Load Balancing in Large-Scale Heterogeneous Systems with Multiple Dispatchers

Email: anonymous.lsq.preprint@gmail.com, Date: 27.12.2018

ABSTRACT

In recent years, the efficiency and even the feasibility of traditional load-balancing policies are challenged by the rapid growth of cloud infrastructure with increasing levels of server heterogeneity and increasing size of cloud services and applications. In such many-software-load-balancers heterogeneous systems, traditional solutions, such as *JSQ*, incur an increasing communication overhead, whereas low-communication alternatives, such as *JSQ*(d) and the recently proposed *JIQ* scheme are either unstable or provide poor performance.

We argue that a better low-communication load balancing scheme can be established by allowing each dispatcher to have a different view of the system and keep using *JSQ*, rather than greedily trying to avoid starvation on a per-decision basis. Accordingly, we introduce the *Loosely-Shortest-Queue* family of load balancing algorithms. Roughly speaking, in *Loosely-Shortest-Queue*, each dispatcher keeps a different approximation of the server queue lengths and routes jobs to the shortest among them. Communication is used only to update the approximations and make sure that they are not too far from the real queue lengths *in expectation*. We formally establish the strong stability of any *Loosely-Shortest-Queue* policy and provide an easy-to-verify sufficient condition for verifying that a policy is *Loosely-Shortest-Queue*. We further demonstrate that the *Loosely-Shortest-Queue* approach allows constructing throughput optimal policies with an arbitrarily low communication budget.

Finally, using extensive simulations that consider homogeneous, heterogeneous and highly skewed heterogeneous systems in scenarios with a single dispatcher as well as with multiple dispatchers, we show that the examined *Loosely-Shortest-Queue* example policies are always stable as dictated by theory. Moreover, it exhibits an appealing performance and significantly outperforms well-known low-communication policies, such as *JSQ*(d) and *JIQ*, while using a similar communication budget.

1 INTRODUCTION

Background. In recent years, due to the rapidly increasing size and heterogeneity of cloud services and applications [4, 8, 13, 20], the design of load balancing algorithms for parallel server systems has become extremely challenging. The goal of these algorithms is to efficiently load-balance incoming jobs to a large number of servers, even though these servers display large *heterogeneity* because of two reasons: First, current large-scale systems increasingly contain, in addition to multiple generations of CPUs (central processing units) [12], various types of accelerated devices such as GPUs (graphics processing units), FPGAs (field-programmable gate arrays) and ASICs (application-specific integrated circuit), with significantly higher processing speeds. Second, VMs (virtual machines) and/or containers are commonly used to deploy different services that share resources on the same servers, potentially leading to significant and unpredictable heterogeneity.

In a traditional server farm, a centralized load-balancer (dispatcher) can rely on a full-state-information policy with strong theoretical guarantees for heterogeneous servers, such as join-the-shortest-queue (*JSQ*), which routes emerging jobs to the server with the shortest queue [5, 6, 13, 29, 30]. This is because in such single-centralized-dispatcher scenarios, the dispatcher forms a single access point to the servers. Therefore, by merely receiving a notification from each server upon the completion of each job, it can track all queue lengths, because it knows the exact arrival and departure patterns of each queue (neglecting propagation times) [15]. The communication overhead between the servers and the dispatcher is at most a single message per job, which is appealing and does not increase with the number of servers.

However, in current clouds, which keep growing in size and thus have to rely on multiple dispatchers [10], implementing a policy like *JSQ* may involve a prohibitive implementation overhead as the number m of dispatchers increases [15]. This is because each server needs to keep all m dispatchers updated as jobs arrive and complete, leading to up to $O(m)$ communication messages per job. This large communication overhead makes scaling the number of dispatchers difficult, and forces cloud dispatchers to rely on heuristics that do not provide any service guarantees with heterogeneous servers. For instance, in L7 load-balancers, multi-dispatcher services are essentially decomposed into several fully-independent single-dispatcher services, where each dispatcher applies either round-robin or *JSQ* reduced to its own jobs only [1, 21, 25]. Unfortunately, such an approach suffers from lack of predictable guarantees, lack of a global view of the system, and communication bursts with potential incast issues.

Related work. Despite their increasing importance, scalable policies for heterogeneous systems with multiple dispatchers have received little attention in the literature. In fact, as we later discuss, the only suggested scalable policies that address the many-dispatcher scenario in an heterogeneous setting are based on join-the-idle-queue (*JIQ*), and none of them is stable [32].

In the *JSQ*(d) (power-of-choice) policy, to make a routing decision, a dispatcher samples $d \geq 2$ queues uniformly at random and chooses the shortest among them [2, 3, 9, 18, 31]. *JSQ*(d) is stable in systems with homogeneous servers. However, with heterogeneous servers, *JSQ*(d) leads to poor performance and even to instability, both with a single and multiple dispatchers [7].

In the *JSQ*(d, m) (power-of-memory) policy, the dispatcher samples the m shortest queues from the previous decision in addition to $d \geq m \geq 1$ new queues chosen uniformly-at-random [17, 22]. The job is then routed to the shortest among these $d + m$ queues. *JSQ*(d, m) has been shown to be stable in the case of a single dispatcher, even with heterogeneous servers. However, it offers poor performance, and has not been considered with multiple dispatchers.

A recent work [32] proposes a class of policies that are both throughput optimal and heavy-traffic delay optimal. However, their assumptions are not aligned with our system model and motivation, due to several reasons: (1) For heterogeneous servers, they require the knowledge of the server service rates, which may not be achievable in practice. (2) They assume that the number of jobs a server may complete in a time slot and that may arrive at a dispatcher in a time slot are deterministically upper-bounded, which rules out important modeling options with unbounded support, such as geometric services or Poisson arrivals. (3) Most importantly, they consider only a single dispatcher, and it is unclear whether their analysis and performance guarantees can be extended to multiple dispatchers.

In addition, to address the communication overhead in systems with multiple dispatchers, the *JIQ* policy has been proposed [15, 16, 23, 24, 27]. Roughly speaking, in *JIQ*, each dispatcher routes jobs to an idle server, if it is aware of any, and to a random server otherwise. Servers may only notify dispatchers when they become idle. *JIQ* achieves low communication overhead of at most a single message per job, irrespective of the number of dispatchers, and good performance at low and moderate loads when servers are homogeneous [15]. However, for heterogeneous servers, *JIQ* is not stable, *i.e.,* it fails to achieve 100% throughput [32].

Finally, a recent manuscript on low-communication load balancing [14] proposes to use local memory not just to remember the last m shortest sampled queues as done by $JSQ(d, m)$, but also to maintain the last known size of all servers. As we later show, this policy is in fact a special case of *Loosely-Shortest-Queue*.[1] However, the manuscript [14] only considers a single dispatcher with homogeneous servers, while we allow for multiple dispatchers and heterogeneous servers, as well as for a general communication policy between servers and dispatchers.

Contributions. This paper makes the following contributions:

Loosely-Shortest-Queue (LSQ). We introduce *Loosely-Shortest-Queue*, a stable low-communication family of load balancing algorithms for large-scale heterogeneous systems with multiple dispatchers. As Figure 1 illustrates, in *Loosely-Shortest-Queue*, each dispatcher keeps a different approximation of the server queue lengths, and routes jobs to the shortest among them. It relies on communication with the servers to keep a bounded expected difference between the approximated server queue lengths and the real ones.

Stability proof. Using a multi-stage argument, we prove that all *Loosely-Shortest-Queue* policies are strongly stable, *i.e.,* keep bounded expected queue lengths. The main difficulty in the proof arises from the fact that the decisions taken by an *Loosely-Shortest-Queue* policy depend on the *local view of each dispatcher*, hence on a potentially long history of system states. To address this challenge, we introduce two additional stable policies into our analysis: (1) *JSQ* and (2) Weighted-Random (*WR*). Broadly speaking, we show that our policy is sufficiently similar to *JSQ* which, in turn, is better than *WR*. We complete the proof by using the fact that in *WR*, the routing decisions do not depend on the system state (unlike *JSQ*).

Sufficient stability condition. It can be challenging to prove that a policy is *Loosely-Shortest-Queue*, *i.e.,* that in expectation, the local dispatcher views are not too far from the real queue lengths. Therefore, we develop a simple sufficiency condition to prove that a policy belongs to the *Loosely-Shortest-Queue* family, and exemplify its use. Intuitively, the condition states that there is a non-zero probability that a server updates a dispatcher at each time-slot.

Example Loosely-Shortest-Queue policies. Since *Loosely-Shortest-Queue* is not restricted to work with either push (*i.e.,* dispatchers sample the servers) or pull (*i.e.,* servers update the dispatchers) based communication, we aim to achieve the same communication overhead as the lowest-overhead/best-known examples in each class. Accordingly, we show how two of the newest existing low communication policies are in fact *Loosely-Shortest-Queue* and how to construct new *Loosely-Shortest-Queue* policies with communication patterns similar to that of other low-communication policies such as the push-based $JSQ(2)$ and the pull-based *JIQ*, but with significantly stronger theoretical guarantees and empirical performance.

Extensive simulations. Using extensive simulations considering homogeneous, heterogeneous, and highly-skewed heterogeneous systems, in scenarios of a single as well as multiple dispatchers, we show how simple *Loosely-Shortest-Queue* policies are always stable in practice, present appealing performance, and significantly outperform other low-communication policies using an equivalent communication budget.

2 SYSTEM MODEL

We consider a system with a set $M = \{1, 2, \dots, m\}$ of dispatchers load-balancing incoming jobs among a set $N = \{1, 2, \dots, n\}$ of possibly-heterogeneous work-conserving servers.

Time slots. We assume a time slotted system with the following order of events within each time slot: (1) jobs arrive at each dispatcher; (2) a routing decision is taken by each dispatcher and it immediately forwards its jobs to one of the servers; (3) each server performs its service for this time-slot.

