On the persistent-idle load distribution policy under batch arrivals and random service capacity

Rami Atar, Isaac Keslassy, Gal Mendelson, Ariel Orda and Shay Vargaftik

Viterbi Faculty of Electrical Engineering, Technion, Israel

Abstract

Practical considerations of present technological uses of load-balancing policies include the capability to function under server heterogeneity and the need to reduce communication overhead associated with information about the queue states. Under such constraints, state-of-the-art load-balancing approaches either do not cover the full stability region or provide poor performance. To address this challenge, the *Persistent-Idle* (PI) policy was recently introduced, where servers only update the dispatcher when they become idle, and the dispatcher always sends jobs to the last server that reported being idle. This policy was proved to achieve the stability region and to operate with low communication overhead (in a sense made precise) for Bernoulli arrivals and heterogeneous servers under the assumption that their service capacities are deterministic and constant. Aiming at a broader range of models that are relevant in applications, we consider in this paper batch arrivals and service capacities that vary over time according to stochastic processes, that may have distinct statistical characteristics across servers. Our main theoretical contribution is to show that the stability region is achieved by this policy in the case of two servers. Then, using extensive simulations, the performance is evaluated for a larger number of heterogeneous servers. PI always appears to be stable and achieves good performance while incurring a negligible communication overhead.

Keywords: Load balancing; load distribution; persistent-idle; join the shortest queue; the supermarket model; positive recurrence; stability; communication overhead

1. Introduction

Background. Load-balancing algorithms play a key role in operating parallel server systems efficiently. In recent years, the efficiency and even feasibility of the traditional load-balancing policies is challenged by the rapid growth of the infrastructure, together with the increasing levels of heterogeneity of the processing servers [1, 2, 3, 4, 5, 6, 7, 8, 9]. Such scaled-out systems (*e.g.,* data centers) typically contain, in addition to multiple generations of CPUs (central processing units) [10], various types of specialized device arrays consisting of devices such as GPUs (graphics processing units), FPGAs (field-programmable gate arrays) and ASICs (application-specific integrated circuit). These devices operate at significantly higher processing speeds and have different sizes of memories [11, 12, 13, 14, 15]. In these large heterogeneous systems, it becomes extremely difficult to come up with an effective load-balancing technique, as it is required to achieve high performance while operating with minimal server state information. In particular, the rapid sampling of server queue states needed for traditional load balancing involves (1) a communication bottleneck both at the dispatcher NIC (network interface card) entrance and at the dispatcher memory, (2) a potentially prohibitive computation complexity at the dispatcher, (3) an increased computation load at the servers, and (4) an increased network load, potentially leading to increased network delay and outdated information with poor resulting performance [16]. Given these limitations, queue lengths at the servers cannot be sampled too often. In addition, the dependence of the job service time on its memory working set and on the server's available memory makes it difficult to accurately estimate service rates and makes efficient load balancing even harder [17].

Related work. When the system state (*i.e.,* the queue length at each of the n servers) is fully observable at all times, *Join-the-Shortest-Queue (JSQ)* appears to be a natural policy due to its appealing versatility and theoretical guarantees [18, 19]. However, this policy is often impractical in large-scale systems, since it requires to continuously observe the state of all queues. Accordingly, state-of-the-art policies are designed to have considerably less communication overhead than JSQ. For example, the power-of-choice *JSQ(d)* policy samples only $d < n$ randomly-chosen server queues, and sends incoming jobs to the shortest one [20, 21, 22, 23, 24]. However, this may come at the price of stability when heterogeneous servers are considered; specifically, the stability region is not achieved by this policy [25, 26]. Another policy is the power-of-memory *JSQ*(d, m). In addition to randomly sampling d new queues, JSQ(d, m) also resamples the previous m shortest queues. It was shown that JSQ(1,1) is stable ([27]). However, our simulations indicate that JSQ(1,1) performs poorly in terms of job completion time compared to other low communication policies.

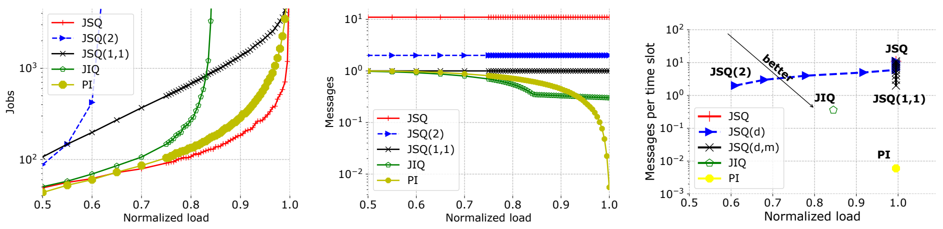
These difficulties have led researchers to seek alternative solutions. A promising recent load-balancing policy is *Join-the-Idle-Queue (JIQ)* [28, 29, 25, 30, 31], that acts by choosing an idle (empty) queue whenever available, and selects a random queue otherwise. In the asymptotic regime of many servers, JIQ has been shown to achieve vanishing steady-state probability of an arriving job experiencing blocking while incurring small communication overhead. However, JIQ can often be unstable in heterogeneous systems, even within the stability region [32].

Recently, *Persistent-Idle* (PI) was introduced in [32], where the dispatcher always sends jobs to idle servers, and if there are none, to the server that was idle most recently (more precise details appear below). It was shown in [32] that this policy achieves the stability region, and that the communication required to maintain idleness information is small. The model considered there assumes Bernoulli arrivals and deterministic, constant service capacities (but allows for distinct capacities across servers).

However, the method of proof developed in [32] does not seem to naturally extend to common models that are more relevant to several applications, such as batch arrivals (*i.e.,* the number of jobs that arrive at the dispatcher at a time slot is not limited to $\{0, 1\}$, *e.g.,* Poisson(λ)) and non-deterministic service capacities (*i.e.,* the number of jobs a server can potentially complete at a time slot is random, *e.g.,* Geometric(μ)).

Contributions. In this paper, we develop a different approach to proving stability, which allows us to handle batch arrivals and varying service capacities in the case of two servers. Our main theoretical contribution is that PI achieves the stability region even when considering batch arrivals and varying service capacities. We leave open the question of extending this result to systems with three or more servers.

The second contribution of this paper is a set of simulation results (Section 4) that compare PI against the aforementioned policies, for batch arrival and random service capacity systems with a varying number of servers. The simulations indicate that PI achieves good average and 99th-



(a) Mean number of jobs in the system (logarithmic scale).

(b) Communication overhead (logarithmic scale).

(c) Communication overhead vs. stability.

