Smart Dispatching in Heterogeneous Systems

Kristen Gardner
Amherst College
Department of Computer Science
kgardner@amherst.edu

Cole Stephens
Amherst College
Department of Computer Science
cstephens19@amherst.edu

1. INTRODUCTION

In multi-server systems, selecting to which server to dispatch an arriving job is a key factor influencing system response time. One of the most widely studied policies is Join-the-Shortest-Queue (JSQ), which is known to minimize mean response time in certain settings [7]. Many variants on JSQ have been proposed, including JSQ-d, under which a job is dispatched to the shortest queue among d servers selected uniformly at random [3, 5]; Join-Idle-Queue (JIQ), under which the dispatcher knows which servers are idle but not the queue lengths of non-idle servers [2]; and others.

The vast majority of work analyzing JSQ and related policies makes a key assumption: that the system is *homogeneous*, meaning that all servers have the same speed. This assumption is inaccurate in most modern computer systems. Server heterogeneity can arise, e.g., when a server farm consists of several generations of hardware, or when many virtual machines contend for resources on the same physical machine. Unfortunately, the wealth of results about how best to dispatch in homogeneous systems does not translate well to heterogeneous systems. Policies like JSQ-d and JIQ, which can achieve near-optimal performance in homogeneous systems, can lead to unacceptably high response times and even instability in heterogeneous systems [4, 8].

Heterogeneous systems offer an opportunity to design policies that leverage differing server speeds to make smarter dispatching decisions. We focus on "power-of-d" policies, which, upon a job's arrival, poll some constant number of servers and dispatch the job to one of the polled servers. There are two primary moments at which such policies can favor faster servers: when selecting which servers to poll, and when selecting to which of the polled servers to dispatch the job. The second approach is used by policies such as Shortest Expected Delay (SED), under which an arriving job is dispatched to the server at which its expected response time (calculated by scaling the number of jobs at the server by the server's speed) is shortest [1, 6]. This requires knowing the speed of every server.

In this paper, we focus on heterogeneous systems in which the dispatcher may not know the exact speed of every server, or even which servers are "fast" and which are "slow." Even absent detailed server speed information, we show that it is possible to design policies that outperform those designed for homogeneous settings. In Section 3, we propose and analyze the JSQ-(d_F, d_S) policy for systems in which the identities of servers ("fast" or "slow") are known, but not the servers' exact speeds. In Section 4, we propose the JSQ-DAS policy, which successfully identifies fast servers when the dispatcher does not know server identities *a priori*. Both policies are successful at achieving low mean response time even when the dispatcher has limited information.

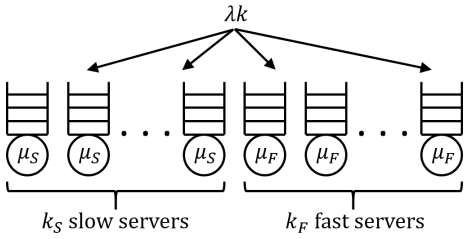

Figure 1: The system model.

2. MODEL

Our system consists of k servers (see Figure 1). Of these, $k_F = k \cdot p$ are "fast" servers and $k_S = k - k_F$ are "slow" servers. Service times are exponentially distributed; fast and slow servers work at rates μ_F and $\mu_S < \mu_F$ respectively. Jobs arrive to the system as a Poisson process with rate λk. We assume that the total system arrival rate is less than the total service rate; i.e., $\lambda k < \mu_F k_F + \mu_S k_S$. Upon arrival, a job is dispatched immediately to a single server according to some policy. Each server works on the jobs in its queue in first-come first-served (FCFS) order; there is no preemption.

3. SETTING 1: UNKNOWN SERVER SPEEDS

We begin with the setting in which the dispatcher knows which servers are "fast" and which are "slow," but does not know the exact speed of any server. We introduce the JSQ-(d_F, d_S) dispatching policy, defined as follows.

DEFINITION 1. *Under the* ***JSQ-(d_F, d_S)*** *dispatching policy, when a job arrives to the system, it polls d_F fast servers, chosen uniformly at random without replacement. If any of these d_F fast servers are idle, the job begins service on an idle fast server (where ties are broken uniformly at random). If none of the d_F polled fast servers are idle, the job then polls d_S slow servers. If any of these d_S slow servers are idle, the job begins service on an idle slow server (where ties are broken uniformly at random). If none of the d_S polled slow servers are idle, the job joins the shortest queue among the d_F polled* fast *servers.*

Intuitively, a job is best off if it gets to begin service immediately, where it is better to run on a fast server than a slow server. However, if there are no idle servers available, the job prefers to wait in the queue at a fast server.