Dispatchers. As mentioned, each of the m dispatchers does not store incoming jobs, and instead immediately forwards them to one of the n servers. We denote by $a^j(t)$ the number of exogenous job arrivals at dispatcher j at the beginning of time slot t. We make the following assumption:

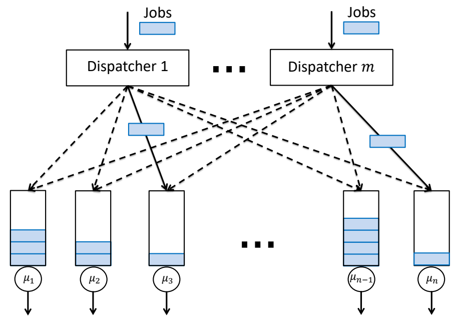
$$\left\{ a(t) = \sum_{j=1}^{m} a^j(t) \right\}_{t=0}^{\infty} \quad \text{is an } i.i.d. \text{ process} \tag{1}$$

$$\mathbb{E}[a(0)] = \lambda^{(1)} \tag{2}$$

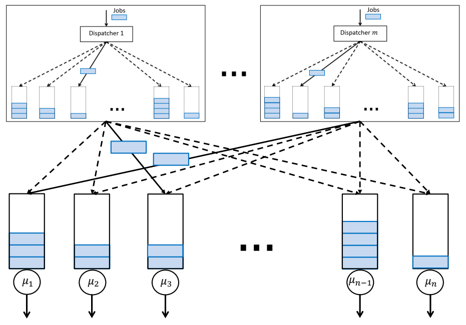
$$\mathbb{E}\left[\Big(a(0)\Big)^2\right] = \lambda^{(2)} \tag{3}$$

That is, we only assume that the total job arrival process to the system is $i.i.d.$ over time slots and admits finite first and second moments. The division of arriving jobs among the dispatchers is assumed to follow any arbitrary policy that does not depend on the system state (*i.e.,* queue lengths). We only assume that there is

[1]In addition, [28] mentions that a preprint [26] introduces another similar low-communication load-balancing, which also seems to be a special case of *Loosely-Shortest-Queue*, as we later discuss. However, the preprint [26] did not seem to be available online.



(a) The *JSQ* approach.



(b) The *Loosely-Shortest-Queue* approach.

Figure 1: *JSQ* vs. *Loosely-Shortest-Queue*. (a) *JSQ* requires instant knowledge of all server queues by all dispatchers, resulting in substantial communication overhead. (b) At each dispatcher, *Loosely-Shortest-Queue* relies on limited current and past information from the servers to build an approximated local view of all the server queue sizes. For instance, dispatcher 1 believes that the queue length at server 3 is 1, while it is 2. It then sends jobs to the shortest queue as dictated by its view (here, to server 3 rather than server n) . Communication is used only to update the local views of the dispatchers, *i.e.,* to improve their approximations.

a positive probability of job arrivals at all dispatchers. That is, we assume that there exists a strictly positive constant ϵ_0 such that

$$\mathbb{P}(a^j(t) > 0) > \epsilon_0 \quad \forall (j,t) \in M \times \mathbb{N}. \tag{4}$$

This, for example, covers complex scenarios with time-varying arrival rates to the different dispatchers that are not necessarily independent. We are not aware of previous work covering such scenarios.

We further denote $a_i^j(t)$ as the number of jobs forwarded by dispatcher j to server i at the beginning of time slot t. Let

$$a_i(t) = \sum_{j=1}^{m} a_i^j(t)$$

be the total number of jobs forwarded to server i at time slot t by all dispatchers.

Servers. Each server has a FIFO queue for storing incoming jobs. Let $Q_i(t)$ be the queue length of server i at the beginning of time slot t (before any job arrivals and departures at time slot t). We denote by $s_i(t)$ the potential service offered to queue i at time slot t. That is, $s_i(t)$ is the maximum number of jobs that can be completed by server i at time slot t. We assume that, for all $i \in N$,

$$\{s_i(t)\}_{t=0}^{\infty} \text{ is } i.i.d. \text{ over time slots} \tag{5}$$

$$\mathbb{E}[s_i(0)] = \mu_i^{(1)} \tag{6}$$

$$\mathbb{E}\left[\left(s_i(0)\right)^2\right] = \mu_i^{(2)} \tag{7}$$

Namely, we assume that the service process of each server is $i.i.d.$ over time slots and admits finite first and second moments. We also assume that all service processes are mutually independent across the different servers and also independent of the arrival processes.

3 *Loosely-Shortest-Queue* LOAD BALANCING

In this section we formally introduce the *Loosely-Shortest-Queue* family of load balancing policies and then prove that any *Loosely-Shortest-Queue* policy is stable.

3.1 The *Loosely-Shortest-Queue* Family

We assume that at each time slot t, each dispatcher $j \in M$ holds a local-view estimation of each server's $i \in N$ queue length, denoted by $\tilde{Q}_i^j(t)$. We begin by introducing the two following assumptions on these estimations and the dispatchers way of operation that define the *Loosely-Shortest-Queue* family of load balancing policies.

Assumption 1 (Local view proximity). *There exists a constant $C > 0$ such that at the beginning of each time slot (before any arrivals and departures), it holds that*

$$\mathbb{E}\left[\left|Q_i(t) - \tilde{Q}_i^j(t)\right|\right] \le C \quad \forall (i,j,t) \in N \times M \times \mathbb{N}. \tag{8}$$

As we later show, this assumption provides some appealing flexibility when designing a load balancing policy.

Assumption 2 (Local view based routing). *At each time slot, each dispatcher j follows the JSQ policy based on its local view of the queue lengths,* i.e., $\{\tilde{Q}_i^j(t)\}_{i=1}^n$. *That is, dispatcher j forwards all its incoming jobs at time slot t to server i^* where $i^* \in argmin_i\{\tilde{Q}_i^j(t)\}_{i=1}^n$ (ties are broken randomly).*

Finally, we term a load balancing policy as an *Loosely-Shortest-Queue* (Local Shortest Queue) policy if it respects Assumptions 1 and 2.

3.2 Stability of *Loosely-Shortest-Queue*

We now prove that any *Loosely-Shortest-Queue* load balancing policy is stable. We begin by formally stating our considered form of stability.

Definition 1 (Strong stability). *We say that the system is strongly stable* iff *there exists a constant $K \ge 0$ such that*

$$\limsup_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \sum_{i=1}^{n} \mathbb{E}\left[Q_i(t)\right] \le K.$$

That is, the system is strongly stable when the expected time averaged sum of queue lengths admits a constant upper bound.

Strong stability is a strong form of stability that implies finite average backlog and (by Little's theorem) finite average delay. Furthermore, under mild conditions, it implies other commonly considered forms of stability, such as steady state stability, rate stability, mean rate stability and more (see [19]). Note that strong stability has been widely used in queueing systems (see [11] and references therein) whose state does not necessarily admit an irreducible and aperiodic Markov chain representation (therefore positive-recurrence may not be considered).

Theorem 1. *Assume the system is sub-critical, i.e., assume that there exists $\epsilon > 0$ such that*

$$\sum_{i=1}^{n} \mu_i^{(1)} - \lambda^{(1)} = \epsilon. \tag{9}$$

Then, any Loosely-Shortest-Queue policy is strongly stable.

Proof. A server can work on a job immediately upon its arrival. Therefore, the queue dynamics at server i are given by

$$Q_i(t+1) = [Q_i(t) + a_i(t) - s_i(t)]^+, \tag{10}$$

where $[\cdot]^+ \equiv \max\{\cdot, 0\}$. Squaring both sides of (10) yields

$$\begin{aligned}\Big(Q_i(t+1)\Big)^2 \leq \Big(Q_i(t)\Big)^2 + \Big(a_i(t)\Big)^2 + \Big(s_i(t)\Big)^2 \\ + 2a_i(t)Q_i(t) - 2s_i(t)Q_i(t) - 2s_i(t)a_i(t).\end{aligned} \tag{11}$$

Rearranging (11) and omitting the last term yields

$$\begin{aligned}&\Big(Q_i(t+1)\Big)^2 - \Big(Q_i(t)\Big)^2 \leq \\ &\Big(a_i(t)\Big)^2 + \Big(s_i(t)\Big)^2 - 2Q_i(t)\Big(s_i(t) - a_i(t)\Big).\end{aligned} \tag{12}$$

Summing over the servers yields

$$\begin{aligned}&\sum_{i=1}^{n}\Big(Q_i(t+1)\Big)^2 - \sum_{i=1}^{n}\Big(Q_i(t)\Big)^2 \leq \\ &B(t) - 2\sum_{i=1}^{n} Q_i(t)\Big(s_i(t) - a_i(t)\Big),\end{aligned} \tag{13}$$

where

$$B(t) = \sum_{i=1}^{n}\Big(a_i(t)\Big)^2 + \sum_{i=1}^{n}\Big(s_i(t)\Big)^2. \tag{14}$$

We would like to proceed in our analysis by taking the expectation of (13). However, first we need to analyze the term

$$\sum_{i=1}^{n} Q_i(t)\Big(s_i(t) - a_i(t)\Big),$$

since $\{Q_i(t)\}_{i=1}^{n}$ and $\{a_i(t)\}_{i=1}^{n}$ are dependent.

Our plan is to use the following recipe. We will introduce two additional policies into our analysis: (1) *JSQ* and (2) Weighted-Random (*WR*). Roughly speaking, we will show that the routing decision that is taken by our policy at each dispatcher and each time slot t is sufficiently similar to the decision that would have been made by *JSQ given the same system state*, which, in turn, is no worse than the decision that *WR* would make at that time slot. Since in *WR* the routing decisions taken at time slot t do not depend on the system state at time slot t, we will obtain the desired independence, which allows us to continue with the analysis. We start by introducing the corresponding *JSQ* and *WR* notations.

Let

$$a_i^{JSQ}(t) = \sum_{j=1}^{m} a_i^{j,JSQ}(t)$$

be the number of jobs that will be routed to server i at time slot t when using *JSQ at time slot t*. That is, each dispatcher forwards its incoming jobs to the server with the shortest queue (ties are broken randomly). Formally, let $i^* \in argmin_i\,\{Q_i(t)\}$, then $\forall j \in M$

$$a_i^{j,JSQ}(t) = \begin{cases} a^j(t), & i = i^* \\ 0, & i \neq i^*. \end{cases} \tag{15}$$

Let

$$a_i^{WR}(t) = \sum_{j=1}^{m} a_i^{j,WR}(t)$$

be the number of jobs that will be routed to server i at time slot t when using *WR at time slot t*. That is, each dispatcher forwards its incoming jobs to a single randomly-chosen server, where the probability of choosing server i is $\frac{\mu_i^{(1)}}{\sum_{i=1}^{n}\mu_i^{(1)}}$. Formally, $\forall j \in M$, $i = i^*$ with probability $\frac{\mu_i^{(1)}}{\sum_{i=1}^{n}\mu_i^{(1)}}$ and

$$a_i^{j,WR}(t) = \begin{cases} a^j(t), & i = i^* \\ 0, & i \neq i^* \end{cases} \tag{16}$$

With these notations at hand, we continue our analysis by adding and subtracting the term $2\sum_{i=1}^{n} a_i^{JSQ}(t)Q_i(t)$ from the right hand side of (13). This yields

$$\begin{aligned}&\sum_{i=1}^{n}\Big(Q_i(t+1)\Big)^2 - \sum_{i=1}^{n}\Big(Q_i(t)\Big)^2 \leq \\ &B(t) - 2\sum_{i=1}^{n} Q_i(t)\Big(s_i(t) - a_i^{JSQ}(t)\Big) + \\ &2\sum_{i=1}^{n} Q_i(t)\Big(a_i(t) - a_i^{JSQ}(t)\Big).\end{aligned} \tag{17}$$

We would like to take the expectation of (17). However, as mentioned, since the actual queue lengths and the local views of the dispatchers and the routing decisions that are made both by our policy and *JSQ* are dependent, we shall rely on the *WR* policy and the expected distance of the local views from the actual queue lengths to evaluate the expected values. To that end, we now introduce the following lemmas.

