Optimal hyper-scalable load balancing with a strict queue limit

Performance Evaluation 149–150 (2021) 102217

Contents lists available at ScienceDirect

Performance Evaluation

journal homepage: www.elsevier.com/locate/peva

Optimal hyper-scalable load balancing with a strict queue limit

Mark van der Boor [a,*], Sem Borst [a], Johan van Leeuwaarden [b,a]

[a] Eindhoven University of Technology, P.O. Box 513, 5600 MB, Eindhoven, The Netherlands
[b] Tilburg University, P.O. Box 90153, 5000 LE, Tilburg, The Netherlands

ARTICLE INFO

Article history:
Received 14 December 2020
Received in revised form 30 April 2021
Accepted 31 May 2021
Available online 7 June 2021

Keywords:
Load balancing
Hyper-scalable
Queueing networks
Throughput-optimal
Communication overhead

ABSTRACT

Load balancing plays a critical role in efficiently dispatching jobs in parallel-server systems such as cloud networks and data centers. A fundamental challenge in the design of load balancing algorithms is to achieve an optimal trade-off between delay performance and implementation overhead (e.g. communication or memory usage). This trade-off has primarily been studied so far from the angle of the amount of overhead required to achieve asymptotically optimal performance, particularly vanishing delay in large-scale systems. In contrast, in the present paper, we focus on an arbitrarily sparse communication budget, possibly well below the minimum requirement for vanishing delay, referred to as the hyper-scalable operating region. Furthermore, jobs may only be admitted when a specific limit on the queue position of the job can be guaranteed.

The centerpiece of our analysis is a universal upper bound for the achievable throughput of any dispatcher-driven algorithm for a given communication budget and queue limit. We also propose a specific hyper-scalable scheme which can operate at any given message rate and enforce any given queue limit, while allowing the server states to be captured via a closed product-form network, in which servers act as customers traversing various nodes. The product-form distribution is leveraged to prove that the bound is tight and that the proposed hyper-scalable scheme is throughput-optimal in a many-server regime given the communication and queue limit constraints. Extensive simulation experiments are conducted to illustrate the results.

1. Introduction

Load balancing provides a crucial mechanism for efficiently distributing jobs among servers in parallel-processing systems. Traditionally, the primary objective in load balancing has been to optimize performance in terms of queue lengths or delays. Due to the immense size of cloud networks and data centers [1–3], however, implementation overhead (e.g. communication or memory usage involved in obtaining or storing state information) has emerged as a further key concern in the design of load balancing algorithms. Indeed, the fundamental challenge in load balancing is to achieve scalability: providing favorable delay performance, while only requiring low implementation overhead in large-scale deployments.

The seminal paper [4] approached the above challenge by imposing the natural performance criterion that the probability of non-zero delay vanishes as the number of servers grows large. It was shown that this can only be achieved

* Corresponding author.
E-mail addresses: markvanderboor@hotmail.nl (M. van der Boor), s.c.borst@tue.nl (S. Borst), j.s.h.vanleeuwaarden@tilburguniversity.edu (J. van Leeuwaarden).

https://doi.org/10.1016/j.peva.2021.102217

with constant communication overhead per job when sufficient memory is available at the dispatcher. There are in fact schemes which achieve a vanishing delay probability with only one message per job [5–7] or even fewer [8], but these rely on server-initiated updates as opposed to dispatcher-driven probes. We defer a more extensive discussion of these papers and the broader literature to a later stage in this introduction.

In the present paper we pursue the same intrinsic trade-off between performance and communication overhead, but focus on the optimal performance for a potentially scarce communication budget, and our perspective is fundamentally different in two respects. First of all, we set the admissible message rate δ to be arbitrary, and in particular to be far lower than one message per job, which we refer as the 'hyper-scalable' operating regime. This range is especially relevant in scenarios with relatively tiny jobs and a correspondingly massive arrival rate which may significantly exceed the message rate that can be sustained between the dispatcher and the servers, prohibiting even just one message per job. Second, jobs may only be admitted when a strict limit K on the queue position of the job can be guaranteed. This queue limit K can have any value and is offered in systems of any size, as opposed to a zero queue length that is only ensured with high probability in a many-server regime. The combination of a low communication budget per job and a strict admission condition is particularly pertinent for high-volume packet processing applications, where zero delay may not be feasible given the admissible message rate, but where an explicit queue limit is crucial.

As the cornerstone of our analysis, we establish a universal upper bound for the achievable throughput of any dispatcher-driven algorithm as function of δ and K, thus capturing the trade-off between performance and communication overhead. We also introduce and analyze a specific hyper-scalable scheme which approaches the latter bound in a many-server regime, demonstrating that the bound is sharp.

Model set-up and hyper-scalable scheme. We adopt the set-up of the celebrated supermarket model which has emerged as the canonical framework in the related literature (as further reviewed below), but add several salient features relevant for our purposes. Specifically, we consider a system with N identical servers of unit exponential rate and a single dispatcher where jobs arrive as a Poisson process of rate $N\lambda$. The dispatcher is unaware of the service requirements of jobs and cannot buffer them, but must immediately forward them to one of the servers or block them. The throughput of the system is defined as the rate of admitted jobs per server.

The blocking option is relevant since the dispatcher must enforce an explicit queue limit K, and is only allowed to admit a job and assign it to a server if it can guarantee that the queue position encountered by that job is at most K. Note that it is not enough for a job to end up in such a position thanks to a lucky guess, but that the dispatcher must have absolute certainty in advance that this is the case, and that a job must be discarded otherwise. Discarding may be the preferred option in packet processing applications when handling a packet beyond a certain tolerance window serves no useful purpose. In that case, processing an obsolete packet results in an unnecessary resource wastage and needlessly contributes to further congestion, and is thus worse than simply dropping the packet upfront.

As mentioned above, the dispatcher is oblivious of the service requirements, which are exponentially distributed and thus have unbounded support. Hence, the dispatcher critically relies on information provided by the servers in order to enforce the queue limit K, and is allowed to send probes for this purpose, requesting queue length reports at a rate $N\delta$. In addition, the dispatcher is endowed with unlimited memory capacity, which it may use to determine which server to probe and when or to which server it will dispatch an arriving job. Servers return instantaneous queue length reports in response to probes from the dispatcher, but are not able to initiate messages or send unsolicited updates when reaching a certain status.

With the above framework in place, we will construct a specific hyper-scalable scheme which is guaranteed to enforce the queue limit K and operate within the communication budget δ. The scheme toggles each individual server between two modes of operation, labeled open and closed. An open period starts when the dispatcher requests a queue length update from the server and the reported queue length is below K; during that period the server is not working, and waits for incoming jobs from the dispatcher, seeing its queue only grow. Once the queue length reaches the limit K, a closed period starts, ending when the dispatcher requests the next update after τ time units; during that period the server is continuously working as long as jobs are available, without receiving any further jobs, thus draining its queue. When the queue length reported at an update is exactly K, the open period has length zero, and the next closed period starts immediately. By construction, the above-described mechanism maintains a queue limit of K at all times and induces a message rate of at most $1/\tau$ per server, which makes $\tau = 1/\delta$ the obvious choice.

Main contributions. The main contributions of the paper may be summarized as follows. First of all, we establish a universal upper bound $\lambda^*(\delta, K)$ for the achievable throughput of any dispatcher-driven algorithm subject to the communication budget per server in terms of δ and the queue limit K. The upper bound relies on a simple yet powerful argument which counts the number of jobs that can be admitted per message given the queue limit K and the message rate δ. While the macroscopic view of the argument covers a broad range of strategies with possibly dynamic and highly complex update rules, the nature of the upper bound strongly points to the superior properties of constant update intervals.

Armed with that insight, we propose a hyper-scalable scheme which can operate at any given message rate δ and enforce any given queue limit K. At the same time, the scheme is specifically designed to produce system dynamics that can be represented in terms of a closed product-form queueing network, in which the servers act as customers traversing various nodes. This furnishes tractable expressions for the relevant stationary distributions and in particular the blocking

probability. The expression for the blocking probability is used to prove that the achieved throughput approaches the minimum of the above-mentioned upper bound and the normalized job arrival rate λ in a many-server regime. This in turn demonstrates that the upper bound is tight and that the proposed hyper-scalable scheme provides optimality in the three-way trade-off among queue limit, communication and throughput.

Background on load balancing algorithms. Load balancing algorithms can be broadly categorized as static (open-loop), dynamic (closed-loop), or some intermediate blend, depending on the amount of state information (e.g. queue lengths or load measurements) that is used in dispatching jobs among servers. Within the category of dynamic policies, one can further distinguish between dispatcher-driven (push-based) and server-oriented (pull-based) approaches. In the former case, the dispatcher 'pushes' jobs to the servers and takes the initiative to collect state information for that purpose, while the servers play a passive role and only provide state information when explicitly requested. In contrast, in server-oriented approaches, the servers may pro-actively share state information with the dispatcher, and indirectly 'pull' in jobs by advertising their availability or load status. The use of state information naturally allows dynamic policies to achieve better performance, but also involves higher implementation complexity (e.g. communication overhead and memory usage) as mentioned earlier. The latter issue has emerged as a pivotal concern due to the deployment of large-scale cloud networks and data centers with immense numbers of servers handling massive amounts of service requests.

The celebrated Join-the-Shortest-Queue (JSQ) policy provides the gold standard in the category of dispatcher-driven algorithms and offers strong stochastic optimality properties. Specifically, in case of identical servers, exponentially distributed service requirements and a service discipline at each server that is oblivious to the actual service requirements, the JSQ policy achieves minimum mean delay among all non-anticipating policies [9,10]. In order to implement the JSQ policy, however, a dispatcher relies on instantaneous knowledge of the queue lengths at all the servers, which may involve a prohibitive communication burden, and may not be scalable. Related is the join-below-threshold scheme [11], which is throughput-optimal, but the dispatcher-driven variant is not scalable either.

The latter issue has spurred a strong interest in so-called JSQ(d) strategies, where the dispatcher assigns incoming jobs to a server with the shortest queue among d servers selected uniformly at random. This involves d message exchanges per job (assuming $d \geq 2$), and thus drastically reduces the communication overhead compared to the full JSQ policy when the number of servers N is large. At the same time, even a value as small as $d = 2$ yields significant performance improvements in the many-server regime $N \to \infty$ compared to purely random assignment ($d = 1$) [12,13]. This is commonly referred to as the "power-of-two" effect. Similar power-of-d effects have been demonstrated for heterogeneous servers, non-exponential service requirements and loss systems in [14–19].

Unfortunately, JSQ(d) strategies lack the ability of the conventional JSQ policy to achieve zero queueing delay as $N \to \infty$ for any finite value of d. In contrast, if d grew with N, making it possible to drive queueing delay to zero [20,21], the communication overhead would grow unboundedly. A noteworthy exception arises for batches of jobs when the value of d and the batch size grow suitably large, as can be deduced from results in [22]. Leaving batch arrivals aside though, it is in fact necessary for d to grow with N in order to achieve zero queueing delay, since results in the seminal paper [4] show that this is fundamentally impossible with a finite communication overhead per job, unless memory is available at the dispatcher to store state information.

The latter feature is exactly at the core of the so-called Join-the-Idle-Queue (JIQ) scheme [5,6], where servers advertise their availability by transferring a 'token' to the dispatcher whenever they become idle, thus generating at most one message per job. The dispatcher assigns incoming jobs to an idle server as long as tokens are outstanding, or to a uniformly at random selected server otherwise. Remarkably, the JIQ scheme has the ability of the full JSQ policy to drive the queueing delay to zero as $N \to \infty$, even for generally distributed service requirements [7,23].

Note that for no single value of d, a JSQ(d) strategy can rival the JIQ scheme which simultaneously provides low communication overhead and asymptotically optimal performance. As alluded to above, this superiority reflects the power of server-oriented approaches in conjunction with memory at the dispatcher. The value of memory in load balancing was already studied in [24,25] in a 'balls-and-bins' context. Related work in [26] examines how much load balancing performance degrades when delayed information is used. A framework for mean-field analysis for JSQ(d) strategies with memory is developed in [27]. The authors of [28] use mean-field limits to determine the minimum required value of d for JSQ(d) strategies with memory to achieve zero queueing delay. The possibilities with limited memories are explored in [29].

