EURANDOM PREPRINT SERIES

2020-006

June 2, 2020

Threshold-based rerouting and replication for resolving job-server affinity relations

Y. Raaijmakers, S. Borst, O. Boxma
ISSN 1389-2355

Threshold-based rerouting and replication for resolving job-server affinity relations

Youri Raaijmakers[a,*], Sem Borst[a], Onno Boxma[a]

[a] *Department of Mathematics and Computer Science, Eindhoven University of Technology, 5600 MB Eindhoven, The Netherlands*

Abstract

We consider a system with several job types and two parallel server pools. Within the pools the servers are homogeneous, but across pools possibly not in the sense that the service speed of a job may depend on its type as well as the server pool. Immediately upon arrival, jobs are assigned to a server pool, possibly based on (partial) knowledge of their type. In case such knowledge is not available, information about the job type can however be obtained while the job is in service; as the service progresses, the likelihood that the service speed of this job type is low increases, creating an incentive to execute the job on different, possibly faster, server(s). Two policies are considered: reroute the job to the other server pool, or replicate it there.

We determine the effective load per server under both the rerouting and replication policy for completely unknown as well as partly known job types. We also examine the impact of these policies on the stability bound, and find that the uncertainty in job types may significantly degrade the performance. For (highly) unbalanced service speeds full replication achieves the largest stability bound while for (nearly) balanced service speeds no replication maximizes the stability bound. Finally, we discuss how the use of threshold-based policies can help improve the expected latency for completely or partly unknown job types.

Keywords: Parallel-processing systems, stability, rerouting, replication, compatibility constraints, server heterogeneity, straggler mitigation

1. Introduction

This paper considers parallel-processing systems with two heterogeneous server pools, in which a dispatcher assigns jobs to one of the server pools immediately upon arrival. However, a job is allowed to be *rerouted* to the other pool when its service has not yet been completed after a certain amount of processing time. We also consider an alternative option in which such a job is *replicated* at the other pool. The jobs are assumed to be of different types; one job type might, e.g., be fast on a server from pool 1 and slow on a server from pool 2, whereas this might be reversed for another job type. We examine the ability of threshold-based policies to deal with such heterogeneity in service speeds: if the job types are unknown, or only partly known, can

*Corresponding author

Email address: y.raaijmakers@tue.nl (Youri Raaijmakers)

we use a threshold for the amount of processing time after which the job should be rerouted (or replicated) in order to improve stability of the system and reduce latency?

Replication schemes as described above were introduced to mitigate the adverse effect of so-called stragglers on the system performance, see for example [3, 7, 17]. Closely related to the present paper is [1] which studies the replication policy under the assumption of i.i.d. replicas and homogeneous servers. An approximation for the expected latency is derived for both exponential and shifted-exponential job size distributions. Moreover, the trade-off between the expected latency and the cost, defined as the sum of the processing times of each replica involved in the job execution, is analyzed via simulation. Replication schemes are shown to reduce both cost and expected latency in case of heavy-tailed job size distributions. Also closely related is [11] which focuses on the throughput-optimal replication policy under the assumption of identical replicas and a continuous speed distribution at each server. The Markov Decision Process formulation for the optimal replication policy is in general intractable, and therefore an upper bound is derived. In addition, a myopic *MaxRate* policy is introduced, which depends on the number of unfinished jobs after a certain action and the expected remaining processing time.

In all the above papers the speed variations experienced by the various replicas are essentially assumed to be purely random in nature. The multi-type set-up that we consider in the present paper allows for intrinsic differences in speed across servers depending on the specific characteristics of each individual job. Such heterogeneity in service speeds captures systematic job-server affinity relations which may arise from data locality issues but may also reflect soft compatibility constraints that are increasingly prevalent in data center environments. We investigate the effectiveness of threshold-based replication and rerouting in the presence of such underlying job-server affinity relations.

The performance of the threshold-based replication policy is strongly linked to the stability of redundancy scheduling, where several replicas are created for each job immediately upon arrival, see for example [10, 13, 14, 15]. Specifically, it is demonstrated in [15] that in case of *known* job types and a probabilistic type-dependent job assignment strategy, *no replication* maximizes the stability bound for New-Better-than-Used (NBU) job size distributions. In contrast, in case of *unknown* job types, *full replication* achieves a larger stability bound than no replication for New-Worse-than-Used (NWU) job size distributions.