Lemma 1. *For all time slots t, it holds that*

$$\sum_{i=1}^{n} a_i^{JSQ}(t)Q_i(t) \leq \sum_{i=1}^{n} a_i^{WR}(t)Q_i(t).$$

Proof. See Section 7.1. □

Lemma 2. *For all servers $i \in N$ and all time slots t, it holds that*

$$\sum_{i=1}^{n} Q_i(t)\Big(a_i(t) - a_i^{JSQ}(t)\Big) \leq \sum_{i=1}^{n}\sum_{j=1}^{m} a(t)\Big|Q_i(t) - \tilde{Q}_i^j(t)\Big|.$$

Proof. See Section 7.2. □

Using Lemmas 1 and 2 in (17) yields

$$\sum_{i=1}^{n}\Big(Q_i(t+1)\Big)^2-\sum_{i=1}^{n}\Big(Q_i(t)\Big)^2\le$$
$$B(t)-2\sum_{i=1}^{n}Q_i(t)\Big(s_i(t)-a_i^{WR}(t)\Big) \tag{18}$$
$$+\,2\sum_{i=1}^{n}\sum_{j=1}^{m}a(t)\Big|Q_i(t)-\tilde{Q}_i^j(t)\Big|.$$

Taking the expectation of (18) yields

$$\mathbb{E}\Big[\sum_{i=1}^{n}\Big(Q_i(t+1)\Big)^2\Big]-\mathbb{E}\Big[\sum_{i=1}^{n}\Big(Q_i(t)\Big)^2\Big]\le$$
$$\mathbb{E}\Big[B(t)\Big]-2\,\mathbb{E}\Big[\sum_{i=1}^{n}Q_i(t)\Big(s_i(t)-a_i^{WR}(t)\Big)\Big] \tag{19}$$
$$+\,2\,\mathbb{E}\Big[\sum_{i=1}^{n}\sum_{j=1}^{m}a(t)\Big|Q_i(t)-\tilde{Q}_i^j(t)\Big|\Big].$$

We now observe that both $a(t)$ (according to (1)) and $\{a_i^{WR}(t)\}_{i=1}^n$ (according to the definition of the WR policy) are independent of $\{Q_i(t)\}_{i=1}^n$ and $\Big\{\tilde{Q}_i^j(t)\,\Big|\,(i,j)\in N\times M\Big\}$. Applying this observation to (19) and using the linearity of expectation yields

$$\mathbb{E}\Big[\sum_{i=1}^{n}\Big(Q_i(t+1)\Big)^2\Big]-\mathbb{E}\Big[\sum_{i=1}^{n}\Big(Q_i(t)\Big)^2\Big]\le$$
$$\mathbb{E}\Big[B(t)\Big]-2\sum_{i=1}^{n}\mathbb{E}\Big[Q_i(t)\Big]\,\mathbb{E}\Big[\Big(s_i(t)-a_i^{WR}(t)\Big)\Big] \tag{20}$$
$$+\,2\sum_{i=1}^{n}\sum_{j=1}^{m}\mathbb{E}\Big[a(t)\Big]\,\mathbb{E}\Big[\Big|Q_i(t)-\tilde{Q}_i^j(t)\Big|\Big].$$

Next, since for any non-negative $\{x_1,x_2,\dots,x_n\}$ such that

$$x=x_1+x_2+\ldots+x_n$$

it always holds that $x^2\ge\sum_{i=1}^n x_i^2$, using (5)-(7), the linearity of expectation and (1)-(3), we obtain

$$\mathbb{E}\Big[B(t)\Big]=\mathbb{E}\Big[\sum_{i=1}^{n}\Big(a_i(t)\Big)^2\Big]+\mathbb{E}\Big[\sum_{i=1}^{n}\Big(s_i(t)\Big)^2\Big]\le$$
$$\mathbb{E}\Big[\Big(a(t)\Big)^2\Big]+\sum_{i=1}^{n}E\Big[\Big(s_i(t)\Big)^2\Big]=\lambda^{(2)}+\sum_{i=1}^{n}\mu_i^{(2)}. \tag{21}$$

Additionally, using (1), (2) and (8) yields

$$\sum_{i=1}^{n}\sum_{j=1}^{m}\mathbb{E}\Big[a(t)\Big]\,\mathbb{E}\Big[\Big|Q_i(t)-\tilde{Q}_i^j(t)\Big|\Big]\le$$
$$\sum_{i=1}^{n}\sum_{j=1}^{m}\lambda^{(1)}C=mn\lambda^{(1)}C. \tag{22}$$

Finally, since the decisions taken by the WR policy are independent of the system state, we can introduce the following lemma.

Lemma 3. *For all $i\in N$ and t it holds that*

$$\mathbb{E}\Big[s_i(t)-a_i^{WR}(t)\Big]=\frac{\epsilon\mu_i^{(1)}}{\sum_{i=1}^n\mu_i^{(1)}}. \tag{23}$$

Proof. See Section 7.3. □

Using (21), (22) and Lemma 3 in (20) yields

$$\mathbb{E}\Big[\sum_{i=1}^{n}\Big(Q_i(t+1)\Big)^2\Big]-\mathbb{E}\Big[\sum_{i=1}^{n}\Big(Q_i(t)\Big)^2\Big]\le$$
$$\lambda^{(2)}+\sum_{i=1}^{n}\mu_i^{(2)}+2mn\lambda^{(1)}C-2\sum_{i=1}^{n}\frac{\epsilon\mu_i^{(1)}}{\sum_{i=1}^n\mu_i^{(1)}}\,\mathbb{E}\Big[Q_i(t)\Big]. \tag{24}$$

For ease of exposition, denote the constants

$$D=\lambda^{(2)}+\sum_{i=1}^{n}\mu_i^{(2)}+2mn\lambda^{(1)}C, \tag{25}$$

and

$$\delta=\min_i\frac{\epsilon\mu_i^{(1)}}{\sum_{i=1}^n\mu_i^{(1)}}. \tag{26}$$

Rearranging (24) and using (25) and (26) yields

$$2\delta\sum_{i=1}^{n}\mathbb{E}\Big[Q_i(t)\Big]\le$$
$$D+\left(\mathbb{E}\Big[\sum_{i=1}^{n}\Big(Q_i(t)\Big)^2\Big]-\mathbb{E}\Big[\sum_{i=1}^{n}\Big(Q_i(t+1)\Big)^2\Big]\right). \tag{27}$$

Summing (27) over time slots $[0,1,\dots,T-1]$, noticing the telescopic series at the right hand side of the inequality and dividing by $2\delta T$ yields

$$\frac{1}{T}\sum_{t=0}^{T-1}\sum_{i=1}^{n}\mathbb{E}\Big[Q_i(t)\Big]\le$$
$$\frac{D}{2\delta}+\frac{1}{2\delta T}\left(\mathbb{E}\Big[\sum_{i=1}^{n}\Big(Q_i(0)\Big)^2\Big]-\mathbb{E}\Big[\sum_{i=1}^{n}\Big(Q_i(T)\Big)^2\Big]\right). \tag{28}$$

Taking limits of (28) and making the standard assumption that the system starts its operation with finite queue lengths, *i.e.,*

$$\mathbb{E}\Big[\sum_{i=1}^{n}\Big(Q_i(0)\Big)^2\Big]<\infty$$

yields

$$\limsup_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}\sum_{i=1}^{n}\mathbb{E}\Big[Q_i(t)\Big]\le\frac{D}{2\delta}. \tag{29}$$

This implies strong stability and concludes the proof. □

3.3 Sufficient stability condition

As mentioned, in order to establish that a policy is *Loosely-Shortest-Queue*, Assumption 1 has to hold. That is

$$\mathbb{E}\Big[\Big|Q_i(t)-\tilde{Q}_i^j(t)\Big|\Big]\le C\quad\forall\,(i,j,t)\in N\times M\times\mathbb{N}.$$

Generally, it may be challenging to establish that this condition holds. To that end, we now develop a simplified sufficient condition. As we later demonstrate, this simplified condition captures a broad family of communication techniques among the servers and the dispatchers and allows for the design of stable policies with appealing performance and extremely low communication budgets.

Theorem 2. *Let* $\mathbb{E}\left[\left|Q_i(0) - \tilde{Q}_i^j(0)\right|\right] \le C_0\ \forall\,(i,j) \in N \times M$, *and let* $\mathbb{1}_i^j(t)$ *be an indicator function that obtains the value* 1 *iff server i updates dispatcher j (via the push-based sampling or the pull-based update message from the server) with its actual queue length at the end of time slot t (after arrivals and departures at time slot t). Assume that there exists* $\bar{\epsilon} > 0$ *such that*

$$\mathbb{E}\left[\mathbb{1}_i^j(t) \;\middle|\; \left|Q_i(t) - \tilde{Q}_i^j(t)\right|\right] > \bar{\epsilon} \quad \forall (i,j,t) \in N \times M \times \mathbb{N}. \tag{30}$$

Then, Assumption 1 holds.

Namely, we assume that there is a strictly positive probability of an update at the end of time slot t, given any gap between the actual queue lengths at the beginning of that time slot and the dispatcher local views.