As described above, the main interest in the line of work sparked by [4] has focused on the amount of communication overhead and/or memory usage required to drive the queueing delay to zero in a many-server regime. While there are known schemes to achieve that with just one message per job, even that may still be prohibitive, especially when jobs do not involve big computational tasks, but tiny data packets which each require little processing. In such situations the sheer message exchange in providing queue length information may be disproportionate to the actual amount of processing required. While the overhead can be reduced, that only appears feasible for sufficiently low load [8], and it remains largely unknown what the best achievable performance is for a given communication budget below one message per job. Motivated by these issues, we focus on dispatcher-driven schemes that can operate at an arbitrarily low communication budget, and that can additionally enforce a specific queue limit for every admitted job. To the best of our knowledge, this hyper-scalable perspective has not been pursued before, with the exception of [30] which however does not consider explicit queue limits or optimality properties.

Finally, [31,32] consider a similar problem but in the case of multiple dispatchers, in which every dispatcher keeps a local estimate of all queue lengths and these estimates are updated infrequently. This is in fact closely related to the hyper-scalable algorithm in this paper, although the analyses and results differ, primarily because of the queue limit we impose.

The literature on load balancing algorithms has ballooned in recent years, and the above discussion provides a non-exhaustive cross-section with some of the classical paradigms and results most pertinent to the present paper. We refer to [33] for a more comprehensive survey discussing related job assignment mechanisms, further model extensions and alternative asymptotic regimes (e.g. heavy-traffic and non-degenerate slow down scalings).

Organization of the paper. The main results are presented in Section 2: the upper bound for the throughput and the analysis of the hyper-scalable scheme, using a closed queueing network. In Section 3 we provide simulation results to further illustrate the behavior of the hyper-scalable scheme. An extension of the hyper-scalable scheme that also aims to minimize queue lengths is introduced in Section 4. In Section 5 we establish product-form distributions for a general closed queueing network scenario which covers both the hyper-scalable scheme and the latter extension as special cases. We conclude with some remarks and suggestions for further research in Section 6.

2. Main results

In this section we discuss the main results, which can be summarized as follows. There is a function λ^* of δ and K, such that subject to a message rate δ and queue limit K,

- the throughput of any dispatcher-driven algorithm is bounded from above by $\min\{\lambda^*(\delta, K), \lambda\}$,
- the throughput of our hyper-scalable scheme approaches $\min\{\lambda^*(\delta, K), \lambda\}$ as $N \to \infty$.

It is worth observing that $\lambda^*(\delta, K)$ equals the product of δ and the expected number of jobs that would leave from a server with K jobs in queue over $1/\delta$ time. The hyper-scalable algorithm will heavily lean on this observation. These two results are covered in Sections 2.1 and 2.2, respectively.

2.1. Universal upper bound

We establish the upper bound for a slightly more general scenario with heterogeneous server speeds. Denote the speed of the nth server by μ_n for $n = 1, \ldots, N$. The next theorem shows that the achievable throughput of any dispatcher-driven algorithm subject to the message rate δ and queue limit K is bounded from above by

$$\lambda^*(\delta, K) = \delta M_K(\bar{\mu}/\delta),$$

with

$$M_K(\tau) = \sum_{k=0}^{K-1}(1 - \alpha_k(\tau)), \tag{1}$$

$\alpha_k(\tau) = \mathrm{e}^{-\tau}\sum_{i=0}^{k}\frac{\tau^i}{i!}$ and $\bar{\mu} = \frac{1}{N}\sum_{n=1}^{N}\mu_n$ denoting the system-wide average server speed. Note that $M_K(\tau)$ may be equivalently written as

$$M_K(\tau) = K - \sum_{k=0}^{K-1}(K-k)\mathrm{e}^{-\tau}\frac{\tau^k}{k!},$$

and may be interpreted as the expected value of the minimum of K and a Poisson distributed random variable with mean τ.

Theorem 1. *The expected number of jobs that any dispatcher-driven algorithm can admit subject to the queue limit K during a period of length T_0 with at most $\delta N T_0$ message exchanges cannot exceed $2KN + \lambda^*(\delta, K) \times NT_0$, for any $\delta > 0$. In particular, the achievable throughput with a message rate of at most $\delta > 0$ is bounded from above by $\lambda^*(\delta, K)$.*

Recall that we defined throughput as the rate of admitted jobs per server, and note that the throughput is naturally bounded from above by the normalized arrival rate λ.

Proof. As noted earlier, since the execution times are exponentially distributed and thus have unbounded support, the dispatcher relies on information provided by the servers in order to enforce the queue limit K. Specifically, the dispatcher earns 'passes' for admitting k jobs when a server reports $k = 0, \ldots, K$ service completions since the previous update, and cannot admit any job without relinquishing a pass. Thus, the number of jobs that the dispatcher can admit during a particular time period cannot exceed the sum of (i) the maximum possible number of KN passes initially available; (ii) the maximum possible number of KN passes earned at the first update from each server during that period, if any; and (iii) the number of additional passes obtained at further updates over intervals that fell entirely during that period, if any. Now

suppose that the dispatcher requests L_n queue length reports from the nth server during a period of length T_0, one after each of the update intervals of lengths $T_{n,1}, \ldots, T_{n,L_n}$, with $\sum_{l=1}^{L_n} T_{n,l} \leq T_0$ for all $n = 1, \ldots, N$ and $L = \sum_{n=1}^{N} L_n \leq \delta N T_0$. Then the number of passes earned at the lth update equals the number of service completions during the time interval $T_{n,l}$, which depends on the queue length at the start of that interval. However, this random variable is stochastically bounded from above by when the queue was full with K jobs at the start of the interval. In the latter case the number of passes earned is given by the minimum of K and a Poisson distributed random variable with parameter $\mu_n T_{n,l}$. We deduce that the expected total number of passes obtained at all these updates is bounded from above by

$$\sum_{n=1}^{N} \sum_{l=1}^{L_n} M_K(\mu_n T_{n,l}), \tag{2}$$

and to prove the first statement of the theorem it thus remains to be shown that this quantity is no larger than $\lambda^*(\delta, K) \times N T_0$. It is easily verified that

$$\frac{\partial^2 M_K(t)}{\partial t^2} = -\mathrm{e}^{-t} \frac{t^{K-1}}{(K-1)!} < 0,$$

implying that $M_K(t)$ is concave as function of t. As an aside, the above expression may be intuitively explained from the fact that the first derivative $\frac{\partial M_K(t)}{\partial t}$ equals the probability that exactly $K-1$ unit-rate Poisson events occur during a period of length t, while the (negative) derivative of the latter probability equals that very same probability by virtue of the Kolmogorov equations for a pure birth process. Because of concavity, we obtain that (2) is no larger than $L \times M_K(\tau)$, with

$$\tau = \frac{1}{L} \sum_{n=1}^{N} \mu_n \sum_{l=1}^{L_n} T_{n,l} \leq \frac{1}{L} \sum_{n=1}^{N} \mu_n T_0 = \bar{\mu} \frac{N T_0}{L}. \tag{3}$$

Invoking the fact that $\frac{\partial M_K(t)}{\partial t} > 0$, i.e., $M_K(t)$ is increasing in t, we may write

$$L \times M_K(\tau) \leq L \times M_K\left(\bar{\mu} \frac{N T_0}{L}\right) = \lambda^*(\gamma, K) \times N T_0, \tag{4}$$

with $\gamma = \frac{L}{N T_0} \leq \delta$. It is easily verified that

$$\frac{\partial \lambda^*(x, K)}{\partial x} = K - K\mathrm{e}^{-1/x} \sum_{k=0}^{K} \frac{(1/x)^k}{k!} > 0, \tag{5}$$

i.e., $\lambda^*(x, K)$ is increasing in x, and hence $\lambda^*(\gamma, K) \leq \lambda^*(\delta, K)$, which completes the proof of the first statement of the theorem.

Finally, to prove the second statement, we consider the long-term scenario $T_0 \to \infty$. The number of jobs that are admitted per time-unit per server then equals $(2KN + \lambda^*(\delta, K) \times N T_0)/(N T_0) \to \lambda^*(\delta, K)$ and the message rate per server equals at most $\delta N T_0/(N T_0) = \delta$. □

Properties of λ^*. We now state some properties of $\lambda^*(\delta, K)$ and discuss their consequences, where we assume without loss of generality that $\bar{\mu} = 1$. In the next subsection we will introduce a hyper-scalable scheme which is able to achieve this throughput in the many-server regime. For now, we will reflect the properties in light of the maximum throughput that is possible for any dispatcher-driven load balancing algorithm given a message rate δ.

Proposition 1. *$\lambda^*(\delta, K)$ has the following properties:*

(i) *$\lambda^*(\delta, K)$ is strictly increasing in both δ and K,*
(ii) *$\lambda^*(\delta, K) \uparrow 1$ as $\delta \to \infty$,*
(iii) *$\lambda^*(\delta, K) \downarrow 0$ and $\lambda^*(\delta, K)/\delta \to K$ as $\delta \downarrow 0$,*
(iv) *for $a \leq 1$, $\lambda^*(a/K, K) \to a$ as $K \to \infty$.*

Proof. $\lambda^*(\delta, K)$ is strictly increasing in δ because of (5) and is strictly increasing in K since $1 - \alpha_k(\tau) > 0$ in (1). For Properties (ii) to (iv), note that

$$\begin{aligned}
&\delta[K - K\mathrm{e}^{-1/\delta} - \mathrm{e}^{-1/\delta}((K-1)/\delta + (K-2)(1/\delta)^{y(\delta)})] \\
&\leq \delta(K - \mathrm{e}^{-1/\delta} \sum_{i=0}^{K-1} (K-i) \frac{(1/\delta)^i}{i!}) = \lambda^*(\delta, K) \leq \min(\delta K, 1),
\end{aligned}$$

with $y(\delta) = 2$ when $\delta \geq 1$ and $y(\delta) = K - 1$ when $\delta < 1$. All limiting statements are true for the LHS and RHS of the previous equation, therefore proving these properties for $\lambda^*(\delta, K)$ too. □

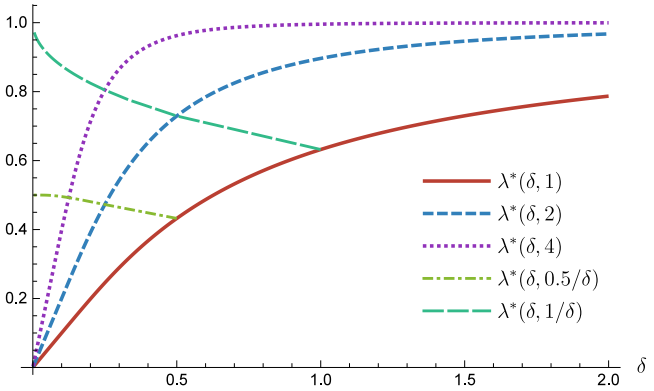


Fig. 1. Visualization of the throughput bound $\lambda^*(\delta, K)$ for various values of K as function of δ. For the fourth and fifth graph, only values of δ are evaluated for which the second argument is integer-valued.

The properties in Proposition 1 are visualized in Fig. 1. They can be interpreted intuitively and practically too. For Property (i), when the communication budget is expanded, i.e. δ is increased, more jobs can be dispatched to queues that are guaranteed to be short. Similarly, more jobs can be admitted into the system if the queue limit is raised, i.e., K is increased. Property (i), in conjunction with Theorem 1, implies that a throughput $\lambda^*(\delta, K)$ cannot be achieved with a message rate strictly below δ, or a queue limit strictly below K.

Property (ii) shows that as the message rate grows large, full server utilization can be achieved. With an unlimited message rate, the dispatcher is able to find idle servers immediately, a necessary requirement for achieving full server utilization irrespective of the queue limit K.

Property (iii) shows that, first, when no communication is allowed, no jobs can be sent to queues that are guaranteed to be short. The further specification of the limit indicates that K jobs are admitted into the system per message. This in turn reveals that when the communication is extremely infrequent, all messages result into finding an idle server, and thus provide the dispatcher with K passes to admit jobs.