Likewise, the threshold-based rerouting policy is closely connected to the problem of learning the type of a job in an online manner as considered in [4]. In their model there is a server pool that is compatible with jobs of all types and a server pool that is either compatible or incompatible with a job depending on its type. Here (in)compatible means that the service of a job can(not) be fulfilled by a server in this pool. As proposed in [4], the job types can be learned via observing the processing time. Indeed, as time elapses and the service has not been completed, the likelihood that the job is incompatible with the specific server it is on increases, and the job is therefore rerouted to the other server pool when the likelihood that the server is incompatible exceeds some value, also called threshold. For related literature on this learning problem we refer to [4] and the references therein.

In this paper we consider similar threshold-based policies, but defined in terms of the received processing time, and extend the model of [4] to allow for completely general type-dependent service speed variations. Furthermore, we include replication as an additional option besides rerouting, resulting in concurrent execution of replicas at possibly different speeds as in [15]. Throughout, we use the term *stability bound* to refer to the maximum arrival rate of jobs for which the effective load per server is smaller than one. The effective load per server is defined as a measure for the amount of time required to serve all offered jobs.

The key contributions and insights can be summarized as follows.

1. We determine the effective load per server under both the rerouting and replication policy. We show that for unknown job types and (highly) unbalanced service speeds the largest stability bound is achieved by full replication. In contrast, for (nearly) balanced service speeds the stability bound is maximized by not replicating at all. Surprisingly, rerouting or replication does not significantly increase the stability bound in case of unknown job types, in part because of the variability in the job sizes.
2. We also determine the effective load per server in a scenario with known job sizes. We find that typically there still is a significant performance loss in terms of the stability bound compared to the case of known job types. This implies that the uncertainty in the job types plays a more pertinent role than the variability in the job sizes.
3. We extend our modeling framework to allow for partly known job types. We observe that decreasing the uncertainty in the job types increases the stability bound in a convex manner.
4. We discuss how the use of threshold-based policies can also help improve the expected latency for completely unknown or partly known job types. While an exact latency analysis is beyond the scope of this paper, we provide approximations that show a reduction of the expected latency.

The remainder of the paper is organized as follows. In Section 2 we provide a detailed model description. Analytical expressions for the effective load per server under threshold-based rerouting or replication in the case of completely unknown and partly known job types are derived in Section 3. In Section 4 we characterize the effective load per server in a scenario where the thresholds depend on the job sizes when these are known in advance. Section 5 presents extensive numerical results which quantify the performance implications due to the uncertainty in job types. In Section 6 we examine how the use of threshold-based policies can also help reduce expected latency. Section 7 contains conclusions and some suggestions for further research.

2. Model description

Consider a system with N servers and a dispatcher where jobs arrive at rate λ. The servers are divided into two server pools, where n_i denotes the number of servers in pool i, with $n_1 + n_2 = N$. The dispatcher assigns jobs immediately upon arrival to one of the two server pools according to a static policy as will be further specified later. Each pool has a central queue that follows a non-idling service discipline, in the sense that the servers always serve jobs when the queue is non-empty. We allow the sizes X_1, X_2 of a generic job on the server pools to be governed by some joint distribution $F_X(x_1, x_2)$, where X_i, $i = 1, 2$, are each distributed as a generic random variable X, but not necessarily independent. This covers the extreme scenarios of perfect dependence (identical replicas) and no dependence at all (i.i.d. replicas), as previously considered in the literature, as special cases. The service speeds R_1, R_2 for a given job in the two pools may differ, but within the pools servers have the same speed. For a particular job on a server in pool i, $i = 1, 2$, with size x_i, x_i/R_i represents the execution time. We allow the service speeds R_1, R_2 of a generic job to be governed by some joint distribution $F_R(r_1, r_2)$, reflecting possible server heterogeneity and job-server affinity relations, thus covering a broad range of common workload models as special cases. Examples are the S&X model [8], a scenario with heterogeneous server speeds and the 'output-queued' flexible server model [16].

