A General “Power-of-d” Dispatching Framework for Heterogeneous Systems

Jazeem Abdul Jaleel*, Alexander Wickeham*, Sherwin Doroudi*, and Kristen Gardner†

*University of Minnesota-Twin Cities †Amherst College

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) [4], 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 [2].

Motivated by the need for dispatching policies that perform well in heterogeneous systems, researchers have designed new policies for this setting. There are two decision points when “power of d” policies can use server speed information: when choosing which d servers to query, and when assigning a job to one of those servers. Many heterogeneity-aware “power of d” policies, such as Shortest-Expected-Delay-d [5] and Balanced Routing [1], use only one of these decision points. While both of these policies lead to better performance than the fully heterogeneity-unaware JSQ-d policy, there is still room for improvement. Recent work has proposed two families of policies, JIQ-(d_F, d_S) and JSQ-(d_F, d_S), that leverage heterogeneity at both decision points [2, 3]. The policies in these families query fixed numbers of “fast” and “slow” servers, then probabilistically choose whether to assign the job to a fast or a slow server based on the idle/busy statuses of the queried servers.

In this paper, we propose a rich general framework of new policies inspired by JIQ-(d_F, d_S). Our framework generalizes existing work in two important directions, necessitating substantially more sophisticated analysis. First, instead of assuming that there are only two server speeds, we allow for *any number of server speeds*. Second, instead of querying deterministic numbers of fast and slow servers, we draw the mix of server speeds to be queried from *any arbitrary distribution*.

Despite the challenges inherent to our general setting, we present an analysis of mean response time; our approach also can be used to derive other metrics. Using our analytical results, we show that the additional flexibility offered by our policy can lead to substantial performance improvements over JIQ-(d_F, d_S).

2. MODEL AND POLICY

We consider a system with k servers. There are s classes of server speeds, $\mathcal{S} \equiv \{1, \ldots, s\}$, 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. The size of a job running on a class-i server is drawn from a general distribution with cdf G_i and mean $1/\mu_i$, where all such distributions have the same “shape,” i.e., $G_i(x) = G_j(\mu_i x/\mu_j)$. Classes are indexed in decreasing order of speed, i.e., $\mu_1 > \cdots > \mu_s$. We assume that $\sum_i \mu_i q_i = 1$. Jobs arrive to the system as a Poisson process with rate λk.

Upon a job’s arrival, the dispatcher (i) queries a given number ($d \ll k$) of servers according to a *querying rule*, 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*. Rules do not use job size information.

When a job arrives, the dispatcher queries d servers at random according to a *querying rule*. Let D_i denote the number of class-i servers in a given query, let $\mathbf{D} \equiv (D_1, \ldots, D_s)$ denote the *query mix*, let d_i and $\mathbf{d} \equiv (d_1, \ldots, d_s)$ denote the realizations of D_i and $\mathbf{D}$, respectively, and finally let $\mathcal{D} \equiv \{\mathbf{d} \colon d_1 + \cdots + d_s = d\}$ be the set of all possible query mixes $\mathbf{d}$. We assume that servers of a given class are *a priori* identical, and so they are equally likely to be queried. A querying rule specifies the distribution over query mixes. Formally, a querying rule is given by a function $p \colon \mathcal{D} \to [0, 1]$ satisfying $\sum_{\mathbf{d} \in \mathcal{D}} p(d) = 1$. The querying rule selects servers so that $\mathbb{P}(\mathbf{D} = \mathbf{d}) = p(\mathbf{d})$.

Once a set of servers has been queried, the job is assigned to one of these servers probabilistically according to an *assignment rule*, which specifies a distribution over the queried classes. The rule is based on the realized query mix $\mathbf{d}$ and the idle/busy statuses of those servers, which we encode by $\mathbf{a} \equiv (a_1, \ldots, a_s)$, where a_i is the number of *idle* class-i servers among the d_i queried. The set of all possible $\mathbf{a}$ vectors is given by $\mathcal{A} \equiv \{\mathbf{a} \colon a_1 + \cdots + a_s \le d\}$. Note that a_i and $\mathbf{a}$ are realizations of the random variable A_i and the random vector $\mathbf{A}$, respectively.