Figure 1: Performance of different policies in a time-slotted heterogeneous system with $n = 11$ servers. At each time slot a Pois(λ) RV determines the number of jobs arriving at the dispatcher, *i.e., batch arrivals*. The number of jobs that server i can potentially complete at a time slot is given by a Geom(μ_i) RV, *i.e., random service capacity*. Specifically, we consider a single strong server with rate $\mu = 10$ and 10 weaker servers with rate 1. **(a)** shows the instability of JSQ(2) and JIQ, together with the poor performance of JSQ$(1, 1)$ that accumulates 5–10$\times$ more jobs at moderate loads. In contrast, PI achieves a reasonable performance, even when compared to JSQ. **(b)** illustrates the communication overhead as a function of the offered load. JSQ prohibitively needs to sample all of the n server queues at each time slot. On the other hand, as the load increases, the communication overhead of PI becomes negligible. **(c)** depicts the tradeoff between performance and communication overhead. It plots the maximum load at which each algorithm appeared to be stable in simulations against its required communication overhead. It also includes the full range of JSQ($d = 2 \dots n$) power-of-choice policies and JSQ($d = 2 \dots n$,1) power-of-memory policies to illustrate the tradeoffs available to load-balancing designers. As seen above, only JSQ, JSQ(d,1) and PI achieve stability within the entire capacity region, and PI does so with a negligible control overhead at high loads.

percentile delays across a wide range of settings. For example, in a heterogeneous system with 110 servers and Pareto arrivals at a load of 90%, PI improves the 99th-percentile job completion time by 2$\times$ when compared to JSQ(1,1) and by 2.9$\times$ when compared to JIQ (beyond a load of 96.5% JIQ becomes unstable). Figure 1 illustrates the performance of a simple heterogeneous system with $n = 11$ servers, including 10 regular servers and one accelerator that works at 10$\times$ the speed (refer to Section 4 for the full simulation settings). It illustrates how JSQ(2) and JIQ suffer from instability, JSQ(1,1) provides poor performance, and JSQ requires a substantial communication overhead. On the other hand, PI appears to achieve the stability region with good performance and low communication overhead.

The remainder of this paper is organized as follows. Section 2 contains the system model and the definition of the PI policy. Section 3 contains our main theoretical result, namely, the stability of PI in a two server system. Section 4 contains the simulation results, comparing PI to other state-of-the-art policies. Finally, the conclusions appear in Section 5.

2. Persistent-idle policy

2.1. System model

Model. We consider a time-slotted parallel-server system model with a single dispatcher and n heterogeneous work conserving servers. New jobs arrive at the dispatcher, which must immediately decide on a server to which they will be assigned in the same time slot. Each server has an unbounded buffer to store the jobs in its queue. Finally, we assume the communication between the dispatcher and the server take a negligible amount of time.

Idle tokens. A single idle token is associated with each server. Thus, the total number of idle tokens in the system is n. Each idle token can be held by either the corresponding server or the dispatcher. Once a server becomes idle, *i.e.,* once its queue becomes empty, it sends its idle token to the dispatcher. Similarly, when the dispatcher chooses an idle server as a destination for a job, it figuratively sends the idle token back to this server along with the job. In practice, an idle server that receives a job knows that it got its idle token back.

2.2. PI policy

Dispatcher state. The dispatcher stores (1) the incoming server idle tokens, and (2) the identity of the last server to which it sent jobs, denoted as *Last-Idle*. At the first time slot, the identity of the *Last-Idle* server is chosen arbitrarily.

Dispatcher policy. Upon the arrival of jobs at the beginning of a time slot, the dispatcher checks whether it owns idle tokens:

- If it does, it chooses one of the corresponding servers using some tie breaking rule (*e.g.,* by lower index or randomly), updates the *Last-Idle* to the corresponding server, and sends all the arrived jobs to the updated *Last-Idle* server (along with its idle token).
- Otherwise, the dispatcher keeps the *Last-Idle* unchanged and persistently sends the arrived jobs to the *Last-Idle* server; namely, the server that was idle most recently.

Servers. Each server keeps a FIFO queue for incoming jobs. As long as a server has pending jobs, it owns its idle token and no communication overhead between the server and the dispatcher is required. Once a server becomes idle, it immediately sends its idle token to the dispatcher to inform about its idleness and availability for new jobs. Then, it waits for new jobs, without any further communication.

Communication overhead. The communication overhead consists only of idle tokens passed from the servers to the dispatcher. It respects two different bounds: (1) it is at most a single message per job (a necessary condition for a transmission of an idle token is the completion of a job), and (2) the number of messages passed up to time slot t is at most t (the dispatcher can return only one idle token per time slot). Note that the same bounds apply for any idle token based policy (*e.g.,* JIQ).

3. Stability with two heterogeneous servers

3.1. Overview

Often, a useful approach to proving stability relies on the construction of a Lyapunov function, mapping the state space to $[0, \infty)$, that satisfies a drift condition within a single time step, outside of some compact set. However, this approach does not appear to be directly applicable for the model under consideration. As discussed at length in [32], the main difficulty arises when considering states where one queue length is large while the other is close to zero. It is possible that the large queue length belongs to the Last-Idle server, which receives all of the work. Moreover, the reliance on remembering the Last-Idle server prevents us from classically looking at only the queue states as a Markov chain, since there is a hidden long-term memory of the Last-Idle server that impacts the analysis.

The proof in [32] relies on sampling the underlying Markov chain at times at which the identity of the Last-Idle server may change. Indeed, since the service capacities are deterministic, given the current system state, the next such time is deterministic as well. This observation allows

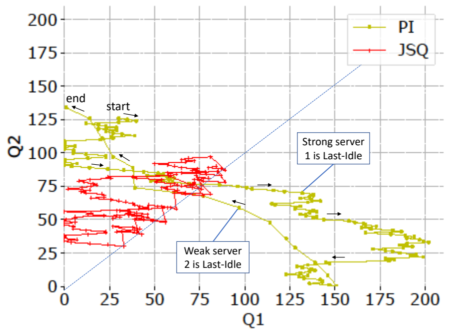


Figure 2: Queue lengths dynamics under PI and JSQ in a 2-server heterogeneous system. The strong server 1 has rate 10, the weaker server 2 has rate 1 and the normalized load is $10.1/11 \approx 92\%$. The arrows point at the queue length evolution over time. **(PI)** The PI plot starts with server 1 as the Last-Idle with about 30 jobs in its queue, while the queue of server 2 has about 125 jobs (*i.e.,* $(Q_1, Q_2) = (30, 125)$). Therefore, as long as server 2 is not idle, server 1 gets all the incoming jobs, even when it is not idle anymore. At first, server 1 processes them fast enough such that it becomes idle a few times. Then, its queue size increases. Meanwhile, server 2 does not get any jobs, and therefore its queue size randomly decreases, until it empties at $(Q_1, Q_2) = (151, 0)$. Server 2 then becomes the Last-Idle server, while server 1 quickly processes its jobs (we can see that Q_1 goes down faster than Q_2 previously did), until server 1 becomes idle again at $(Q_1, Q_2) = (0, 135)$. Therefore, PI essentially keeps bouncing back from one axis to the other. **(JSQ)** As expected, JSQ pushes the queue lengths towards the diagonal line where $Q_1 = Q_2$. Since the first server is stronger, JSQ tends to stay on the left upper side of the diagonal, and the first server becomes idle for a few times.

to obtain a state dependent drift. However, in the model considered in this paper, since the service capacities are random, the above argument fails.