For convenience, we consider the case where the joint distribution $F_R(r_1, r_2)$ is discrete, and has mass in a finite number of, say, J points (r_{1j}, r_{2j}) with corresponding probabilities p_j, $j = 1, \ldots, J$. This system may equivalently be thought of as having J job types, where r_{ij} is the service speed of type-j jobs at servers in pool i.

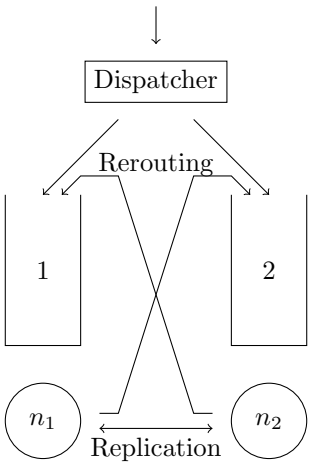

Figure 1: Representation of the model that illustrates the difference between the rerouting and the replication policy.

As an important feature of our model we consider the following two resource allocation policies: *rerouting* and *replication*, see also Figure 1. In both policies, a job exits the system once its service requirement has been fully fulfilled. However, when the job has received a given amount of processing time (we focus on the case of a fixed service time threshold $\boldsymbol{\tau} = (\tau_1, \tau_2)$), under the rerouting policy, the job is rerouted to the other server pool. Under the replication policy, the job is replicated in the other server pool while also staying in service at its original pool. In both policies the service does not carry over, i.e., the entire service requirement should be fulfilled by a server in the 'new' pool. Thus, the main difference between the policies is that in the case of replication, when the job is in service for an amount of processing time larger than the threshold, there is still a replica in service at the original pool.

We make the following assumption: replicas of a job after replication have preemptive-resume priority over jobs that are not yet replicated, if they can get simultaneous service at both server pools. If both replicas cannot immediately get simultaneous service, then they wait until one server in each pool is available. This ensures that after replication the replicas will always receive simultaneous service. This assumption additionally helps eschew stability issues that can arise due to local priorities in scenarios with simultaneous resource possession, see for example [12, Section 8.4].

For equally large server pools, i.e., $n_1 = n_2$, the preemptive-resume priority implies that jobs never have to wait after replication. Indeed, the numbers of servers handling replicated jobs are equal at all times and therefore it can never occur that a job is replicated, while all the servers in the other server pool are serving jobs that have already been replicated. For unequal sizes of the server pools, i.e., $n_1 \neq n_2$, it might occur that after replication both replicas have to wait before getting service.

Replicas are abandoned as soon as one finishes service. The replication policy with threshold $\boldsymbol{\tau} = \mathbf{0}$ is referred to as the *full redundancy* policy. For threshold $\boldsymbol{\tau} = \boldsymbol{\infty}$ the rerouting and

replication policy are equivalent, and will be called the *zero redundancy* policy.

In this paper we restrict ourselves to the scenario with two server pools. Extension to an arbitrary number of server pools is left as a topic for further research. We focus on increasing the achievable stability bound by optimizing threshold values in terms of the received processing times.

3. Stability for unknown job sizes

In Section 3.1 we derive analytical expressions for the effective load per server for both the rerouting and replication policy under a given assignment and threshold values. In Section 3.2 we specify these allocation fractions for three cases: Completely unknown job types, partly unknown job types and completely known job types. Throughout we assume that the job sizes are unknown.

We attach superscripts *Rer* and *Rep* to metrics that correspond to the *rerouting* and *replication* policy, respectively.

3.1. Load of the system under a given assignment

Let A denote the $2 \times J$ stochastic assignment matrix with elements α_{ij}. Under the $S(A, \boldsymbol{\tau})$ policy we assign a fraction α_{ij} of type-j jobs to server pool i and reroute (or replicate) at this pool as soon as the processing time equals τ_i for $i = 1, 2$. We define $l := 3 - i$ as the other server pool than i, so that $\alpha_{ij} + \alpha_{lj} = 1$ for all $j = 1, \ldots, J$.