Finally, Property (iv) is somewhat similar to Property (iii). When the queue limit K increases, one needs fewer messages in order to achieve a server utilization level a. With $a = 1$, Property (iv) shows that one message per K jobs is needed in order to achieve full server utilization, which is a somewhat similar conclusion as the one from Property (iii).

2.2. The hyper-scalable scheme

We now introduce the hyper-scalable scheme in full detail for the case of homogeneous servers.

At all times, the dispatcher remembers the most recent queue length that was reported by every server. Furthermore, the dispatcher records the number of jobs that have been sent to every server since the last update from that server. When the sum of these two numbers is strictly less than the queue limit K, a server is labeled open, and otherwise closed.

Whenever a job arrives to the dispatcher, it is assigned to an open server, if possible. There are two options for how to select an open server. Either an open server is selected uniformly at random (random case), or the open server that was interacted with (i.e. updated or received job) the longest ago is selected (FCFS case). The job is dropped when no open servers exist.

Exactly τ time units after a server was labeled closed, the dispatcher will request a queue length update of the server. The server becomes open when this queue length is strictly less than K, and the server remains closed for another τ time units if the queue length equals K, in which case the dispatcher will request the next queue length update after another τ time units. The hyper-scalable scheme is a dispatcher-driven algorithm, since only the dispatcher initiates messages and every server can track itself when it is labeled open by the dispatcher: exactly when the sum of the queue length during the latest update and the number of jobs received since then is strictly below K.

Note that by construction the hyper-scalable scheme respects the queue limit K at all times and involves a message rate of at most $1/\tau$. In addition, the scheme has been specifically designed to allow explicit analysis and derivation of provable capacity benchmarks. As it turns out, a crucial feature in that regard is for the servers to refrain from executing jobs while being marked open. This feature ensures that the queue length is exactly K at the moment a server becomes closed. The average number of job completions in an interval of length τ then equals $M_K(\tau)$, so one message leads to $M_K(\tau)$ admitted jobs on average, immediately yielding the following result.

Corollary 1. *The average number of messages per admitted job is equal to* $1/M_K(\tau)$*, regardless of* λ *and* N.

While the forced idling of servers during open periods may seem inefficient, the next theorem shows that the proposed hyper-scalable scheme is in fact throughput-optimal in large-scale systems, given the message rate δ and queue limit K, with the choice $\tau = 1/\delta$.

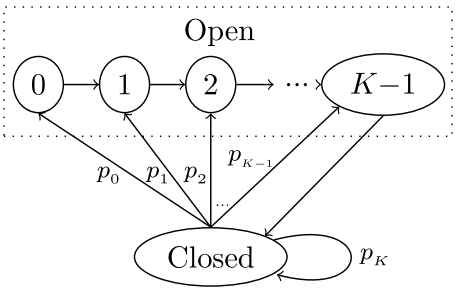


Fig. 2. Schematic representation of the circulation of an individual customer in the closed queueing network.

Theorem 2. *For any $\delta > 0$, the throughput achieved by the hyper-scalable scheme with $\tau = 1/\delta$ approaches* $\min\{\lambda^*(\delta, K), \lambda\}$ *as $N \to \infty$.*

Since the hyper-scalable scheme obeys the queue limit K and involves a message rate of at most δ, Theorems 1 and 2 combined imply that it is throughput-optimal as $N \to \infty$.

According to Theorem 1 and Property (i) of Proposition 1, one would require a message rate of at least δ to achieve a throughput of $\lambda^*(\delta, K)$. Theorem 2 shows that the throughput of the hyper-scalable scheme approaches $\lambda^*(\delta, K)$ as $N \to \infty$ when $\lambda \geq \lambda^*(\delta, K)$. A combination of these two observations (and the fact that $\lambda^*(\delta, K)$ is continuous in δ) indicates that the message rate of the hyper-scalable scheme must approach δ as $N \to \infty$ when $\lambda \geq \lambda^*(\delta, K)$. This in turn implies that the expected duration of an open period must become negligible, compare to the length τ of a closed period, i.e. the fraction of time that a server is marked open vanishes. This will also be shown numerically in Section 3.

We now proceed with an outline of the proof of Theorem 2.

Analysis. For brevity, a server is said to be in state k when the sum of the queue length at its latest update epoch and the number of jobs the server has received since, equals k. This means that all servers in state $k < K$ are labeled open and servers in state K are labeled closed. In view of the homogeneity of the servers, it is useful to further introduce $N(t) = (N_0(t), N_1(t), \ldots, N_{K-1}(t), N_K(t))$, with $\sum_k N_k(t) = N$, where $N_k(t)$ stands for the number of servers in state k at time t. While the vector $N(t)$ provides a convenient representation, it is worth emphasizing that it does not provide a Markovian state description.

We now explain how individual servers transition between various states. When a job is accepted into the system, the state of an open server will change from $k < K$ to $k + 1$. An update of a server may cause the server to change state too. The new state of the server equals the number of jobs that are left in queue after the update interval of τ time units. The number of jobs that were served follows a truncated Poisson distribution, so the probability p_k that exactly k jobs remain, equals $p_k := \mathrm{e}^{-\tau} \frac{\tau^{K-k}}{(K-k)!}$ for $k > 0$ and $p_0 := 1 - \mathrm{e}^{-\tau} \sum_{i=0}^{K-1} \frac{\tau^i}{i!}$. When $k < K$ jobs are left, the state of the server becomes k. When there are K jobs left, the state of the server does not change and remains K.

It is important to observe that service completions of jobs do not cause direct transitions in server states. The reason is twofold. When a server is open, it stops working on jobs, so there are no such completions at open servers. For closed servers, all servers are aggregated; the number of jobs in queue is not taken into account. Only after the period of length τ, the number of jobs left in queue is determined indirectly by using the transition probabilities as specified above.

Although the vector $N(t)$ does not provide a Markovian state description as noted above, its evolution can be described in terms of a closed queueing network, in which the servers act as customers in the network, traversing various nodes corresponding to their states. Specifically, the closed queueing network consists of one multi-class "single-server" node with service rate λN in which the customers can be of classes $0, 1, \ldots, K-1$, and one "infinite-server" node with deterministic service time τ that holds all class-K customers. A service completion at the single-server node makes one customer transition. The class of the customer changes from k to $k + 1$ if $k < K - 1$, or the customer transitions to the infinite-server node if its class was $K - 1$. When multiple customers are present at the single-server node, the customer that transitions is either selected uniformly at random (random case), or the customer that has been in the single-server node for the longest time is selected (FCFS case). Finally, upon a service completion at the infinite-server node a customer moves to the single-server node as class $k < K$ with probability p_k, or directly re-enters the infinite-server node with probability p_K.

A schematic representation is shown in Fig. 2. We define γ_k as the relative throughput value of class-k customers. With $\gamma_K = 1$, it follows that $\gamma_k = p_0 + \ldots + p_k = 1 - \alpha_{K-1-k}(\tau)$ for $k < K$.

By virtue of the above-described equivalence, the process $N(t)$ representing the server states under the hyper-scalable scheme inherits the product-form equilibrium distribution of the closed network as stated in the next proposition.

Proposition 2. *The equilibrium distribution of the system with N servers is*

$$\pi(n_0, n_1, \ldots, n_{K-1}, n_K) = G_N^{-1} \frac{(n_0 + \ldots + n_{K-1})!}{n_0! \ldots n_{K-1}!} \left(\prod_{i=0}^{K-1} \left(\frac{\gamma_i}{\lambda N} \right)^{n_i} \right) \frac{\tau^{n_K}}{n_K!} \tag{6}$$

if $n_0 + \ldots + n_K = N$, *with normalization constant*

$$G_N = \sum_{v_0+\ldots+v_{K-1}+w=N} \frac{(v_0+\ldots+v_{K-1})!}{v_0!\ldots v_{K-1}!} \left(\prod_{i=0}^{K-1} \left(\frac{\gamma_i}{\lambda N}\right)^{v_i}\right) \frac{\tau^w}{w!}.$$

A proof of Proposition 2 can be found in Section 5, and the product-form equilibrium distribution may be informally understood as follows. The infinite-server node allows a product-form distribution even for deterministic service times. While traditionally exponentially distributed service times are considered, the equilibrium distribution is insensitive to the service time distribution at the infinite-server node and only depends on its mean, see Section 5 for details. As mentioned above, the service discipline at the single-server node with exponentially distributed service times may either be FCFS or random order of service. In the case of the FCFS discipline, albeit not being reversible [34], the single-server node with multiple classes can be represented as an order-independent queue [35,36]. According to Theorem 2.2 in [36], the queue is quasi-reversible, which is sufficient for a product-form distribution. For random order of service, which is a symmetric service discipline, the single-server node is reversible, yielding a product-form as well.

The equilibrium distribution (6) can be simplified when only the number of open and closed servers matters. This immediately yields an expression for the blocking probability C_N as provided in the next corollary.

Corollary 2. *The equilibrium probability of there being n open servers and* $N - n$ *closed servers under the hyper-scalable scheme equals*

$$\pi(n, N-n) = \sum_{n_0+\ldots+n_{K-1}=n} \pi(n_0, \ldots, n_{K-1}, N-n) = \frac{\left(\frac{M_K(\tau)}{\lambda N}\right)^n \frac{\tau^{N-n}}{(N-n)!}}{\sum_{w=0}^{N} \left(\frac{M_K(\tau)}{\lambda N}\right)^w \frac{\tau^{N-w}}{(N-w)!}}. \tag{7}$$

In particular, because of the PASTA property, the blocking probability is given by

$$C_N = \pi(0, N) = \frac{\frac{(xN)^N}{N!}}{\sum_{w=0}^{N} \frac{(xN)^w}{w!}}, \tag{8}$$

with $x = \lambda\tau/M_K(\tau) = \lambda/\lambda^*(1/\tau, K)$. *Finally,*

$$C_N \overset{N\to\infty}{\to} \max\{0, 1 - \lambda^*(1/\tau, K)/\lambda\}.$$

Specifically, $C_N \downarrow 0$ *as* $N \to \infty$ *when* $\lambda \le \lambda^*(1/\tau, K)$.

Suppose that the allowed message rate is δ as stated in Theorem 2, then put $\tau = 1/\delta$. When $\lambda \le \lambda^*(1/\tau, K)$, the blocking probability vanishes in the many-server regime according to Corollary 2, and thus the throughput approaches λ. When $\lambda > \lambda^*(1/\tau, K)$, the acceptance probability tends to $\lambda^*(1/\tau, K)/\lambda$ and the throughput approaches $\lambda \times \lambda^*(1/\tau, K)/\lambda = \lambda^*(1/\tau, K)$. These two statements combined yield Theorem 2.

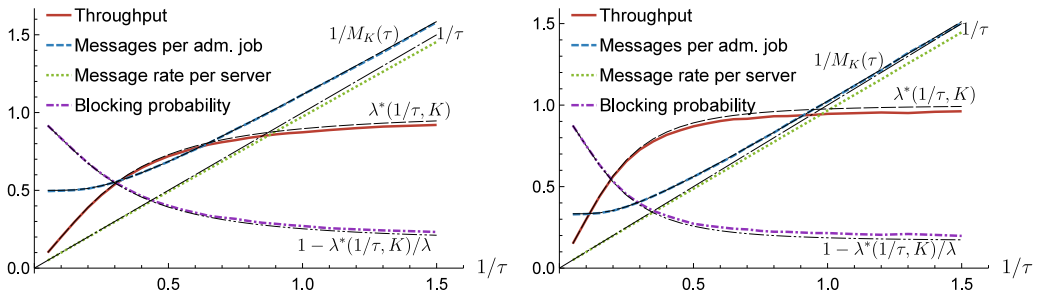
Theorem 2 allows us to equivalently view $\lambda^*(\delta, K)$ as the throughput that is achieved by the hyper-scalable scheme as $N \to \infty$ when $\lambda \ge \lambda^*(\delta, K)$. We now revisit properties (ii) and (iii) as stated in Proposition 1 from that perspective. In the limiting scenario $\delta \to \infty$, $\tau \downarrow 0$, servers are updated after an infinitesimally small time, which in turn alerts the dispatcher immediately when even a single job has been served. This ensures that all servers can work at full capacity.

In the scenario $\delta \downarrow 0$, $\tau \to \infty$, update periods become extremely long. Every update that does happen, will most likely find an idle server and allow for K admitted jobs, explaining why $\lambda^*(\delta, K) \approx K\delta$ for small δ.