Formally, an assignment rule is given by a family of functions $\alpha_i\colon \mathcal{A} \times \mathcal{D} \to [0,1]$ parameterized by $i \in \mathcal{S}$. For all $\mathbf{a} \in \mathcal{A}$ and $\mathbf{d} \in \mathcal{D}$ such that $\mathbf{a} \le \mathbf{d}$ (elementwise) these families must satisfy $\sum_{i\in\mathcal{S}} \alpha_i(\mathbf{a},\mathbf{d}) = 1$ and $\alpha_i(\mathbf{a},\mathbf{d}) = 0$ if $d_i = 0$. Given such a family of functions (together with a query resulting in vectors $\mathbf{a} \in \mathcal{A}$ and $\mathbf{d} \in \mathcal{D}$) the dispatcher sends the job to a class-i server with probability $\alpha_i(\mathbf{a},\mathbf{d})$. 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, $\alpha_i(\mathbf{a},\mathbf{d}) = 0$ whenever there is a class $j < i$ such that $a_j \ge 1$. Moreover, whenever $\mathbf{a} \ne \mathbf{0}$, the value of $\alpha_i(\mathbf{a},\mathbf{d})$ depends only on $j^* = \min\{j \in \mathcal{S}\colon a_j > 0\}$ and on $\mathbf{d}$ (specifically, on $\{j \le j^*\colon d_j > 0\}$). This structure allows us to introduce the following abuse of notation that will facilitate the discussion of our analysis: $\alpha_i(j^*,\mathbf{d}) \equiv \alpha_i(\mathbf{a},\mathbf{d})$ for all $j^* \in \mathcal{S}$ and $\mathbf{a} \in \mathcal{A}$ such that $j^* = \min\{j \in \mathcal{S}\colon a_j > 0\}$. When $\mathbf{a} = \mathbf{0}$, we use our original notation: $\alpha_i(\mathbf{0},\mathbf{d})$.

3. ANALYSIS

We carry out all analysis in steady-state. We let $k \to \infty$ and assume that *asymptotic independence* holds in this regime, meaning that (i) the states of all servers (i.e., the number of jobs and their remaining sizes) are independent of one another, and (ii) all servers of the same class behave stochastically identically. 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 arbitrary $i \in \mathcal{S}$) each class-i server experiences a Poisson arrival stream with status-dependent rate: $\lambda_i^{\mathbf{I}}$ (when idle) and $\lambda_i^{\mathbf{B}}$ (when busy).

In light of the above, each class-i server, when busy, operates exactly like a standard M/G_i/1 system with arrival rate $\lambda_i^{\mathbf{B}}$ under the chosen scheduling rule. Recalling that our scheduling rule must be work-conserving, standard M/G/1 analysis gives the expected busy period at such a server as follows: $\mathbb{E}[B_i] = 1/(\mu_i - \lambda_i^{\mathbf{B}})$. Letting ρ_i denote the fraction of time that a class-i server is busy, applying the Renewal Reward Theorem yields

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

Next observe that the mean response time experienced by jobs at a class-i server, $\mathbb{E}[T_i]$ is given by

$$\mathbf{E}[T_i] = (1-\rho_i)\mathbf{E}\left[T_i^{\mathbf{I}}\right] + \rho_i \mathbf{E}\left[T_i^{\mathbf{B}}\right], \tag{2}$$

where $\mathbf{E}\left[T_i^{\mathbf{I}}\right]$ (respectively $\mathbf{E}\left[T_i^{\mathbf{B}}\right]$) denotes the mean response time experienced by a job arriving to an idle (respectively busy) M/G_i/1 system under the chosen scheduling rule. Expressions for these latter two quantities under a variety of scheduling rules (e.g., First-Come-First-Served, Processor Sharing, Foreground-Background) are available in the queueing literature.