Proposition 1. *The effective load of server pool i, for $i = 1, 2$, in the system with rerouting under the $S(A, \boldsymbol{\tau})$ policy is*

$$\rho^{Rer}_{i,S(A,\boldsymbol{\tau})} = \frac{\lambda \mathbb{E}\left[B^{Rer}_i(A, \boldsymbol{\tau})\right]}{n_i}, \tag{1}$$

where the expected service time requirement of an arbitrary job, assigned either to pool i or pool l, at pool i is

$$\mathbb{E}\left[B^{Rer}_i(A, \boldsymbol{\tau})\right] = \sum_{j=1}^{J} p_j \alpha_{ij} \mathbb{E}\left[\min\left\{\frac{X_i}{r_{ij}}, \tau_i\right\}\right] + \sum_{j=1}^{J} p_j \alpha_{lj} \mathbb{E}\left[\frac{X_i}{r_{ij}} \mathbb{1}\left\{\frac{X_l}{r_{lj}} > \tau_l\right\}\right]. \tag{2}$$

Proof: When allocating a job to server pool i, there are two possibilities: i) the job finishes before rerouting; ii) the job is rerouted, in which case the job receives τ_i units of service at this server pool. However, jobs that started in server pool l can be rerouted to pool i as well. The proof then follows from noting that $\mathbb{E}\left[\min\left\{\frac{X_i}{r_{ij}}, \tau_i\right\}\right]$ represents the expected received amount of processing time of a job in server pool i before completing or being rerouted to server pool l, and $\mathbb{E}\left[\frac{X_i}{r_{ij}} \mathbb{1}\left\{\frac{X_l}{r_{lj}} > \tau_l\right\}\right]$ represents the expected received amount of processing time of a job in server pool i, if any, after having received τ_l units in pool l first. □

To simplify the expressions for the replication policy, we introduce some notation:

$$k^m_{il,j}(y) = \mathbb{E}\left[\left(\min\left\{\frac{X_i}{r_{ij}} - y, \frac{X_l}{r_{lj}}\right\}\right)^m \mathbb{1}\left\{\frac{X_i}{r_{ij}} > y\right\}\right].$$

This represents the m-th moment of the amount of time that a job will be processed in both server pools, if any, after having been processed up to y time units in server pool i first. In case of $m = 1$ the superscript is omitted.

Proposition 2. *The effective load per server in pool i, for $i = 1, 2$, in the system with replication under the $S(A, \boldsymbol{\tau})$ policy is*

$$\rho^{Rep}_{i,S(A,\boldsymbol{\tau})} = \frac{\lambda \mathbb{E}\left[B^{Rep}_i(A, \boldsymbol{\tau})\right]}{n_i}, \tag{3}$$

where the expected service time requirement of an arbitrary job, assigned either to pool i or pool l, at pool i is

$$\mathbb{E}\left[B^{Rep}_i(A, \boldsymbol{\tau})\right] = \sum_{j=1}^{J} p_j \alpha_{ij} \left(\mathbb{E}\left[\min\left\{ \frac{X_i}{r_{ij}}, \tau_i \right\} \right] + k_{il,j}(\tau_i) \right) + \sum_{j=1}^{J} p_j \alpha_{lj} k_{li,j}(\tau_l). \tag{4}$$

Proof: When allocating a job to server pool i, there are two possibilities: i) the job finishes before replication; ii) the job is replicated, in which case the job spends at least τ_i time units in service at this server pool. The (remaining) expected service time requirement of jobs starting at server pool i after replication, if any, is $k_{il,j}(\tau_i)$. However, jobs that started in server pool l can be replicated to pool i as well. The (remaining) expected service time requirement of these jobs, if any, is $k_{li,j}(\tau_l)$. □

Evidently, $\rho^{\text{Rer}}_{i,S(A,\boldsymbol{\tau})} < 1$ (or $\rho^{\text{Rep}}_{i,S(A,\boldsymbol{\tau})} < 1$), for $i = 1, 2$, is a necessary condition for stability. It is quite plausible that this condition is in fact also sufficient for the system to be stable (under the preemptive-resume priority policy for replicated jobs).