Remark 1. Note that with the queue limit K in force we may assume each server to have a finite buffer of size K. In case of a finite buffer, the queue limit K would automatically be enforced, even if the dispatcher were allowed to forward jobs without any advance guarantee. With the option of "(semi)-blind guesses", where the dispatcher may even choose servers that are full (and a job will be discarded when it arrives at the full server), the throughput bound would trivially become 1 (the average server speed), and Property (iii) indicates that the achievable throughput $\lambda^*(\delta, K)$ without lucky guesses could be (substantially) lower when δ is (significantly) smaller than $1/K$. However the throughput of 1 can only be approached for a high arrival rate (much larger than the system capacity), in which severe blocking is unavoidable. In contrast, the hyper-scalable scheme does not need an arrival rate larger than the system capacity, and in fact only needs an arrival rate of $\lambda^*(\delta, K)$ to deliver the throughput $\lambda^*(\delta, K)$, thus having negligible blocking asymptotically.

3. Simulation experiments and optimality benchmarks

In this section we conduct various simulation experiments to further benchmark the properties of the hyper-scalable scheme and make several comparisons. Most results are for $K = 2$, yielding the throughput bound $\lambda^*(\delta, 2) = 2\delta - 2\delta e^{-1/\delta} - e^{-1/\delta}$ as function of the message rate δ. Furthermore, all simulation results emulate the random case, i.e. a job is sent to an open server selected uniformly at random.



(a) $K = 2$. (b) $K = 3$.

Fig. 3. Simulation results for the hyper-scalable scheme for $\lambda = 1.2$ and $N = 100$. Numerical values of the throughput bound $\lambda^*(1/\tau, K)$, the associated blocking probability bound $1 - \lambda^*(1/\tau, K)/\lambda$, the average number of messages per admitted job $1/M_K(\tau)$ and $1/\tau$ are also shown with thin black lines. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

3.1. Baseline version of the hyper-scalable scheme

First, we evaluate the hyper-scalable scheme itself in Figs. 3(a) and 3(b) for $K = 2$ and $K = 3$ respectively. We note that the message rate stays below the line $y = 1/\tau$, confirming that it never exceeds $1/\tau$. The throughput and blocking probability achieved by the hyper-scalable scheme are nearly indistinguishable from the respective asymptotic values (upper and lower bounds, respectively), especially at lower and medium values of the communication budget $1/\tau$. For higher values of the communication budget, the throughput and blocking probability slightly diverge from the asymptotic values but remain remarkably close nevertheless. This demonstrates that the asymptotic optimality properties of the hyper-scalable scheme as stated in Theorems 1 and 2 already manifest themselves in moderately large systems.

In order to provide further insight in the asymptotic optimality, we compare the baseline version of the hyper-scalable scheme with several variants and alternative scenarios that are not analytically tractable.

Specifically, in the next two subsections, we examine the following variants through simulations:

- "non-idling"; open servers continue working, but convey their queue length as if they had not been working while being open,
- "work-conserving"; open servers continue working and convey their actual queue lengths at update epochs.

At first sight, one might suspect that these variants achieve a possibly larger throughput. As we will see however, the differences are small and are only observable at low load (less than $\lambda^*(\delta, K)$) or in systems with few servers. When the arrival rate is large enough (larger than $\lambda^*(\delta, K)$) and the number of servers too, the fraction of servers that are open becomes negligible, and hence the difference between the baseline version and the variants vanishes.

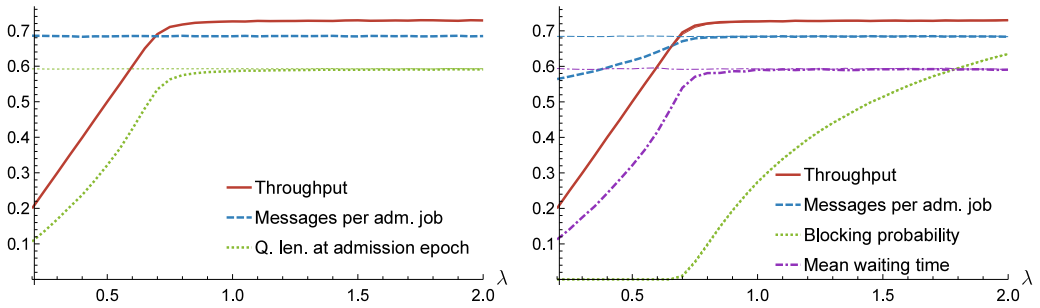
In Section 3.4 we make a comparison with the AUJSQ$^{det}(\delta)$ scheme considered in [30], which is not analytically tractable either but seems to be asymptotically throughput-optimal as well.

3.2. Non-idling variant

Open servers do not work on jobs in the baseline version of the hyper-scalable scheme. While Theorem 2 showed that the forced idling does not affect the achieved throughput in large-scale systems, it is still interesting to investigate the consequences of this design. In the non-idling variant, open servers do work on jobs, but they convey their queue length to the dispatcher as if they had not been working on jobs while being labeled open. While this variant may seem fundamentally different, the information that the dispatcher has is exactly the same as in the baseline version: the sets of open servers and their respective states coincide in both scenarios, as long as jobs are sent to the same open server.

In particular, the equilibrium distribution of the server states as provided in Proposition 2 applies to the non-idling variant as well. The throughput and the number of messages exchanged per admitted job are identical in both scenarios. The only difference arises in the expected queue length encountered by admitted jobs: they are somewhat smaller in the non-idling scenario. This is illustrated by the simulation results presented in Fig. 4(a), in which the expected queue length encountered by admitted jobs in the hyper-scalable algorithm (thin green line) is smaller than in its non-idling variant (thick green line).

At low load values, there are instants where there is time for servers to execute jobs when they are open. This causes a distinction between the two variants, since in the non-idling variant jobs join shorter queues. In Section 4, we will consider a tractable extension of the hyper-scalable scheme that aims to reduce the queue lengths. As the number of servers grows however, an overflow of arrivals will cause open servers to have less time to execute jobs, which causes the queue lengths to be similar in both scenarios. This viewpoint provides further intuition why the hyper-scalable scheme is still asymptotically optimal.



(a) Non-idling variant. (b) Work-conserving variant.

Fig. 4. Simulation results: mean values of the throughput, messages per job and queue position (and waiting time) are compared for the two variants (thick lines) and the baseline scenario (thin lines), for $K = 2$, $\tau = 2$ and $N = 500$, yielding a throughput bound $\lambda^*(1/2, 2) \approx 0.73$. Note that for the number of messages and throughput in Fig. 4(a) and the throughput and blocking probability in Fig. 4(b), the difference between the baseline scenario and the variant is indistinguishable. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

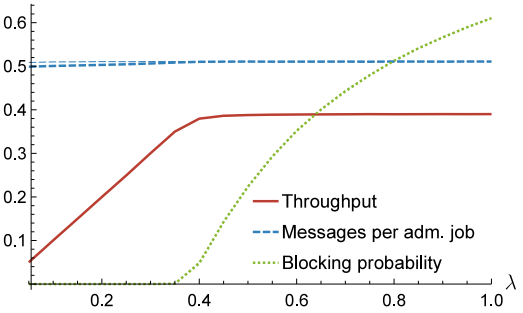


Fig. 5. Simulation results: comparison between the baseline scenario (thin lines) and the work-conserving variant (thick lines) for $K = 2$, $\tau = 5$ and $N = 500$, so that $\lambda^*(1/5, 2) \approx 0.39$.

3.3. Work-conserving variant

We now turn to a work-conserving variant of the hyper-scalable scheme, in which open servers also work on jobs, and in fact convey their actual queue length at an update epoch. In this case the evolution of the server states is different, and the equilibrium distribution provided in Proposition 2 no longer applies.

The throughput and blocking probability are similar in both scenarios. This may be intuitively explained as follows. When $\lambda \geq \lambda^*(1/2, 2)$, Theorem 2 shows that there are hardly ever any open servers, and hence there should not be any substantial difference between the two variants, which is corroborated by Fig. 4(b).

When $\lambda < \lambda^*(1/2, 2)$, there can be a significant number of open servers. Theorem 2 however implies that the hyper-scalable scheme approaches zero blocking and throughput λ in this case. While it is plausible that the work-conserving variant achieves that as well, as attested by Fig. 4(b), it is simply not feasible to achieve lower blocking or higher throughput. There is room for improvement in the number of message exchanges per admitted job, and Fig. 4(b) demonstrates that the work-conserving variant indeed provides some gain compared to the hyper-scalable scheme in that regard. To put that observation in perspective, consider Corollary 1. As one can see, the communication overhead is strictly decreasing in τ. For such a low arrival rate, the hyper-scalable scheme permits to choose the update interval τ much larger. Fig. 5 confirms that the choice $\tau = 5$ largely eliminates the difference in communication overhead between the work-conserving variant and the baseline version. Finally, note that the work-conserving variant achieves a significantly lower mean waiting time for these values of λ, since servers no longer idle unnecessarily when the arrival rate is low.

We finally show some auxiliary results in Figs. 6 and 7. Fig. 6 shows the average fraction of servers that are open. There is a slight difference between the baseline scenario and the work-conserving variant, but it is most important to observe that the fraction of open servers is large when λ is small (this fraction is close to $1 - \lambda$). When λ (and N) become large, the fraction of servers that are open tends to zero. In Fig. 7 we show the mean sojourn time of the baseline scenario and the two variants. The mean sojourn time of the baseline scenario grows indefinitely as $\lambda \downarrow 0$, because in this regime, servers are waiting for long times until the next job arrives, and in this time period they are not processing the jobs in queue. The two variants perform significantly better. There is no visible difference between the two variants, which may be explained by noting that when the arrival rate is low, it does not matter that servers are unnecessarily idling, while if the arrival rate is larger, servers are almost never idling because they receive jobs at a high rate.

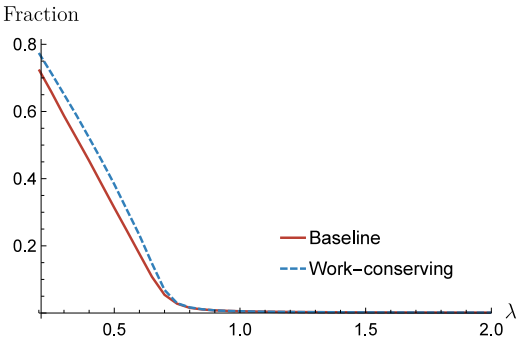


Fig. 6. Simulation results: the average fraction of servers that are open, for $K = 2$, $\tau = 2$ and $N = 500$, so that $\lambda^*(1/2, 2) \approx 0.90$.

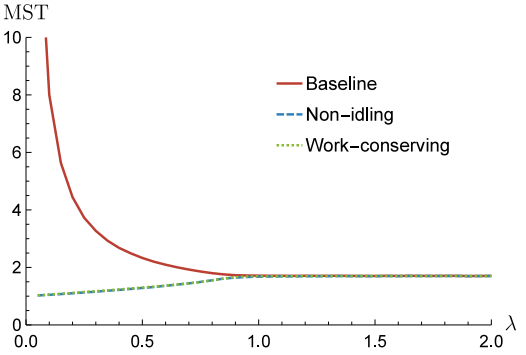


Fig. 7. Simulation results: the mean sojourn time of jobs in the baseline, non-idling and work-conserving scenarios for $K = 2$, $\tau = 2$ and $N = 500$, so that $\lambda^*(1/2, 2) \approx 0.90$.

