



Stability, Memory, and Messaging Trade-Offs in Heterogeneous Service Systems

 David Gamarnik,^a John N. Tsitsiklis,^b Martin Zubeldia^c
^aOperations Research Center, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139; ^bLaboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, Massachusetts, 02139; ^cIndustrial and Systems Engineering Department, Georgia Institute of Technology, Atlanta, Georgia 30332

Contact: gamarnik@mit.edu,  <https://orcid.org/0000-0001-8898-8778> (DG); jnt@mit.edu (JNT); m.zubeldia.suarez@tue.nl,  <https://orcid.org/0000-0003-1320-9893> (MZ)

Received: July 10, 2020
Revised: April 16, 2021
Accepted: June 11, 2021
Published Online in Articles in Advance:
 November 9, 2021

MSC2000 Subject Classification: Primary: 60G52; secondary: 90B22
OR/MS Subject Classification: Information systems; analysis and design

<https://doi.org/10.1287/moor.2021.1191>
Copyright: © 2021 INFORMS

Abstract. We consider a heterogeneous distributed service system consisting of n servers with unknown and possibly different processing rates. Jobs with unit mean arrive as a renewal process of rate proportional to n and are immediately dispatched to one of several queues associated with the servers. We assume that the dispatching decisions are made by a central dispatcher with the ability to exchange messages with the servers and endowed with a finite memory used to store information from one decision epoch to the next, about the current state of the queues and about the service rates of the servers. We study the fundamental resource requirements (memory bits and message exchange rate) in order for a dispatching policy to be always stable. First, we present a policy that is always stable while using a positive (but arbitrarily small) message rate and $\log_2(n)$ bits of memory. Second, we show that within a certain broad class of policies, a dispatching policy that exchanges $o(n^2)$ messages per unit of time, and with $o(\log(n))$ bits of memory, cannot be always stable.

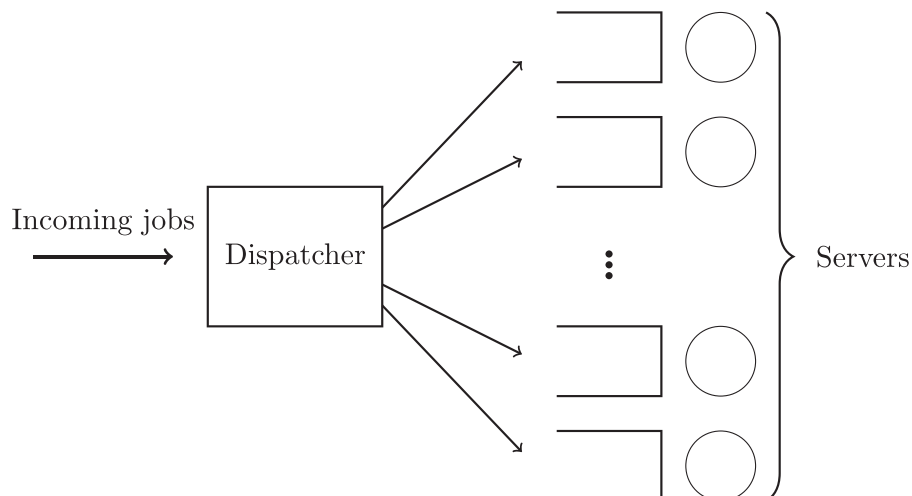
Funding: This work was supported by Nederlandse Organisatie voor Wetenschappelijk Onderzoek [NETWORKS-024.002.003].

Keywords: load balancing • stability • memory • communication overhead

1. Introduction

Distributed service systems are pervasive, from the checkout lines at supermarkets to server farms for cloud computing. At a high level, many of these systems involve a stream of incoming jobs that are dispatched to a distinct queue associated with one of the servers (see Figure 1 for a stylized model). Naturally, the behavior and performance of such systems depend on the dispatching policy.

Although delay performance and stability are important factors when choosing how to operate these systems, the huge number of servers in applications such as multicore processors and data centers has led to a desire for low communication overhead. On the other hand, communication between the dispatcher and the servers, as

Figure 1. Parallel server queuing system with a central dispatcher.


well as memory at the dispatcher, allows the dispatcher to obtain and store information about the current state of the queues and about the characteristics of the servers, leading to better dispatching decisions. This points to a trade-off between the resources utilized (in terms of communication overhead and memory) and the attainable delay performance and stability of the system.

In this paper, we consider a heterogeneous distributed service system, where servers can have different and unknown processing rates, and explore the trade-off between the stability region of the system and the amount of communication overhead and memory used to gather and store relevant information. This complements the work in Gamarnik et al. [6, 7], where the authors explore the trade-off between the delay performance and the amount of communication overhead and memory in a system with identical servers. In particular, in the setting of Gamarnik et al. [6, 7], stability was easy to achieve (even with a static, randomized policy), and the focus was on the queueing delay going to zero (as the arrival rate and the number of servers jointly increase). In the present context, stability becomes an issue: the dispatcher must either “learn” the rates of the different servers (and store this information in its memory), or must use some dynamic queue-size information to stabilize the system.

Remark 1. It is important to note that we use the term “memory” throughout this paper to refer to the memory used to store information from one decision epoch to the next, about the current state of the queues and about the service rates of the servers. In particular, it does not refer to the total memory requirements (at a hardware level) of a practical implementation of a policy, which can be much greater. Indeed, practical implementations of a central controller could require more memory to store, for example, the addresses of the servers to which it is attached. Moreover, even larger amounts of memory could be required to make computations, or to temporarily store information about the current state of the queues (e.g., to compare multiple queue lengths at a decision epoch).

The reason behind our focus on this type of memory is twofold. First, it allows us to concentrate on the amount of information stored that is relevant to the stability of the system. Second, the type of memory that we consider is application agnostic, allowing our results to be applicable in broader settings.

1.1. Previous Work

There is a wide range of policies for operating the system described above, which result in different delay performances, stability regions, and resource utilizations. For example, a most simple policy is to dispatch jobs uniformly at random. This policy requires no message exchanges and no memory, but it is unstable if some server is slow enough. At the opposite extreme, the server can use dynamically available information and send incoming jobs to a shortest queue. This policy results in small delay and is maximally stable (Bramson [4]) but requires substantial communication overhead and an unbounded memory.

Many intermediate policies have been proposed and analyzed in the past, with a focus on low resource usage. Most notably, the power-of- d -choices algorithm (also known as [Shortest Queue] SQ(d)) was introduced and analyzed in Mitzenmacher [9] and Vvedenskaya et al. [15], and results in relatively low average delays for the jobs, while requiring a message rate proportional to the arrival rate and no memory. However, the blind randomization used by the policy renders it unstable if there is at least one sufficiently slow server. Another popular policy is Join-Idle-Queue (Lu et al. [8], Stolyar [12]), which leverages the power of memory (one bit per server) to obtain vanishing queueing delays (as the arrival rate and the number of servers jointly increase) while using roughly the same amount of communication overhead as the power-of- d -choices algorithm. However, this policy also utilizes blind randomization that renders it unstable if there is at least one sufficiently slow server (Atar et al. [3]).

Recently, there has been a focus on policies that attain a vanishing queueing delay while minimizing their resource usage. In particular, in Mukherjee et al. [10], a variation of the power-of- d -choices algorithm was shown to yield a vanishing queueing delay while using no memory and a message rate that is superlinear in the arrival rate. Moreover, variations of Join-Idle-Queue were shown to have vanishing queueing delays with either a memory of size (in bits) superlogarithmic in the number of servers and a message rate equal to the arrival rate (Gamarnik et al. [6]), or a memory size (in bits) equal to the number of servers and a message rate strictly smaller (but still proportional) to the arrival rate (van der Boor et al. [13]). Last but not least, a novel combination of size-based load balancing and round-robin was shown to have vanishing queueing delay using unbounded memory and no communication overhead (Anselmi [1]).