Proof. Fix server i and dispatcher j. Denote

$$Z(t) = \left|Q_i(t) - \tilde{Q}_i^j(t)\right|.$$

Now, for all t it holds that

$$\begin{aligned}&Z(t+1) \le \left(1 - \mathbb{1}_i^j(t)\right) \cdot \left(Z(t) + a_i(t) + s_i(t)\right) \le \\ &\left(1 - \mathbb{1}_i^j(t)\right) \cdot Z(t) + a(t) + s_i(t).\end{aligned} \tag{31}$$

Taking expectation of (31) yields

$$\mathbb{E}[Z(t+1)] \le \mathbb{E}[(1 - \mathbb{1}_i^j(t)) \cdot Z(t)] + \lambda^{(1)} + \mu_i^{(1)}. \tag{32}$$

Next, using the law of total expectation

$$\begin{aligned}&\mathbb{E}[(1 - \mathbb{1}_i^j(t)) \cdot Z(t)] = \mathbb{E}\left[\,\mathbb{E}\left[(1 - \mathbb{1}_i^j(t)) \cdot Z(t) \,\middle|\, Z(t)\right]\right] = \\ &\mathbb{E}\left[Z(t) \cdot \mathbb{E}\left[(1 - \mathbb{1}_i^j(t)) \,\middle|\, Z(t)\right]\right] \le (1 - \bar{\epsilon})\,\mathbb{E}[Z(t)],\end{aligned} \tag{33}$$

where the last inequality follows from the linearity of expectation and (30). Now, using (33) in (32) yields

$$\begin{aligned}&\mathbb{E}[Z(t+1)] \le (1 - \bar{\epsilon})\,\mathbb{E}[Z(t)] + \lambda^{(1)} + \mu_i^{(1)} \le \\ &(1 - \bar{\epsilon})\,\mathbb{E}[Z(t)] + \lambda^{(1)} + \max_i \left\{\mu_i^{(1)}\right\}.\end{aligned} \tag{34}$$

With this result at hand, we can finish our proof by an inductive claim. Fix

$$C^* = \max\left\{\frac{\lambda^{(1)} + \max_i \left\{\mu_i^{(1)}\right\}}{\bar{\epsilon}}, C_0\right\}.$$

We now show that for all t it holds that $\mathbb{E}[Z(t)] \le C^*$.
Basis. For $t = 0$ the claim trivially holds since $\mathbb{E}[Z(0)] \le C_0 \le C^*$.
Induction hypothesis. Assume that $\mathbb{E}[Z(t)] \le C^*$.
Inductive step. Using the induction hypothesis in (34) yields

$$\begin{aligned}&\mathbb{E}[Z(t+1)] \le (1 - \bar{\epsilon})\,\mathbb{E}[Z(t)] + \lambda^{(1)} + \max_i \left\{\mu_i^{(1)}\right\} \le \\ &(1 - \bar{\epsilon})C^* + \lambda^{(1)} + \max_i \left\{\mu_i^{(1)}\right\} \le C^*,\end{aligned} \tag{35}$$

where we used the fact that by the definition of C^* it holds that $\lambda^{(1)} + \max_i \left\{\mu_i^{(1)}\right\} \le \bar{\epsilon}C^*$. Note that the obtained bound is not dependent on (i,j,t). This concludes the proof. □

4 EXAMPLE *Loosely-Shortest-Queue* POLICIES

Since *Loosely-Shortest-Queue* is not restricted to work with either pull- or push-based communications, in this section we provide examples for both. In a push-based policy, the dispatchers sample the servers for their queue length whereas in a pull-based policy the servers may update the dispatchers with their queue length. While empirically, we will see that the pull-based approach can provide better performance in many scenarios, it may also incur additional implementation overhead because it requires the servers to actively update the dispatchers given some state conditions, rather than passively answer sample queries. Therefore we are inclined to consider both the push and pull frameworks.

In both frameworks we consider policies with low communication overhead that can be unstable even with a single dispatcher, and provide an alternative *Loosely-Shortest-Queue* policy with similar communication budget that is strongly stable for any number of dispatchers.

4.1 Push based *Loosely-Shortest-Queue* example

The *JSQ*(d) policy forms a popular low-communication push-based load balancing approach. As mentioned, *JSQ*(d) is not stable in heterogeneous systems even for a single dispatcher.

Instead, we will now extend a push-based *Loosely-Shortest-Queue* policy that uses exactly the same communication pattern between the servers and the dispatchers. Specifically, each dispatcher holds a local array of the server queue length approximations and sends jobs to the minimum one among them. The approximations are updated as follows: (1) when a dispatcher sends jobs to a server, these jobs are added to the respective local approximation; (2) at each time slot, if new jobs arrive, the dispatcher randomly samples d distinct queues and uses this information only to update the respective d distinct entries in its local array to their actual value. Algorithm 1 (termed *Loosely-Shortest-Queue-Sample*(d)) depicts the actions taken by each dispatcher at each time slot.

Remark 1. *Note that Loosely-Shortest-Queue-Sample*(d) *is not a new policy but is considered for a single dispatcher and homogenous servers in [14].*

The simplicity of *Loosely-Shortest-Queue-Sample*(d) may be surprising. For instance, there is no attempt to guess or estimate how the other dispatchers send traffic or how the queue drains to get a better estimate, *i.e.,* our estimate is based only on the jobs that the specific dispatcher sends and the last time it sampled a queue. We also do not take the age of the information into account.

Furthermore, as we find below, the stability proof of *Loosely-Shortest-Queue-Sample*(d) only relies on the sample messages and not on the job increments. We empirically find that these increments help improve the estimation quality and therefore the performance.

We now prove that using *Loosely-Shortest-Queue-Sample*(d) at each dispatcher results in strong stability in multi-dispatcher heterogeneous systems. Remarkably, this result holds even for $d = 1$.

Proposition 1. *Assume that the system is sub-critical and each dispatcher uses Loosely-Shortest-Queue-Sample*(d)*. Then, the system is strongly stable.*

Proof. Fix dispatcher j and server i. Consider time slot t. By (4), with probability of at least ϵ_0, dispatcher j samples d out of n

Algorithm 1: *Loosely-Shortest-Queue-Sample*(d) (push-based *Loosely-Shortest-Queue* example)

```
Code for dispatcher $j \in M$;
Route jobs:
    foreach time slot t do
        Forward jobs to server $i^* \in argmin_i \left\{\tilde{Q}_i^j(t)\right\}$;
        Update $\tilde{Q}_{i^*}^j(t) \leftarrow \tilde{Q}_{i^*}^j(t) + a^j(t)$;
    end
end
Update local state:
    foreach time slot t do
        if new jobs arrive at time slot t then
            Uniformly at random pick distinct $i_1, \ldots, i_d \in N$;
            For each $i \in \{i_1, \ldots, i_d\}$ update $\tilde{Q}_i^j(t) \leftarrow Q_i(t)$;
        end
    end
```

servers uniformly at random disregarding the system state at time slot t. Therefore, we obtain

$$\mathbb{E}\left[\mathbb{1}_i^j(t) \;\middle|\; \left|Q_i(t) - \tilde{Q}_i^j(t)\right|\right] \geq \frac{\epsilon_0 \cdot d}{n}.$$

This respects the simplified sufficiency condition and thus concludes the proof. □

4.2 Pull based *Loosely-Shortest-Queue* example

JIQ is a popular, recently proposed, low-communication pull-based load balancing policy. It offers a low communication overhead that is upper-bounded by a single message per job [15]. However, as mentioned, for heterogeneous systems, *JIQ* is not stable even for a single dispatcher.

We now propose a different pull-based *Loosely-Shortest-Queue* policy that conforms with the same communication upper bound, namely a single message per job, and leverages the important idleness signals from the servers. Specifically, each server, upon the completion of one or several jobs at the end of a time slot, sends its queue length to a dispatcher, which is chosen uniformly at random, using the following rule: (1) if the server becomes idle, then the message is sent with probability 1; (2) otherwise, the message is sent with probability $0 < p \leq 1$ where p is a fixed parameter. Algorithm 2 (termed *Loosely-Shortest-Queue-Update*(p)) depicts the actions taken by each dispatcher at each time slot.

The intuition behind this approach is to always leverage the idleness signals in order to avoid immediate starvation as done by *JIQ*; yet, in contrast to *JIQ*, even when no servers are idle, we want to make sure that the local views are not too far from the approximations, which provides significant advantage at high loads.

Remark 2. *Note that Loosely-Shortest-Queue-Update*(p) *is not a new policy but, to the best of our knowledge, is similar to the policy considered in [26] for a single dispatcher and homogeneous servers.*

We now formally prove that using *Loosely-Shortest-Queue-Update*(p) results in strong stability in multi-dispatcher heterogeneous systems. Remarkably, this result holds for any $p > 0$.

Algorithm 2: *Loosely-Shortest-Queue-Update*(p) (pull-based *Loosely-Shortest-Queue* example)

```
Code for dispatcher $j \in M$;
Route jobs:
    foreach time slot t do
        Forward jobs to server $i^* \in argmin_i \left\{\tilde{Q}_i^j(t)\right\}$;
        Update $\tilde{Q}_{i^*}^j(t) \leftarrow \tilde{Q}_{i^*}^j(t) + a^j(t)$;
    end
end
Update local state:
    foreach arrived message $\langle i, q \rangle$ at time slot t do
        Update $\tilde{Q}_i^j(t) \leftarrow q$;
    end
end
Code for server $i \in N$;
Send update message:
    foreach time slot t do
        if completed jobs at time slot t then
            Uniformly at random pick $j \in M$;
            if idle then
                Send $\langle i, Q_i(t) \rangle$ to dispatcher $j$;
            else
                Send $\langle i, Q_i(t) \rangle$ to dispatcher $j$ $w.p.$ $p$;
            end
        end
    end
end
```

Proposition 2. *Assume that the system is sub-critical and each dispatcher uses Loosely-Shortest-Queue-Update*(p)*. Then, the system is strongly stable.*

Proof. Fix dispatcher j, server i and time slot t. Our goal is to prove that (30) holds. To do so, we examine two possible events at the beginning of time slot t: (1) $Q_i(t) = 0$ and (2) $Q_i(t) > 0$.

(1) Since $Q_i(t) = 0$, the server updated at least one dispatcher in a previous time slot, *i.e.,* for at least one dispatcher j^* we have that $\tilde{Q}_i^{j^*}(t) = 0$. This must hold since there is a dispatcher that received the update message after this queue got empty (that is, when a server becomes idle, a message is sent $w.p.$ 1). Now consider the event $A_1 = \left\{a_i^{j^*}(t) > 0 \cap s_i(t) > 0\right\}$. Since the tie breaking rule is random, by (1), (4), (2), (5) and (6), there exists $\bar{\epsilon}_i > 0$ such that $\mathbb{P}(A_1) > \bar{\epsilon}_i$. Since $a(t)$ and $s_i(t)$ are not dependent on any system information at the beginning of time slot t we obtain

$$\mathbb{E}\left[\mathbb{1}_i^j(t) \;\middle|\; \left|Q_i(t) - \tilde{Q}_i^j(t)\right|, Q_i(t) = 0\right] \geq p \cdot \mathbb{P}(A_1) > p \cdot \bar{\epsilon}_i. \quad (36)$$

(2) Since $Q_i(t) > 0$, there is a strictly positive probability that a job would be completed at this time slot. That is, since $s_i(t)$ is not dependent on any system information at the beginning of time slot t we obtain

$$\mathbb{E}\left[\mathbb{1}_i^j(t) \;\middle|\; \left|Q_i(t) - \tilde{Q}_i^j(t)\right|, Q_i(t) > 0\right] \geq p \cdot P(s_i(t) > 0) > p \cdot \bar{\epsilon}_i. \quad (37)$$

Finally, since $\mathbb{E}\left[\mathbb{1}_i^j(t) \;\middle|\; \left|Q_i(t) - \tilde{Q}_i^j(t)\right|\right]$ is a convex combination of the left hand sides of (36) and (37) we obtain that

$$\mathbb{E}\left[\mathbb{1}_i^j(t) \;\middle|\; \left|Q_i(t) - \tilde{Q}_i^j(t)\right|\right] > p \cdot \bar{\epsilon}_i.$$

Now fix $\bar{\epsilon} = \min_i \{\bar{\epsilon}_i\}$. We have that

$$\mathbb{E}\left[\mathbb{1}_i^j(t) \;\middle|\; \left|Q_i(t) - \tilde{Q}_i^j(t)\right|\right] > p \cdot \bar{\epsilon} \quad \forall (i,j,t) \in N \times M \times \mathbb{N}.$$

This concludes the proof. □

Table 1 summarizes the stability properties and the *worst case* communication requirements of the evaluated load balancing techniques as dictated by theory and verified by our evaluations.