Our proof combines several tools. We consider the dynamics of the model as a Markov process with state description that includes queue length information as well as the Last-Idle server. We were unable to come up with a function that may serve as a Lyapunov function adhering to the negative drift condition in a single-step. Instead, we rely on a technique that allows one to verify such a condition over a sufficiently long fixed time period. In verifying this condition, an important role is played by a suitable reflected random walk, for which we provide a bound on its average drift over time. Figure 2 provides some intuition on the behavior of queue lengths under PI in a two-server heterogeneous system compared to that of JSQ. It illustrates how under PI the queue length of the Last-Idle server follows a reflected random walk, while the queue length of the other server randomly decreases.

3.2. Notation and definitions

Arrivals. The system evolves in discrete time $t \in \mathbb{N}$. The job arrival process is given by a sequence of *i.i.d.* random variables (RVs) $\{a(t)\}_{t=0}^{\infty}$. Namely, at each time slot t, $a(t)$ jobs arrive at the dispatcher. We assume non-zero probabilities $\mathbb{P}(a(0) = k) > 0$ for $k = 0$ and $k = 1$ to avoid dealing with a periodic Markov chain. We denote the mean and standard deviation of $a(0)$ by λ and σ_a, respectively.

Servers. We consider two work-conserving servers, each with its own unbounded FIFO queue holding pending jobs. The service process of server $i \in \{1, 2\}$ is given by a sequence of *i.i.d.* RVs $\{s_i(t)\}_{t=0}^{\infty}$, representing the potential number of jobs that a server can complete in a time slot. We assume that the service capability of each server is bounded, *i.e.,* there exists s_{max} such that $s_i(t) \leq s_{max}$ for all i and t. We also assume for each i that $\mathbb{P}(s_i(0) = k) > 0$ for $k = 0$ and $k = 1$. We denote the mean and standard deviation of $s_i(0)$ by μ_i and σ_{s_i}, respectively.

Queue lengths. Let $Q_i(t)$ denote the number of pending jobs in queue i at the beginning of time slot t, before any arrivals and departures. Define $Q(t) = (Q_1(t), Q_2(t))$, and $Q = (Q(t))_{t=0}^{\infty}$.

Last-Idle. Denote $\sigma_i(t)$ as the last time slot at which server i is idle before time slot t. Formally,

$$\sigma_i(t) = \max\{s \leq t : Q_i(s) = 0\},$$

where if the set is empty, $\sigma_i(t) = 0$. Then, the Last-Idle server at time t is given by

$$LI(t) = argmax_i\{\sigma_i(t)\},$$

where in case of a tie, the Last-Idle server is chosen by some tie-breaking rule. For ease of exposition, we assume that the tie-breaking rule is that the Last-Idle server is chosen to be the idle server with the lowest index. Define the Last-Idle process as $LI = (LI(t))_{t=0}^{\infty}$.

PI policy. At time slot t, the PI dispatcher assigns arriving jobs to the server specified by $LI(t)$.

Dynamics. The queue length and Last-Idle processes satisfy the following recursions,

$$\begin{cases} Q_i(t+1) = [Q_i(t) + a(t)\mathbb{1}_{\{LI(t)=i\}} - s_i(t)]^+ \\ LI(t+1) = \begin{cases} LI(t), & \text{if } Q_j(t+1) > 0 \ \forall j, \\ \min\{j : Q_j(t+1) = 0\}, & \text{otherwise.} \end{cases} \end{cases} \tag{1}$$

Equation (1) describes the evolution of the process

$$X = (Q, LI).$$

This equation expresses $X(t+1)$ in terms of $X(t)$ and $(a(t), s_1(t), s_2(t))$. Since the latter tuple forms an *i.i.d.* sequence, it follows that the process X is a Markov chain on the state space

$$\mathcal{A} = \mathbb{N}^2 \times \{1, 2\}.$$

A member α of the state space $\mathcal{A}$ is of the form

$$\alpha = (\alpha_1, \alpha_2, l),$$

where α_1 and α_2 are queue lengths, and l is the identity of the Last-Idle server. It follows from our assumptions that the Markov chain X is irreducible and aperiodic.

Departures. Let $d_i(t)$ denote the actual number of jobs completed by server i during time slot t. Note that $d_i(t) \leq s_i(t)$. Then $d_i(t)$ is given by

$$d_i(t) = \min\{s_i(t), Q_i(t) + a(t)\mathbb{1}_{\{LI(t)=i\}}\},$$

and the first equation in recursion (1) can also be written as

$$Q_i(t+1) = Q_i(t) + a(t)\mathbb{1}_{\{LI(t)=i\}} - d_i(t). \tag{2}$$

We introduce the following notation. For $\alpha \in \mathcal{A}$, we write $\mathbb{E}_\alpha[\,\cdot\,]$ and $\mathbb{P}_\alpha(\,\cdot\,)$ for the conditional expectation $\mathbb{E}[\,\cdot\,|X(0){=}\alpha]$ and probability $\mathbb{P}(\,\cdot\,|X(0){=}\alpha)$, respectively.

3.3. Stability result

Theorem 3.1. *Assume $\lambda < \mu_1 + \mu_2$. Then:*
(i) X is positive recurrent. Consequently,
(ii) X has a unique stationary distribution, denoted by π_X, and
(iii) For any initial state $\alpha \in \mathcal{A}$ and any $B \subset \mathcal{A}$, $\mathbb{P}_\alpha(X(t) \in B) \to \pi_X(B)$ as $t \to \infty$.

Proof. We prove that there exist a non-negative function $f(\alpha)$, $\alpha \in \mathcal{A}$, a positive constant ϵ, a positive integer T_0 and a finite set $A \subset \mathcal{A}$, such that the following inequalities hold:

$$\begin{aligned} &\mathbb{E}_\alpha[f(X(T_0)) - f(X(0))] < -\epsilon T_0, \quad && \text{if } \alpha \notin A, \\ &\mathbb{E}_\alpha[f(X(T_0)) - f(X(0))] < \infty, && \text{if } \alpha \in A. \end{aligned} \tag{3}$$

By Theorem 2.2.4 of [33], (3) implies the positive recurrence of the Markov chain X, proving part *(i)* of Theorem 3.1. Note that [33] defines ergodic as positive recurrent, and that the result requires that the chain is irreducible and aperiodic, which we confirmed previously. Part (ii) then follows by I.3.6 of [34]. Finally, by the ergodic theorem for Markov chains, Theorem I.4.2 of [34], part (iii) follows because the chain is aperiodic.