On the other hand, there are few policies in the literature that maximize the stability region. In Anselmi and Dufour [2] and Shah and Prabhakar [11], the authors present and analyze a variation of power-of- d -choices algorithm that utilizes memory (of size logarithmic in the number of servers) that guarantees maximal stability. Furthermore, in Atar et al. [3], the authors propose yet another variation of Join-Idle-Queue, dubbed Persistent-Idle, that achieves maximal stability, without any randomization. This policy requires a message rate proportional to the arrival rate and a memory of size (in bits) at least proportional to the number of servers.

Finally, independent simultaneous work (Vargaftik et al. [14], Zhou et al. [16]) has also focused on minimizing the communication overhead while still being maximally stable. These policies are designed to keep (possibly outdated) estimates of the queue lengths in memory, which are updated sporadically. They require arbitrarily small message rates and a memory size (in bits) that grows faster than the number of servers.

1.2. Our Contribution

Instead of focusing on yet another policy or decision-making architecture, we step back and address a more fundamental question: What are the message rate (from dispatcher to servers and from servers to dispatcher combined) and/or memory size requirements that are necessary and sufficient in order for a policy to be maximally stable? We are able to provide a fairly complete answer to this question:

a. For the case of Poisson arrivals and exponential service times, if the message rate is positive *and* the memory size (in bits) is logarithmic in the number of servers, we provide a fairly simple and natural policy that is maximally stable.

b. If the message rate is sublinear in the square of the arrival rate *and* the number of memory bits is sublogarithmic in the number of servers, we show that *no* decision-making architecture and policy, within a certain broad class of policies, is maximally stable. The main constraint that we impose on the policies that we consider is that they are “weakly symmetric,” in a sense to be defined later.

In a nutshell, with as little as a logarithmic number of bits of memory, the message rate requirement for maximal stability decreases from being proportional to the square of the arrival rate to being merely positive.

Remark 2. Our proposed policy is more economical than the most efficient maximally stable policies analyzed in earlier literature, requiring minimal communication overhead and a remarkably small memory. On the one hand, the power-of- d -choices algorithm with memory policy (Anselmi and Dufour [2], Shah and Prabhakar [11]) is maximally stable, but requires a memory of size (in bits) at least logarithmic in the number of servers and a message rate proportional to the arrival rate. On the other hand, policies based on estimated queue lengths (Vargaftik et al. [14], Zhou et al. [16]) are also maximally stable, while requiring a memory size (in bits) superlinear in the number of servers and an arbitrarily small message rate. In contrast, our proposed policy requires a memory size (in bits) logarithmic in the number of servers and an arbitrarily small message rate.

Remark 3. Given its singular focus on being maximally stably while utilizing minimal memory and communication overhead, for a given message rate, the delay performance of our policy is inferior to the delay performance of the policies in Atar et al. [3], Vargaftik et al. [14], and Zhou et al. [16], which use significantly more memory. Hence, this is not a policy meant for practical implementation in applications such as data centers, where memory is not an issue, but it serves as a counterpart to our impossibility result.

2. Model and Main Results

In this section, we present our modeling assumptions and main results. We present a unified framework for a broad set of dispatching policies, which includes most of the policies studied in the previous literature, and then present our negative result on the failure of maximal stability to hold for resource-constrained policies within this framework.

2.1. Modeling Assumptions

We consider a distributed service system consisting of n parallel servers, where each server is associated with an infinite capacity first-in, first-out queue. For each $i \in \{1, \dots, n\}$, the i th server has constant (but unknown) service rate $\mu_i > 0$. Despite the heterogeneity in the service rates, we assume that the total processing power of all servers is equal to n . Thus, the set of possible service rate vectors is

$$\Sigma_n \triangleq \left\{ \mu \in (0, \infty)^n : \sum_{i=1}^n \mu_i = n \right\}. \quad (1)$$

Jobs arrive to the system as a single renewal process of rate λn (for some fixed $\lambda \in (0, 1)$), and their sizes are independent and identically distributed (i.i.d.), independent from the arrival process, and have a general distribution with unit mean. A central controller (dispatcher) is responsible for routing every incoming job to a queue, immediately upon arrival. We assume that the dispatcher can rely only on a limited amount of local memory and on messages that provide partial information about the state and parameters of the system. These messages (which are assumed to be instantaneous) can be sent from a server to the dispatcher at any time, or from the dispatcher to a server (in the form of queries) at the time of an arrival or at the time of a spontaneous message from a server.

Messages from a server i can contain information only about the state of its own queue (number of remaining jobs and the remaining workload of each one) and about its processing rate μ_i . Within this context, a system designer has the freedom to choose a messaging policy, as well as the rules for updating the memory and for selecting the destination of an incoming job.

Regarding the performance metric, our focus is on the *stability region* of a policy under the arrival rate λ , that is, the largest subset of server rates $\Gamma_n(\lambda) \subset \Sigma_n$ such that the policy is stable for all $\mu \in \Gamma_n(\lambda)$. We will formalize this definition in Subsection 2.4.

2.2. A Maximally Stable Policy

In this subsection, we propose a simple dispatching policy with the largest possible stability region (i.e., with stability region equal to Σ_n , for all $\lambda \in (0, 1)$).

2.2.1. Policy Description. For any fixed value of n , we consider the following policy. At any time, the dispatcher stores the ID of a single server in its memory. This ID is initialized in an arbitrary way, and it is updated based on spontaneous messages from the servers. In particular, each server sends messages to the dispatcher as an independent Poisson process of rate $\alpha_n > 0$, informing the dispatcher of its queue length (i.e., of the number of jobs in its queue or in service). When a message from a server arrives to the dispatcher, the dispatcher stores the ID of this server only if the sender's queue is shorter than the queue of the server that is currently stored in memory. In order to make this comparison, the queue length of the currently stored server is obtained by sending a query to it. Finally, whenever a new job arrives to the system, it is sent to the server whose ID is stored in the dispatcher's memory (the server ID in memory does not change at this point).

Remark 4. This policy requires only $\lceil \log_2(n) \rceil$ bits of memory and an arbitrarily small (but positive) average message rate of $3\alpha_n n$.

2.2.2. Main Result. When the arrival process is Poisson and the service times are exponentially distributed, the behavior of the system under this policy can be modeled as a continuous-time Markov chain $(\mathbf{Q}(\cdot), I(\cdot))$, where $\mathbf{Q}(\cdot) = (\mathbf{Q}_1(\cdot), \dots, \mathbf{Q}_n(\cdot))$ is the vector of queue lengths, and $I(\cdot)$ is the ID of the server stored in memory. In this setting, the stability of the policy is established in the following result.

Theorem 1. *Suppose that the arrival process is Poisson, and that the job sizes are exponentially distributed. For any n , if $\alpha_n > 0$, then the stability region of the policy described above is Σ_n for all $\lambda \in (0, 1)$.*

This stability result is established by constructing an appropriate Lyapunov function. The proof is given in Appendix A.

Theorem 1 states that, at least in the Markovian case, the stability region of our proposed policy is the whole set of admissible rates Σ_n . Moreover, it implies that $\lceil \log_2(n) \rceil$ bits of memory and an average message rate of $3\alpha_n n$ (which can be arbitrarily small) are sufficient for a policy to be always stable. We conjecture that our policy is always stable, even with renewal arrivals and generally distributed service times.

Although Theorem 1 ensures stability even when the message rate is arbitrarily small, a small message rate will result in poor steady-state delay performance of the policy. Indeed, because the policy sends all incoming jobs to the same queue between consecutive messages, a small message rate leads to large buildups of jobs in the queues. In particular, we conjecture that the steady-state queueing delay of our policy is of order $\Theta(1/\alpha_n)$.