We find the system's overall mean response time by taking a weighted average of the the mean response times at each server class. Let $\lambda_i \equiv (1-\rho_i)\lambda_i^{\mathbf{I}} + \rho_i\lambda_i^{\mathbf{B}}$ denote the average arrival rate experienced by a class-i server. 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

$$\mathbf{E}[T] = \sum_{i=1}^{s} \left(\frac{q_i\lambda_i}{\lambda}\right) \mathbf{E}[T_i]. \tag{3}$$

We now proceed to find $\lambda_i^{\mathbf{I}}$ and $\lambda_i^{\mathbf{B}}$ via mean-field analysis by tagging a class-i server. Recall that the rate at which the tagged server is queried does not (and in fact, *cannot*) depend on its idle/busy status. Given query mix $\mathbf{d}$, the probability that the query includes the tagged server is d_i/k_i. Because a query is of mix $\mathbf{d}$ with probability $p(\mathbf{d})$, the tagged server is queried at rate $\lambda k \sum_{\mathbf{d}\in\mathcal{D}} p(\mathbf{d})(d_i/k_i)$. Of course, the tagged server's presence in the query does not guarantee that the job will be sent to it. Denote by $r_i^{\mathbf{I}}(\mathbf{d})$ (respectively $r_i^{\mathbf{B}}(\mathbf{d})$) the probability that the job is sent to the tagged server under mix $\mathbf{d}$ given that the tagged server is in the query and is idle (respectively busy); $r_i^{\mathbf{I}}(\mathbf{d}) = r_i^{\mathbf{B}}(\mathbf{d}) = 0$ when d_i=0. Hence, the arrival rate *from queries with mix* $\mathbf{d}$ observed by the tagged server when it is idle is $\lambda k p(\mathbf{d})(d_i/k_i) r_i^{\mathbf{I}}(\mathbf{d}) = (\lambda/q_i) d_i p(\mathbf{d}) r_i^{\mathbf{I}}(\mathbf{d})$, with the analogous expression holding when the tagged server is busy. The overall arrival rates to an idle and busy class-i server are then

$$\lambda_i^{\mathbf{I}} = \frac{\lambda}{q_i} \sum_{\mathbf{d}\in\mathcal{D}} d_i p(\mathbf{d}) r_i^{\mathbf{I}}(\mathbf{d}) \tag{4}$$

$$\lambda_i^{\mathbf{B}} = \frac{\lambda}{q_i} \sum_{\mathbf{d}\in\mathcal{D}} d_i p(\mathbf{d}) r_i^{\mathbf{B}}(\mathbf{d}). \tag{5}$$

Next, we determine $r_i^{\mathbf{I}}(\mathbf{d})$ and $r_i^{\mathbf{B}}(\mathbf{d})$ (assuming that $d_i > 0$), beginning with $r_i^{\mathbf{I}}(\mathbf{d})$. Observe that the job can be sent to the tagged server only if all faster servers in the query are busy, which occurs with probability $b_i(\mathbf{d}) \equiv \mathbb{P}(A_1 = \cdots = A_{i-1} = 0 | \mathbf{D} = \mathbf{d}) = \prod_{j=1}^{i-1} \rho_j^{d_j}$ for a given query mix $\mathbf{d}$. If this is the case, then with probability $\alpha_i(i,\mathbf{d})$ the job is sent 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 $A_i = a_i$ (noting that $A_i \ge 1$ as the tagged server itself is idle and part of the query), we find the probability that the tagged server gets the job when queried with mix $\mathbf{d}$:

$$r_i^{\mathbf{I}}(\mathbf{d}) = b_i(\mathbf{d})\alpha_i(i,\mathbf{d}) \sum_{a_i=1}^{d_i} \frac{\mathbb{P}(A_i = a_i | \mathbf{D} = \mathbf{d}, A_i \ge 1)}{a_i}.$$