To prove (3), we start by specifying the choices of f, ϵ, T_0 and A. We choose the function $f : \mathcal{A} \to \mathbb{N}$ as the sum of the queue lengths. Specifically, for each $\alpha \in \mathcal{A}$,

$$f(\alpha) = f(\alpha_1, \alpha_2, l) = \alpha_1 + \alpha_2.$$

Since $\lambda < \mu_1 + \mu_2$, there exists $\delta > 0$ such that $\lambda + \delta = \mu_1 + \mu_2$. Fix

$$\epsilon = \min\{\mu_1, \mu_2, \delta\}/2. \tag{4}$$

Let

$$C = 2\sqrt{\sigma_a^2 + \sigma_{s_1}^2 + \sigma_{s_2}^2}, \tag{5}$$

and fix a large enough T_0 such that $C\sqrt{T_0} + s_{max} \leq \epsilon T_0$, or written differently,

$$-\min\{\mu_1, \mu_2, \delta\}T_0 + C\sqrt{T_0} + s_{max} \leq -\epsilon T_0. \tag{6}$$

Finally, define

$$A = \{\alpha = (\alpha_1, \alpha_2, l) \in \mathcal{A} : \max\{\alpha_1, \alpha_2\} \leq s_{max} T_0\}. \tag{7}$$

We begin with the second inequality in (3). For ease of exposition define $Y(t) = f(X(t))$, and denote

$$\Delta Y(t_1, t_2) \overset{\triangle}{=} Y(t_2) - Y(t_1) = f(X(t_2)) - f(X(t_1)).$$

For any $\alpha \in \mathcal{A}$, the difference in the sum of the queue lengths is upper-bounded by the number of arrivals,

$$\Delta Y(0, T_0) \leq \sum_{t=0}^{T_0 - 1} a(t).$$

Therefore

$$\mathbb{E}_\alpha[\Delta Y(0, T_0)] \leq \lambda T_0 < \infty.$$

Next, to prove the first inequality in (3), consider an initial state $X(0){=}\alpha$ such that $\alpha \notin A$. By the definition of A, we have $\alpha_i > s_{max} T_0$ for at least one server i. Without loss of generality, assume i=1. Since s_{max} is the maximum number of jobs that server 1 can complete in a time slot, its queue length cannot hit zero within T_0 time slots. However, the queue length at server 2 may hit zero during this time period. Define the first time that queue 2 hits zero as

$$\hat{\sigma} = \inf\{t : t \geq 0, Q_2(t) = 0\}, \tag{8}$$

where the infimum over the empty set is ∞. Note that $\hat{\sigma}$ is not necessarily in $[0, T_0]$. Define

$$\sigma = \hat{\sigma} \wedge T_0. \tag{9}$$

Note that $\sigma < T_0$ means that queue 2 hits zero during $[0, T_0 - 1]$. If $\sigma = T_0$, then queue 2 is non empty during $[0, T_0 - 1]$ and may be empty at time slot T_0. By definition of ΔY we obtain

$$\mathbb{E}_\alpha[\Delta Y(0, T_0)] = \mathbb{E}_\alpha[\Delta Y(0, \sigma)] + \mathbb{E}_\alpha[\Delta Y(\sigma, T_0)], \tag{10}$$

where we use the conventions that if $a > b$, then $\sum_a^b = 0$ and $[a, b] = \emptyset$. The argument proceeds by analyzing the members of the right-hand side of (10) separately. By the definition of T_0, regardless of the identity of the last-idle server l at time 0, queue 1 does not hit zero during the interval $[0, T_0]$. Therefore,

$$d_1(t) = s_1(t) \text{ for } t \in [0, \sigma - 1].$$

In addition, since σ is the first time queue 2 is empty, it holds that

$$d_2(t) = s_2(t) \text{ for } t \in [0, \sigma - 2].$$

We next use the fact that the difference between s_2 and d_2 is at most s_{max} to obtain the following upper bound,

$$\begin{aligned}\Delta Y(0, \sigma) &= \sum_{t=0}^{\sigma-1} \Big(a(t) - d_1(t) - d_2(t)\Big) \\ &\leq \sum_{t=0}^{\sigma-1} \Big(a(t) - s_1(t) - s_2(t)\Big) + s_{max}. \end{aligned} \tag{11}$$

Define $g(t) \overset{\triangle}{=} a(t) - s_1(t) - s_2(t)$. Note that

$$\sum_{t=0}^{\sigma-1} g(t) = \sum_{t=0}^{T_0-1} g(t) \mathbb{1}_{\{t<\sigma\}}. \tag{12}$$

The random variable $\mathbb{1}_{\{t<\sigma\}}$ equals zero if queue 2 hit zero during $[0, t]$, and equals one otherwise. Since we look at the queue length at the beginning of each time slot, the value of $\mathbb{1}_{\{t<\sigma\}}$ is determined by the primitive random variables a, s_1 and s_2 during $[0, t - 1]$, which are independent of $a(t)$, $s_1(t)$ and $s_2(t)$. Hence, $g(t)$ and $\mathbb{1}_{\{t<\sigma\}}$ are independent random variables. Therefore,

$$\mathbb{E}_\alpha\left[\sum_{t=0}^{T_0-1} g(t) \mathbb{1}_{\{t<\sigma\}}\right] = \sum_{t=0}^{T_0-1} \mathbb{E}_\alpha[g(t)] \mathbb{P}_\alpha(t < \sigma). \tag{13}$$

Now, since a, s_1 and s_2 are independent of the initial state α, we obtain

$$\mathbb{E}_\alpha[g(t)] = \mathbb{E}[g(t)] = \lambda - \mu_1 - \mu_2 = -\delta. \tag{14}$$

By definition,

$$\sum_{t=0}^{T_0-1} \mathbb{P}_\alpha(t < \sigma) = \mathbb{E}_\alpha[\sigma]. \tag{15}$$

Using (11), (12), (13), (14) and (15), we obtain

$$\mathbb{E}_\alpha[\Delta Y(0,\sigma)] \leq -\delta \mathbb{E}_\alpha[\sigma] + s_{max}. \tag{16}$$