Given this apparent trade-off between the average message rate and the delay performance of the policy, the proposed policy is most useful for applications where a large stability region and a small communication overhead are preferred, and where there is tolerance for large delays. Furthermore, because the policy does not depend explicitly on the rates of the servers (or estimates thereof), it would continue to work even if the service rates changed slowly over time, which makes it robust.

2.3. A General Class of Dispatching Policies

In this subsection, we present a unified framework that describes memory-based dispatching policies in systems with heterogeneous servers, which is slightly more general than the one introduced in Gamarnik et al. [7].

Let c_n be the number of memory bits available to the dispatcher. We define the corresponding set of memory states to be $\mathcal{M}_n \triangleq \{1, \dots, 2^{c_n}\}$. Furthermore, we define the set of possible states at a server as the set of nonnegative sequences $\mathcal{Q} \triangleq \mathbb{R}_+^{\mathbb{Z}_+}$, where a sequence specifies the remaining workload of each job in that queue, including the one that is being served. (In particular, an idle server is represented by the zero sequence.) As long as a queue has a finite number of jobs, the queue state is a sequence that has only a finite number of nonzero entries. The

reason that we include the workload of the jobs in the state is that we wish to allow for a broad class of policies that can take into account the remaining workload in the queues. In particular, we allow for information-rich messages that describe the full workload sequence at the server that sends the message. We are interested in the process

$$\mathbf{Q}(\cdot) = (\mathbf{Q}_1(\cdot), \dots, \mathbf{Q}_n(\cdot)) = \left((\mathbf{Q}_{1,j}(\cdot))_{j=1}^{\infty}, \dots, (\mathbf{Q}_{n,j}(\cdot))_{j=1}^{\infty} \right),$$

which takes values in the set \mathcal{Q}^n and describes the evolution of the workload of each job in each queue. We are also interested in the process $M(\cdot)$ that describes the evolution of the memory state, and in a process $Z(\cdot)$ that describes the elapsed time since the arrival of the previous job.

2.3.1. Fundamental Processes and Initial Conditions. All processes of interest are driven by the following common fundamental processes:

1. *Arrival process:* A delayed renewal counting process $A_n(\cdot)$ with rate λn and event times $\{T_k\}_{k=1}^{\infty}$, defined on a probability space $(\Omega_A, \mathcal{A}_A, \mathbb{P}_A)$.
2. *Spontaneous messages process:* A Poisson counting process $R_n(\cdot)$ with rate β_n , and event times $\{T_k^s\}_{k=1}^{\infty}$, defined on a probability space $(\Omega_R, \mathcal{A}_R, \mathbb{P}_R)$.
3. *Job sizes:* A sequence of i.i.d. random variables $\{W_k\}_{k=1}^{\infty}$ with mean one, defined on a probability space $(\Omega_W, \mathcal{A}_W, \mathbb{P}_W)$.
4. *Randomization variables:* Eight independent and individually i.i.d. sequences of random variables $\{U_{1,k}\}_{k=1}^{\infty}, \dots, \{U_{8,k}\}_{k=1}^{\infty}$, uniform on $[0, 1]$, defined on a common probability space $(\Omega_U, \mathcal{A}_U, \mathbb{P}_U)$.
5. *Initial conditions:* Random variables $\mathbf{Q}(0)$, $M(0)$, and $Z(0)$, defined on a common probability space $(\Omega_0, \mathcal{A}_0, \mathbb{P}_0)$.

The whole system will be defined on the associated product probability space

$$(\Omega_A \times \Omega_R \times \Omega_W \times \Omega_U \times \Omega_0, \mathcal{A}_A \times \mathcal{A}_R \times \mathcal{A}_W \times \mathcal{A}_U \times \mathcal{A}_0, \mathbb{P}_A \times \mathbb{P}_R \times \mathbb{P}_W \times \mathbb{P}_U \times \mathbb{P}_0),$$

to be denoted by $(\Omega, \mathcal{A}, \mathbb{P})$. All of the randomness in the system originates from these fundamental processes, and everything else is a deterministic function of them. In particular, the *randomization variables* are used to randomize the actions of the policies, independently of the rest of the system.

2.3.2. A Construction of Sample Paths. We provide a construction of a Markov process $(\mathbf{Q}(\cdot), M(\cdot), Z(\cdot))$, taking values in the set $\mathcal{Q}^n \times \mathcal{M}_n \times \mathbb{R}_+$. The memory process $M(\cdot)$ is piecewise constant and can jump only at the time of an event. All processes considered will have the càdlàg property (right continuous with left limits) either by assumption (e.g., the underlying fundamental processes) or by construction.

There are three types of events: job arrivals, spontaneous messages, and service completions. We now describe the sources of these events, and what happens when they occur.

2.3.2.1. Job Arrivals. At the time of the k th event of the arrival process A_n , which occurs at time T_k and involves a job with size W_k , the following transitions happen sequentially but instantaneously:

1. First, the dispatcher chooses a set S_k of distinct servers, from which it solicits information about their state, according to

$$S_k = f_1(M(T_k^-), W_k, U_{1,k}),$$

where $f_1 : \mathcal{M}_n \times \mathbb{R}_+ \times [0, 1] \rightarrow \mathcal{P}(\{1, \dots, n\})$ is a measurable function defined by the policy. Here, and in the sequel, $\mathcal{P}(A)$ stands for the power set of a set A .

2. Second, messages are sent to the servers in the set S_k , and the servers respond with messages containing their queue states and their service rates. This results in a total of $2|S_k|$ messages exchanged. Using this information, the destination of the incoming job is chosen to be

$$D_k = f_2(M(T_k^-), W_k, \{(\mathbf{Q}_i(T_k^-), \mu_i, i) : i \in S_k\}, U_{2,k}),$$

where $f_2 : \mathcal{M}_n \times \mathbb{R}_+ \times \mathcal{B}_n \times [0, 1] \rightarrow \{1, \dots, n\}$ is a measurable function defined by the policy, with $\mathcal{B}_n \subset \mathcal{P}(\mathcal{Q} \times \mathbb{R}_+ \times \{1, \dots, n\})$ comprised of those sets of triples such that the triples in a set have different third coordinates. Note that the destination of a job can depend not only on the current memory state, the job size, the set of queried servers, and the state of their queues, but also on the rates of the queried servers.

3. Third, the memory state is updated according to

$$M(T_k) = f_3(M(T_k^-), W_k, \{(\mathbf{Q}_i(T_k^-), \mu_i, i) : i \in S_k\}, D_k, U_{3,k}),$$

where $f_3 : \mathcal{M}_n \times \mathbb{R}_+ \times \mathcal{B}_n \times \{1, \dots, n\} \times [0, 1] \rightarrow \mathcal{M}_n$ is a measurable function defined by the policy. Note that the new memory state is obtained using the same information as for selecting the destination, including the rates of the queried servers, plus the destination of the job.

2.3.2.2. Spontaneous Messages. At the time of the k th event of the spontaneous message process R_n , which occurs at time T_k^s , the i th server sends a spontaneous message to the dispatcher if and only if

$$g_1(\mathbf{Q}(T_k^s), \mu, U_{4,k}) = i,$$

where $g_1 : \mathcal{Q}^n \times \mathbb{R}_+^n \times [0, 1] \rightarrow \{0, 1, \dots, n\}$ is a measurable function defined by the policy. On the other hand, no message is sent when $g_1(\mathbf{Q}(T_k^s), \mu, U_{4,k}) = 0$. Note that the dependence of g_1 on \mathbf{Q} and μ allows the message rate at each server to depend on all servers' current workloads and on their rates. This allows for policies that let servers with higher service rates send messages at a higher rate than servers with slower service rates.

