advantage of **CLD**-based assignment outweighs the benefit of jointly optimizing the querying and assignment rules. The existence of such a regime is a result of the fact that, in heavy traffic, all querying rules that maintain the system's stability must result in similar λ_i^{B} values, meaning that dispatching policies that stabilize the system are distinguished from one another primarily in terms of their assignment rules.

Two of the dispatching policies under consideration emerge as the consistently strongest performers: $\left\langle \mathsf{IPOptQ}^{\mathsf{SEED}}_{\mathbf{GEN}}, \mathsf{SEW}^{\star} \right\rangle$ and $\mathsf{IPOptD}_{\langle \mathbf{SRC}, \mathsf{JSQ} \rangle}$. It may appear surprising that $\left\langle \mathsf{IPOptQ}^{\mathsf{SEED}}_{\mathbf{GEN}}, \mathsf{SEW}^{\star} \right\rangle$ consistently outperforms its counterparts that make use of SED or SED*. It turns out that (assuming a judiciously chosen querying rule) it is crucial to make use of idle queried servers whenever possible; unlike SED and SED*, SEW and SEW* never send jobs to busy servers when an idle server has been queried. The strong performance of $\left\langle \mathsf{IPOptQ}^{\mathsf{SEED}}_{\mathbf{GEN}}, \mathsf{SEW}^{\star} \right\rangle$ also highlights the value of our analysis of **CID**-based dispatching policies: even though such length-unaware policies do not themselves necessarily achieve the best performance (especially at high λ), we see that the **CID**-based optimization of the querying rule allows for the development of considerably stronger **CLD**-based policies. Such policies are likely to be difficult to discover using, e.g., a grid search approach.

Remark 15 In fact, the best-performing $\langle \mathsf{QR}, \mathsf{SEW}^{\star} \rangle$ policies found by a simulation-driven grid search (over $\mathsf{QR} \in \mathbf{GEN}$) performed no better than $\left\langle \mathsf{IPOptQ}^{\mathsf{SEED}}_{\mathbf{GEN}}, \mathsf{SEW}^{\star} \right\rangle$. We performed this grid search to validate the performance of our heuristic policies, but, even with a fairly coarse search, this took on the order of an hour for a single value of λ, while one can obtain a better performing $\mathsf{IPOptQ}^{\mathsf{SEED}}_{\mathbf{GEN}}$ policy in mere seconds by leveraging our optimization problems rather than simulations. Our experience leads us to conclude that the high dimensionality of the **GEN** family renders such searches poorly suited for practice.

Meanwhile, $\mathsf{IPOptD}_{\langle \mathbf{SRC}, \mathsf{JSQ} \rangle}$ also exhibits consistently strong performance across the range of λ values. We can attribute its excellent performance to the fact that—unlike the other **CLD**-based policies under consideration—$\mathsf{IPOptD}_{\langle \mathbf{SRC}, \mathsf{JSQ} \rangle}$ features a querying rule that is optimized for use with its own assignment rule, rather than for use with a **CID** assignment rule.

8 Conclusion

This paper provides a comprehensive framework for dispatching in scalable systems in the presence of heterogeneous servers, by examining two separate components of the dispatching policy: the querying rule and the assignment rule. We highlight tradeoffs associated with the choice of each rule: less restrictive families of querying rules allow for lower mean response times at the cost of increased solution runtime. Meanwhile, some assignment rules lend themselves to tractable analysis, while others boast better performance (insofar as observed from simulations). Moreover, at some system loads, both the querying and assignment decision can be crucial, while at more extreme loads, one decision plays a more dominant role over the other (subject to stability constraints).