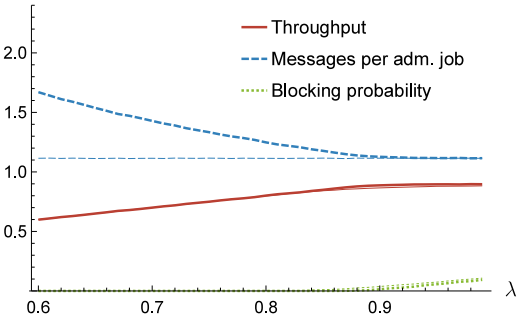
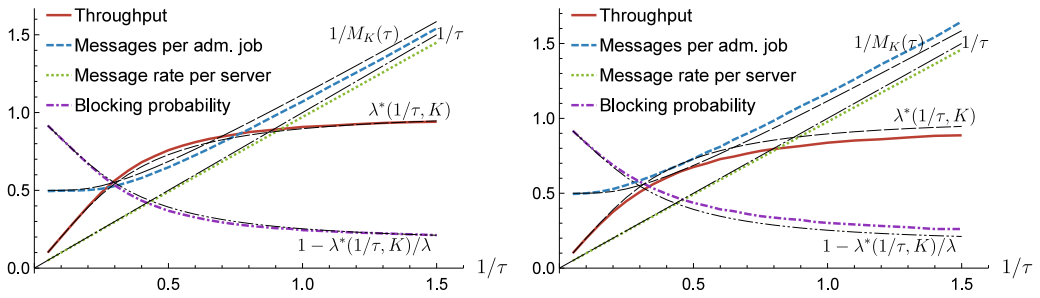


Fig. 8. Simulation results: comparison between the baseline scenario (thin lines) and the AUJSQ$^{det}(\delta)$ scheme (thick lines) for $K = 2$, $\tau = 1$ and $N = 500$, so that $\lambda^*(1, 2) \approx 0.90$.

3.4. Comparison with the AUJSQ$^{det}(\delta)$ *scheme*

We now compare the hyper-scalable scheme with the AUJSQ$^{det}(\delta)$ scheme [30], in which the dispatcher forwards incoming jobs to a server with the lowest queue estimate, and this estimate for a server is updated *exactly* every $\tau = 1/\delta$ time units based on a timer. Thus the AUJSQ$^{det}(\delta)$ scheme might update servers even when they are known to have strictly less than $K = 2$ jobs in queue. In contrast to [30], we consider a variant of the AUJSQ$^{det}(\delta)$ scheme in which jobs are blocked when the dispatcher is not aware of any servers that are guaranteed to have strictly less than $K = 2$ jobs in queue. The comparison is shown in Fig. 8.

It is important to note that in the hyper-scalable scheme the expected number of messages *per admitted job* is independent of λ, while in the AUJSQ$^{det}(\delta)$ scheme the expected number of messages *per time unit* is independent of λ. We observe that the average number of messages per admitted job coincides when $\lambda > \lambda^*(1/\tau, K)$. While it is natural to expect that the AUJSQ$^{det}(\delta)$ scheme offers similar asymptotic optimality properties, it lacks the mathematical tractability of the hyper-scalable scheme to facilitate a rigorous proof argument.



(a) Gamma(2, 2). (b) Gamma(1/2, 1/2).

Fig. 9. Simulation results for the hyper-scalable scheme for $K = 2$, $\lambda = 1.2$ and $N = 100$, and non-exponential service times. Numerical values of the throughput bound $\lambda^*(1/\tau, K)$, the associated blocking probability bound $1 - \lambda^*(1/\tau, K)/\lambda$, the average number of messages per admitted job $1/M_K(\tau)$ and $1/\tau$ are also shown with thin black lines. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

3.5. Non-exponential service times

We conclude our simulation experiments with analyzing the hyper-scalable scheme for non-exponential service time distributions. In Fig. 9(a), the service times are Gamma(2,2) distributed. The throughput of the hyper-scalable algorithm slightly exceeds $\lambda^*(1/\tau, K)$, the maximum throughput when job sizes are exponential. The number of messages per admitted job is also lower than $1/M_K(\tau)$. This is all explained by the fact that the tail of the Gamma(2,2) distribution is smaller than the tail of the exponential distribution. This means that more jobs are completed in a fixed time interval, which increases the effectiveness of the messages sent. The service time distribution in Fig. 9(b) is Gamma(1/2,1/2). The opposite effect is observed: the throughput is lower compared to Fig. 3(a) and the message rate is larger, because of the heavier tail.

4. Extension aimed at minimizing queue lengths

While the hyper-scalable scheme is asymptotically throughput-optimal given the message rate δ and queue limit K, it does not make any explicit effort beyond that to minimize queue lengths or delays experienced by jobs. Motivated by that observation, we now consider an extension of the hyper-scalable scheme aimed at minimizing waiting times. In this extension, a server that receives its ith job after an update at which its queue length was j, becomes closed for $\tau_{j,j+i}$ time units. After this time, it becomes open if $j + i < K$ and it is updated if $j + i = K$. Thus, servers are not only closed when they become full, but are closed for a while after every job they receive.

Henceforth, we focus on the case $K = 2$ for the ease of exposition, and we set $\tau_{0,0} = 0$, $\tau_{0,1} = \tau_{1,1} = \tau_1$, $\tau_{0,2} = \tau_{1,2} = \tau_2$ and $\tau_{2,2} = \tau_3$. We can put $\tau_{0,0}$ to zero without loss of generality as it makes no sense to have a cool-down period for an empty server. As a consequence there is no difference between servers that had zero jobs or one job at the previous update epoch, so we can set $\tau_{0,1} = \tau_{1,1}$, and $\tau_{0,2} = \tau_{1,2}$ as well. Let p_{2j} be the probability that j jobs remain after an update, when there were zero or one jobs just after the latest update epoch. This means that the server had $\tau_{0,1}$ time units to work on the first job and another $\tau_{0,2}$ time units after both jobs were dispatched to it. This gives $p_{20} = e^{-\tau_1}(1-\tau_2 e^{-\tau_2} - e^{-\tau_2}) + (1-e^{-\tau_1})(1-e^{-\tau_2})$, $p_{22} = e^{-\tau_1} e^{-\tau_2}$ and $p_{21} = 1 - p_{20} - p_{22}$. Let q_{2j} be the probability that j jobs remain after an update, when there were two jobs just after the latest update epoch. This gives $q_{20} = 1 - e^{-\tau_3} - \tau_3 e^{-\tau_3}$, $q_{22} = e^{-\tau_3}$ and $q_{21} = 1 - q_{20} - q_{22}$.

Servers can be in either of the five following states.

- A_1 The server is idle and open.
- B_1 The server had zero jobs during the previous update moment and received one job since, or the server had one job during the previous update moment and received no jobs since. The server is now marked closed for τ_1 time units.
- A_2 The server had zero jobs during the previous update moment and received one job since, or the server had one job during the previous update moment and received no jobs since. The server was marked closed for τ_1 but is now open.
- B_2 The server had zero jobs during the previous update moment and received two jobs since, or the server had one job during the previous update moment and received one job since. The server is now marked closed for τ_2 time units.
- B_3 The server had two jobs during the previous update moment and is now marked closed for τ_3 time units.

The transitions are schematically represented in Fig. 10, with the transition probabilities as defined earlier.

The system dynamics under this extension of the hyper-scalable scheme can also be represented in terms of a closed queueing network with one single-server node that holds two classes of customers and three infinite-server nodes. The

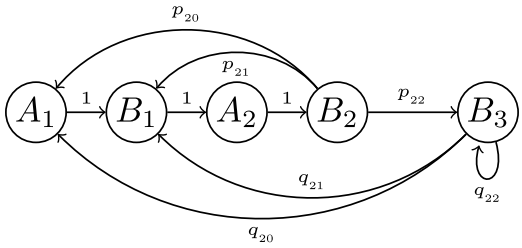


Fig. 10. Schematic representation of the server states and transitions when $K = 2$.

states A_1 and A_2 correspond to the two classes that customers can be of when they are present at the single-server node. The states B_1, B_2 and B_3 each correspond to one of the three infinite-server nodes in the network, with deterministic service times τ_1, τ_2 and τ_3, respectively.

Proposition 3. *The equilibrium distribution of the system with N servers is*

$$\pi(n_1, n_2, m_1, m_2, m_3) = H_N^{-1} \frac{(n_1+n_2)!}{n_1!n_2!} \left(\frac{\gamma_1}{\lambda N}\right)^{n_1} \left(\frac{\gamma_2}{\lambda N}\right)^{n_2} \frac{(\kappa_1\tau_1)^{m_1}}{m_1!} \frac{(\kappa_2\tau_2)^{m_2}}{m_2!} \frac{(\kappa_3\tau_3)^{m_3}}{m_3!} \tag{9}$$

if $n_1 + n_2 + m_1 + m_2 + m_3 = N$, *with* $(\gamma_1, \gamma_2, \kappa_1, \kappa_2, \kappa_3) = (p_{20} + \frac{p_{22}q_{20}}{1-q_{22}}, 1, 1, 1, \frac{p_{22}}{1-q_{22}})$ *and normalization constant* $H_N =$

$$\sum_{v_1+v_2+w_1+w_2+w_3=N} \frac{(v_1+v_2)!}{v_1!v_2!} \left(\frac{\gamma_1}{\lambda N}\right)^{v_1} \left(\frac{\gamma_2}{\lambda N}\right)^{v_2} \frac{(\kappa_1\tau_1)^{w_1}}{w_1!} \frac{(\kappa_2\tau_2)^{w_2}}{w_2!} \frac{(\kappa_3\tau_3)^{w_3}}{w_3!},$$

where n_i *is the number of open servers in state* A_i *and* m_i *the number of closed servers in state* B_i.

The vector $(\gamma_1, \gamma_2, \kappa_1, \kappa_2, \kappa_3)$ represents the vector of relative throughputs of the various states. The proof of Proposition 3 is provided in Section 5.

The equilibrium distribution (9) can be simplified when only the number of open and closed servers are counted, as shown in the next corollary.

Corollary 3.

- *The equilibrium probability of there being n open servers and* $N - n$ *closed servers under the extension equals*

$$\pi(n, N-n) = \frac{\left(\frac{\gamma_1+\gamma_2}{\lambda N}\right)^n \frac{(\kappa_1\tau_1+\kappa_2\tau_2+\kappa_3\tau_3)^{N-n}}{(N-n)!}}{\sum_{w=0}^{N} \left(\frac{\gamma_1+\gamma_2}{\lambda N}\right)^w \frac{(\kappa_1\tau_1+\kappa_2\tau_2+\kappa_3\tau_3)^{N-w}}{(N-w)!}}.$$

  In particular, because of the PASTA property, the blocking probability is given by

$$\pi(0, N) = \frac{\frac{(xN)^N}{N!}}{\sum_{w=0}^{N} \frac{(xN)^w}{w!}},$$

  with $x = \lambda \times \frac{\kappa_1\tau_1+\kappa_2\tau_2+\kappa_3\tau_3}{\gamma_1+\gamma_2}$, *and* $\pi(0, N) \to \max\{0, 1 - \lambda^*(\tau_1, \tau_2, \tau_3)/\lambda\}$ *as* $N \to \infty$, *which equals zero when* $\lambda \le \lambda^*(\tau_1, \tau_2, \tau_3) := \frac{\gamma_1+\gamma_2}{\kappa_1\tau_1+\kappa_2\tau_2+\kappa_3\tau_3}$.
- *The average number of messages per admitted job equals*

$$u(\tau_1, \tau_2, \tau_3) := \frac{\kappa_2 + \kappa_3}{\gamma_1 + \gamma_2}.$$

- *The average queue position of an admitted job equals*

$$q(\tau_1, \tau_2, \tau_3) := \frac{e^{-\tau_1}}{\gamma_1 + \gamma_2}.$$

The last two statements follow directly from the relative throughput values. These exact expressions for the maximum throughput λ^*, the average number of updates per admitted job u and the average queue position q of admitted jobs, allow us to evaluate the performance of this extension.

In Fig. 11(a), the value of τ_1 is varied while the values of τ_2 and τ_3 are kept constant. Since τ_1 represents the time that a server is closed when it has one job in queue, the result is that the second job that is sent to the server experiences a

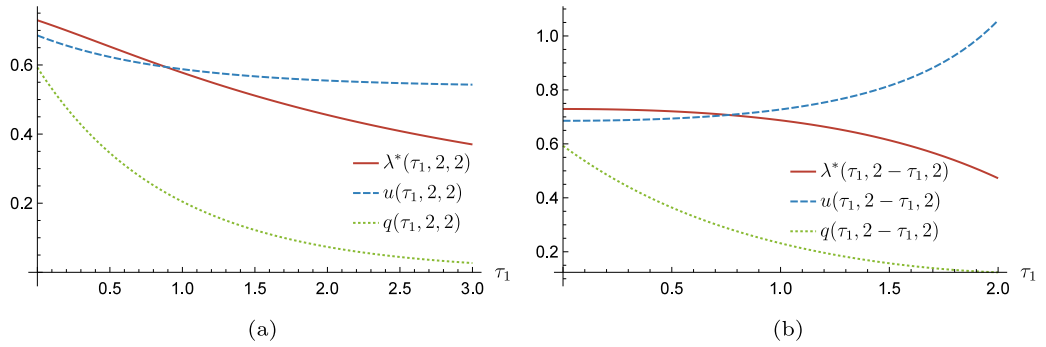


Fig. 11. Maximum throughput λ^*, average number of updates per admitted job u and average queue position of admitted jobs q as a function of τ_1.