When a spontaneous message from server i arrives to the dispatcher, the following transitions happen sequentially but instantaneously:

1. First, the dispatcher chooses a set of distinct servers S_k^s , from which it solicits information about their state, according to

$$S_k^s = g_2(M(T_k^-), i, \mathbf{Q}_i(T_k^s), \mu_i, U_{5,k}),$$

where $g_2 : \mathcal{M}_n \times \{1, \dots, n\} \times \mathcal{Q} \times \mathbb{R}_+ \times [0, 1] \rightarrow \mathcal{P}(\{1, \dots, n\})$ is a measurable function defined by the policy. Note that the set of servers that are sampled not only depends on the current memory state but also on the index, queue state, and rate of the server that sent the message.

2. Second, messages are sent to the servers in the set S_k^s , and the servers respond with messages containing their queue states and their service rates. This results in a total of $2|S_k^s|$ messages exchanged. Using this information, the memory is updated to the new memory state

$$M(T_k^s) = g_3(M(T_k^{s-}), i, \mathbf{Q}_i(T_k^s), \mu_i, \{(\mathbf{Q}_j(T_k^s), \mu_j, j) : j \in S_k^s\}, U_{6,k}),$$

where $g_3 : \mathcal{M}_n \times \{1, \dots, n\} \times \mathcal{Q} \times \mathbb{R}_+ \times \mathcal{B}_n \times [0, 1] \rightarrow \mathcal{M}_n$ is a measurable function defined by the policy.

2.3.2.3. Service Completions. Let $\{T_k^d(i)\}_{k=1}^\infty$ be the sequence of departure times at the i th server. At those times, the i th server sends a message to the dispatcher if and only if

$$h_1(\mathbf{Q}_i(T_k^d(i)), \mu_i, U_{7,k}) = 1,$$

where $h_1 : \mathcal{Q} \times \mathbb{R}_+ \times [0, 1] \rightarrow \{0, 1\}$ is a measurable function defined by the policy. In that case, the memory is updated to the new memory state

$$M(T_k^d(i)) = h_2(M(T_k^d(i)^-), i, \mathbf{Q}_i(T_k^d(i)), \mu_i, U_{8,k}),$$

where $h_2 : \mathcal{M}_n \times \{1, \dots, n\} \times \mathcal{Q} \times \mathbb{R}_+ \times [0, 1] \rightarrow \mathcal{M}_n$ is a measurable function defined by the policy. Finally, no message is sent when $h_1(\mathbf{Q}_i(T_k^d(i)), \mu_i, U_{7,k}) = 0$.

Remark 5. Note that this framework allows for policies that are more general than those considered in Gamarnik et al. [7]. In particular, (i) some decisions can depend on the rates of the different servers, (ii) the dispatcher can sample servers whenever a spontaneous message arrives, and (iii) memory updates may involve randomization.

We now introduce a symmetry assumption on the policies.

Assumption 1 (Weakly Symmetric Policies).

We assume that the dispatching policy is weakly symmetric, in the following sense. For any given permutation of the servers σ , there exists a corresponding (not necessarily unique) permutation σ_M of the memory states \mathcal{M}_n that satisfies both of the following properties:

1. For every $m \in \mathcal{M}_n$ and $w \in \mathbb{R}_+$, and if U is a uniform random variable on $[0, 1]$, then

$$\sigma(f_1(m, w, U)) \stackrel{d}{=} f_1(\sigma_M(m), w, U),$$

where $\stackrel{d}{=}$ stands for equality in distribution. Note that this equality in distribution is only with respect to U .

2. For every $m \in \mathcal{M}_n$, $w \in \mathbb{R}_+$, $S \in \mathcal{P}(\{1, \dots, n\})$, $\mathbf{q} \in \mathcal{Q}^n$, and $\mu \in \mathbb{R}_+^n$, and if V is a uniform random variable on $[0, 1]$, then

$$\sigma(f_2(m, w, \{(\mathbf{q}_i, \mu_i, i) : i \in S\}, V)) \stackrel{d}{=} f_2(\sigma_M(m), w, \{(\mathbf{q}_i, \mu_i, \sigma(i)) : i \in S\}, V).$$

Remark 6. This assumption prevents any bias for or against a server, unless it is encoded in the memory in a sufficiently detailed way so that the assumption is satisfied. For example, in order to implement (in a weakly symmetric way) the randomized dispatching policy where incoming jobs are sent to a server with a probability proportional to its processing rate, the second condition in Assumption 1 requires the dispatching probabilities to be encoded in memory, in a sufficiently detailed way.

Remark 7. Note that the universally stable policy introduced in Subsection 2.2.1 falls within the class of policies defined by this general framework, and it satisfies Assumption 1.

2.4. Instability of Resource-Constrained Policies

In this subsection, we state the main result about the instability of general weakly symmetric dispatching policies. Before stating this main result, we first define the *average message rate* between the dispatcher and the servers as

$$\limsup_{t \rightarrow \infty} \frac{1}{t} \left[\sum_{k=1}^{A_n(t)} 2 |S_k| + \sum_{k=1}^{R_n(t)} (1 + 2 |S_k^s|) \mathbb{1}_{\{1, \dots, n\}}(g_1(\mathbf{Q}(T_k^s), \mu, U_{4,k})) + \sum_{i=1}^n \sum_{k: T_k^d(i) < t} \mathbb{1}_{\{1\}}(h_1(\mathbf{Q}_i(T_k^d(i)), \mu_i, U_{7,k})) \right]. \quad (2)$$

Second, we provide a formal definition of our performance metric: the stability region of a policy. For each n , given a policy and an arrival rate λ , the stability region of the policy under the arrival rate λ , denoted by $\Gamma_n(\lambda)$, is the set of all server rates in Σ_n for which the process $(\mathbf{Q}(\cdot), M(\cdot), Z(\cdot))$ is positive Harris recurrent.

We are now ready to state our main negative result. It asserts that within the class of weakly symmetric policies that we consider, and under some constraints on the memory size and the message rate, the stability region does not contain all possible rates.

Theorem 2 (Instability of Resource-Constrained Policies).

For any constant $\lambda \in (0, 1)$ and positive sequence $\{\alpha_n\}_{n \geq 1}$, there exists a sequence of stability regions $\{\Gamma_n(\lambda, \alpha_n)\}_{n \geq 1}$, where $\Gamma_n(\lambda, \alpha_n) \subsetneq \Sigma_n$ for all $n \geq 1$, with the following property. Consider a sequence of weakly symmetric memory-based dispatching policies, that is, that satisfy Assumption 1, with at most $c_n \in o(\log(n))$ bits of memory, and with an average message rate (see Equation (2)) upper bounded by $\alpha_n \in o(n^2)$ almost surely. Then, for all n large enough, the stability region of the policies under the arrival rate λ are contained in $\Gamma_n(\lambda, \alpha_n)$.

The proof consists of showing that when half of the servers have a sufficiently small service rate $\epsilon_n \in \Theta(e^{-\alpha_n/n})$, the total workload of the system diverges, as a function of time, for all n large enough. It also relies heavily on a combinatorial result (Proposition B.1 in Appendix B) from Gamarnik et al. [7] on the limitations imposed by symmetry on a limited memory. The proof is given in Appendix B.

Remark 8. Theorem 2 states that the stability region of a policy is contained in a proper subset of Σ_n , which depends only on n , the arrival rate λ , and the message rate α_n . This means that for all n large enough, and as long as $\alpha_n \in o(n^2)$, there is at least one vector of processing rates for which the system is unstable under any weakly symmetric policy with $o(\log(n))$ bits of memory and a message rate that is upper bounded by α_n .