Indeed, for multiserver queues it is well known that the system is stable if and only if the load per server is smaller than one, where the arrival process can be quite general (see for example [6, Chapter 1] or [18, Chapter 7]). Moreover, in [18, Proposition 7.4.12] it is proved that, under the assumption that the sequence of inter-arrival and service times of the jobs at the queues is ergodic and stationary, the stability of two $G/G/1$ queues in tandem is independent of the departure process of the first queue. However, in our system both initially assigned *and* rerouted (or replicated) jobs arrive at a server pool. In addition, replicated jobs must receive service simultaneously at both server pools, implying that servers may be idling even when there are jobs waiting. As a result, it is hard to rigorously establish that a load per server smaller than one is sufficient for the system to be stable.

Remark 1. *The expected service time requirement at server pool i for the zero redundancy policy is equal to*

$$\mathbb{E}\left[B^{Rer}_i(A, \infty)\right] = \mathbb{E}\left[B^{Rep}_i(A, \infty)\right] = \sum_{j=1}^{J} p_j \alpha_{ij} \mathbb{E}\left[\frac{X_i}{r_{ij}}\right], \tag{5}$$

and the expected service time requirement at server pool i for the full redundancy policy is equal to

$$\mathbb{E}\left[B^{Rep}_i(A, \mathbf{0})\right] = \sum_{j=1}^{J} p_j \mathbb{E}\left[\min\left\{\frac{X_i}{r_{ij}}, \frac{X_l}{r_{lj}}\right\}\right]. \tag{6}$$

Observe that the expected service time requirement for the full redundancy policy is independent of the assigned fractions. The achievable stability bound for this policy coincides with that derived in [15]. Furthermore, note that in case of identical replicas the effective load per server for both the zero redundancy and full redundancy policy is insensitive to the job size distribution, given its mean.

3.2. Specific assignment fractions

In this subsection, we specify the assignment fractions in Equations (2) and (4) for the three cases of completely unknown, partly unknown and completely known job types. Let q_j^i denote the fraction of type-j jobs that are assigned to server pool i in these three cases. The proofs are straightforward, and hence omitted.

Corollary 1. *For the case of unknown job types, Propositions 1 and 2 hold with*

$$\alpha_{ij} = q^i, \qquad \text{for } i = 1, 2 \text{ and } j = 1, \ldots, J. \tag{7}$$

We proceed with the case of partly known job types by which we mean that every arriving job is *believed* to be of a specific type. This can be thought of as a *label* indicating the likely type of a job. Let $p_{j\to j^*}$ denote the probability that a job of type j is believed to be of type j^*, with $\sum_{j^*=1}^{J} p_{j\to j^*} = 1$. The scenario $p_{j\to j} = 1$ corresponds to the case of known job types, whereas $p_{j\to j^*} = p_{j^*}$, for all $j = 1, \ldots, J$, corresponds to the case of unknown job types.

Corollary 2. *For the case of partly known job types, Propositions 1 and 2 hold with*

$$\alpha_{ij} = \sum_{j^*=1}^{J} p_{j\to j^*} q_{j^*}^i, \qquad \text{for } i = 1, 2 \text{ and } j = 1, \ldots, J. \tag{8}$$

Observe that in the case of completely unknown job types Equation (8) becomes $\alpha_{ij} = \sum_{j^*=1}^{J} p_{j^*} q_{j^*}^i$ for $i = 1, 2$, which is equivalent to Equation (7) since it is independent of the job types.

Corollary 3. *For the case of known job types, Propositions 1 and 2 hold with*

$$\alpha_{ij} = q_j^i, \qquad \text{for } i = 1, 2 \text{ and } j = 1, \ldots, J. \tag{9}$$

Substituting these assignment fractions in Equation (5) gives

$$\mathbb{E}\left[B_i^{\mathrm{Rer}}(A, \infty)\right] = \mathbb{E}\left[B_i^{\mathrm{Rep}}(A, \infty)\right] = \sum_{j=1}^{J} p_j q_j^i \mathbb{E}\left[\frac{X_i}{r_{ij}}\right].$$

This coincides with the stability condition derived in [15] for no replication and known job types. Moreover, in [15] it is proved that no replication gives a strictly larger stability bound than replication in the case of NBU job size distributions, independent of the server speeds. For NWU job size distributions examples show that both no replication and full replication can give a larger stability bound depending on the server speeds. Examples in which neither no replication nor full replication gives a larger stability bound have not been found, see [15].