3.1 Analysis

We derive an approximation for mean response time under this policy. Throughout, we will assume that $k \to \infty$, and that, in this limiting regime, all servers are idle independently with probability $1-\rho_F$ for fast servers and with probability $1-\rho_S$ for slow servers.

We begin by deriving ρ_F and ρ_S:

$$\rho_S = \lambda k \mathbf{P}\left\{\substack{\text{job runs on}\\\text{a slow server}}\right\} \cdot \frac{1}{\mu_S k_S} = \frac{\lambda \rho_F^{d_F}(1-\rho_S^{d_S})}{\mu_S(1-p)}$$

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

Solving this system of equations, numerically if an exact analytical solution is not possible, yields ρ_F and ρ_S.

To find $\mathbf{E}[T]$, we first condition on the number of idle servers found by an arriving job (here we use the assumption that servers are idle independently):

$$\begin{aligned}\mathbf{E}[T] &= \mathbf{E}[T|\text{idle fast servers}] \cdot \mathbf{P}\{\text{idle fast servers}\}\\ &\quad + \mathbf{E}\left[T\left|\substack{\text{no idle fast servers,}\\\text{idle slow servers}}\right.\right] \cdot \mathbf{P}\left\{\substack{\text{no idle fast servers,}\\\text{idle slow servers}}\right\}\\ &\quad + \mathbf{E}\left[T\left|\substack{\text{no idle fast servers,}\\\text{no idle slow servers}}\right.\right] \cdot \mathbf{P}\left\{\substack{\text{no idle fast servers,}\\\text{no idle slow servers}}\right\}\\ &= \frac{1}{\mu_F}\cdot(1-\rho_F^{d_F}) + \frac{1}{\mu_S}\cdot\rho_F^{d_F}(1-\rho_S^{d_S})\\ &\quad + \rho_F^{d_F}\rho_S^{d_S}\cdot\mathbf{E}\left[T\left|\substack{\text{no idle fast servers,}\\\text{no idle slow servers}}\right.\right]. \qquad (1)\end{aligned}$$

We next need to derive $\mathbf{E}\left[T\left|\substack{\text{no idle fast servers,}\\\text{no idle slow servers}}\right.\right]$. In this case, the job joins the shortest queue among the d_F polled fast servers, all of which are busy. To derive response time, we first need to determine the distribution of the number of jobs in a fast server's queue.

We take an approach similar to Mitzenmacher's analysis of JSQ in homogeneous systems [3]. Let $n_i(t)$ denote the number of fast servers with exactly i jobs in the queue (including the job in service, if there is one) at time t. Let $m_i(t)$ denote the number of fast servers with *at least* i jobs at time t. Let $f_i(t) = n_i(t)/k_F$ be the fraction of fast servers with exactly i jobs, and let $s_i(t) = \sum_{j=i}^{\infty} f_i(t) = m_i(t)/k_F$ be the fraction of fast servers with at least i jobs, so the s_i's are the tails of the f_i's. Note that $s_0(t) = 1$ for all t.

As in [3], we consider a deterministic limiting system and express its behavior using a system of differential equations. We consider the expected change in the number of queues with at least $i > 1$ jobs over a small interval of time dt. This number will increase if an arriving job joins the queue at a fast server with exactly $i-1$ jobs. The rate at which jobs arrive to the system is λk; with probability $s_{i-1}^{d_F} - s_i^{d_F}$ all d_F of the servers polled by the arriving job have at least $i-1$ jobs, but not all d_F of the servers have at least i jobs (that is, the shortest queue among the d_F servers contains exactly i jobs); and with probability $\rho_S^{d_S}$ all d_s of the polled slow servers are busy. The number of queues with at least $i > 1$ jobs will decrease if a job departs from a queue with exactly i jobs. This happens with rate $\mu_F k_F(s_i - s_{i+1})$. Putting this together, we have, for $i > 1$:

$$\frac{dm_i}{dt} = \lambda k\left(s_{i-1}^{d_F} - s_i^{d_F}\right)\rho_S^{d_S} - \mu_F k_F(s_i - s_{i+1}).$$

The case where $i = 1$ is similar, except here an arriving job that finds a server with $i-1=0$ jobs in the queue will enter service on that server instead of polling slow servers:

$$\frac{dm_1}{dt} = \lambda k\left(s_0^{d_F} - s_1^{d_F}\right) - \mu_F k_F(s_1 - s_2).$$

Dividing all terms by k_F gives us a system of differential equations for the s_i terms:

$$\frac{ds_1}{dt} = \frac{\lambda}{p}\left(s_0^{d_F} - s_1^{d_F}\right) - \mu_F(s_1 - s_2) \qquad (2)$$

$$\frac{ds_i}{dt} = \frac{\lambda}{p}\left(s_{i-1}^{d_F} - s_i^{d_F}\right)\rho_S^{d_S} - \mu_F(s_i - s_{i+1}), \quad i > 1 \qquad (3)$$

$$s_0 = 1,$$

recalling that $p = \frac{k_F}{k}$ is the fraction of servers that are fast.