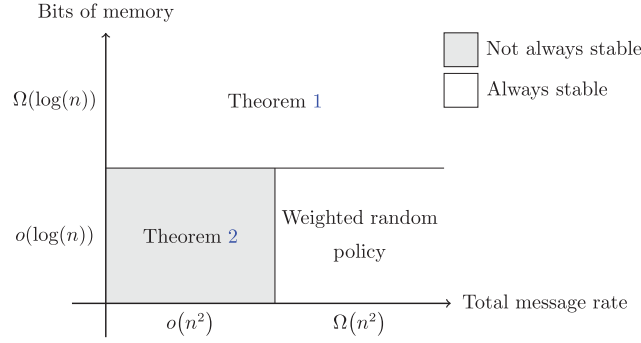
Remark 9. The most interesting regime is the one where $\alpha_n \in O(n)$, that is, when we have a constant number of messages per job. In this regime, when half of the servers have rate $\epsilon_n \in \Theta(1)$, weakly symmetric policies with $o(\log(n))$ memory are unstable. In particular, resource-constrained policies become unstable for a significant portion of the possible service rate vectors Σ_n .

On the other hand, when $\alpha_n \in \omega(n)$, our result requires half of the servers to have rate $\epsilon_n \in \Theta(e^{-\alpha_n/n})$, which is exponentially small in $\alpha_n/n \in \omega(1)$. This suggests that when the average message grows faster than n , it is only a very small portion of the possible service rate vectors that can destabilize all resource-constrained policies.

2.5. Stability vs. Resources Trade-Off

In this subsection, we provide a visual summary of our results on the trade-off between the stability region and the memory and communication overhead of weakly symmetric dispatching policies.

First, according to Theorem 1, with a memory size of at least $\lceil \log_2(n) \rceil$ bits and with an arbitrarily small message rate, we can obtain a weakly symmetric policy that is always stable (for any service rate vector in Σ_n). Second, Theorem 2 states that weakly symmetric policies with $o(\log(n))$ bits of memory and a message rate of order $o(n^2)$ cannot be always stable. Finally, note that both the join-shortest-queue policy and a policy that sends

Figure 2. Resource requirements for stable policies.

incoming jobs to each server with a probability proportional to the server's rate can be implemented by querying all servers at the time of each arrival. These policies require a message rate of order $\Theta(n^2)$ and no memory, and they are always stable. The three regimes are depicted in Figure 2.

The only remaining question in this setting is whether stability can be guaranteed with zero communication overhead and $\Omega(\log(n))$ bits of memory. In this case, no messages are exchanged, and the dispatcher can never obtain information about the rate of the servers. As a result, it can only dispatch jobs blindly, and stability fails for some server rates.

3. Conclusions and Future Work

In this paper, we proposed a simple but efficient dispatching policy that requires a memory of size (in bits) logarithmic in the number of servers and an arbitrarily small message rate, and showed that it has the largest possible stability region. The key to the stability properties of this policy is the fact that it never chooses the destination of a job by random sampling of the servers (like the power-of- d -choices algorithm) or by random dispatching of the job (like Join-Idle-Queue). On the other hand, we showed that when we have a memory size (in bits) sublogarithmic in the number of servers and a message rate sublinear in the square of the arrival rate, all weakly symmetric dispatching policies have a suboptimal stability region.

There are several interesting directions for future research, such as the following, for example:

- i. Although policies can have the largest possible stability region using an arbitrarily small message rate and logarithmic memory, their delay performance could be arbitrarily bad. We conjecture that the average delay of a policy is at least inversely proportional to its average message rate per server.
- ii. In light of the symmetry assumption in Theorem 2, a natural question is whether the result still holds without it. In that case, the sampling of servers and dispatching of jobs need not be uniform (as established in Propositions B.1 and B.2 in Appendix B using the symmetry assumption), and it becomes unclear whether maximal stability is still impossible in the same regime.

Appendix A. Proof of Theorem 2.1

Let us fix some n and some arbitrary vector of processing rates in Σ_n . Let μ_{min} and μ_{max} be the smallest and largest processing rates in the chosen vector, respectively. In particular, note that they are positive.

We will use the Foster–Lyapunov criterion to show that the continuous-time Markov chain $(\mathbf{Q}(\cdot), I(\cdot))$ is positive recurrent. First, note that this process has state space $\mathbb{Z}_+^n \times \{1, \dots, n\}$. Its transition rates, denoted by $r_{\cdot \rightarrow \cdot}$, are as follows, where we use \mathbf{e}_j to denote the j th unit vector in \mathbb{Z}_+^n :

1. Because incoming jobs are sent to the queue whose ID is stored in memory, each queue sees arrivals with rate

$$r_{(\mathbf{q}, i) \rightarrow (\mathbf{q} + \mathbf{e}_i, i)} = \lambda n.$$

2. Transitions due to service completions occur according to the processing rate of each server, and they do not affect the ID stored in memory:

$$r_{(\mathbf{q}, i) \rightarrow (\mathbf{q} - \mathbf{e}_i, i)} = \mu_i \mathbb{1}_{[1, \infty)}(\mathbf{q}_i).$$

3. Spontaneous messages are sent from each server to the dispatcher at a rate equal to α_n , but the ID stored in memory changes only if the sender of the message has a shorter queue:

$$r_{(\mathbf{q}, i) \rightarrow (\mathbf{q}, j)} = \alpha_n \mathbb{1}_{[0, \mathbf{q}_i - 1]}(\mathbf{q}_j).$$

4. Any transitions that do not appear in the above have zero rate.

Note that the Markov process $(\mathbf{Q}(\cdot), I(\cdot))$ on the state space $\mathbb{Z}_+^n \times \{1, \dots, n\}$ is irreducible, with all states reachable from each other. To show positive recurrence, we define the Lyapunov functions

$$\begin{aligned}\Xi_1(\mathbf{q}, i) &\triangleq \frac{2\mu_{\max}}{\alpha_n} \mathbf{q}_i, \\ \Xi_2(\mathbf{q}, i) &\triangleq \sum_{j=1}^n \mathbf{q}_j^2,\end{aligned}$$

and

$$\Xi(\mathbf{q}, i) \triangleq \Xi_1(\mathbf{q}, i) + \Xi_2(\mathbf{q}, i), \tag{A.1}$$

and note that

$$\sum_{(\mathbf{q}', i') \neq (\mathbf{q}, i)} \Xi(\mathbf{q}', i') r_{(\mathbf{q}, i) \rightarrow (\mathbf{q}', i')} < \infty, \quad \forall (\mathbf{q}, i) \in \mathbb{Z}_+^n \times \{1, \dots, n\}.$$

We also define the finite set

$$F_n \triangleq \left\{ (\mathbf{q}, i) \in \mathbb{Z}_+^n \times \{1, \dots, n\} : \sum_{j=1}^n \mathbf{q}_j < \frac{\lambda n \left(1 + \frac{2\mu_{\max}}{\alpha_n}\right) + n + 1}{2 \min\{1 - \lambda, \mu_{\min}\}} \right\}. \tag{A.2}$$