Our framework illuminates several potentially fruitful areas of future work. First, this paper restricts attention to symmetric and static querying and assignment rules. There has been little study of asymmetric rules (of either kind) in the literature when all jobs are *ex ante* identical (i.e., when dispatching is size blind and all jobs are equally important). Yet we believe that the explicit and separate consideration of—and study of the interaction between—querying and assignment rules suggests how asymmetry might be exploited to develop superior dispatching policies even when jobs are *ex ante* identical. A judiciously chosen asymmetric assignment rule may be able to synergistically exploit the asymmetry introduced by the querying rule. Meanwhile, future research could allow for dynamic, rather than merely static, querying and/or assignment rules, permitting the incorporation of round-robin-like dispatching decisions into our framework, which would necessitate novel analysis. Another direction for future work involves generalizing our framework to heterogeneous systems with multiple dispatchers, as considered in [29, 39]. Such a generalization likely would require a different approach for selecting policy parameters, as each dispatcher possesses only a partial view of the system's arrival process.

While this paper presents a comprehensive examination of querying rules within the space restricted by the aforementioned assumptions, the bulk of our analysis focused on the **CID** family, where assignment rules eschew making decisions on the basis of detailed queue length information in favor of idleness information. The performance analysis of even **CLD**-based dispatching policies remains an open problem, and while the explicit analysis of the set of all **CLD** assignment rules (in conjunction with querying rules coming from, say **GEN**) may prove intractable, we anticipate that many policies incorporating more detailed—if still restricted—queue length information are amenable to analysis. Moreover, we imagine that many such policies may outperform the **CID**-driven dispatching policies studied in this paper.

Finally, there remain open problems on the theoretical front. For example, throughout our analysis asymptotic independence remains an assumption (although one that is validated by simulation) for which future work may provide a universal rigorous justification (as past work has for more restricted special cases of our framework). There is also ample room for optimization theory to shed further light on the structure of the optimization problems presented in this work.

A Appendix: Tables of notation

See Tables 2, 3, 4 and 5.

Table 2 Querying rule and policy abbreviations

Querying rule families		
DET	$\equiv$	**Deterministic Class Mix**
GEN	$\equiv$	**General Class Mix**
IND	$\equiv$	**Independent Querying**
IID	$\equiv$	**Independent and Identically Distributed Querying**
QRF	$\equiv$	Generic notation for an arbitrary querying rule family
SFC	$\equiv$	**Single Fixed Class**
SRC	$\equiv$	**Single Random Class**
Individual querying rules		
BR	$\equiv$	Balanced Routing querying rule
$\mathsf{DQR}_{\boldsymbol{d}}$	$\equiv$	Individual querying rule in **DET** that always queries according to class mix $\boldsymbol{d}$
$\mathsf{IPOptQ}_{\langle \mathbf{QRF}, \mathbf{ARF} \rangle}$	$\equiv$	Querying rule used by the "optimal" policy in $\langle$**QRF**, **ARF**$\rangle$ found by IPOPT
$\mathsf{IPOptQ}_{\mathbf{QRF}}$	$\equiv$	Abbreviated notation for $\mathsf{IPOptQ}_{\langle \mathbf{QRF}, \mathbf{CID} \rangle}$
$\mathsf{IPOptQ}^{\mathsf{SEED}}_{\mathbf{GEN}}$	$\equiv$	Querying rule used by the $\mathsf{IPOptD}^{\mathsf{SEED}}_{\mathbf{GEN}}$ dispatching policy
QR	$\equiv$	Generic notation for an arbitrary individual querying rule
UNI	$\equiv$	Uniform Querying

Table 3 Assignment rule and policy abbreviations

Assignment rule families		
ARF	≡	Generic notation for an arbitrary assignment rule family
CD	≡	**Class Differentiated**
CID	≡	**Class and Idleness Differentiated**
CLD	≡	**Class and Length Differentiated**
ID	≡	**Idleness Differentiated**
LD	≡	**Length Differentiated**
Individual assignment rules		
AR	≡	Generic notation for an arbitrary individual assignment rule
$\mathsf{IPOptA}_{\langle \mathbf{QRF}, \mathbf{ARF} \rangle}$	≡	Assignment rule used by the "optimal" policy in ⟨**QRF**, **ARF**⟩ found by IPOPT
$\mathsf{IPOptA}_{\mathbf{QRF}}$	≡	Abbreviated notation for $\mathsf{IPOptA}_{\langle \mathbf{QRF}, \mathbf{CID} \rangle}$
$\mathsf{IPOptA}^{\mathsf{SEED}}_{\mathbf{GEN}}$	≡	Assignment rule used by the $\mathsf{IPOptD}^{\mathsf{SEED}}_{\mathbf{GEN}}$ dispatching policy
JSQ	≡	Join the Shortest Queue assignment rule
JSQ⋆	≡	Variant of JSQ where ties are broken in favor of faster classes
ND	≡	Non-Differentiated
SED	≡	Shortest Expected Delay assignment rule
SED⋆	≡	Variant of SED where ties are broken in favor of faster classes
SEW	≡	Shortest Expected Wait assignment rule
SEW⋆	≡	Variant of SEW where ties are broken in favor of faster classes