By setting $\frac{ds_i}{dt} = 0$ for all i, we can find a fixed point for this system and solve for the s_i terms. We start by summing equation (2) and (3) for all $i > 1$:

$$\begin{aligned}\sum_{i=1}^{\infty}\frac{ds_i}{dt} = 0 &= \frac{\lambda}{p}\left(s_0^{d_F} - s_1^{d_F}\right) - \mu_F(s_1 - s_2)\\ &\quad + \sum_{i=2}^{\infty}\left(\frac{\lambda}{p}\left(s_{i-1}^{d_F} - s_i^{d_F}\right)\rho_S^{d_S} - \mu_F(s_i - s_{i+1})\right)\\ 0 &= \frac{\lambda}{p}\left(s_0^{d_F} - s_1^{d_F} + \rho_S^{d_S}\sum_{i=2}^{\infty}\left(s_{i-1}^{d_F} - s_i^{d_F}\right)\right)\\ &\quad - \mu_F\sum_{i=1}^{\infty}(s_i - s_{i+1})\\ 0 &= \frac{\lambda}{p}\left(1 - s_1^{d_F} + \rho_S^{d_S}s_1^{d_F}\right) - \mu_F s_1\\ 0 &= \frac{\lambda}{p}\left(1 - s_1^{d_F}\left(1-\rho_S^{d_S}\right)\right) - \mu_F s_1, \qquad (4)\end{aligned}$$

from which we can now solve for s_1. For some values of d_F, a closed-form solution exists. In other cases that do not permit a closed-form solution, we can solve for s_1 numerically.

We then find the remaining s_i values using the recurrence

$$\frac{\lambda}{p}(s_{i-1}^{d_F} - s_i^{d_F})\rho_S^{d_S} - \mu_F(s_i - s_{i+1}) = 0.$$

Once we have determined the s_i terms, we are ready to return to equation (1) to find $\mathbf{E}\left[T\left|\substack{\text{no idle fast servers,}\\\text{no idle slow servers}}\right.\right]$:

$$\begin{aligned}\mathbf{E}\left[T\left|\substack{\text{no idle fast servers,}\\\text{no idle slow servers}}\right.\right] &= \sum_{i=1}^{\infty}\mathbf{P}\{\text{job joins queue with } i \text{ jobs}\}\cdot i\cdot\frac{1}{\mu_F}\\ &= \frac{1}{\mu_F}\sum_{i=1}^{\infty} i\cdot\frac{s_i^{d_F} - s_{i+1}^{d_F}}{s_1^{d_F}}. \qquad (5)\end{aligned}$$

Note that the probability that a job joins a queue with i servers is *not* the same as the probability that a server has i jobs in its queue. Combining (1) and (5) gives the overall system mean response time.

4. SETTING 2: UNKNOWN SERVER IDENTITIES

We now turn to a setting in which the dispatcher has even less information: here we assume that the dispatcher knows what fraction of servers, p, are fast, but does not know which servers are fast or the speeds of fast and slow servers.

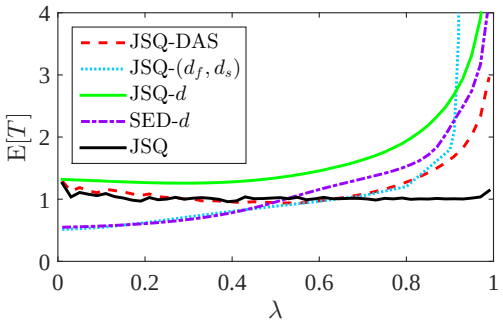

Figure 2: Mean response time as a function of λ under five dispatching policies.

To identify the fast servers, we use a novel heuristic called *accomplishment sampling*. We define a server's accomplishment to be the number of jobs that have been dispatched to that server. The JSQ-DAS (JSQ-d with Accomplishment Sampling) policy uses the idea of server accomplishment to decide which servers to poll.