For any state (\mathbf{q}, i) , we have

$$\begin{aligned}\sum_{(\mathbf{q}', i') \in \mathbb{Z}_+^n \times \{1, \dots, n\}} [\Xi_1(\mathbf{q}', i') - \Xi_1(\mathbf{q}, i)] r_{(\mathbf{q}, i) \rightarrow (\mathbf{q}', i')} \\ = \lambda n \left(\frac{2\mu_{\max}}{\alpha_n} \right) - \frac{2\mu_{\max}}{\alpha_n} \mu_i \mathbb{1}_{[1, \infty)}(\mathbf{q}_i) - \sum_{j=1}^n 2\mu_{\max} (\mathbf{q}_i - \mathbf{q}_j)^+ \\ \leq \lambda n \left(\frac{2\mu_{\max}}{\alpha_n} \right) - \sum_{j=1}^n 2\mu_{\max} (\mathbf{q}_i - \mathbf{q}_j)^+, \end{aligned} \tag{A.3}$$

and

$$\begin{aligned}\sum_{(\mathbf{q}', i') \in \mathbb{Z}_+^n \times \{1, \dots, n\}} [\Xi_2(\mathbf{q}', i') - \Xi_2(\mathbf{q}, i)] r_{(\mathbf{q}, i) \rightarrow (\mathbf{q}', i')} &= \lambda n (2\mathbf{q}_i + 1) - \sum_{j=1}^n \mu_j (2\mathbf{q}_j - 1) \mathbb{1}_{[1, \infty)}(\mathbf{q}_j) \\ &= \lambda n (2\mathbf{q}_i + 1) + \sum_{j=1}^n \mu_j \mathbb{1}_{[1, \infty)}(\mathbf{q}_j) - 2 \sum_{j=1}^n \mu_j \mathbf{q}_j \\ &\leq \lambda n (2\mathbf{q}_i + 1) + n - 2 \sum_{j=1}^n \mu_j \mathbf{q}_j, \end{aligned} \tag{A.4}$$

where in the last inequality we used that the vector of server rates μ is in Σ_n , so that

$$\sum_{j=1}^n \mu_j = n. \tag{A.5}$$

Combining Equations (A.1), (A.3), and (A.4), for any state $(\mathbf{q}, i) \notin F_n$, we have

$$\begin{aligned}\sum_{(\mathbf{q}', i') \in \mathbb{Z}_+^n \times \{1, \dots, n\}} [\Xi(\mathbf{q}', i') - \Xi(\mathbf{q}, i)] r_{(\mathbf{q}, i) \rightarrow (\mathbf{q}', i')} \\ \leq \lambda n \left(1 + \frac{2\mu_{\max}}{\alpha_n}\right) + n + 2\lambda n \mathbf{q}_i - 2 \sum_{j=1}^n [\mu_j \mathbf{q}_j + \mu_{\max} (\mathbf{q}_i - \mathbf{q}_j)^+] \\ \leq \lambda n \left(1 + \frac{2\mu_{\max}}{\alpha_n}\right) + n + 2\lambda n \mathbf{q}_i - 2 \sum_{j=1}^n \mu_j [\mathbf{q}_j + (\mathbf{q}_i - \mathbf{q}_j)^+] \\ = \lambda n \left(1 + \frac{2\mu_{\max}}{\alpha_n}\right) + n + 2\lambda n \mathbf{q}_i - 2 \sum_{j=1}^n \mu_j \max\{\mathbf{q}_j, \mathbf{q}_i\} \\ = \lambda n \left(1 + \frac{2\mu_{\max}}{\alpha_n}\right) + n + 2\lambda n \mathbf{q}_i - 2 \sum_{j=1}^n \mu_j [\mathbf{q}_i + (\mathbf{q}_j - \mathbf{q}_i)^+] \\ = \lambda n \left(1 + \frac{2\mu_{\max}}{\alpha_n}\right) + n + 2\lambda n \mathbf{q}_i - 2 \mathbf{q}_i \sum_{j=1}^n \mu_j - 2 \sum_{j=1}^n \mu_j (\mathbf{q}_j - \mathbf{q}_i)^+\end{aligned}$$

$$\begin{aligned}
&\stackrel{(*)}{=} \lambda n \left(1 + \frac{2\mu_{\max}}{\alpha_n}\right) + n - 2(1 - \lambda)n\mathbf{q}_i - 2 \sum_{j=1}^n \mu_j (\mathbf{q}_j - \mathbf{q}_i)^+ \\
&\leq \lambda n \left(1 + \frac{2\mu_{\max}}{\alpha_n}\right) + n - 2(1 - \lambda)n\mathbf{q}_i - 2\mu_{\min} \sum_{j=1}^n (\mathbf{q}_j - \mathbf{q}_i)^+ \\
&\leq \lambda n \left(1 + \frac{2\mu_{\max}}{\alpha_n}\right) + n - 2\min\{1 - \lambda, \mu_{\min}\} \sum_{j=1}^n [\mathbf{q}_i + (\mathbf{q}_j - \mathbf{q}_i)^+] \\
&= \lambda n \left(1 + \frac{2\mu_{\max}}{\alpha_n}\right) + n - 2\min\{1 - \lambda, \mu_{\min}\} \sum_{j=1}^n \max\{\mathbf{q}_i, \mathbf{q}_j\} \\
&\leq \lambda n \left(1 + \frac{2\mu_{\max}}{\alpha_n}\right) + n - 2\min\{1 - \lambda, \mu_{\min}\} \sum_{j=1}^n \mathbf{q}_j \\
&\leq -1,
\end{aligned}$$

where in equality (*) we used Equation (A.5), and in the last inequality we used the fact that $(\mathbf{q}, i) \notin F_n$ and the definition of the finite set F_n (Equation (A.2)). Then, the Foster–Lyapunov criterion (Foster [5]) implies the positive recurrence of the Markov chain $(\mathbf{Q}(\cdot), I(\cdot))$. Finally, because this is true for all server rates in Σ_n , we conclude that Σ_n is the stability region of the policy.

Appendix B. Proof of Theorem 2.2

Fix λ , and consider a vector of server rates in Σ_n , where $\lfloor n/2 \rfloor$ servers have rate $\epsilon_n > 0$. We will show that for any given λ and for all ϵ_n small enough, every resource-constrained dispatching policy that is weakly symmetric (i.e., satisfies Assumption 1) overloads the slow servers.

The high-level outline of the proof is as follows. In Subsection B.1, we show that under our weak symmetry assumption, the constraint on the number of bits available implies that the dispatcher treats all servers in a symmetric way, in some appropriate sense. Then, in Subsection B.2, we combine the results obtained in Subsection B.1 with the bound on the average message rate to show that jobs are sent to slow servers (i.e., to servers with service rate ϵ_n) with a positive rate that is bounded away from zero. This implies that the total workload of the servers diverges when ϵ_n is small enough, thus completing the proof.

In the proof that follows, we assume that the sequences c_n (memory size) and α_n (message rate) have been fixed and are of order $o(\log n)$ and $o(n^2)$, respectively.

B.1. Local Limitations of Symmetry and Finite Memory

In this subsection, we will show how the constraint of having only $o(\log(n))$ bits of memory affects the distribution of the sampled servers and the distribution of the dispatched jobs. The results that we provide are corollaries or special cases of results in Gamarnik et al. [7].

We first note that if the dispatcher has $o(\log(n))$ bits of memory and if n is large enough, then the distribution of the sampled servers is uniform among all sets of the same size.

Proposition B.1. *Let U be a uniform random variable over $[0, 1]$. For all n large enough, for every memory state $m \in \mathcal{M}_n$ and every possible job size $w \in \mathbb{R}_+$, the following holds. Consider any set of servers $S \in \mathcal{P}(\{1, \dots, n\})$ with $|S| \in o(n)$. Consider the event*

$$B(m, w; S) = \{f_1(m, w, U) = S \cup \{i\}, \text{ for some } i \notin S\},$$

and assume that the conditional probability measure

$$\mathbb{P}(\cdot \mid B(m, w; S))$$

is well defined. Then,

$$\mathbb{P}(j \in f_1(m, w, U) \mid B(m, w; S))$$

is the same for all $j \notin S$.

Proof. This is a special case of proposition 5.1 in Gamarnik et al. [7], noting that although the statement of that proposition requires $|S| \leq \sqrt{n}$, its proof goes through under the weaker assumption $|S| \in o(n)$. \square

Corollary B.1. *Let U be a uniform random variable over $[0, 1]$. For all n large enough, for every memory state $m \in \mathcal{M}_n$, for every possible job size $w \in \mathbb{R}_+$, and for any set of servers $S \in \mathcal{P}(\{1, \dots, n\})$ with $|S| \in o(n)$, we have*

$$\mathbb{P}(f_1(m, w, U) = S) = \mathbb{P}(f_1(m, w, U) = \sigma(S)),$$

for every permutation σ .