Definition 1. *A rerouting policy corresponds to a stochastic assignment matrix A with elements α_{ij}, where we assign a fraction α_{ij} of type-j jobs to server pool i and vectors $(\tau_{i1}, \tau_{i2}, \ldots, \tau_{iK_i}) \in \mathbb{R}_+^{K_i}$ with possibly $K_i = 0$ or $K_i = \infty$, $i = 1, 2$. Here a job of size $\boldsymbol{x}$ that is assigned to server pool i, is processed for up to τ_{i1} time units, and then successively restarted and processed in server pool l for up to τ_{i2} time units, restarted and processed in server pool i for up to τ_{i3} time units, etc. Eventually this job is restarted and processed for up to τ_{iK_i} time units in server pool i or l if K_i is odd or even, respectively, and ultimately processed for any amount of time in server pool i or l if K_i is even or odd, respectively, until the job is completed, whichever occurs first (or, as long as the job has not been completed).*

In this section we focused on single-threshold policies. The effective load per server for the rerouting policy can also be derived for the case of an arbitrary n-threshold policy as defined in Definition 1, see Appendix B. However, this would make the optimization over all the parameters, as done numerically in Section 5, much more complex.

4. Stability for known job sizes

In the previous section we derived the effective load per server for unknown job sizes. In this section, we consider the case of known job sizes, which is indicated by an additional superscript *KS* in the notation. We allow the assignment fractions and the threshold values to depend on the size of the job at the server pool, which is indicated by a superscript x. Thus, the threshold at a server pool may differ for every arriving job, whereas in the previous section the threshold at the pool was equal for all jobs. The joint density of the job sizes at the server pools is denoted by $f_X(\cdot, \cdot)$. The effective load per server for the rerouting and replication policy are derived in Propositions 3 and 4, respectively.

Proposition 3. *The effective load per server in pool i, for $i = 1, 2$, in the system with rerouting under the $S(A^x, \boldsymbol{\tau}^x)$ policy, in case of known job sizes, is*

$$\rho_{i,S(A^x,\boldsymbol{\tau}^x)}^{Rer,KS} = \frac{\lambda \mathbb{E}\left[B_i^{Rer,KS}(A^x, \boldsymbol{\tau}^x)\right]}{n_i}, \tag{10}$$

where

$$\mathbb{E}\left[B_i^{Rer,KS}(A^x, \boldsymbol{\tau}^x)\right] = \iint_{\mathbb{R}^2} \left(\sum_{j=1}^{J} p_j \alpha_{ij}^x \min\left\{ \frac{x_i}{r_{ij}}, \tau_i^x \right\} + \sum_{j=1}^{J} p_j \alpha_{lj}^x \frac{x_i}{r_{ij}} \mathbb{1}\left\{ \frac{x_l}{r_{lj}} > \tau_l^x \right\} \right) f_X(x_1, x_2) dx_1 dx_2. \tag{11}$$

Proof: The proof follows along the same lines as the proof of Proposition 1. □

Proposition 4. *The effective load per server in pool i, for $i = 1, 2$, in the system with replication under the $S(A^x, \boldsymbol{\tau}^x)$ policy, in case of known job sizes, is*

$$\rho_{i,S(A^x,\boldsymbol{\tau}^x)}^{Rep,KS} = \frac{\lambda \mathbb{E}\left[B_i^{Rep,KS}(A^x, \boldsymbol{\tau}^x)\right]}{n_i}, \tag{12}$$

where

$$\mathbb{E}\left[B_i^{Rep,KS}(A^x,\boldsymbol{\tau}^x)\right] = \iint_{\mathbb{R}^2}\left(\sum_{j=1}^{J} p_j\alpha_{ij}^x\left(\min\left\{\frac{x_i}{r_{ij}},\tau_i^x\right\} + k_{il,j}^{KS}(\boldsymbol{x},\tau_i^x)\right) + \sum_{j=1}^{J} p_j\alpha_{lj}^x k_{li,j}^{KS}(\boldsymbol{x},\tau_l^x)\right) f_X(x_1,x_2)dx_1dx_2, \tag{13}$$

where $k_{il,j}^{KS}(\boldsymbol{x},y) = \min\left\{\frac{x_i}{r_{ij}} - y, \frac{x_l}{r_{lj}}\right\}\mathbb{1}\left\{\frac{x_i}{r_{ij}} > y\right\}$.