DEFINITION 2. *Under the JSQ-DAS dispatching policy, when a job arrives to the system, it polls d servers. Of these, $0 < a < d$ are chosen uniformly at random without replacement from among the pk most accomplished servers. The remaining $d - a$ servers are chosen uniformly at random without replacement from among the $(1-p)k$ least accomplished servers. The job then joins the queue at the server with the shortest queue among the d polled servers. Ties are broken in favor of more accomplished servers.*

The idea behind accomplishment sampling is that fast servers are less likely to build up long queues than slow servers, making a polled fast server more likely to have the shortest queue. Thus, fast servers are likely to accumulate higher accomplishment values than slow servers, giving them a better chance of being polled in the future. Empirical results indicate that at $\lambda > 0.6$, accomplishment sampling correctly identifies 99.9% of fast servers, allowing the policy to nearly guarantee that at least one fast server is polled.

5. RESULTS AND DISCUSSION

In this section we evaluate our two proposed policies, JSQ-(d_F, d_S) and JSQ-DAS, by comparing their performance to that of several other well-known dispatching policies. In addition to JSQ-(d_F, d_S) and JSQ-DAS, we consider JSQ, JSQ-d, and SED-d (a low-communication variant on SED wherein an arriving job polls d servers chosen uniformly at random, and joins the queue at the server at which it has the shortest expected delay). We consider a system with $k = 1000$ servers and $p = 0.3$ (so there are $k_F = 300$ fast servers and $k_S = 700$ slow servers); we set $\mu_F = 1.95$ and $\mu_S = 0.6$. Hence a necessary condition for stability is $\lambda < 1$. For all policies except JSQ, we set $d = 4$; for JSQ-(d_F, d_S) we set $d_F = d_S = 2$, and for JSQ-DAS we set $a = 2$.

Figure 2 shows our results. At low load, JSQ-(d_F, d_S) performs as well as SED-d. When λ is low many fast servers are idle, and, like SED-d, JSQ-(d_F, d_S) gives preference to idle fast servers over idle slow servers. Surprisingly, JSQ-(d_F, d_S) outperforms SED-d at moderate values of λ. We hypothesize that this is because JSQ-(d_F, d_S) favors fast servers more strongly than SED-d does when polling. Unfortunately, JSQ-(d_F, d_s) appears to have a smaller stability region than the other policies studied here. Intuitively, this is because as λ gets high, it becomes impossible to maintain finite queue lengths for the fast servers while also not allowing there to be any queue at the slow servers. Precisely characterizing the stability region under JSQ-(d_F, d_S) remains an open problem.

Both JSQ-(d_F, d_S) and SED-d require some knowledge of server speeds, or at least of which servers are slow and which are fast. JSQ-DAS requires no such knowledge, yet manages to outperform the former two policies in many settings. At high load, JSQ-DAS has the best performance of any of the polling policies. The accomplishment sampling heuristic is extremely effective at correctly identifying the fast servers when λ is sufficiently high. Like JSQ-(d_F, d_S), JSQ-DAS then guarantees that some number of polled servers are fast, overcoming one of the weaknesses of SED-d. On the other hand, by allowing queues to build up at the slow servers, JSQ-DAS maintains a larger stability region than JSQ-(d_F, d_S). At low load, JSQ-DAS does worse than SED-d and JSQ-(d_F, d_S), but similarly to JSQ.

Our results are promising: both policies are able to match or even outperform policies like JSQ and JSQ-d, which do not use any knowledge of server heterogeneity when making dispatching decisions, as well as policies like SED-d, which require much more detailed heterogeneity information. While many open questions remain, we hope that this work provides a starting point for designing and analyzing other dispatching policies that leverage server heterogeneity to achieve better performance.

6. REFERENCES

[1] S. Banawan and N. Zeidat. A comparative study of load sharing in heterogeneous multicomputer systems. In *Proceedings. 25th Annual Simulation Symposium*, pages 22–31. IEEE, 1992.

[2] Y. Lu, Q. Xie, G. Kliot, A. Geller, J. Larus, and A. Greenberg. Join-idle-queue: A novel load balancing algorithm for dynamically scalable web services. *Performance Evaluation*, 68(11):1056–1071, 2011.

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

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

[5] N. Vvedenskaya, R. Dobrushin, and F. Karpelevich. Queueing system with selection of the shortest of two queues: An asymptotic approach. *Problemy Peredachi Informatsii*, 32(1):20–34, 1996.

[6] W. Whitt. Deciding which queue to join: Some counterexamples. *Operations research*, 34(1):55–62, 1986.

[7] W. Winston. Optimality of the shortest line discipline. *Journal of Applied Probability*, 14(1):181–189, 1977.

[8] X. Zhou, F. Wu, J. Tan, Y. Sun, and N. Shroff. Designing low-complexity heavy-traffic delay-optimal load balancing schemes: Theory to algorithms. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 1(2):39, 2017.