Proof. In order to simplify notation, we omit the dependence of f_1 on m and w . Let us fix a set S , and a transposition τ . If $\tau(S) = S$, then it is trivially true that

$$\mathbb{P}(f_1(U) = S) = \mathbb{P}(f_1(U) = \tau(S)).$$

On the other hand, if $\tau(S) \neq S$, then there exists some $i \in S$ such that $\tau(i) \notin S$. In that case, we have

$$\begin{aligned} \mathbb{P}(f_1(U) = S) &= \mathbb{P}(f_1(U) = S \mid |f_1(U)| = |S|) \mathbb{P}(|f_1(U)| = |S|) \\ &= \mathbb{P}(i \in f_1(U) \mid \{|f_1(U)| = |S|\} \cap \{S \setminus \{i\} \subset f_1(U)\}) \\ &\quad \cdot \mathbb{P}(S \setminus \{i\} \subset f_1(U) \mid |f_1(U)| = |S|) \mathbb{P}(|f_1(U)| = |S|) \\ &= \mathbb{P}(\tau(i) \in f_1(U) \mid \{|f_1(U)| = |S|\} \cap \{S \setminus \{i\} \subset f_1(U)\}) \\ &\quad \cdot \mathbb{P}(S \setminus \{i\} \subset f_1(U) \mid |f_1(U)| = |S|) \mathbb{P}(|f_1(U)| = |S|) \\ &= \mathbb{P}(f_1(U) = \tau(S)), \end{aligned}$$

where in the second to last equality, we used Proposition B.1.

Finally, because any permutation σ can be obtained as a sequence of transpositions, applying the previous argument iteratively yields

$$\mathbb{P}(f_1(U) = S) = \mathbb{P}(f_1(U) = \sigma(S)),$$

for every permutation σ . \square

Remark B.1. Although all sets of servers of the same size have the same probability of being sampled, the memory state and the incoming job size can influence the number of sampled servers.

Similarly, if the dispatcher has $o(\log(n))$ bits of memory, then the distribution of the destination of the incoming job is uniform (possibly zero) outside the set of sampled servers.

Proposition B.2. Let V be a uniform random variable over $[0, 1]$. For all n large enough and for every memory state $m \in \mathcal{M}_n$, every set of indices $S \in \mathcal{P}(\{1, \dots, n\})$ with $|S| \in o(n)$, every queue vector state $\mathbf{q} \in \mathcal{Q}^n$, every rate vector $\mu \in \mathbb{R}_+^n$, and every job size $w \in \mathbb{R}_+$, we have that

$$\mathbb{P}(f_2(m, w, \{(\mathbf{q}_i, \mu_i, i) : i \in S\}, V) = j)$$

is the same for all $j \notin S$.

Proof. This is a special case of proposition 5.2 in Gamarnik et al. [7].

B.2. High Arrival Rate to Slow Servers

In this subsection, we will leverage the results of the previous subsection to show that the total workload in the system diverges in time.

For every $t \geq 0$, let $\mathcal{W}^n(t)$ be the total remaining workload in the system at time t .

Lemma B.1. Fix some $\lambda > 0$, and suppose that the service rate of $\lfloor n/2 \rfloor$ servers is equal to some $\epsilon_n > 0$. Then, there exists a positive sequence $\{b_n(\lambda)\}_{n \geq 1}$, which is completely determined by λ (i.e., independent of ϵ_n) such that $b_n \in \Theta(e^{-\alpha_n/n})$, and

$$\liminf_{t \rightarrow \infty} \frac{\mathcal{W}^n(t)}{t} \geq [b_n(\lambda) - \epsilon_n]n, \quad \text{a.s.},$$

for all n large enough.

Proof. Let $\bar{A}_n(t)$ be the counting process of arrivals with a job size of at least $1/2$, and let us define

$$p_{1/2} \triangleq \mathbb{P}\left(W_1 \geq \frac{1}{2}\right).$$

Because the arrivals are modeled as a renewal process of rate λn , and the job sizes $\{W_k\}_{k=1}^\infty$ are i.i.d. with unit mean, it follows that $\bar{A}_n(t)$ is a renewal counting process with rate $\lambda n p_{1/2} > 0$. On the other hand, because the average message rate (see Equation (2)) is upper bounded by α_n almost surely, we have

$$\limsup_{t \rightarrow \infty} \frac{1}{t} \sum_{k=1}^{\bar{A}_n(t)} 2 |S_k| \leq \alpha_n, \quad \text{a.s.}$$

Combining this with the fact that

$$\limsup_{t \rightarrow \infty} \frac{1}{t} \sum_{k=1}^{\bar{A}_n(t)} 2 \left(\frac{\alpha_n}{\lambda n p_{1/2}} \right) \mathbb{1}_{\left\{ |S_k| > \frac{\alpha_n}{\lambda n p_{1/2}} \right\}} \leq \limsup_{t \rightarrow \infty} \frac{1}{t} \sum_{k=1}^{\bar{A}_n(t)} 2 |S_k|,$$

we obtain

$$\limsup_{t \rightarrow \infty} \frac{1}{t} \sum_{k=1}^{\bar{A}_n(t)} \mathbb{1}_{\left\{ |S_k| > \frac{\alpha_n}{\lambda n p_{1/2}} \right\}} \leq \frac{\lambda n p_{1/2}}{2}.$$

This in turn implies that

$$\begin{aligned} \liminf_{t \rightarrow \infty} \frac{1}{t} \sum_{k=1}^{\bar{A}_n(t)} \mathbb{1}_{\left\{ |S_k| \leq \frac{\alpha_n}{\lambda n p_{1/2}} \right\}} &= \liminf_{t \rightarrow \infty} \frac{1}{t} \sum_{k=1}^{\bar{A}_n(t)} \left(1 - \mathbb{1}_{\left\{ |S_k| > \frac{\alpha_n}{\lambda n p_{1/2}} \right\}} \right) \\ &\geq \liminf_{t \rightarrow \infty} \frac{\bar{A}_n(t)}{t} + \liminf_{t \rightarrow \infty} \frac{1}{t} \sum_{k=1}^{\bar{A}_n(t)} -\mathbb{1}_{\left\{ |S_k| > \frac{\alpha_n}{\lambda n p_{1/2}} \right\}} \\ &= \lambda n p_{1/2} - \limsup_{t \rightarrow \infty} \frac{1}{t} \sum_{k=1}^{\bar{A}_n(t)} \mathbb{1}_{\left\{ |S_k| > \frac{\alpha_n}{\lambda n p_{1/2}} \right\}} \\ &\geq \frac{\lambda n p_{1/2}}{2}, \quad \text{a.s.} \end{aligned} \tag{B.1}$$

Let $N_{\epsilon_n} \subset \{1, \dots, n\}$ be the set of servers with service rate ϵ_n , which was assumed to have cardinality $\lfloor n/2 \rfloor$. Let s be a non-negative integer upper bounded by $s_n^* \triangleq \alpha_n / \lambda n p_{1/2}$. Because $s_n^* \in o(n)$, Corollary B.1 applies, and we obtain

$$\begin{aligned} \mathbb{P}(S_k \subset N_{\epsilon_n} \mid |S_k| = s) &= \frac{\binom{\lfloor n/2 \rfloor}{s}}{\binom{n}{s}} \\ &= \frac{\lfloor n/2 \rfloor (\lfloor n/2 \rfloor - 1) \cdots (\lfloor n/2 \rfloor - s + 1)}{n(n-1) \cdots (n-s+1)} \\ &\geq \left(\frac{1}{3}\right)^s, \end{aligned}$$

for all n large enough, where in the last inequality we used that $s_n^* \in o(n)$. Because this is true for all s in the given range, we obtain

$$\mathbb{P}(S_k \subset N_{\epsilon_n} \mid |S_k| \leq s_n^*) \geq \left(\frac{1}{3}\right)^{s_n^*},$$

for all $k \geq 1$ and for all n large enough. Combining this with Equation (B.1), we obtain

$$\liminf_{t \rightarrow \infty} \frac{1}{t} \sum_{k=1}^{\bar{A}_n(t)} \mathbb{1}_{\{|S_k| \leq s_n^*, S_k \subset N_{\epsilon_n}\}} \geq \frac{\lambda n p_{1/2}}{2} \left(\frac{1}{3}\right)^{s_n^*}, \tag{B.2}$$

almost surely, for all n large enough.