Table 1
Throughput a, average number of messages per job u and average queue position of arrivals q for various schemes.

$K = 2$, $\lambda = 0.4$	a	u	q
Baseline scenario, $\tau = 5$	0.390567	0.512076	0.508626
Extension, $\tau_1 = 1$, $\tau_2 = 4$ and $\tau_3 = 5$	0.389605	0.51334	0.187575
Extension, $\tau_1 = 2$, $\tau_2 = 3$ and $\tau_3 = 5$	0.384692	0.519896	0.0698862
Benchmark scheme A	0.358974	N/A	0.285714
Benchmark scheme B	0.336	N/A	0.285714

shorter queue in expectation. Indeed, for larger values of τ_1, the mean experienced queue length q decreases. As a further benefit, the mean number of updates decreases as well, since an idle server will take at least $\tau_1 + \tau_2$ time units to be updated. The penalty incurred for these advantages is that the maximum throughput, λ^*, drops below the value of $\lambda^*(\delta, K)$ as asymptotically achieved by the baseline version of the hyper-scalable scheme, since servers may become idle during the τ_1 time in which they will not receive any more jobs.

Finally, in Fig. 11(b) we show that a trade-off between the parameters is possible. τ_1 is increased while τ_2 is decreased, and this leads to interesting behavior. Around the point $\tau_1 = 0$, the values of λ^* and u do not change when the parameters are altered, but the value of q does change. Such a trade-off might be worth it in scenarios where mean queue lengths play an important role.

4.1. Numerical example

We will now compare the throughput, average number of messages per job and the average queue position of arriving jobs of the baseline scenario of Section 2, the extension from Section 4 and two benchmark schemes that are derived from random routing. In benchmark scheme A, a server is selected uniformly at random for an incoming job. When this server has a queue length strictly less than K, the job is sent to this server, and is blocked otherwise. The throughput of this scheme equals $\lambda \frac{1+\lambda}{1+\lambda+\lambda^2}$ and the average queue position equals $\frac{\lambda}{1+\lambda}$. In benchmark scheme B, a server is selected uniformly at random for an incoming job and is accepted regardless of the queue length at the server. However, for the throughput, only jobs are counted that were added when the queue length was strictly below K. The throughput of this scheme equals $\lambda((1-\lambda)+\lambda(1-\lambda))$ and the average queue position equals $\frac{\lambda}{1+\lambda}$ (only counting jobs below the queue limit K). Note that the mean sojourn time may be computed using the average queue position of arriving jobs, assuming the non-idling variant is applied.

Corollary 3 is used to generate Table 1, in which one can see that the throughputs of the hyper-scalable scheme and its extension are higher than the two benchmark schemes. However, it is more interesting to compare the baseline scenario with the two extension schemes. While the throughput and number of messages are nearly equal, the average queue position of arriving jobs becomes extremely low when the parameters of the extension are chosen properly. This in turn implies that the sojourn time of jobs is also much lower. This behavior was already visible in Fig. 11(b): the queue position can decrease rapidly without hurting the other two performance measure too much.

4.2. General analysis

We will now discuss a method to calculate the throughput, average number of messages and average queue position of an admitted job for any value of K. We will use $K = 2$ as example for conciseness, but this idea can easily be extended. Fig. 2 will be helpful for the analysis. In the analysis, we will consider one cycle from state A_1 to A_1, and compute the expected time of such a cycle, the average number of admitted jobs, average number of updates and the average queue

position. We will exploit the following observation: since a server stops working when it is open and waiting for incoming jobs, one can assume that the time stops for this server at this time. Consequently, one can assume that a server receives a job immediately after becoming open. This method yields the results as in Corollary 3.

Let $t(S)$ be the expected time (excluding time the server is waiting for jobs) for a server to reach state A_1, given that it is in state S now. The expected cycle length then equals $t(A_1) = 0$. This gives rise to the following set of equations:

$$\begin{aligned} t(B_1) &= &&+t(A_1),\\ t(A_2) &= \tau_1 &&+t(B_1),\\ t(A_2) &= &&+t(B_2),\\ t(B_2) &= \tau_2 &&+p_{21}\times t(B_1)+p_{22}\times t(B_3),\\ t(B_3) &= \tau_3 &&+q_{21}\times t(B_1)+q_{22}\times t(B_3). \end{aligned} \tag{10}$$

Similarly, the average number of arrivals per cycle may be computed by replacing $(0, \tau_1, 0, \tau_2, \tau_3)$ (those are the symbols read vertically after the $=$-symbol in (10)) by $(1, 1, 0, 0, 0)$, the average number of updates by replacing it with $(0, 0, 0, 1, 1)$ and the average queue position by replacing it with $(0, e^{-\tau_1}, 0, 0, 0)$.

In the notation of Corollary 3, one would obtain λ^* by dividing the average number of arrivals per cycle by the average time of a cycle. The average number of messages per admitted job equals the fraction of the average number of updates per cycle divided by the average number of arrivals per cycle. Finally, the average queue position equals the average queue position per cycle divided by the average number of arrivals per cycle.

5. Closed queueing network and further proofs

In this section we establish product-form distributions for a general closed queueing network scenario which captures the network representations of the hyper-scalable scheme and the extension considered in the previous section as special cases. This provides the proofs of Propositions 2 and 3.

The closed queueing network consists of N customers circulating among one single-server node and B infinite-server nodes. Customers can be of A classes while at the single-server node, denoted by $\mathcal{A}_1, \ldots, \mathcal{A}_A$. Denote the infinite-server nodes by $\mathcal{B}_1, \ldots, \mathcal{B}_B$. The routing probabilities are denoted by $p_{x\to y}$; this is the probability that a customer transitions from x to y (x and y may correspond to either a class or an infinite-server node).

Service completions at the multi-class single-server node occur at an exponential rate λN. The customer that completes service is either selected uniformly at random, or in a FCFS manner, where the next customer is the one that transitioned last. If the selected customer is of class i, then it immediately returns to the single-server node as a class-j customer with probability $p_{\mathcal{A}_i\to\mathcal{A}_j}$ or it moves to node $\mathcal{B}_j$ with probability $p_{\mathcal{A}_i\to\mathcal{B}_j}$. The service times at the infinite-server node $\mathcal{B}_i$ are deterministic and equal to τ_i. Upon completing service at node $\mathcal{B}_i$, a customer moves to the single-server node as a class-j customer with probability $p_{\mathcal{B}_i\to\mathcal{A}_j}$, or to node $\mathcal{B}_j$ with probability $p_{\mathcal{B}_i\to\mathcal{B}_j}$.

The relative throughput values may be calculated from the traffic equations,

$$\begin{cases} \gamma_i = \sum_{j=1}^{A} p_{\mathcal{A}_j\to\mathcal{A}_i}\times\gamma_j + \sum_{j=1}^{B} p_{\mathcal{B}_j\to\mathcal{A}_i}\times\kappa_j,\\ \kappa_i = \sum_{j=1}^{A} p_{\mathcal{A}_j\to\mathcal{B}_i}\times\gamma_j + \sum_{j=1}^{B} p_{\mathcal{B}_j\to\mathcal{B}_i}\times\kappa_j, \end{cases}$$

where γ_i stands for the relative throughput of class $\mathcal{A}_i$ at the single-server node and κ_i for the relative throughput at node $\mathcal{B}_i$. We assume a "single-chain network", where the routing probability matrix is irreducible, meaning that all customers can reach all classes and nodes.

Proposition 4.

(a) The equilibrium distribution of the system with N customers is

$$\begin{aligned} &\pi(n_1, n_2, \ldots, n_A, m_1, m_2, \ldots, m_B)\\ &= F_N^{-1}\frac{(n_1+\ldots+n_A)!}{n_1!\cdots n_A!}\prod_{i=1}^{A}\left(\frac{\gamma_i}{\lambda N}\right)^{n_i}\prod_{j=1}^{B}\frac{(\kappa_j\tau_j)^{m_j}}{m_j!} \end{aligned} \tag{11}$$

if $n_1+\ldots+n_A+m_1+\ldots+m_B = N$, with normalization constant

$$F_N = \sum_{v_1+\ldots+v_A+w_1+\ldots+w_B=N}\frac{(v_1+\ldots+v_A)!}{v_1!\cdots v_A!}\prod_{i=1}^{A}\left(\frac{\gamma_i}{\lambda N}\right)^{v_i}\prod_{j=1}^{B}\frac{(\kappa_j\tau_j)^{w_j}}{w_j!},$$

where n_i is the number of customers of class $\mathcal{A}_i$ at the single-server node and m_j the number of customers at infinite-server node $\mathcal{B}_j$.

(b) *The equilibrium probability of there being n customers at the single-server node and $N-n$ customers in total at all the infinite-server nodes equals*

$$\pi(n, N-n) = \sum_{\substack{n_1+\ldots+n_A=n \\ m_1+\ldots+m_B=N-n}} \pi(n_1, \ldots, n_A, m_1, \ldots, m_B) = \frac{\left(\frac{\sum_{i=1}^{A}\gamma_i}{\lambda N}\right)^n \frac{\left(\sum_{j=1}^{B}\kappa_j\tau_j\right)^{N-n}}{(N-n)!}}{\sum_{w=0}^{N}\left(\frac{\sum_{i=1}^{A}\gamma_i}{\lambda N}\right)^w \frac{\left(\sum_{j=1}^{B}\kappa_j\tau_j\right)^{N-w}}{(N-w)!}}. \tag{12}$$

In particular, with $R = \frac{\gamma_1+\ldots+\gamma_A}{\kappa_1\tau_1+\ldots+\kappa_B\tau_B}$ and $x = \lambda/R$, because of the PASTA property, the probability that no customer resides at the single-server node is

$$\pi(0, N) = \frac{\frac{(xN)^N}{N!}}{\sum_{w=0}^{N}\frac{(xN)^w}{w!}},$$

and $\pi(0, N) \to \max\{0, 1-R/\lambda\}$ as $N \to \infty$ which equals zero when $\lambda \le R$.

In order to prove Proposition 4, we will verify that the equilibrium distribution (11) satisfies the balance equations of the closed queueing network.

5.1. Proof of Proposition 4

In order to verify the balance equations, we may assume that the service times of the infinite-server nodes are exponentially distributed even though in our closed queueing network, the service times are deterministic. This is because the equilibrium distribution (11) is insensitive to the service time distribution of nodes and only depends on the means of them (see Chapter 3 of [36] for a further discussion on this).

To see this, consider one infinite-server node D with exponential service rate μ_D and throughput value κ_D. This node adds the term

$$\frac{(\kappa_D/\mu_D)^d}{d!} \tag{13}$$

to the product-form equilibrium distribution, representing the presence of d customers in the infinite-server node. We now replace this infinite-server node by a series of infinite-server nodes, denoted by $D_1, \ldots, D_M$, each with an exponential service rate $M\mu_D$. The transition probabilities are altered in such a way that every transition previously to node D, now transitions to node D_1 instead. Customers then transition from node D_i to D_{i+1} for $i = 1, \ldots, D-1$ with probability one. Finally, any transition previously from node D, will now transition from node D_M. This construction makes every customer stay in this collection of nodes for M exponentially distributed phases, which is an Erlang$(M, M\mu)$ distributed random variable. All other throughput values in the network remain equal.