Table 4 Dispatching rule and policy abbreviations

Dispatching Policy Families		
⟨**QRF**, **ARF**⟩	≡	Dispatching policy family using **QRF** querying and **ARF** assignment
⟨**QRF**, AR⟩	≡	Disp. policy family using **QRF** querying with the individual AR assignment rule
⟨QR, **ARF**⟩	≡	Disp. policy family using the individual QR querying rule with **ARF** assignment
DPF	≡	Generic notation for an arbitrary dispatching policy
Individual dispatching policies		
⟨QR, AR⟩	≡	Dispatching policy using the QR querying rule and AR assignment rule
DP	≡	Generic notation for an individual dispatching policy
$\mathsf{IPOptD}_{\langle \mathbf{QRF}, \mathbf{ARF} \rangle}$	≡	"Optimal" dispatching policy in ⟨**QRF**, **ARF**⟩ found by IPOPT
$\mathsf{IPOptD}_{\mathbf{QRF}}$	≡	Abbreviated notation for $\mathsf{IPOptD}_{\langle \mathbf{QRF}, \mathbf{CID} \rangle}$
$\mathsf{IPOptD}^{\mathsf{SEED}}_{\mathbf{GEN}}$	≡	Variant of the $\mathsf{IPOptD}_{\mathbf{GEN}}$ dispatching policy, where the associated optimization problem is "seeded" with the parameters of the $\mathsf{IPOptD}_{\mathbf{IND}}$ policy