4.3 Practical considerations

Before moving to simulations, we discuss several practical considerations when considering different load balancing approaches.

Instantaneous routing. An appealing property of the *Loosely-Shortest-Queue* policy, similarly to *JIQ*, is that a dispatcher can immediately take routing decisions upon a job arrival. This is in contrast to common push-based policies that have to wait for a response from the sampled servers to be able to make a decision. For example, when using the *JSQ*(2) policy, when a job arrives the dispatcher cannot immediately send the job to a server but must pay the additional delay of sampling two servers. Note that Assumption 1 trivially applies for any additional constant time delay in update messages among the servers and the dispatchers.

Space requirements. To implement the *Loosely-Shortest-Queue* policy, similarly to *JSQ*, each dispatcher has to hold an array of size n with all server queue length estimations. It is important to note that such a space requirement incurs negligible overhead on a modern server. For example, nowadays, any commodity server has tens to hundreds GB of DRAM. But even a hypothetical cluster with 10^6 servers requires only a few MB of the dispatcher's memory, which is negligible in comparison to the DRAM size.

Computational complexity. To implement the *Loosely-Shortest-Queue* policy, similarly to *JSQ*, each dispatcher has to find the minimum (approximated) queue length and route the incoming jobs to it. At first glance, it might seem that this requires $O(n)$ operations for each decision making, which is a disadvantage in comparison to *JIQ* and *JSQ*(2) that require a constant number of operations per decision, irrespective of the size of the system. However, by using a priority queue (*e.g.,* min-heap), finding the minimum results in only a single operation (i.e., simply looking at the head of the priority queue). For a queue length update operation, $O(\log n)$ operations are required in the worst case (*e.g.,* decrease-key operation in a min-heap). Even with $n = 10^6$, just a few operations are required in the worst case per queue length update. This results in a single commodity core being able to perform tens to hundreds of millions of such updates per second, hence resulting in negligible overhead, especially for a low-communication policy in which queue length updates are not too frequent.

5 EVALUATION

In this section we conduct an extensive evaluation of the *Loosely-Shortest-Queue* approach, using both the *Loosely-Shortest-Queue-Sample*(d) (with $d = 1$ and $d = 2$) and *Loosely-Shortest-Queue-Update*(p) (with $p = 1$ and $p = 0.01$) families of *Loosely-Shortest-Queue* algorithms. We compare them to the baseline full-information *JSQ* (for *JSQ* only, dispatchers take decisions sequentially to allow for a "water-filling" behaviour and avoid incast) and to the low-communication *JSQ*(2) and *JIQ*. We consider heterogeneous systems with low and high skew, in small single-dispatcher and larger-scale multi-dispatcher scenarios. For completeness, we also present the results for homogeneous systems in Appendix A.

For each scenario and each policy, we measure the time-averaged number of jobs in the system. This measure also translates to the mean delay by Little's Law, and reveals the stability region of each policy. Additionally, even though we have theoretical guarantees for the *worst-case* communication overhead incurred by each policy, we also measure the total average communication overhead per time slot.

5.1 Low heterogeneity

We start by considering a mix of weak and strong servers with a ratio of 2 between their service rates, thus exhibiting a moderate degree of heterogeneity.

In this subsection the job arrival process at a dispatcher is a Poisson process with parameter λ and the server service processes are geometrically distributed with a parameter $2p$ for a weak server and a parameter p for a strong server. In a simulation with n_s strong servers, n_w weak servers and m dispatchers we set $p = \frac{n_s + 0.5 n_w}{100m}$ and sweep $0 \leq \lambda < 100$.

5.1.1 Small scale scenario. In this evaluation, we consider a small scale scenario with a single dispatcher and 10 heterogeneous servers. The results are depicted in Figure 2. As expected, *Loosely-Shortest-Queue-Update*(1) performs identically to *JSQ* since in a single dispatcher scenario both are aware of all queue lengths at all times. It is evident that, in all three scenarios, *JIQ* is not stable whereas our *LSQ-Update*(0.01) is stable with a similar communication overhead. Additionally, in all three scenarios, *Loosely-Shortest-Queue-Sample*(2) performs better than *JSQ*(2), which appears to be stable in this scenario (we tested up to a normalized load of 0.995).

5.1.2 Larger scale scenario. In this evaluation, we consider a larger scale scenario with 10 dispatchers and 100 heterogeneous servers. The results are depicted in Figure 3 and show similar trends. *Loosely-Shortest-Queue-Update*(1) performs slightly worse than *JSQ* but with a significantly lower communication budget (by an order of magnitude). Similarly to the previous scenario, it is evident that, in all three scenarios, *JIQ* is not stable again, and its stability region decreases as the proportion of weak servers decreases. In contrast, our *LSQ-Update*(0.01) is always stable, with a similar communication overhead. Again, in all three scenarios, *Loosely-Shortest-Queue-Sample*(2) performs better then *JSQ*(2), which appears to be stable in this scenario as well.

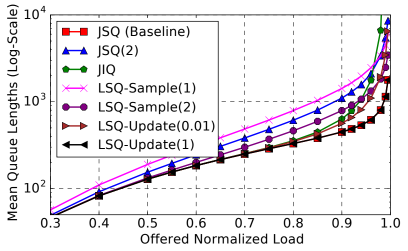
5.2 High heterogeneity

In this subsection, we consider systems with highly skewed heterogeneous servers. That is, we test the different approaches with a mix of weak and strong servers, and assume a ratio of 10 between their service rates.

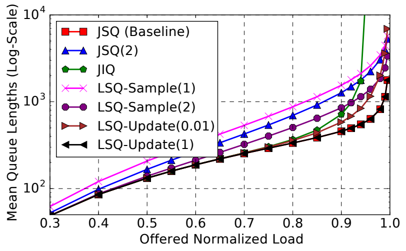
Again, in this subsection the job arrival process at a dispatcher is a Poisson process with parameter λ but the server service processes are geometrically distributed with parameter $10p$ for a weak server and a parameter p for a strong server. In a simulation with n_s strong

	Throughput optimality		Comm. overhead	
	Homo-geneous	Hetero-geneous	Per time slot	Per job arrival
JSQ	✓	✓	$m \cdot n$	m
JSQ(d)	✓	×	$d \cdot m$	d
JIQ	✓	×	n	1
Loosely-Shortest-Queue-Sample(d)	✓	✓	$d \cdot m$	d
Loosely-Shortest-Queue-Update(p)	✓	✓	n	1

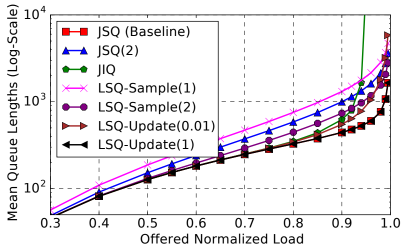
Table 1: Comparing stability and *worst case* communication overhead of the evaluated load balancing techniques.



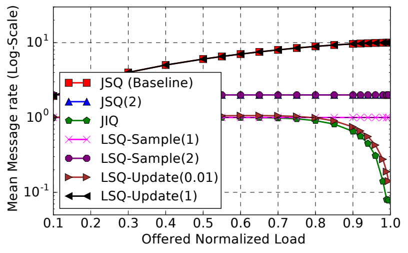
(a) 1 strong server, 9 weak servers.



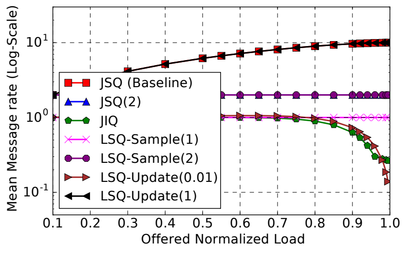
(b) 5 strong servers, 5 weak servers.



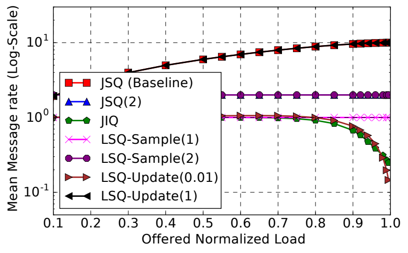
(c) 9 strong servers, 1 weak server.



(d) 1 strong server, 9 weak servers.

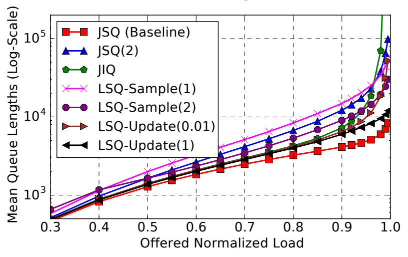


(e) 5 strong servers, 5 weak servers.

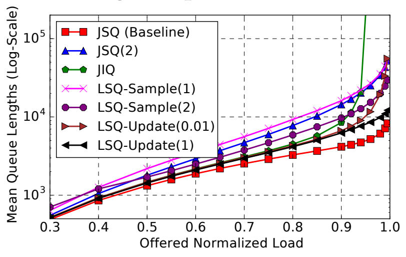


(f) 9 strong servers, 1 weak server.

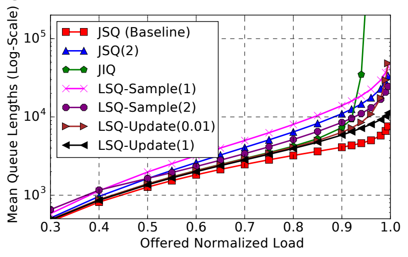
Figure 2: Scenario with a single dispatcher and 10 heterogeneous servers.



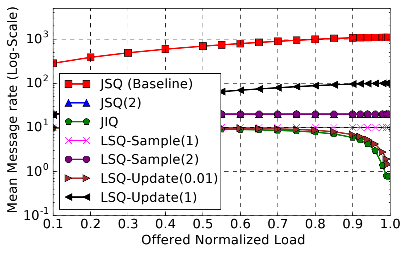
(a) 10 strong servers, 90 weak servers.