As $(A_i | \mathbf{D} = \mathbf{d}, A_i \ge 1) \sim \text{Binomial}(d_i - 1, 1-\rho_i) + 1$, we have the explicit expression

$$r_i^{\mathbf{I}}(\mathbf{d}) = b_i(\mathbf{d})\alpha_i(i,\mathbf{d}) \sum_{a_i=1}^{d_i} \binom{d_i - 1}{a_i - 1} \frac{(1-\rho_i)^{a_i-1}\rho_i^{d_i-a_i}}{a_i}. \tag{6}$$

We determine $r_i^{\mathbf{B}}(\mathbf{d})$ by taking the tagged server to be busy and conditioning on the class containing the fastest idle queried server $J \equiv \min\{j \in \mathcal{S}\colon A_j > 0\}$ and

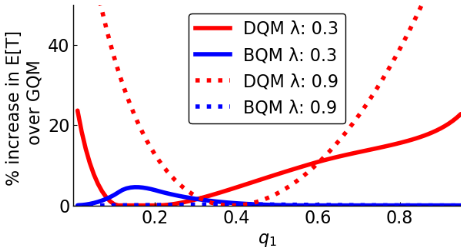

Figure 1: Percent increase in $\mathbf{E}[T]$ of DQM and BQM over GQM as a function of q_1. Here $s = d = 2$ and job sizes are exponential with $\mu_1/\mu_2 = 2$.

the special case where all servers are busy (i.e., when $\mathbf{A} = \mathbf{0}$ and J is not defined). If $J < i$, the job will not be sent to a class-i server, and if $J = i$, then the job will not be sent to the *tagged* server in particular (as it is busy). The probability that the fastest idle queried server belongs to class $j > i$ (given a query $\mathbf{d}$ which includes the busy tagged server) is

$$\mathbb{P}(J = j | \mathbf{D} = \mathbf{d}, A_i < D_i) = b_j(\mathbf{d}) \left(1 - \rho_j^{d_j}\right) \Big/ \rho_i,$$

where the division by ρ_i is due to the fact the tagged server is already known to be busy. Now recall that when $J = j > i$ the job is sent to a class-i server in general (respectively, the tagged server in particular) with probability $\alpha_i(j, \mathbf{d})$ (respectively, $\alpha_i(j, \mathbf{d})/d_i$). The special case where all other queried servers are busy (given $\mathbf{d}$) arises with probability $\left(\prod_{j=1}^{s} \rho_j^{d_j}\right)/\rho_i$; in this case, the job is sent to the tagged server with probability $\alpha_i(\mathbf{0}, \mathbf{d})/d_i$. Letting $y_i(\mathbf{d}) \equiv \left(\prod_{j=1}^{s} \rho_j^{d_j}\right) \alpha_i(\mathbf{0}, \mathbf{d}) \Big/ (\rho_i d_i)$,

$$r_i^{\mathbf{B}}(\mathbf{d}) = \sum_{j=i+1}^{s} \frac{b_j(\mathbf{d}) \left(1 - \rho_j^{d_j}\right) \alpha_i(j, \mathbf{d})}{\rho_i d_i} + y_i(\mathbf{d}) \qquad (7)$$

Finally, we can simultaneously solve Equations (1,4,5, 6,7) for $\lambda_i^{\mathbf{I}}$, $\lambda_i^{\mathbf{B}}$, ρ_i, $r_i^{\mathbf{I}}(\mathbf{d})$, $r_i^{\mathbf{B}}(\mathbf{d})$ (taking $b_i(\mathbf{d})$ as defined). All such equations and variables are parameterized by $i \in \mathcal{S}$, with Equations (6,7) and variables $r_i^{\mathbf{I}}(\mathbf{d})$ and $r_i^{\mathbf{B}}(\mathbf{d})$ further parameterized by $\mathbf{d} \in \mathcal{D}$; note that we do not need to solve for $r_i^{\mathbf{I}}(\mathbf{d})$ and $r_i^{\mathbf{B}}(\mathbf{d})$ when i and $\mathbf{d}$ are such that $d_i = 0$. Once determined, the $\lambda_i^{\mathbf{I}}$, $\lambda_i^{\mathbf{B}}$, and ρ_i values can be used in conjunction with (2) and (3) to determine the system's overall mean response time under a variety of scheduling rules.