Table 5 List of notations

$\alpha_i(\boldsymbol{a}, \boldsymbol{d})$	$\equiv$	Probability that the job is assigned to a class-i server when $\boldsymbol{A} = \boldsymbol{a}$ and $\boldsymbol{D} = \boldsymbol{d}$
$\alpha_i(j, \boldsymbol{d})$	$\equiv$	Notation for $\alpha_i(\boldsymbol{a}, \boldsymbol{d})$ when $\boldsymbol{A} = \boldsymbol{a}$ is such that the fastest idle queried server belongs to class j
$\boldsymbol{A}$	$\equiv$	$(A_1, \ldots, A_s)$; random vector representing the number of queried idle servers of each class
A_i	$\equiv$	Random variable representing the number of queried idle class-i servers
$\mathcal{A}$	$\equiv$	$\{\boldsymbol{a}\colon a_1 + \cdots + a_s \leq d\}$; Set of all possible values of the random vector $\boldsymbol{A}$ (given a fixed d)
$\boldsymbol{a}$	$\equiv$	$(a_1, \ldots, a_s)$; realization of the random vector $\boldsymbol{A}$
a_i	$\equiv$	Realization of the random variable A_i
B_i	$\equiv$	Busy period duration at a class-i server
$b_i(\boldsymbol{d})$	$\equiv$	$\prod_{\ell=1}^{i-1} \rho_\ell^{d_\ell}$; probability that all queried servers faster than those of class-i in are busy
d	$\equiv$	Total number of servers to be queried
$\boldsymbol{D}$	$\equiv$	$(D_1, \ldots, D_s)$; random vector representing the number of queried servers of each class
D_i	$\equiv$	Random variable representing the number of queried class-i servers
$\mathcal{D}$	$\equiv$	$\{\boldsymbol{d}\colon d_1 + \cdots + d_s = d\}$; set of all possible values of the random vector $\boldsymbol{D}$ (given a fixed d)
$\boldsymbol{d}$	$\equiv$	$(d_1, \ldots, d_s)$; realization of $\boldsymbol{D}$ representing the class mix
d_i	$\equiv$	Realization of D_i representing the number of class-i servers in the query
$\gamma(j, \boldsymbol{d})$	$\equiv$	Mapping where $\alpha_i(j, \boldsymbol{d}) = \alpha_i(j, \gamma(j, \boldsymbol{d}))$ under our assignment rule pruning for all $(i, j, \boldsymbol{d})$
$h(\boldsymbol{d})$	$\equiv$	$\min\{\ell \in \mathcal{S}\colon d_\ell > 0\}$; the fastest class included in a query when $\boldsymbol{D} = \boldsymbol{d}$
J	$\equiv$	$\min\{j \in \mathcal{S}\colon A_j > 0\}$ (with $\min \emptyset \equiv s + 1$); class of the fastest idle queried server
$\mathcal{J}_i(\boldsymbol{d})$	$\equiv$	Set of classes $j > i$ for which $(i, j, \gamma(j, \boldsymbol{d})) \in \mathcal{T}$
k	$\equiv$	Total number of servers
k_i	$\equiv$	Number of class i servers for $i \in \mathcal{S}$
λ	$\equiv$	Overall mean arrival rate to a server
λ_i	$\equiv$	Mean arrival rate to a class-i server
λ_i^{B}	$\equiv$	Mean arrival rate to a busy class-i server
λ_i^{I}	$\equiv$	Mean arrival rate to an idle class-i server
μ_i	$\equiv$	Speed of a class-i server
$\mathcal{P}$	$\equiv$	Set of $(j, \boldsymbol{d})$ pairs that can form a triple with some $i \in \mathcal{S}$ so that $(i, j, \boldsymbol{d}) \in \mathcal{T}$
$\mathcal{P}(\boldsymbol{d})$	$\equiv$	Set of server classes j that can form a triple with $\boldsymbol{d}$ and some $i \in \mathcal{S}$ so that $(i, j, \boldsymbol{d}) \in \mathcal{T}$
$p(\boldsymbol{d})$	$\equiv$	$\mathbb{P}(\boldsymbol{D} = \boldsymbol{d})$; the probability that $\boldsymbol{D} = \boldsymbol{d}$; the function $p(\cdot)$ uniquely specifies the querying rule

Table 5 continued

$\mathcal{Q}$	$\equiv$	$\{1, 2, \ldots, d\}$; set of indices for each queried server in a query
q_i	$\equiv$	k_i/k; proportion of servers which belong to class i for $i \in \mathcal{S}$
ρ_i	$\equiv$	Fraction of time a class-i server is busy
R_i	$\equiv$	μ_i/μ_s; speed of class-i servers normalized by that of the slowest (i.e., class-s) servers
$r_i^{\mathrm{B}}(\boldsymbol{d})$	$\equiv$	Probability that a busy queried tagged class-i server is assigned the job when $\boldsymbol{D} = \boldsymbol{d}$
$r_i^{\mathrm{I}}(\boldsymbol{d})$	$\equiv$	Probability that an idle queried tagged class-i server is assigned the job when $\boldsymbol{D} = \boldsymbol{d}$
s	$\equiv$	Number of server classes
$\mathcal{S}$	$\equiv$	$\{1, \ldots, s\}$; set of server class indices
$\bar{\mathcal{S}}$	$\equiv$	$\{1, \ldots, s+1\}$; set of all possible values for the random variable J
$\mathcal{S}(\boldsymbol{d})$	$\equiv$	$\{i \in \mathcal{S}\colon d_i > 0\}$; Indices of server classes included in the query
$\mathcal{T}$	$\equiv$	Set of triples $(i, j, \boldsymbol{d})$ for which each $\alpha_i(j, \boldsymbol{d})$ can take a distinct value in our pruning
$\mathcal{T}(\boldsymbol{d})$	$\equiv$	Set of pairs (i, j) (given $\boldsymbol{d}$) for which each $\alpha_i(j, \boldsymbol{d})$ can take a distinct value in our pruning
T	$\equiv$	Response time of a job (not conditioned on the class of the server on which the job runs)
T_i	$\equiv$	Response time of a job that runs at a class-i server