Let us fix a particular set S that satisfies $|S| \leq s_n^*$, and $S \subset N_{\epsilon_n}$. For any such set, Proposition B.2 implies

$$\begin{aligned} \mathbb{P}(D_k \in N_{\epsilon_n} \mid S_k = S) &= \mathbb{P}(D_k \in N_{\epsilon_n} \mid D_k \in S, S_k = S) \mathbb{P}(D_k \in S \mid S_k = S) \\ &\quad + \mathbb{P}(D_k \in N_{\epsilon_n} \mid D_k \in S^c, S_k = S) \mathbb{P}(D_k \in S^c \mid S_k = S) \\ &= \mathbb{P}(D_k \in S \mid S_k = S) + \frac{|N_{\epsilon_n} \cap S^c|}{|S^c|} \mathbb{P}(D_k \in S^c \mid S_k = S) \\ &\geq \frac{|N_{\epsilon_n} \cap S^c|}{|S^c|} \\ &= \frac{\lfloor \frac{n}{2} \rfloor - |S|}{n - |S|} \\ &\geq \frac{\lfloor \frac{n}{2} \rfloor - s_n^*}{n} \\ &\geq \frac{1}{3}, \end{aligned}$$

for all n large enough, where in the last inequality we used that $s_n^* \in o(n)$. Because this is true for every set S with the given properties, we conclude that

$$\mathbb{P}(D_k \in N_{\epsilon_n} \mid S_k \subset N_{\epsilon_n}, |S_k| \leq s_n^*) \geq \frac{1}{3},$$

for all $k \geq 1$ and for all n large enough. Combining this with Equation (B.2), we obtain

$$\begin{aligned} \liminf_{t \rightarrow \infty} \frac{1}{t} \sum_{k=1}^{\bar{A}_n(t)} \mathbb{1}_{\{D_k \in N_{\epsilon_n}\}} &\geq \liminf_{t \rightarrow \infty} \frac{1}{t} \sum_{k=1}^{\bar{A}_n(t)} \mathbb{1}_{\{D_k \in N_{\epsilon_n}, |S_k| \leq s_n^*, S_k \subset N_{\epsilon_n}\}} \\ &\geq \frac{\lambda n p_{1/2}}{6} \left(\frac{1}{3} \right)^{s_n^*}, \quad \text{a.s.}, \end{aligned}$$

for all n large enough. Note that this is a lower bound on the average rate of arrival of jobs with size at least $1/2$ to the servers with service rate ϵ_n . On the other hand, those servers have a total processing rate of $\epsilon_n \lfloor n/2 \rfloor$ units of workload per unit of time. Then, because the total workload of the system is at least as much as the workload of the servers with rate ϵ_n , we have

$$\begin{aligned} \liminf_{t \rightarrow \infty} \frac{\mathcal{W}^n(t)}{t} &\geq \liminf_{t \rightarrow \infty} \frac{1}{t} \sum_{k=1}^{\bar{A}_n(t)} \frac{1}{2} \mathbb{1}_{\{D_k \in N_{\epsilon_n}\}} - \epsilon_n \left\lfloor \frac{n}{2} \right\rfloor \\ &\geq \left[\frac{\lambda p_{1/2}}{6} \left(\frac{1}{3} \right)^{s_n^*} - \epsilon_n \right] n, \end{aligned}$$

for all n large enough. This establishes the desired result, with $b_n(\lambda)$ equal to the first term in the bracketed expression above. \square

Lemma B.1 implies that for all n large enough, the total workload in the system increases at least linearly with time, as long as $\lfloor n/2 \rfloor$ of the servers have rate $\epsilon_n < b_n(\lambda)$. In particular, this will happen if $\epsilon_n \in O(e^{-\alpha_n/n})$.

Because the above is true for every weakly symmetric policy with $o(\log n)$ bits of memory, and with an average message rate upper bounded by $\alpha_n \in o(n^2)$ almost surely, it follows that for all n large enough, the stability region of any such policy is contained in a proper subset $\Gamma_n(\lambda, \alpha_n)$ of Σ_n that excludes service rate vectors for which $\lfloor n/2 \rfloor$ of the servers have rate $\epsilon_n < b_n(\lambda)$.

References

- [1] Anselmi J (2019) Combining size-based load balancing with round-robin for scalable low latency. *IEEE Trans. Parallel Distributed Systems* 31(4):886–896.
- [2] Anselmi J, Dufour F (2020) Power-of- d -choices with memory: Fluid limit and optimality. *Math. Oper. Res.* 45(3):862–888.
- [3] Atar R, Keslassy I, Mendelson G, Orda A, Vargaftik S (2020) Persistent-Idle load-distribution. *Stochastic Systems* 10(2):152–169.
- [4] Bramson M (2011) Stability of join the shortest queue network. *Ann. Appl. Probab.* 21(4):1568–1625.
- [5] Foster F (1953) On the stochastic matrices associated with certain queueing processes. *Ann. Math. Statist.* 24(3):355–360.
- [6] Gamarnik D, Tsitsiklis JN, Zubeldia M (2018) Delay, memory, and messaging tradeoffs in distributed service systems. *Stochastic Systems* 8(1):45–74.
- [7] Gamarnik D, Tsitsiklis JN, Zubeldia M (2020) A lower bound on the queueing delay in resource constrained load balancing. *Ann. Appl. Probab.* 30(2):870–901.
- [8] Lu Y, Xie Q, Kliot G, Geller A, Larus J, Greenberg A (2011) Join-Idle-Queue: A novel load balancing algorithm for dynamically scalable web services. *Performance Evaluation* 68(11):1056–1071.
- [9] Mitzenmacher M (1996) The power of two choices in randomized load balancing. Unpublished PhD thesis, University of California, Berkeley.
- [10] Mukherjee D, Borst S, van Leeuwen J, Whiting P (2016) Universality of power-of- d load balancing schemes. *Workshop Math. Perform. Model. Anal. ACM SIGMETRICS Performance Evaluation Rev.* 44(2):36–38.
- [11] Shah D, Prabhakar B (2002) The use of memory in randomized load balancing. *Proc. IEEE Internat. Sympos. Inform. Theory* (Institute of Electrical and Electronics Engineers, Piscataway, NJ).
- [12] Stolyar A (2015) Pull-based load distribution in large-scale heterogeneous service systems. *Queueing Systems* 80(4):341–361.
- [13] van der Boer M, Zubeldia M, Borst S (2020) Zero-wait load balancing with sparse messaging. *Oper. Res. Lett.* 48(3):368–375.
- [14] Vargaftik S, Keslassy I, Orda A (2020) LSQ: Load balancing in large-scale heterogeneous systems with multiple dispatchers. *IEEE/ACM Trans. Networking* 28(3):1186–1198.
- [15] Vvedenskaya ND, Dobrushin RL, Karpelevich FI (1996) Queueing system with selection of the shortest of two queues: An asymptotic approach. *Problems Inform. Transmission* 32(1):15–27.
- [16] Zhou X, Shroff N, Wierman A (2021) Asymptotically optimal load balancing in large-scale heterogeneous systems with multiple dispatchers. *ACM SIGMETRICS Performance Evaluation Rev.* 48(3):57–58.