4. DISCUSSION

This paper introduces a very general framework for dispatching in the presence of heterogeneous servers, extending existing work by considering multiple speed classes and probabilistic querying. As a result of this generality, response time evaluation and rule optimization can become intractable as d and/or s grow large. Specifically, the system of equations we solve to find mean response time consists of $s\left(3 + 2\binom{s+d-2}{d-1}\right)$ nonlinear equations in the same number of unknowns. Moreover the space of possible querying rules forms a polytope of dimensionality $\binom{s+d-1}{d} - 1$, while the space of assignment rules forms a polytope of dimensionality

$$\sum_{k=1}^{\min\{s,d\}-1} \binom{s-1}{k} k + \sum_{j=1}^{s} \sum_{k=1}^{\min\{j,d\}-1} \binom{j-1}{k} k.$$

In practice, "power of d" policies often operate in the low-d regime. In particular, $d = 2$ typically yields the biggest marginal response time improvement at the lowest communication cost. In heterogeneous settings, however, deterministic querying rules can be too coarse when d is low. In contrast, our probabilistic querying offers much more flexibility; hence, we anticipate our policies will have the greatest advantage over deterministic querying in this regime. Fortuitously, this regime also offers computational benefits for our policies: when $d = 2$ the system (1,4,5,6,7) reduces to only $3s + 2s^2$ equations, while querying and assignment rules span only $s(s + 1)/2 - 1$ and $s^2 - s$ dimensions, respectively.

Fig. 1 compares mean response time under three querying rules, assuming First-Come-First-Served scheduling, exponential job sizes, $s = d = 2$, and $\mu_1/\mu_2 = 2$. Deterministic Query Mix (DQM) always uses $p(1, 1) = 1$ (i.e., it queries one fast and one slow server). Binomial Query Mix (BQM) uses $p(d_1, 2 - d_1) = \binom{2}{d_1} \ell^{d_1} (1 - \ell)^{2-d_1}$ with ℓ chosen optimally. General Query Mix (GQM) uses the optimal choice of $p(\cdot)$. In each case assignment is optimal given the querying rule. GQM typically outperforms DQM substantially; surprisingly, BQM often captures most of this benefit. Notably, a multinomial generalization of BQM to higher values of s reduces the space of querying rules to only $s-1$ dimensions, rendering optimization significantly more tractable than GQM.

The good performance and low computational cost of binomial (multinomial) querying suggest that the investigation of other heuristics for pruning the policy space would be a worthwhile endeavor. Additionally, we anticipate that favoring shorter queues when deciding among busy servers of the same class will allow for improved performance, providing another avenue for exploration.

5. REFERENCES

[1] H. Chen and H.-Q. Ye. Asymptotic optimality of balanced routing. *Operations research*, 60(1):163–179, 2012.

[2] K. Gardner, J. Jaleel, A. Wickeham, and S. Doroudi. Scalable load balancing in the presence of heterogeneous servers. Unpublished work, https://kgardner.people.amherst.edu/papers/disp20.pdf, 2020.

[3] K. Gardner and C. Stephens. Smart dispatching in heterogeneous systems. *ACM SIGMETRICS Performance Evaluation Review*, 47(2):12–14, 2019.

[4] M. Mitzenmacher. The power of two choices in randomized load balancing. *IEEE Transactions on Parallel and Distributed Systems*, 12(10):1094–1104, 2001.

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