Next, $\Delta Y(\sigma, T_0)$ satisfies,

$$\begin{aligned}\Delta Y(\sigma, T_0) &= \sum_{t=\sigma}^{T_0-1} \Big(a(t) - d_1(t) - d_2(t)\Big) \\ &= \sum_{t=\sigma}^{T_0-1} \Big(a(t) - d_2(t)\Big) - \sum_{t=\sigma}^{T_0-1} s_1(t).\end{aligned} \tag{17}$$

We use the same arguments as in (12), (13) and (15) to obtain

$$\mathbb{E}_\alpha\left[\sum_{t=\sigma}^{T_0-1} s_1(t)\right] = \mu_1(T_0 - \mathbb{E}_\alpha[\sigma]). \tag{18}$$

We turn to analyze the first member of the right-hand side of (17). Define,

$$R(s,T) \triangleq \sum_{t=s}^{T-1} \Big(a(t) - d_2(t)\Big). \tag{19}$$

By our conventions,

$$R(T,T) = 0. \tag{20}$$

Write

$$\mathbb{E}_\alpha[R(\sigma, T_0)] = \sum_{y=0}^{T_0} \mathbb{E}_\alpha[R(\sigma, T_0) | \sigma = y] \cdot \mathbb{P}_\alpha(\sigma = y). \tag{21}$$

By Lemma 3.2, whose statement and proof appear below, it holds that

$$\mathbb{E}_\alpha[R(\sigma, T_0)|\sigma = y] \leq [\lambda - \mu_2]^+(T_0 - y) + C\sqrt{T_0}. \tag{22}$$

Using (22) together with (17), (18), (21) and the fact that,

$$[\lambda - \mu_2]^+ - \mu_1 = [\mu_1 - \delta]^+ - \mu_1 = -\min\{\mu_1, \delta\},$$

we obtain

$$\mathbb{E}_\alpha[\Delta Y(\sigma, T_0)] \leq -\min\{\mu_1, \delta\}(T_0 - \mathbb{E}_\alpha[\sigma]) + C\sqrt{T_0}. \tag{23}$$

Finally, by (10), (16) and (23), we obtain

$$\begin{aligned}\mathbb{E}_\alpha[\Delta Y(0, T_0)] \leq& -\min\{\mu_1, \delta\}T_0 + (\min\{\mu_1, \delta\} - \delta)\mathbb{E}_\alpha[\sigma] \\ &+ C\sqrt{T_0} + s_{max} \\ \leq& -\min\{\mu_1, \mu_2, \delta\}T_0 + C\sqrt{T_0} + s_{max} \\ \leq& -\epsilon T_0,\end{aligned} \tag{24}$$

where the last inequality is due to the specific choice of T_0 in (6). This concludes the proof. $\square$

Lemma 3.2. *We have*

$$\mathbb{E}_\alpha[R(\sigma, T_0)|\sigma = y] \le [\lambda - \mu_2]^+(T_0 - y) + C\sqrt{T_0}, \tag{25}$$

where C is defined in (5).

Proof. By (20), for $y = T_0$, the bound holds trivially. Next, consider $0 \le y < T_0$. Server 2 starts empty at time slot y, and all of the work is assigned to it during the $T_0 - y$ time slots in $[y, T_0 - 1]$. Therefore, $R(y, T_0)$ is the position of a *reflected random walk* starting at zero, after $T_0 - y$ steps and step size $a(t) - s_2(t)$ (where $y \le t < T_0$). Define the random walk

$$S_y(t) = \sum_{s=y}^{t-1}(a(s) - s_2(s)).$$

By (19) and proposition 6.3 in [34],

$$R(y, T_0) = S_y(T_0) + \max_{t\in[y,T_0]} \big(-S_y(t)\big). \tag{26}$$

Note that $S_y(y) = 0$. Denote by $\rho = \lambda - \mu_2$ the average step size of the random walk. Then,

$$\begin{aligned}
\max_{t\in[y,T_0]} \big(-S_y(t)\big) &= \max_{t\in[y,T_0]} \big(-S_y(t) + \rho(t-y) - \rho(t-y)\big) \\
&\le \max_{t\in[y,T_0]} \big(-S_y(t) + \rho(t-y)\big) + \max_{t\in[y,T_0]} \big(-\rho(t-y)\big) \\
&\le \max_{t\in[y,T_0]} |-S_y(t) + \rho(t-y)| + [-\rho]^+(T_0 - y)
\end{aligned} \tag{27}$$

The first term in the right-hand side of (27) is the maximum of the absolute value of a centered random walk, which we denote by

$$\hat{S}_y(t) \triangleq -S_y(t) + \rho(t-y).$$

Since $\hat{S}$ is a martingale, by the $\mathcal{L}^p$ maximum inequality (Theorem 5.4.3 of [35]) with $p = 2$,

$$\mathbb{E}[(\max|\hat{S}_y(t)|)^2] \le 4(\sigma_a^2 + \sigma_{s_2}^2)(T_0 - y). \tag{28}$$

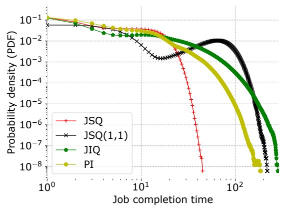
Using Jensen's inequality, we obtain

$$\begin{aligned}
\mathbb{E}[\max|\hat{S}_y(t)|] &\le \sqrt{4(\sigma_a^2 + \sigma_{s_2}^2)(T_0 - y)} \\
&\le 2\sqrt{(\sigma_a^2 + \sigma_{s_1}^2 + \sigma_{s_2}^2)}\sqrt{T_0} \\
&= C\sqrt{T_0}
\end{aligned} \tag{29}$$

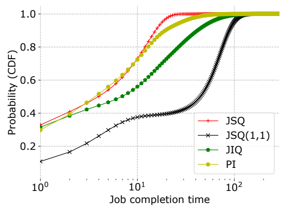
By (26), (27), (29) and the fact that $\mathbb{E}[S_y(T_0)] = \rho(T_0 - y)$,

$$\begin{aligned}
\mathbb{E}[R(y, T_0)] &\le \rho(T_0 - y) + [-\rho]^+(T_0 - y) + C\sqrt{T_0} \\
&= [\rho]^+(T_0 - y) + C\sqrt{T_0} \\
&= [\lambda - \mu_2]^+(T_0 - y) + C\sqrt{T_0}.
\end{aligned} \tag{30}$$

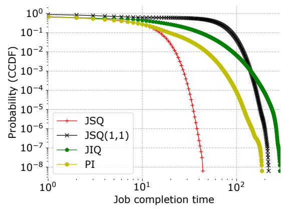
This concludes the proof. □



(a) PDF.

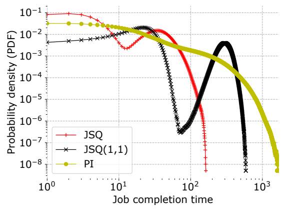


(b) CDF.