References

1. Banawan, S., Zeidat, N.: A comparative study of load sharing in heterogeneous multicomputer systems. In: Proceedings of 25th Annual Simulation Symposium, pp. 22–31. IEEE (1992)
2. Banawan, S.A., Zahorjan, J.: Load sharing in heterogeneous queueing systems. In: Proceedings of IEEE INFOCOM'89, pp. 731–739 (1989)
3. Bonomi, F.: On job assignment for a parallel system of processor sharing queues. IEEE Trans. Comput. **39**(7), 858–869 (1990)
4. Chen, H., Ye, H.Q.: Asymptotic optimality of balanced routing. Oper. Res. **60**(1), 163–179 (2012)
5. Dunning, I., Huchette, J., Lubin, M.: Jump: a modeling language for mathematical optimization. SIAM Rev. **59**(2), 295–320 (2017)
6. Feng, H., Misra, V., Rubenstein, D.: Optimal state-free, size-aware dispatching for heterogeneous m/g/-type systems. Perform. Eval **62**(1), 475–492 (2005). https://doi.org/10.1016/j.peva.2005.07.031
7. Gardner, K., Jaleel, J.A., Wickeham, A., Doroudi, S.: Scalable load balancing in the presence of heterogeneous servers. Performance Evaluation p. 102151 (2020)
8. Gupta, V., Harchol-Balter, M., Sigman, K., Whitt, W.: Analysis of join-the-shortest-queue routing for web server farms. Perform. Eval. **64**(9–12), 1062–1081 (2007)
9. Hellemans, T., Bodas, T., Van Houdt, B.: Performance analysis of workload dependent load balancing policies. In: Proceedings of the ACM on Measurement and Analysis of Computing Systems (2019). https://doi.org/10.1145/3341617.3326150
10. Hyytiä, E.: Optimal routing of fixed size jobs to two parallel servers. INFOR: Inf. Syst. Oper. Res. **51**(4), 215–224 (2013). https://doi.org/10.3138/infor.51.4.215
11. Izagirre, A., Makowski, A.: Light traffic performance under the power of two load balancing strategy: the case of server heterogeneity. SIGMETRICS Perform. Eval. Rev. **42**(2), 18–20 (2014)