The throughput values of all these nodes will be equal to κ_D (since they are in series). Finally, similarly to the simplification of (11) to (12), all nodes $D_1, \ldots, D_M$ may be aggregated, which would lead to a term

$$\frac{(\sum_{i=1}^{M}\kappa_D/(M\mu_D))^d}{d!} = \frac{(\kappa_D/\mu_D)^d}{d!}$$

in the equilibrium probability, representing the presence of d customers in total in the infinite-server nodes $D_1, \ldots, D_M$. Note that the term in the RHS exactly matches the term (13), that appears when the node D has an exponentially distributed service time. This shows that the equilibrium distribution does not change when an exponential node is replaced by an Erlang$(M, M\mu)$ node, for any integer M, which can also be verified by substitution in the balance equations. Of course, each infinite-server node B_i with $\mu_i = 1/\tau_i$ can be replaced by such an Erlang distribution using this construction.

Because an Erlang$(M, M\mu)$ random variable converges to a deterministic quantity $1/\mu$ as M tends to infinity, this indicates that the equilibrium distribution also holds with infinite-server nodes that have deterministic service times. In fact, the node D may be replaced by any phase-type distribution, and every distribution may be approximated arbitrarily closely by phase-type distributions, implying that the equilibrium distribution in (11) in fact holds for generally distributed service times with mean τ_i at the infinite-server node B_i as well, although that is not directly relevant for our purposes.

We will now verify that (11) indeed solves the balance equations for the random order of service case, and we will use $\mu_i = 1/\tau_i$, representing the rates of the infinite-server nodes. The proof for the FCFS case is quite similar, but involves a more detailed state representation, and is deferred to Appendix.

Proof of part (a) — ROS. For conciseness, denote by (a, b) the vector $(a_1, \ldots, a_A, b_1, \ldots, b_B)$ and by e_i the ith unit vector.

Note that (11) is a proper distribution by definition. Since the equilibrium distribution is unique, it suffices to verify that (11) satisfies the following set of balance equations:

$$\begin{aligned}
&\left(\mathbb{1}_{\{a_1+\ldots+a_A>0\}}\lambda N + b_1\mu_1 + \ldots + b_B\mu_B\right)\pi(a,b)\\
&= \sum_{i=1}^{A}\sum_{j=1}^{A}\mathbb{1}_{\{a_j>0\}}p_{\mathcal{A}_i\to\mathcal{A}_j}\frac{a_i+\mathbb{1}_{\{i\neq j\}}}{a_1+\ldots+a_A}\lambda N\pi(a+e_i-e_j,b)\\
&+ \sum_{i=1}^{A}\sum_{j=1}^{B}\mathbb{1}_{\{b_j>0\}}p_{\mathcal{A}_i\to\mathcal{B}_j}\frac{a_i+1}{a_1+\ldots+a_A+1}\lambda N\pi(a+e_i,b-e_j)\\
&+ \sum_{i=1}^{B}\sum_{j=1}^{A}\mathbb{1}_{\{a_j>0\}}p_{\mathcal{B}_i\to\mathcal{A}_j}(b_i+1)\mu_i\pi(a-e_j,b+e_i)\\
&+ \sum_{i=1}^{B}\sum_{j=1}^{B}\mathbb{1}_{\{b_j>0\}}p_{\mathcal{B}_i\to\mathcal{B}_j}(b_i+\mathbb{1}_{\{i\neq j\}})\mu_i\pi(a,b+e_i-e_j).
\end{aligned}$$

The first line of the RHS refers to transitions where a customer at the single-server node transitions to the same node and may change class. The second line refers to transitions from the single-server node to one of the infinite-server nodes. Lines three and four correspond to transitions from a infinite-server node, to the single-server node or to another infinite-server node, respectively.

We will show that (11) satisfies the balance equations. By using the definition of (11) in the RHS, we obtain

$$\begin{aligned}
&\sum_{i=1}^{A}\sum_{j=1}^{A}\mathbb{1}_{\{a_j>0\}}p_{\mathcal{A}_i\to\mathcal{A}_j}\frac{a_i+1}{a_1+\ldots+a_A}\lambda N\pi(a,b)\frac{a_j}{a_i+1}\frac{\gamma_i}{\lambda N}\frac{\lambda N}{\gamma_j}\\
&+\sum_{i=1}^{A}\sum_{j=1}^{B}\mathbb{1}_{\{b_j>0\}}p_{\mathcal{A}_i\to\mathcal{B}_j}\frac{a_i+1}{a_1+\ldots+a_A+1}\lambda N\pi(a,b)\frac{a_1+\ldots+a_A+1}{a_i+1}\frac{\gamma_i}{\lambda N}\frac{b_j}{\kappa_j\tau_j}\\
&+\sum_{i=1}^{B}\sum_{j=1}^{A}\mathbb{1}_{\{a_j>0\}}p_{\mathcal{B}_i\to\mathcal{A}_j}(b_i+1)\mu_i\pi(a,b)\frac{a_j}{a_1+\ldots+a_A}\frac{\lambda N}{\gamma_j}\frac{\kappa_i\tau_i}{b_i+1}\\
&+\sum_{i=1}^{B}\sum_{j=1}^{B}\mathbb{1}_{\{b_j>0\}}p_{\mathcal{B}_i\to\mathcal{B}_j}(b_i+1)\mu_i\pi(a,b)\frac{\kappa_i/\mu_i}{b_i+1}\frac{b_j}{\kappa_j/\mu_j}.
\end{aligned}$$

Next, we combine the inside sums, resulting in

$$\begin{aligned}
&\sum_{j=1}^{A}\mathbb{1}_{\{a_j>0\}}\frac{a_j}{a_1+\ldots+a_A}\frac{\lambda N}{\gamma_j}\left[\sum_{i=1}^{A}p_{\mathcal{A}_i\to\mathcal{A}_j}\gamma_i+\sum_{i=1}^{B}p_{\mathcal{B}_i\to\mathcal{A}_j}\kappa_i\right]\pi(a,b)\\
&+\sum_{j=1}^{B}\mathbb{1}_{\{b_j>0\}}\frac{b_j}{\kappa_j/\mu_j}\left[\sum_{i=1}^{A}p_{\mathcal{A}_i\to\mathcal{B}_j}\gamma_i+\sum_{i=1}^{B}p_{\mathcal{B}_i\to\mathcal{B}_j}\kappa_i\right]\pi(a,b)\\
&=\left[\sum_{j=1}^{A}\mathbb{1}_{\{a_j>0\}}\frac{a_j}{a_1+\ldots+a_A}\lambda N+\sum_{j=1}^{B}b_j\mu_j\right]\pi(a,b)\\
&=\left(\mathbb{1}_{\{a_1+\ldots+a_A>0\}}\lambda N+\sum_{j=1}^{B}b_j\mu_j\right)\pi(a,b). \quad \square
\end{aligned}$$

6. Conclusion

We established a universal upper bound for the achievable throughput of any dispatcher-driven algorithm for a given communication budget and queue limit. We also introduced a specific hyper-scalable scheme which can operate at any given message rate and enforce any given queue limit, while allowing the system dynamics to be captured via a closed product-form network. We leveraged the product-form distribution to show that the bound is tight, and that the proposed hyper-scalable scheme provides asymptotic optimality in the three-way trade-off among performance, communication and throughput. Extensive simulation experiments were presented to illustrate the results and make comparisons with various alternative design options.

The work-conserving variant covered in Section 3.3 is especially worth discussing further. Intuitively, letting servers work all the time seems better than pausing the servers when they become open, but this remains to be rigorously proven.

The extension aimed at minimizing waiting times that was introduced in Section 4 warrants further attention as well. For the baseline scenario, we were able to prove a strict relationship between the amount of communication and the throughput. Likewise, there might exist a result, similar in spirit to Theorem 1, which provides an upper bound for the throughput *and* the average queue position of admitted jobs, given a certain communication budget. The main point of concern in this regard is that the concavity argument no longer seems to hold.

Finally, it would be worth investigating whether the current framework could be broadened further. It may be possible for example to extend the category of algorithms considered, specifically allowing for pull-based schemes. While the results in [8] imply that Theorem 1 does not hold for pull-based schemes, there might be a larger upper bound covering such algorithms as well. For further extensions, other performance metrics might be considered too, such as the mean waiting time as opposed to the throughput subject to a queue limit.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work is supported by the Nederlandse Organisatie voor Wetenschappelijk Onderzoek (NWO), Netherlands Gravitation Networks grant 024.002.003 and an ERC Starting Grant, Netherlands. We would like to thank Céline Comte and Martin Zubeldia for several helpful discussions and suggestions.

Appendix. Proof of Proposition 4 — FCFS case

Proof of part (a) — FCFS. The proof for the FCFS case consists of multiple steps. First we define a more detailed state space. A state is represented by $(c, b) = ((c_1, \ldots, c_m), (b_1, \ldots, b_B))$, which represents the situation where m customers are at the single-server node, and the order of the classes of customers is saved as well: the kth customer at the single-server node has class c_k. We will sometimes refer to the number of customers of a specific class with $a_i = \sum_j \mathbb{1}_{\{c_j = A_i\}}$. Furthermore, b_i customers are at the infinite-server node B_i.

Equilibrium distribution for the extended state space. We will show that the equilibrium distribution (modulo normalization constant) of state (c, b) equals

$$\tilde{\pi}(c, b) = \left(\frac{\gamma_1}{\lambda N}\right)^{a_1} \cdots \left(\frac{\gamma_A}{\lambda N}\right)^{a_A} \frac{(\kappa_1/\mu_1)^{b_1}}{b_1!} \cdots \frac{(\kappa_B/\mu_B)^{b_B}}{b_B!} \tag{A.1}$$

with a_k the number of customers of class A_k.

We assume FCFS arrivals of customers at the single-server node: customers arrive at the end of the line at the single-server node and only the customer first in line is able to transition.

Balance equations. First, we introduce the balance equations, in which the symbol m is used to denote the length of vector c,

$$\begin{aligned}
&\left(\mathbb{1}_{\{m>0\}}\lambda N + b_1\mu_1 + \ldots + b_B\mu_B\right)\tilde{\pi}(c, b) \\
&= \sum_{i=1}^{A} \mathbb{1}_{\{m>0\}} p_{A_i \to c_m} \lambda N \tilde{\pi}((A_i, c_1, \ldots, c_{m-1}), b) \\
&+ \sum_{i=1}^{A}\sum_{j=1}^{B} \mathbb{1}_{\{b_j>0\}} p_{A_i \to B_j} \lambda N \tilde{\pi}((A_i, c_1, \ldots, c_m), b - e_j) \\
&+ \sum_{i=1}^{B} \mathbb{1}_{\{m>0\}} p_{B_i \to c_m} (b_i + 1)\mu_i \tilde{\pi}((c_1, \ldots, c_{m-1}), b + e_i) \\
&+ \sum_{i=1}^{B}\sum_{j=1}^{B} \mathbb{1}_{\{b_j>0\}} p_{B_i \to B_j} (b_i + \mathbb{1}_{\{i\neq j\}})\mu_i \tilde{\pi}(c, b + e_i - e_j).
\end{aligned} \tag{A.2}$$

The term before $\pi(c, b)$ on the LHS represents the outgoing rate of state (c, b), which equals a rate of λN for the single-server node (if at least one customer is present there) plus a rate of $b_j\mu_j$, for each infinite-server node B_j.

On the RHS, four possible transitions to state (c, b) are shown preceded by the rate of the transitions. First, a transition from the non-empty single-server node makes the then first customer change its class from c_{m-1} to class c_m. If the previous

class order at the single-server node is $c_m - 1, c_1, \ldots, c_{m-1}$, then a transition to that node will make the class order exactly c. Second, if the previous class order at the single-server node is $K - 1, c_1, \ldots, c_m$, then a transition from that node to a infinite-server node will make the class order exactly c. Additionally, if the number of customers at infinite-server node B_j was $b_j - 1$, then it will become b_j as the infinite-server node receives an extra customer. Third, any of the customers at the infinite-server nodes might transition to the single-server node. Finally, customers might transition from and to one of the infinite-server nodes.