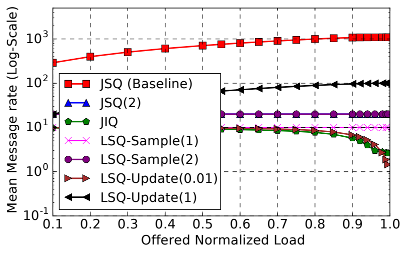
(b) 50 strong servers, 50 weak servers.



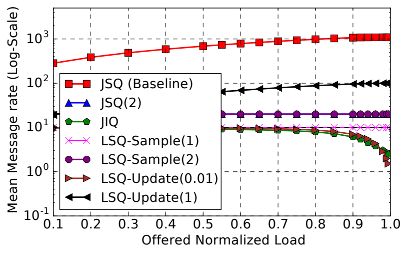
(c) 90 strong servers, 10 weak servers.



(d) 10 strong servers, 90 weak servers.

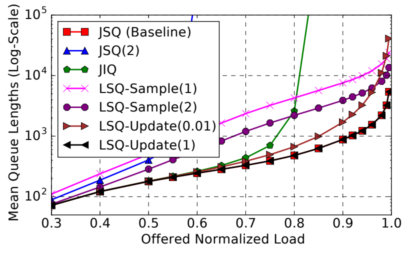


(e) 50 strong servers, 50 weak servers.

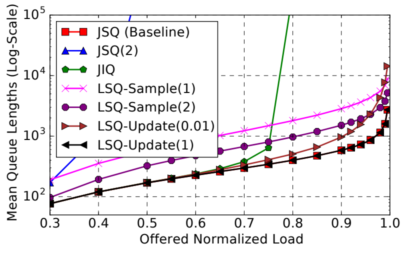


(f) 90 strong servers, 10 weak servers.

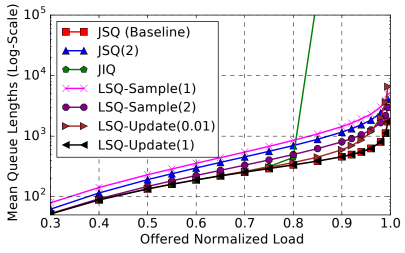
Figure 3: Scenario with 10 dispatchers and 100 heterogeneous servers.



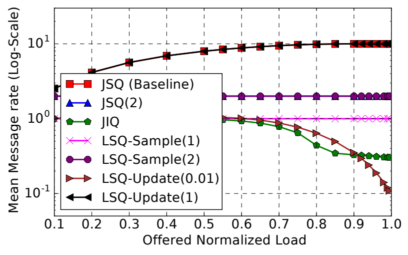
(a) 1 strong server, 9 weak servers.



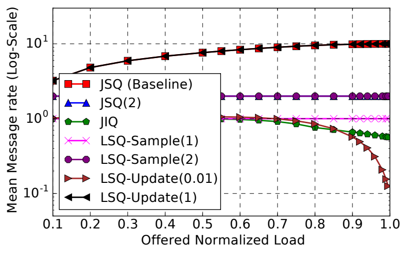
(b) 5 strong servers, 5 weak servers.



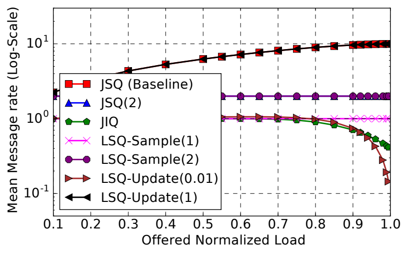
(c) 9 strong servers, 1 weak server.



(d) 1 strong server, 9 weak servers.

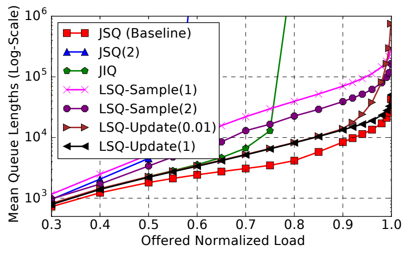


(e) 5 strong servers, 5 weak servers.

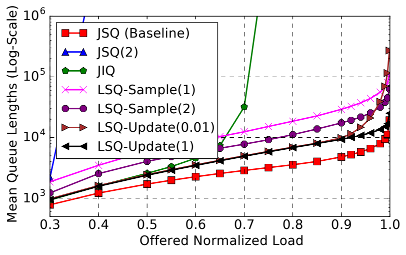


(f) 9 strong servers, 1 weak server.

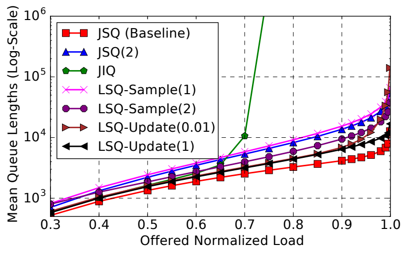
Figure 4: Scenario with a single dispatcher and 10 highly skewed heterogeneous servers.



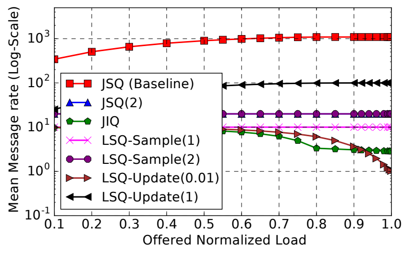
(a) 10 strong servers, 90 weak servers.



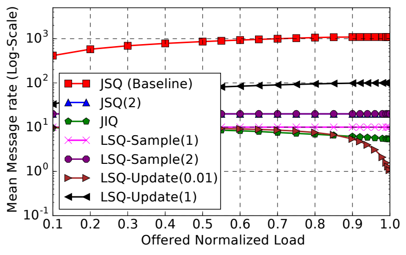
(b) 50 strong servers, 50 weak servers.



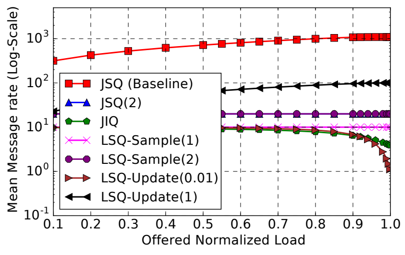
(c) 90 strong servers, 10 weak servers.



(d) 10 strong servers, 90 weak servers.



(e) 50 strong servers, 50 weak servers.



(f) 90 strong servers, 10 weak servers.

Figure 5: Scenario with 10 dispatchers and 100 highly skewed heterogeneous servers.

servers, n_w weak servers and m dispatchers we set $p = \frac{n_s+0.1n_w}{100m}$ and sweep $0 \le \lambda < 100$.

5.2.1 Small scale scenario. In this evaluation, we consider a small scale scenario with a single dispatcher and 10 highly skewed heterogeneous servers. The results are depicted in Figure 4.

As expected, *Loosely-Shortest-Queue-Update*(1) performs identically to *JSQ*. Again, in all three scenarios, *JIQ* is not stable, and its stability region has decreased dramatically for this larger skew in server service rates. For example, for a mix of 5 strong and 5 weak servers, its stability region is only up to ≈78%. In contrast, our *LSQ-Update*(0.01) is always stable with a similar communication overhead. *JSQ*(2) is not stable as well with a dramatic degradation, especially when the number of strong and weak servers is similar. For example, in a scenario with 5 strong and 5 weak servers, the stability region of *JSQ*(2) is only up to ≈44%. On the other hand, *Loosely-Shortest-Queue-Sample*(2) is always stable with an identical communication overhead. Note that, as dictated by theory, even *Loosely-Shortest-Queue-Sample*(1) is always stable with even less communication overhead.

5.2.2 Larger scale scenario. In this evaluation, we consider a larger scale scenario with 10 dispatchers and 100 highly skewed heterogeneous servers. The results are depicted in Figure 5. The results show a similar trend to the results in the small-scale simulation. It is evident that the stability regions of both *JSQ*(2) and *JIQ* suffer from a significant degradation that appears to be consistent with an increasing number of dispatchers. On the other hand, all *LSQ* approaches are stable and keep performing well using similar communication overhead. It is evident that *LSQ-Update*(1) is only slightly worse than *JSQ* but, again, with a significantly lower communication overhead (approximately by an order of magnitude). Again *LSQ-Update*(0.01) and *LSQ-Sample*(2) use similar communication budgets as *JIQ* and *JSQ*(2), respectively, but are stable with good performance.

5.3 Evaluation takeaways

The *Loosely-Shortest-Queue* approach always guarantees stability and it achieves this using the same communication budget as other non-throughput-optimal low-communication techniques. Additionally, the simulations indicate that, under identical low-communication requirements, *Loosely-Shortest-Queue* consistently exhibits good performance in different scenarios whereas other low-communication techniques are either unstable or offer poor performance.

Remark 3. *Interestingly, by allowing different dispatchers to have a different view of the system, Loosely-Shortest-Queue indirectly appears to solve the incast problem JSQ may incur in a parallel multi-dispatcher system.*

Also, evaluation results present an interesting trade-off between the push-based *LSQ-Sample*(d) and the pull-based *LSQ-Update*(p) approaches for small d and p values (*i.e.,* extremely low communication overhead). *LSQ-Update*(p) appears to be consistently better at low loads (where servers keep getting idle frequently), whereas *LSQ-Sample*(d) appears to be consistently better at high loads where idleness becomes rare and the approximations of *LSQ-Update*(p) are less effective than in the push approach.

6 ARBITRARILY LOW COMMUNICATION

We have shown, both formally and in simulations, that the *Loosely-Shortest-Queue* approach offers strong theoretical guarantees and appealing performance with low communication overhead. Interestingly, by virtue of Theorem 2, we can construct various strongly stable *Loosely-Shortest-Queue* policies with any arbitrarily low communication budget, disregarding whether the system uses pull or push messages (or both). This, in fact, appears to generalize a result from [26] to multiple dispatchers and heterogeneous servers.

Let $M(t)$ be the number of queue length updates performed by all dispatchers up to time t. Fix any arbitrary small $r > 0$. Suppose that we want to achieve strong stability, such that the average message rate is at most r, *i.e.,* for all t we have that $\mathbb{E}[M(t)] \leq rt$. Then, the two following per-time-slot dispatcher sampling rules trivially achieve strong stability (by Theorem 2) and respect the desired communication bound, *i.e.,* $\mathbb{E}[M(t)] \leq rt$.