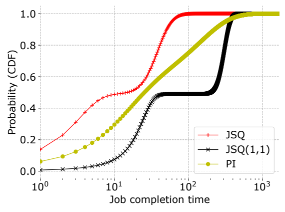


(c) CCDF.

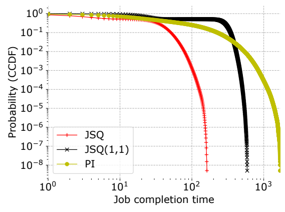
Figure 3: PDF, CDF and CCDF of the job completion time at 80% normalized load for the heterogeneous case with one strong server and ten slower servers. Note that the PDF and CCDF are in log scale. JSQ(2) is unstable and therefore not shown. **(a)** The PDF shows how the probability density for PI decreases faster at the tail than for JSQ(1,1) and JIQ. We also note the two-humped shape of JSQ(1,1), which may be related to whether the server in memory is a slow or a fast one. Indeed, we found that this phenomena does not occur in the homogeneous case (not displayed). **(b)** the CDF shows that PI is close to JSQ up to about the 80th percentile; **(c)** and the CCDF stresses how PI outperforms the other algorithms at the 90th and 99th percentiles (corresponding to 10^{-1} and 10^{-2} in the CCDF y-axis).



(a) PDF.



(b) CDF.



(c) CCDF.

Figure 4: PDF, CDF and CCDF of the job completion time at 98% normalized load for the heterogeneous case with 11 servers. Note that the PDF and CCDF are in log scale. JSQ(2) and JIQ are unstable and not displayed. **(a)** The PDF for JSQ(1,1) declines faster at the tail than for PI; however, **(b)** the CDF shows how PI out-performs JSQ(1,1) until the 96th percentile; **(c)** the CCDF illustrates this more clearly.

4. Evaluation

4.1. Evaluation setup

Algorithms. We evaluate the performance of the PI policy and several policies from the literature: Join-the-Shortest-Queue (*JSQ*) [18, 19], the power-of-choice *JSQ(d)* [20, 21, 22, 23, 24], the power-of-memory *JSQ*(d, m) [27], and Join-the-Idle-Queue (*JIQ*) [28, 29, 25, 30, 31].

Arrivals and services. We consider a discrete-time system consisting of one dispatcher and n heterogeneous servers. A sequence of i.i.d. Pois(λ) RVs (and later a heavy-tailed Pareto RVs with tail index $\alpha = 2$ and rate λ) determine how many jobs arrive at the dispatcher at each time slot. All jobs that arrive at the dispatcher are immediately forwarded to a unique server within the same time-slot (there is no possibility to load-balance within a time-slot). For every server $i \in \{1, ..., n\}$, a sequence of i.i.d Geom(μ_i) RVs determines the number of jobs it can potentially process at a time slot. Thus, the processing rate of server i is given by μ_i. We define the *normalized load* as $\lambda / \sum \mu_i$.

Systems. We consider several different scenarios (*e.g.,* heterogeneity and scale) to better understand the performance of PI in systems with random service capacity. First, we consider a heterogeneous setting with 10 servers of rate $\mu_1 = 1$ and one strong server (*e.g.,* accelerator) of rate $\mu_{11} = 10$. Next, we consider the impact of scale and multiply by 10 the number of servers. We first evaluate the homogeneous case with 110 servers of rate $\mu_1 = 1$. Then, we consider the heterogeneous case with 100 servers of rate $\mu_1 = 1$ and 10 strong servers of rate $\mu_{11} = 10$. Finally, we consider the same heterogeneous system with heavy-tailed Pareto arrivals.

Yardsticks. We measure several parameters to monitor performance. First, the mean number of jobs in the system as a function of the normalized load, to analyze stability and performance. Second, the mean rate of control messages (*e.g.,* queue sampling messages) as a function of the normalized load, to evaluate the control communication overhead. Finally, at given normalized loads, we plot the distributions of the job completion time, to better understand whether the sticky nature of PI impacts, for instance, the 90th or 99th percentiles of the completion time.

4.2. Heterogeneous case

We start by evaluating the system performance in the heterogeneous case consisting of 1 strong server and 10 slower servers. Figure 3 plots the PDF (probability density function), CDF (cumulative distribution function) and CCDF (complementary CDF) of the job completion time at 80% normalized load. JSQ(2) is unstable in this setting and therefore is omitted from this plot. The results show how PI manages to outperform JSQ(1,1) and JIQ at the median as well as 90th and 99th percentiles of the job completion time despite its sticky nature. Figure 4 provides the job completion time distributions at 98% normalized load. JSQ(2) and JIQ are unstable in this setting and therefore are omitted from this plot. The results show how PI outperforms JSQ(1,1) until the 95th job-completion-time percentile, but under-performs beyond.

4.3. Homogeneous case at scale

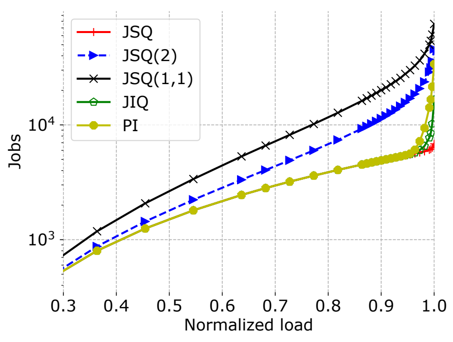
Figure 5 shows the performance in a homogeneous system with 110 identical servers. As expected, in this scenario all policies appear to be stable. JSQ(2) and JSQ(1,1) perform poorly compared to the other policies. JSQ, JIQ and PI are indistinguishable until 97% of the maximal load. The communication overhead of JSQ is two orders of magnitude larger than for the other policies.

4.4. Heterogeneous case at scale

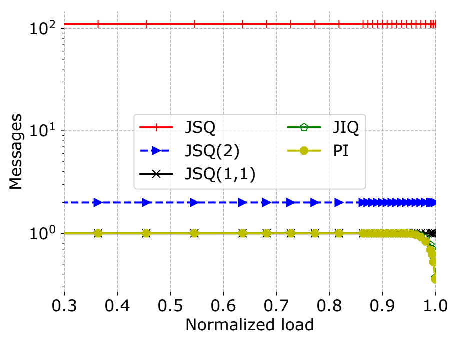
Figure 6 shows the performance in the scaled-up heterogeneous system. The results are similar to those in the smaller system (depicted in Figure 1). For high loads, JIQ performs poorly compared to PI and appears to be unstable. JSQ(2) appears to be unstable even at moderate loads. JSQ(1,1) appears to be stable but performs poorly compared to PI at any load (note the logarithmic y-scale). Figure 7 displays the CCDF of the job completion time in the scaled-up heterogeneous system, at 80% and at 96% normalized loads (such that JIQ is still stable). As the load increases, the 90th and 99th percentiles of the completion time under JSQ(1,1) and JIQ are higher than under PI. As expected, the communication overhead of JSQ is by two orders of magnitude larger compared to the other policies.