12. Jaleel, J.A., Doroudi, S., Gardner, K., Wickeham, A.: A general "power-of-d" dispatching framework for heterogeneous systems (2021). https://arxiv.org/abs/2112.05823
13. Koole, G.: A simple proof of the optimality of a threshold policy in a two-server queueing system. Syst. Control Lett. **26**(5), 301–303 (1995)
14. Larsen, R.L.: Control of Multiple Exponential Servers with Application to Computer Systems. Ph.D. thesis, College Park, MD, USA (1981)
15. Lin, W., Kumar, P.R.: Optimal control of a queueing system with two heterogeneous servers. IEEE Trans. Autom. Control **29**(8), 696–703 (1984)
16. Lu, Y., Xie, Q., Kliot, G., Geller, A., Larus, J., Greenberg, A.: Join-idle-queue: a novel load balancing algorithm for dynamically scalable web services. Perform. Eval. **68**(11), 1056–1071 (2011)
17. Lubin, M., Dunning, I.: Computing in operations research using Julia. INFORMS J. Comput. **27**(2), 238–248 (2015). https://doi.org/10.1287/ijoc.2014.0623
18. Luh, H.P., Viniotis, I.: Threshold control policies for heterogeneous server systems. Math. Methods Oper. Res. **55**(1), 121–142 (2002)
19. Mitzenmacher, M.: The power of two choices in randomized load balancing. IEEE Trans. Parallel Distrib. Syst. **12**(10), 1094–1104 (2001)
20. Mukhopadhyay, A., Mazumdar, R.: Analysis of randomized join-the-shortest-queue (JSQ) schemes in large heterogeneous processor-sharing systems. IEEE Trans. Control Netw. Syst. **3**(2), 116–126 (2016)
21. Nelson, R.D., Philips, T.K.: An Approximation to the Response Time for Shortest Queue Routing, vol. 17. ACM, New York (1989)
22. Rubinovitch, M.: The slow server problem. J. Appl. Probab. **22**(1), 205–213 (1985)
23. Rubinovitch, M.: The slow server problem: a queue with stalling. J. Appl. Probab. **22**(4), 879–892 (1985)
24. Rykov, V.V., Efrosinin, D.V.: On the slow server problem. Autom. Remote. Control. **70**(12), 2013–2023 (2009)
25. Selen, J., Adan, I., Kapodistria, S.: Approximate performance analysis of generalized join the shortest queue routing. In: Proceedings of the 9th EAI International Conference on Performance Evaluation Methodologies and Tools, pp. 103–110. ICST (Institute for Computer Sciences, Social-Informatics and ... (2016)
26. Selen, J., Adan, I., Kapodistria, S., van Leeuwaarden, J.: Steady-state analysis of shortest expected delay routing. Queueing Syst. **84**(3–4), 309–354 (2016)
27. Sethuraman, J., Squillante, M.S.: Optimal stochastic scheduling in multiclass parallel queues. SIGMETRICS Perform. Eval. Rev. **27**(1), 93–102 (1999). https://doi.org/10.1145/301464.301483
28. Stolyar, A.: Pull-based load distribution in large-scale heterogeneous service systems. Queueing Syst. **80**(4), 341–361 (2015)
29. Stolyar, A.L.: Pull-based load distribution among heterogeneous parallel servers: the case of multiple routers. Queueing Syst. **85**(1–2), 31–65 (2017)
30. Tantawi, A.N., Towsley, D.: Optimal static load balancing in distributed computer systems. J. ACM (JACM) **32**(2), 445–465 (1985)
31. Vargaftik, S., Keslassy, I., Orda, A.: LSQ: load balancing in large-scale heterogeneous systems with multiple dispatchers. IEEE/ACM Transactions on Networking, **28**(3), 1186–1198 (2020). https://urldefense.com/v3/. https://doi.org/10.1109/TNET.2020.2980061
32. Vvedenskaya, N., Dobrushin, R., Karpelevich, F.: Queueing system with selection of the shortest of two queues: an asymptotic approach. Problemy Peredachi Informatsii **32**(1), 20–34 (1996)
33. Wächter, A., Biegler, L.T.: On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. Math. Program. **106**(1), 25–57 (2006). https://doi.org/10.1007/s10107-004-0559-y
34. Wang, C., Feng, C., Cheng, J.: Distributed join-the-idle-queue for low latency cloud services. IEEE/ACM Trans. Netw. **26**(5), 2309–2319 (2018)
35. Weber, R.R.: On the optimal assignment of customers to parallel servers. J. Appl. Probab. **15**(2), 406–413 (1978)
36. Weng, W., Zhou, X., Srikant, R.: Optimal load balancing with locality constraints. Proc. ACM Meas. Anal. Comput. Syst. **4**(3), 1–37 (2020)
37. Whitt, W.: Deciding which queue to join: some counterexamples. Oper. Res. **34**(1), 55–62 (1986)
38. Winston, W.: Optimality of the shortest line discipline. J. Appl. Probab. **14**(1), 181–189 (1977)

39. Zhou, X., Shroff, N., Wierman, A.: Asymptotically optimal load balancing in large-scale heterogeneous systems with multiple dispatchers. Perform. Eval. **145**, 102146 (2021)
40. Zhou, X., Wu, F., Tan, J., Sun, Y., Shroff, N.: Designing low-complexity heavy-traffic delay-optimal load balancing schemes: theory to algorithms. Proc. ACM Measu. Anal. Comput. Syst. **1**(2), 39 (2017)