Example 1 (Push-based communication example.). *Dispatcher sampling rule upon job(s) arrival: (1) pick a server $i \in N$ uniformly at random; (2) sample server i with probability $\frac{r}{m}$.*

Example 2 (Pull-based communication example.). *Server messaging rule upon job(s) completion: (1) pick a dispatcher $j \in M$ uniformly at random; (2) update dispatcher j with probability $\frac{r}{n}$.*

Theorem 2 also enables to design strongly stable *Loosely-Shortest-Queue* policies with hybrid communication (*e.g.,* push and pull) that can attempt to maximize the benefits of both approaches (for example, as we have seen, for extremely low communication overhead requirements, at low loads *LSQ-Update*(p) appears to be more effective whereas at high loads *LSQ-Sample*(d) appears to be more effective). For example, the following policy leverages both the advantages of *JIQ* (*i.e.,* being immediately notified that a server becomes idle) and *JSQ*(d) (*i.e.,* random exploration of shallow queues when no servers are idle).

Example 3 (Hybrid communication example.). *Dispatcher sampling rule: (1) pick a server $i \in N$ uniformly at random; (2) sample server i with probability $\frac{r}{m}$.*
Server messaging rule: (1) if got idle, pick a dispatcher $j \in M$ uniformly at random and send it an update message.

The above three examples demonstrate the wide range of possibilities that the *Loosely-Shortest-Queue* approach offers to the design of stable, scalable policies with low communication overhead.

7 PROOFS OF LEMMAS

This section provides the proofs of the various lemmas that we employed towards establishing our main theoretical result, *i.e.,* Theorem 1.

7.1 Proof of Lemma 1

First, by definition

$$\sum_{i=1}^{n} a_i^{JSQ}(t) = \sum_{i=1}^{n} a_i^{WR}(t) = a(t).$$

Therefore, both $\left\{a_i^{JSQ}(t)\right\}_{i=1}^{n}$ and $\left\{a_i^{WR}(t)\right\}_{i=1}^{n}$ are feasible solutions to the optimization problem given by

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \sum_{i=1}^{n} x_i(t)Q_i(t) \\ \text{subject to} \quad & \sum_{i=1}^{n} x_i(t) = a(t), \quad x_i(t) \geq 0 \ \forall i \in N \end{aligned} \tag{38}$$

The optimal solution to this problem is simply

$$a(t) \min_i \{Q_i(t)\},$$

which is exactly the way *JSQ* policy operates. That is,

$$\sum_{i=1}^{n} a_i^{JSQ}(t)Q_i(t) = a(t) \min_i \{Q_i(t)\}.$$

Clearly, any other feasible solution, *e.g.,* $\left\{a_i^{WR}(t)\right\}_{i=1}^{n}$, cannot be better. This concludes the proof. □

7.2 Proof of Lemma 2

Expanding the term $\sum_{i=1}^{n} Q_i(t)\Big(a_i(t) - a_i^{JSQ}(t)\Big)$ yields

$$\sum_{i=1}^{n} Q_i(t)\Big(a_i(t) - a_i^{JSQ}(t)\Big) = \sum_{i=1}^{n}\sum_{j=1}^{m} Q_i(t)\Big(a_i^j(t) - a_i^{j,JSQ}(t)\Big). \quad (39)$$

We now substitute $Q_i(t)$ by $Q_i(t) - \tilde{Q}_i^j(t) + \tilde{Q}_i^j(t)$. This yields

$$\sum_{i=1}^{n} Q_i(t)\Big(a_i(t) - a_i^{JSQ}(t)\Big) = \sum_{i=1}^{n}\sum_{j=1}^{m} \Big(Q_i(t) - \tilde{Q}_i^j(t) + \tilde{Q}_i^j(t)\Big)\Big(a_i^j(t) - a_i^{j,JSQ}(t)\Big). \quad (40)$$

We now introduce the following lemma:

Lemma 4. *For all t it holds that*

$$\sum_{i=1}^{n}\sum_{j=1}^{m} \tilde{Q}_i^j(t)\Big(a_i^j(t) - a_i^{j,JSQ}(t)\Big) \le 0.$$

Proof. See Section 7.4. □

Using Lemma 4 in (40) yields

$$\sum_{i=1}^{n} Q_i(t)\Big(a_i(t) - a_i^{JSQ}(t)\Big) \le \sum_{i=1}^{n}\sum_{j=1}^{m} \Big(Q_i(t) - \tilde{Q}_i^j(t)\Big)\Big(a_i^j(t) - a_i^{j,JSQ}(t)\Big). \quad (41)$$

Now, using the fact that $xy \le |x||y|$ for all $(x, y) \in \mathbb{R}^2$ on (41) yields

$$\sum_{i=1}^{n} Q_i(t)\Big(a_i(t) - a_i^{JSQ}(t)\Big) \le \sum_{i=1}^{n}\sum_{j=1}^{m} \Big|Q_i(t) - \tilde{Q}_i^j(t)\Big|\Big|a_i^j(t) - a_i^{j,JSQ}(t)\Big|. \quad (42)$$

Finally, it trivially holds that

$$a(t) \ge \Big|a_i^j(t) - a_i^{j,JSQ}(t)\Big|. \quad (43)$$

Using (43) in (42) concludes the proof. □

7.3 Proof of Lemma 3

Each dispatcher applies the WR policy independently. Therefore, by applying (5), (6), (1) and (2) we have that the expected number of jobs arriving at each server i is

$$\mathbb{E}\Big[a_i^{WR}(t)\Big] = \mathbb{E}\Big[\sum_{j=1}^{m} a_i^{j,WR}(t)\Big] = \frac{\mu_i^{(1)}}{\sum_{i=1}^{n}\mu_i^{(1)}}\mathbb{E}\Big[\sum_{j=1}^{m} a^j(t)\Big] = \frac{\lambda^{(1)}\mu_i^{(1)}}{\sum_{i=1}^{n}\mu_i^{(1)}}. \quad (44)$$

Using (5), (6), (9) and (44) yields

$$\mathbb{E}\Big[s_i(t) - a_i^{WR}(t)\Big] = \mu_i^{(1)} - \frac{\lambda^{(1)}\mu_i^{(1)}}{\sum_{i=1}^{n}\mu_i^{(1)}} = \mu_i^{(1)}\frac{\sum_{i=1}^{n}\mu_i^{(1)} - \lambda^{(1)}}{\sum_{i=1}^{n}\mu_i^{(1)}} = \frac{\epsilon\mu_i^{(1)}}{\sum_{i=1}^{n}\mu_i^{(1)}}. \quad (45)$$

This concludes the proof. □

7.4 Proof of Lemma 4

Fix $j = j^*$. It is sufficient to show that

$$\sum_{i=1}^{n} \tilde{Q}_i^{j^*}(t)\Big(a_i^j(t) - a_i^{j^*,JSQ}(t)\Big) \le 0. \quad (46)$$

The proof now follows similar lines to the proof of Lemma 1. By definition

$$\sum_{i=1}^{n} a_i^{j^*}(t) = \sum_{i=1}^{n} a_i^{j^*,JSQ}(t) = a^{j^*}(t).$$

Therefore, both $\Big\{a_i^{j^*}(t)\Big\}_{i=1}^{n}$ and $\Big\{a_i^{j^*,JSQ}(t)\Big\}_{i=1}^{n}$ are feasible solutions to the optimization problem given by

$$\begin{aligned} &\underset{x}{\text{minimize}} && \sum_{i=1}^{n} x_i(t)\tilde{Q}_i^{j^*}(t) \\ &\text{subject to} && \sum_{i=1}^{n} x_i(t) = a^{j^*}(t), \quad x_i(t) \ge 0\ \forall i \in N \end{aligned} \quad (47)$$

The optimal solution to this problem is simply $a^{j^*}(t)\min_i\Big\{\tilde{Q}_i^{j^*}(t)\Big\}$, which is exactly the way our policy operates since it performs *JSQ* considering $\Big\{\tilde{Q}_i^{j^*}(t)\Big\}$ instead of $\{Q_i(t)\}$. That is

$$\sum_{i=1}^{n} a_i^{j^*}(t)Q_i(t) = a^{j^*}(t)\min_i\Big\{\tilde{Q}_i^{j^*}(t)\Big\}.$$

Any other feasible solution including $\Big\{a_i^{JSQ}(t)\Big\}_{i=1}^{n}$ cannot be better when considering $\Big\{\tilde{Q}_i^{j^*}(t)\Big\}$ instead of $\{Q_i(t)\}$. This proves the inequality in (46) and thus concludes the proof. □

8 CONCLUSIONS

In this paper, we introduced the *Loosely-Shortest-Queue* family of load balancing algorithms. We formally established that any *Loosely-Shortest-Queue* policy is strongly stable and further developed a simplified sufficient condition for establishing that a policy is *Loosely-Shortest-Queue*. We then demonstrated that the *Loosely-Shortest-Queue* approach allows to construct stable policies with arbitrary low communication budgets for system with multiple dispatchers and heterogeneous servers.

Using extensive simulations that consider homogeneous, heterogeneous and highly skewed heterogeneous systems in small single-dispatcher and larger-scale multi-dispatcher scenarios, we illustrated how simple low-communication *Loosely-Shortest-Queue* known policies are stable and at the same time exhibit appealing performance. Our example policies significantly outperform well-known low-communication policies such as *JSQ*(2) and *JIQ*, while obeying the same constraints on the communication overhead.

Given the strength of the *Loosely-Shortest-Queue* approach in large-scale multi-dispatcher heterogeneous systems, we believe that it has the potential to open a new thread in the research of scalable load balancing policies.

REFERENCES

[1] 2017. NetScaler 11.1. The Least Connection Method. https://docs.citrix.com/en-us/netscaler/11-1/load-balancing/load-balancing-customizing-algorithms/leastconnection-method.html. (2017).

[2] Maury Bramson, Yi Lu, and Balaji Prabhakar. 2010. Randomized load balancing with general service time distributions. In *ACM SIGMETRICS Performance Evaluation Review*, Vol. 38. ACM, 275–286.

[3] Maury Bramson, Yi Lu, and Balaji Prabhakar. 2012. Asymptotic independence of queues under randomized load balancing. *Queueing Systems* 71, 3 (2012), 247–292.

[4] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. *Commun. ACM* 51, 1 (2008), 107–113.

[5] Robert D Foley and David R McDonald. 2001. Join the shortest queue: stability and exact asymptotics. *Annals of Applied Probability* (2001), 569–607.

[6] G Foschini and J Salz. 1978. A basic dynamic routing problem and diffusion. *IEEE Transactions on Communications* 26, 3 (1978), 320–327.

[7] Serguei Foss and Natalia Chernova. 1998. On the stability of a partially accessible multi-station queue with state-dependent routing. *Queueing Systems* 29, 1 (1998), 55–73.

[8] Ian Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. 2008. Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop, 2008. GCE'08*. Ieee, 1–10.