4.5. Heterogeneous case with Pareto arrivals

We now consider again the heterogeneous system with 110 servers, but with a more realistic data-center heavy-tailed arrival process [36]. Specifically, we use the Pareto distribution with a tail index of $\alpha = 2$. Figure 8(a) shows results that are similar to the previous case. JSQ(1,1), PI and JSQ appear to be stable. Figure 8(b) displays the CCDF of the job completion time at 90%

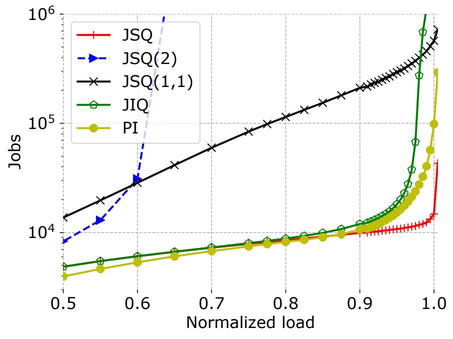


(a) Mean total number of jobs.

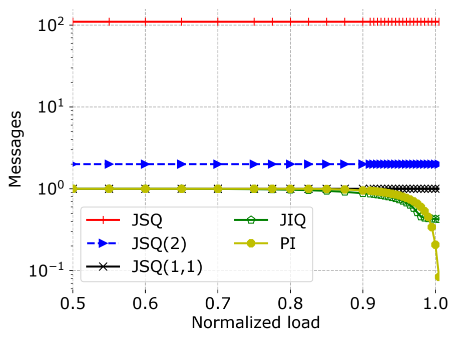


(b) Communication overhead.

Figure 5: Performance evaluation in an homogeneous setting with 110 identical servers of rate 1. As expected, all policies appear to be stable in this scenario. **(a)** The mean number of jobs is similar for JSQ, JIQ and PI. There is a slight advantage to JIQ over PI at the highest loads. Both JSQ(2) and JSQ(1,1) show poor performance, with a clear advantage to JSQ(2). **(b)** The control overhead of JSQ is larger than the overhead in other algorithms by two orders of magnitude, reflecting its impractical nature at scale. The control overhead of both JIQ and PI dips as the load approaches the capacity region.



(a) Mean total number of jobs.



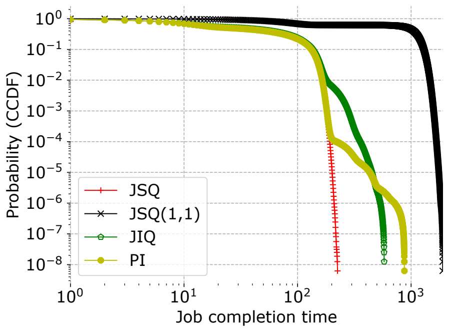
(b) Communication overhead.

Figure 6: Performance evaluation in the scaled-up heterogeneous case with 110 servers, *i.e.,* 10 strong servers of rate 10 and 100 weaker servers of rate 1. The results are similar to those of the heterogeneous case with 11 servers depicted in Figure 1. **(a)** Again, JSQ(2) and JIQ appear to be unstable, although the critical load for JIQ is now higher. In addition, the performance of JSQ(1,1) continues to be poor compared to PI. **(b)** The communication overhead of JSQ is by two orders of magnitude larger compared to the other policies.

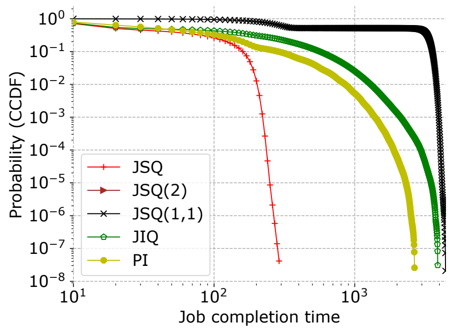
normalized load. The 99th percentile of the completion time for PI is lower by $2.9\times$ than for JIQ and by $2\times$ than for JSQ(1,1).

5. Conclusions.

This paper considers PI in a model with batch arrivals and random service capacity. Out main theoretical result is the stability of PI in a two server system. Since the technique presented in [32] does not seem to extend to the considered model, we used a different approach that relies

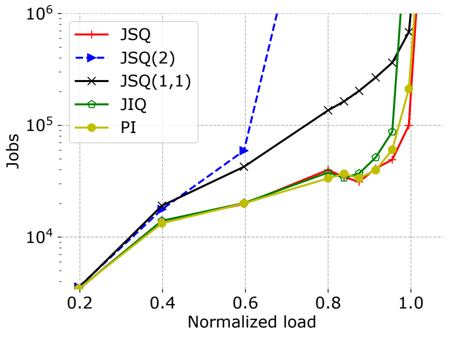


(a) CCDF at 80% load.

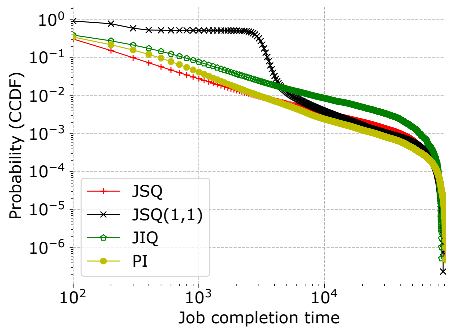


(b) CCDF at 96% load.

Figure 7: CCDF of the scaled-up heterogeneous case with 110 servers. **(a)** CCDF at 80% normalized load. PI follows JSQ surprisingly closely, until the 99.99th job-completion-time percentile (*i.e.,* 10^{-4} in the CCDF plot). JIQ also performs similarly, under-performing then out-performing PI. However, the performance of JSQ(1,1) is poor, with about an order or magnitude higher delay at most percentiles. **(b)** CCDF at 96% normalized load (this load is chosen so that JIQ is still stable). JSQ(1,1) performs poorly. JIQ's completion time exceeds that of PI by more than $3\times$ at the 90th percentile (*i.e.,* 10^{-1}).



(a) Mean total number of jobs.



(b) CCDF at 90% load.

Figure 8: Performance evaluation in the scaled-up heterogeneous system with heavy-tailed Pareto arrivals. **(a)** JSQ(2) appears to lose stability at 60% normalized load. JIQ shows similar performance to PI and JSQ, but degrades as the load increases (it appears to be unstable beyond 96.5%). JSQ(1,1) shows poor performance at reasonable loads, but performs surprisingly well just before the critical load; **(b)** illustrates the job completion time CCDF at 90% normalized load. PI closely follows JSQ. The 99th delay percentile of PI is by $2.9\times$ lower than for JIQ and by $2\times$ lower than for JSQ(1,1).

on analyzing the queue dynamics over sufficiently long time intervals.