We will show that $\tilde{\pi}$ satisfies the balance equations. By using the definition of $\tilde{\pi}$ in the RHS, we obtain

$$\begin{aligned}
&\sum_{i=1}^{A} \mathbb{1}_{\{m>0\}} p_{A_i \to c_m} \lambda N \tilde{\pi}(c, b) \frac{\gamma_i}{\lambda N} \frac{\lambda N}{\gamma_{c_m}} \\
&+ \sum_{i=1}^{A} \sum_{j=1}^{B} \mathbb{1}_{\{b_j>0\}} p_{A_i \to B_j} \lambda N \tilde{\pi}(c, b) \frac{\gamma_i}{\lambda N} \frac{b_j}{\kappa_j \tau_j} \\
&+ \sum_{i=1}^{B} \mathbb{1}_{\{m>0\}} p_{B_i \to c_m} (b_i + 1) \mu_i \tilde{\pi}(c, b) \frac{\lambda N}{\gamma_{c_m}} \frac{\kappa_i/\mu_i}{b_i + 1} \\
&+ \sum_{i=1}^{B} \sum_{j=1}^{B} \mathbb{1}_{\{b_j>0\}} p_{B_i \to B_j} (b_i + 1) \mu_i \tilde{\pi}(c, b) \frac{\kappa_i/\mu_i}{b_i + 1} \frac{b_j}{\kappa_j/\mu_j}.
\end{aligned} \tag{A.3}$$

Next, we reorganize terms, yielding

$$\begin{aligned}
&\mathbb{1}_{\{m>0\}} \frac{\lambda N}{\gamma_{c_m}} \tilde{\pi}(c, b) \left[\sum_{i=1}^{A} p_{A_i \to c_m} \gamma_i + \sum_{i=1}^{B} p_{B_i \to c_m} \kappa_i \right] \\
&+ \sum_{j=1}^{B} \mathbb{1}_{\{b_j>0\}} \frac{b_j}{\kappa_j/\mu_j} \tilde{\pi}(c, b) \left[\sum_{i=1}^{A} p_{A_i \to B_j} \gamma_i + \sum_{i=1}^{B} p_{B_i \to B_j} \kappa_i \right] \\
&= \left[\mathbb{1}_{\{m>0\}} \lambda N + \sum_{j=1}^{B} b_j \mu_j \right] \tilde{\pi}(c, b),
\end{aligned} \tag{A.4}$$

which shows that $\tilde{\pi}$ is the equilibrium distribution of the extended state space.

Finally, note that in the original state space, only the number of customers of certain classes is tracked. Thus, $\pi(a, b)$ is an enumeration of $\tilde{\pi}(c, b)$ over all possible orders with the correct number of customers of a certain class. The number of possible orders is $\binom{a_1+\ldots+a_A}{a_1 \ldots a_A}$, which leads to $\pi(a, b) = \binom{a_1+\ldots+a_A}{a_1 \ldots a_A} \pi(c, b)$; the description of π as presented in the statement of the proposition. □

References

[1] R. Gandhi, H.H. Liu, Y.C. Hu, G. Lu, J. Padhye, L. Yuan, M. Zhang, Duet: Cloud scale load balancing with hardware and software, ACM SIGCOMM Comput. Commun. Rev. 44 (4) (2014) 27–38.
[2] S.T. Maguluri, R. Srikant, L. Ying, Stochastic models of load balancing and scheduling in cloud computing clusters, in: 2012 IEEE Conference on Computer Communications, INFOCOM, IEEE, 2012, pp. 702–710.
[3] P. Patel, D. Bansal, L. Yuan, A. Murthy, A. Greenberg, D.A. Maltz, R. Kern, H. Kumar, M. Zikos, H. Wu, K. Changhoon, N. Karri, Ananta: Cloud scale load balancing, ACM SIGCOMM Comput. Commun. Rev. 43 (4) (2013) 207–218.
[4] D. Gamarnik, J.N. Tsitsiklis, M. Zubeldia, Delay, memory, and messaging tradeoffs in distributed service systems, ACM SIGMETRICS Perform. Eval. Rev. 44 (1) (2016) 1–12.
[5] R. Badonnel, M. Burgess, Dynamic pull-based load balancing for autonomic servers, in: Network Operations and Management Symposium, 2008. NOMS 2008. IEEE, IEEE, 2008, pp. 751–754.
[6] Y. Lu, Q. Xie, G. Kliot, A. Geller, J.R. Larus, A. Greenberg, Join-Idle-Queue: A novel load balancing algorithm for dynamically scalable web services, Perform. Eval. 68 (11) (2011) 1056–1071.
[7] A.L. Stolyar, Pull-based load distribution in large-scale heterogeneous service systems, Queueing Syst. 80 (4) (2015) 341–361.
[8] M. van der Boor, M. Zubeldia, S.C. Borst, Zero-wait load balancing with sparse messaging, Oper. Res. Lett. 48 (3) (2020) 368–375.
[9] A. Ephremides, P. Varaiya, J. Walrand, A simple dynamic routing problem, IEEE Trans. Automat. Control 25 (4) (1980) 690–693.
[10] W. Winston, Optimality of the shortest line discipline, J. Appl. Probab. 14 (1) (1977) 181–189.
[11] Xingyu Zhou, Fei Wu, Jian Tan, Yin Sun, Ness Shroff, Designing low-complexity heavy-traffic delay-optimal load balancing schemes: Theory to algorithms, Proc. ACM Meas. Anal. Comput. Syst. 1 (2) (2017) 39.
[12] M. Mitzenmacher, The power of two choices in randomized load balancing, IEEE Trans. Parallel Distrib. Syst. 12 (10) (2001) 1094–1104.
[13] N.D. Vvedenskaya, R.L. Dobrushin, F.I. Karpelevich, Queueing system with selection of the shortest of two queues: An asymptotic approach, Probl. Pereda. Inf. 32 (1) (1996) 20–34.
[14] M. Bramson, Y. Lu, B. Prabhakar, Randomized load balancing with general service time distributions, ACM SIGMETRICS Perform. Eval. Rev. 38 (1) (2010) 275–286.
[15] M. Bramson, Y. Lu, B. Prabhakar, Asymptotic independence of queues under randomized load balancing, Queueing Syst. 71 (3) (2012) 247–292.
[16] A. Mukhopadhyay, A. Karthik, R.R. Mazumdar, Randomized assignment of jobs to servers in heterogeneous clusters of shared servers for low delay, Stoch. Syst. 6 (1) (2016) 90–131.

[17] A. Mukhopadhyay, A. Karthik, R.R. Mazumdar, F. Guillemin, Mean field and propagation of chaos in multi-class heterogeneous loss models, Perform. Eval. 91 (2015) 117–131.
[18] A. Mukhopadhyay, R.R. Mazumdar, Randomized routing schemes for large processor sharing systems with multiple service rates, ACM SIGMETRICS Perform. Eval. Rev. 42 (1) (2014) 555–556.
[19] Q. Xie, X. Dong, Y. Lu, R. Srikant, Power of d choices for large-scale bin packing: A loss model, ACM SIGMETRICS Perform. Eval. Rev. 43 (1) (2015) 321–334.
[20] D. Mukherjee, S.C. Borst, J.S.H. van Leeuwaarden, P.A. Whiting, Universality of power-of-*d* load balancing in many-server systems, Stoch. Syst. 8 (4) (2018) 265–292.
[21] X. Liu, L. Ying, On achieving zero delay with power-of-*d*-choices load balancing, in: 2018 IEEE Conference on Computer Communications, INFOCOM, 2018, pp. 297–305.
[22] L. Ying, R. Srikant, X. Kang, The power of slightly more than one sample in randomized load balancing, in: 2015 IEEE Conference on Computer Communications, INFOCOM, IEEE, 2015, pp. 1131–1139.
[23] S.G. Foss, A.L. Stolyar, Large-scale Join-Idle-Queue system with general service times, J. Appl. Probab. 54 (4) (2017) 995–1007.
[24] N. Alon, O. Gurel-Gurevich, E. Lubetzky, Choice-memory tradeoff in allocations, Ann. Appl. Probab. 20 (4) (2010) 1470–1511.
[25] M. Mitzenmacher, B. Prabhakar, D. Shah, Load balancing with memory, in: The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings, 2002, pp. 799–808.
[26] M. Mitzenmacher, How useful is old information? IEEE Trans. Parallel Distrib. Syst. 11 (1) (2000) 6–20.
[27] M.J. Luczak, J.R. Norris, Averaging over fast variables in the fluid limit for Markov chains: application to the supermarket model with memory, Ann. Appl. Probab. 23 (3) (2013) 957–986.
[28] Jonatha Anselmi, Francois Dufour, Power-of-*d*-choices with memory: Fluid limit and optimality, Math. Oper. Res. 45 (3) (2020) 862–888.
[29] T. Hellemans, B. Van Houdt, Performance analysis of load balancing policies with memory, in: Proceedings of the 13th EAI International Conference on Performance Evaluation Methodologies and Tools, VALUETOOLS '20, 2020, pp. 27–34.
[30] M. van der Boor, S.C. Borst, J.S.H. van Leeuwaarden, Hyper-scalable JSQ with sparse feedback, Proc. ACM Meas. Anal. Comput. Syst. 3 (1) (2019).
[31] Shay Vargaftik, Isaac Keslassy, Ariel Orda, LSQ: Load balancing in large-scale heterogeneous systems with multiple dispatchers, IEEE/ACM Trans. Netw. PP (2020) 1–13.
[32] Xingyu Zhou, Ness Shroff, Adam Wierman, Asymptotically optimal load balancing in large-scale heterogeneous systems with multiple dispatchers, Perform. Eval. 145 (2021) 102146.
[33] M. van der Boor, S.C. Borst, J.S.H. van Leeuwaarden, D. Mukherjee, Scalable load balancing in networked systems: Universality properties and stochastic coupling methods, in: Proceedings of the International Congress of Mathematicians, ICM 2018, Vol. 4, 2018, pp. 3911–3942.
[34] F.P. Kelly, Reversibility and Stochastic Networks, Cambridge University Press, 2011, pp. 6–7.
[35] S.A. Berezner, C.F. Kriel, A.E. Krzesinski, Quasi-reversible multiclass queues with order independent departure rates, Queueing Syst. 19 (1995) 345–359.
[36] A.E. Krzesinski, Order independent queues, in: R. Boucherie, N. Van Dijk (Eds.), Queueing Networks, in: International Series in Operations Research and Management Science, vol. 154, 2011.

Mark van der Boor is a former Ph.D. -student in the Department of Mathematics and Computer Science, Eindhoven University of Technology (TU/e) and he obtained his degree on March 26th, 2021. He received his Master's degree in Mathematics in 2016 at the Eindhoven University of Technology, specialized in Stochastics. His research focuses on queueing networks, stochastic simulation and load balancing.

Sem Borst has been a Full Professor in Stochastic Operations Research in the Department of Mathematics & Computer Science at Eindhoven University of Technology (TU/e) since 1998. He also had a (part-time) position at Bell Laboratories in Murray Hill, USA, from 1995 to 2019, and was a Senior Researcher at the Center for Mathematics & Computer Science (CWI) in Amsterdam from 1998 to 2006. His main research interests are in the area of performance evaluation and resource allocation algorithms for large-scale stochastic networks, in particular computer-communication systems. Sem has over 200 refereed publications, and was (co-)recipient of the best-paper awards at ACM SIGMETRICS/Performance 1992 and IEEE Infocom 2003, the 2001 Yosef Levy Prize, the 2005 Van Dantzig Prize, and the 2017 ACM SIGMETRICS Achievement Award. He serves or has served on the editorial boards of several journals, such as ACM Transactions on Modeling and Performance of Computing Systems, IEEE/ACM Transactions on Networking, Operations Research, Queueing Systems and Stochastic Models, and has been a program committee member of numerous conferences.

Johan van Leeuwaarden is a Full Professor and Chair of Stochastic Networks in the Department of Mathematics and Computer Science, Eindhoven University of Technology (TU/e) and a Full Professor in the Department of Operations Research, Tilburg University. His research focuses on stochastic or probability theory, the mathematics of chance and uncertainty. His main research interests are in probability theory, stochastic networks, enumerative combinatorics, and complex and asymptotic analysis. Applications that motivate his research are congestion phenomena (queueing), resource allocation in communication networks, distributed control of complex networks, random graphs and the spread of epidemics over networks. The challenge is to model, analyze and, where necessary, improve network behavior. This research is relevant to, for example, the Internet, wireless networks, traffic, energy and light networks, as well as networks of people or animals that spread viruses or information. Johan's research group works closely together with researchers from other disciplines, as well as with experts working for companies or governments.