[9] David Gamarnik, John N Tsitsiklis, and Martin Zubeldia. 2018. Delay, memory, and messaging tradeoffs in distributed service systems. *Stochastic Systems* 8, 1 (2018), 45–74.

[10] Rohan Gandhi, Hongqiang Harry Liu, Y Charlie Hu, Guohan Lu, Jitendra Padhye, Lihua Yuan, and Ming Zhang. 2015. Duet: Cloud scale load balancing with hardware and software. *ACM SIGCOMM Computer Communication Review* 44, 4 (2015), 27–38.

[11] Leonidas Georgiadis, Michael J Neely, Leandros Tassiulas, et al. 2006. Resource allocation and cross-layer control in wireless networks. *Foundations and Trends® in Networking* 1, 1 (2006), 1–144.

[12] Ramesh Govindan, Ina Minei, Mahesh Kallahalla, Bikash Koley, and Amin Vahdat. 2016. Evolve or die: High-availability design principles drawn from googles network infrastructure. ACM, 58–72.

[13] Varun Gupta, Mor Harchol Balter, Karl Sigman, and Ward Whitt. 2007. Analysis of join-the-shortest-queue routing for web server farms. *Performance Evaluation* 64, 9-12 (2007), 1062–1081.

[14] Anselmi Jonatha and Dufour Francois. 2018. Power-of-d-Choices with Memory: Fluid Limit and Optimality. *arXiv preprint arXiv:1802.06566* (2018).

[15] Yi Lu, Qiaomin Xie, Gabriel Kliot, Alan Geller, James R Larus, and Albert Greenberg. 2011. Join-Idle-Queue: A novel load balancing algorithm for dynamically scalable web services. *Performance Evaluation* 68, 11 (2011), 1056–1071.

[16] Michael Mitzenmacher. 2016. Analyzing distributed Join-Idle-Queue: A fluid limit approach. In *Communication, Control, and Computing (Allerton), 2016 54th Annual Allerton Conference on*. IEEE, 312–318.

[17] Michael Mitzenmacher, Balaji Prabhakar, and Devavrat Shah. 2002. Load balancing with memory. In *null*. IEEE, 799.

[18] Arpan Mukhopadhyay, A Karthik, and Ravi R Mazumdar. 2016. Randomized assignment of jobs to servers in heterogeneous clusters of shared servers for low delay. *Stochastic Systems* 6, 1 (2016), 90–131.

[19] Michael J Neely. 2012. Stability and probability 1 convergence for queueing networks via Lyapunov optimization. *Journal of Applied Mathematics* Volume 2012 (2012), Article ID 831909, 35 pages.

[20] Rajesh Nishtala, Hans Fugal, Steven Grimm, Marc Kwiatkowski, Herman Lee, Harry C Li, Ryan McElroy, Mike Paleczny, Daniel Peek, Paul Saab, et al. 2013. Scaling Memcache at Facebook.. In *nsdi*, Vol. 13. 385–398.

[21] Will Reese. 2008. Nginx: the high-performance web server and reverse proxy. *Linux Journal* 2008, 173 (2008), 2.

[22] Devavrat Shah and Balaji Prabhakar. 2002. The use of memory in randomized load balancing. In *Information Theory, 2002. Proceedings. 2002 IEEE International Symposium on*. IEEE, 125.

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

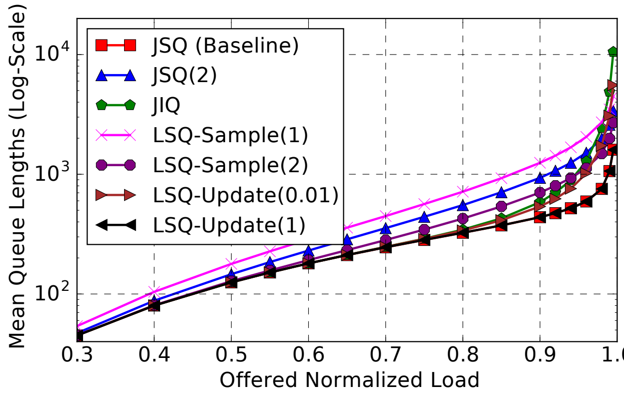
[24] Alexander L Stolyar. 2017. Pull-based load distribution among heterogeneous parallel servers: the case of multiple routers. *Queueing Systems* 85, 1-2 (2017), 31–65.

[25] Willy Tarreau et al. 2012. HAProxy-the reliable, high-performance TCP/HTTP load balancer. (2012).

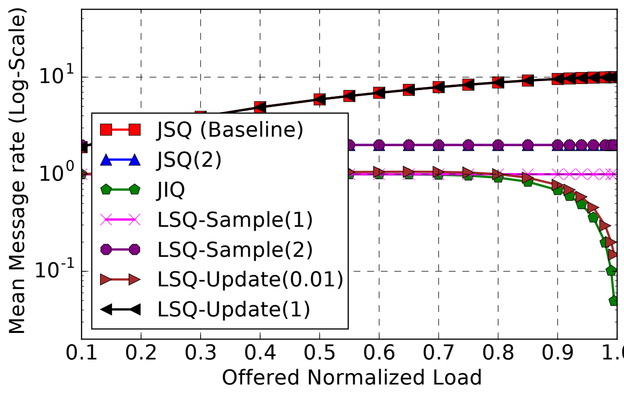
[26] M Van der Boor, SC Borst, and JSH Van Leeuwaarden. 2017. Hyper-scalable JSQ with sparse feedback. *Preprint* (2017).

[27] Mark van der Boor, Sem Borst, and Johan van Leeuwaarden. 2017. Load balancing in large-scale systems with multiple dispatchers. In *INFOCOM 2017-IEEE Conference on Computer Communications, IEEE*. IEEE, 1–9.

[28] Mark van der Boor, Sem C Borst, Johan SH van Leeuwaarden, and Debankur Mukherjee. 2018. Scalable load balancing in networked systems: A survey of recent advances. *arXiv preprint arXiv:1806.05444* (2018).



(a) Mean queue lengths.



(b) Mean message rate.

Figure 6: Scenario with a single dispatcher and 10 homogeneous servers.

[29] Richard R Weber. 1978. On the optimal assignment of customers to parallel servers. *Journal of Applied Probability* 15, 2 (1978), 406–413.

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

[31] Lei Ying, R Srikant, and Xiaohan Kang. 2017. The power of slightly more than one sample in randomized load balancing. *Mathematics of Operations Research* 42, 3 (2017), 692–722.

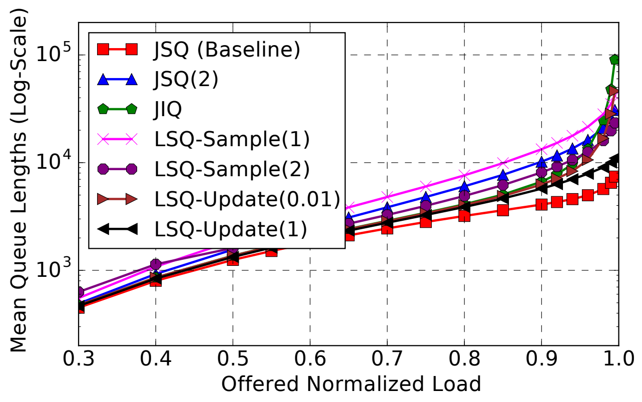
[32] Xingyu Zhou, Fei Wu, Jian Tan, Yin Sun, and Ness Shroff. 2017. Designing Low-Complexity Heavy-Traffic Delay-Optimal Load Balancing Schemes: Theory to Algorithms. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 1, 2 (2017), 39.

A SIMULATION RESULTS FOR THE HOMOGENEOUS CASE

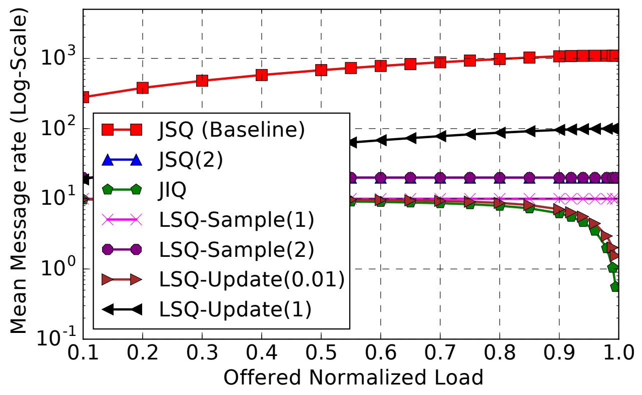
We now consider systems with identical servers. In these simulations, the job arrival process at a dispatcher is a Poisson process with parameter λ, and the server service processes are geometrically distributed with parameter p. In a simulation with n servers and m dispatchers we set $p = \frac{n}{100m}$ and sweep $0 \le \lambda < 100$.

A.1 Small scale scenario

In this evaluation, we consider a small scale scenario with a single dispatcher and 10 homogeneous servers. The results are depicted in Figure 6. As dictated by theory, all tested approaches are stable in this scenario. Our *Loosely-Shortest-Queue-Update*(1) achieves



(a) Mean queue lengths.



(b) Mean message rate.

Figure 7: Scenario with 10 dispatchers and 100 homogeneous servers.

the best performance which is identical to the baseline, *i.e., JSQ*. This is because in a single dispatcher scenario *Loosely-Shortest-Queue-Update*(1) is always aware of the exact queue length of all queues.

Loosely-Shortest-Queue-Update(0.01) offers better performance than *JIQ* especially as the load increases. This is achieved with similar average communication overhead. It is notable that *JIQ* performs similarly to *JSQ* at low loads, but its performance quickly degrades as the load increases. This complies with the latest theoretical results indicating that *JIQ* is asymptotically worse than *JSQ* at high loads (*i.e., JIQ* is not heavy traffic delay optimal) [32].

Finally, *Loosely-Shortest-Queue-Update*(2) is always better than its *JSQ*(2) counterpart using exactly the same communication overhead. *Loosely-Shortest-Queue-Update*(1) is slightly worse in this scenario but with a lesser communication overhead.

A.2 Larger scale scenario

In this evaluation, we consider a larger scale scenario with 10 dispatcher and 100 homogeneous servers. The results are depicted in Figure 7. Similarly to the previous scenario, as dictated by theory, it is evident that all tested approaches are stable in this scenario as well. Our *Loosely-Shortest-Queue-Update*(1) achieves good performance which is slightly worse than *JSQ* while using by an order of magnitude less communication. The performance slightly degrades since in this multiple dispatchers scenario *Loosely-Shortest-Queue-Update*(1) cannot always be aware of the exact queue length of all queues.

Again, *Loosely-Shortest-Queue-Update*(0.01) is always better than *JIQ* with similar average communication overhead and *Loosely-Shortest-Queue-Update*(2) is always better than its *JSQ*(2) counterpart using exactly the same communication overhead.