The second contribution of this paper is a set of simulation results that compare PI to other state-of-the-art policies. Simulation results indicate that the performance of JSQ(2), JSQ(1,1) and JIQ is sensitive to system parameters (*e.g.,* service heterogeneity and system size). PI, on the other hand, performs well for all tested system parameters. Moreover, when the number of servers is large, the policy achieves a good balance between performance and communication overhead, including for the average and 99th-percentile job completion time.

[1] J. Dean, L. A. Barroso, The tail at scale, Communications of the ACM 56 (2) (2013) 74–80.

[2] P. Patel, D. Bansal, L. Yuan, A. Murthy, A. Greenberg, D. A. Maltz, R. Kern, H. Kumar, M. Zikos, H. Wu, et al., Ananta: Cloud scale load balancing, in: ACM SIGCOMM CCR, Vol. 43, 2013, pp. 207–218.

[3] D. E. Eisenbud, C. Yi, C. Contavalli, C. Smith, R. Kononov, E. Mann-Hielscher, A. Cilingiroglu, B. Cheyney, W. Shang, J. D. Hosein, Maglev: A fast and reliable software network load balancer., in: Usenix NSDI, 2016, pp. 523–535.

[4] J. Yang, L. Ling, H. Liu, A hierarchical load balancing strategy considering communication delay overhead for large distributed computing systems, Mathematical Problems in Engineering 2016.

[5] J. O. Gutierrez-Garcia, A. Ramirez-Nafarrate, Agent-based load balancing in cloud data centers, Cluster Computing 18 (3) (2015) 1041–1062.

[6] A. Bhadani, S. Chaudhary, Performance evaluation of web servers using central load balancing policy over virtual machines on cloud, in: ACM Bangalore Conference, 2010, p. 16.

[7] M. Randles, D. Lamb, A. Taleb-Bendiab, A comparative study into distributed load balancing algorithms for cloud computing, in: IEEE WAINA, 2010, pp. 551–556.

[8] K. Al Nuaimi, N. Mohamed, M. Al Nuaimi, J. Al-Jaroodi, A survey of load balancing in cloud computing: Challenges and algorithms, in: IEEE NCAA, 2012, pp. 137–142.

[9] Y. Desmouceaux, P. Pfister, J. Tollet, M. Townsley, T. Clausen, SRLB: The power of choices in load balancing with segment routing, in: IEEE ICDCS, 2017, pp. 2011–2016.

[10] R. Govindan, I. Minei, M. Kallahalla, B. Koley, A. Vahdat, Evolve or die: High-availability design principles drawn from googles network infrastructure, in: ACM SIGCOMM, 2016, pp. 58–72.

[11] A. M. Caulfield, E. S. Chung, A. Putnam, H. Angepat, J. Fowers, M. Haselman, S. Heil, M. Humphrey, P. Kaur, J.-Y. Kim, et al., A cloud-scale acceleration architecture, in: IEEE/ACM MICRO, 2016, pp. 1–13.

[12] C. Kachris, D. Soudris, A survey on reconfigurable accelerators for cloud computing, in: IEEE FPL, 2016, pp. 1–10.

[13] L.-W. Chang, J. Gómez-Luna, I. El Hajj, S. Huang, D. Chen, W.-m. Hwu, Collaborative computing for heterogeneous integrated systems, in: ACM/SPEC ICPE, 2017, pp. 385–388.

[14] I. Magaki, M. Khazraee, L. V. Gutierrez, M. B. Taylor, Asic clouds: specializing the datacenter, in: ACM/IEEE ISCA, 2016, pp. 178–190.

[15] W. Wang, B. Li, B. Liang, Dominant resource fairness in cloud computing systems with heterogeneous servers, in: IEEE Infocom, 2014, pp. 583–591.

[16] D. Breitgand, R. Cohen, A. Nahir, D. Raz, On cost-aware monitoring for self-adaptive load sharing, IEEE Journal on Selected Areas in communications 28 (1).

[17] T. Heath, B. Diniz, E. V. Carrera, W. Meira Jr, R. Bianchini, Energy conservation in heterogeneous server clusters, in: ACM SIGPLAN PPoPP, 2005, pp. 186–195.

[18] S. Foss, N. Chernova, On the stability of a partially accessible multi-station queue with state-dependent routing, Queueing Systems 29 (1) (1998) 55–73.

[19] M. Bramson, et al., Stability of join the shortest queue networks, The Annals of Applied Probability 21 (4) (2011) 1568–1625.

[20] M. Bramson, Y. Lu, B. Prabhakar, Randomized load balancing with general service time distributions, in: ACM SIGMETRICS, Vol. 38, 2010, pp. 275–286.

[21] C. Graham, Chaoticity on path space for a queueing network with selection of the shortest queue among several, Journal of Applied Probability 37 (1) (2000) 198–211.

[22] M. J. Luczak, C. McDiarmid, On the maximum queue length in the supermarket model, The Annals of Probability (2006) 493–527.

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

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

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

[26] R. Atar, I. Keslassy, G. Mendelson, Randomized load balancing in heavy traffic, Submitted.

[27] D. Shah, B. Prabhakar, The use of memory in randomized load balancing, in: IEEE ISIT, 2002, p. 125.

[28] R. Badonnel, M. Burgess, Dynamic pull-based load balancing for autonomic servers, in: IEEE NOMS, 2008, pp. 751–754.

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

[30] M. van der Boor, S. Borst, J. van Leeuwaarden, Load balancing in large-scale systems with multiple dispatchers, IEEE Infocom (2017) 613–621.

[31] A. L. Stolyar, Pull-based load distribution among heterogeneous parallel servers: the case of multiple routers,

Queueing Systems 85 (1-2) (2017) 31–65.

[32] R. Atar, I. Keslassy, G. Mendelson, A. Orda, S. Vargaftik, Persistent-idle load distribution, preprint. URL `https://rami.net.technion.ac.il/files/2018/09/PI.pdf`

[33] G. Fayolle, V. A. Malyshev, M. V. Menshikov, Topics in the constructive theory of countable Markov chains, Cambridge university press, 1995.

[34] S. Asmussen, Applied probability and queues, Vol. 51, Springer Science & Business Media, 2008.

[35] R. Durrett, Probability: theory and examples, Cambridge university press, 2010.

[36] S. Di, D. Kondo, F. Cappello, Characterizing cloud applications on a google data center, in: IEEE ICPP, 2013, pp. 468–473.