Proof: The proof follows along the same lines as the proof of Proposition 2. □

Again, the effective load per server can also be derived for the case of an arbitrary n-threshold policy, see Appendix B.

Definition 2. *Let* $(n_{i1}^x, n_{i2}^x, \ldots, n_{iK_i}^x) \in \mathbb{N}^{K_i}$ *be vectors, with* $\sum_{k=1}^{K_i} n_{ik}^x \leq J-1$, $n_{ik}^x \geq 1$ *for all* $k = 1,\ldots,K_i$ *and* $i = 1,2$. *Given the set* J_{ik-1}, *with* $J_{i0} = \{1,2,\ldots,J\}$, *let* r_{ik} *be the* n_{ik}^x*-th highest service speed on server pool* $i(k)$ *among the job types in* J_{ik-1} *and let* J_{ik} *be the job types in* J_{ik-1} *that do not have the* n_{ik}^x *highest service speeds on server pool* $i(k)$, *where* $i(k) = i$ *if* k *is odd and* $i(k) = l$ *if* k *is even and* $i = 1,2$ *The set* J_{ik} *corresponds to the remaining possible job types of a job that is initially assigned to server pool* i *and rerouted* k *times (according to a specific policy). Then the rerouting policy that corresponds to some stochastic assignment matrix* A^x *and vectors* $(\tau_{i1}^x, \tau_{i2}^x, \ldots, \tau_{iK_i}^x) \in \mathbb{R}_+^{K_i}$, *with* $\tau_{ik}^x = \min_{j\in J_{ik-1}}^{(n_{ik})} x_{i(k)}/r_{i(k)j}$ *for* $k = 1,\ldots,K_i$ *and* $i = 1,2$, *where* $\min^{(n)}$ *denotes the n-th order statistic, is said to be associated with the vectors* $(n_{i1}^x, n_{i2}^x, \ldots, n_{iK_i}^x) \in \mathbb{N}^{K_i}$.

Any rerouting policy that is associated with two particular vectors as described in Definition 2 is said to be a rational policy. Note that a rational policy involves at most $J-1$ threshold values, and the next observation provides an intuitive explanation for that.

Observation 1. *In the case of known job sizes and J job types, for each job there are at most J time points, referred to as rational decision points, at which the job could possibly fulfill its service requirement, which depend on the job size, server speeds and the threshold values. Moreover, we (only) gain information about the job type at these (rational) decision points.*

The next lemma establishes that in order to achieve maximum stability for known job sizes, we may restrict within the class of all rerouting policies as specified in Definition 1, to the subclass of rational policies as defined in Definition 2.

Lemma 1. *For any arbitrary rerouting policy there exists a rational rerouting policy that is better in the sense that for a job of any size, given the initial assignment to one of the server pools, it uses at most the same cumulative amount of processing time in each of the server pools.*

Proof by induction in the number of remaining possible job types:
Base case: Suppose that for a job initially assigned to server pool i, the rerouting policy is rational up to (measured in processing time) $t_{J-1} = \min_{j\in J_{iK_i-1}}^{(1)} x_{i(K_i-1)}/r_{i(K_i-1)j}$ time units, at which we know that the job is of the one remaining job type and without rerouting the job would finish at

t_J. Then rerouting *exactly at* t_{J-1} uses at most the same cumulative amount of processing time in each of the server pools as rerouting *between* t_{J-1} and t_J.
Inductive step: Show that for any $m \geq 1$, if the lemma holds for m remaining possible job types, then it also holds for $m+1$ remaining possible job types.

Suppose that for a job initially assigned to server pool i, the rerouting policy is rational up to (measured in processing time) $t_{J-m-1} = \min_{j \in J_{ik}}^{(n_{ik})} x_{i(k)}/r_{i(k)j}$ time units, with k (the number of reroutings after which the job can possibly be of the $m+1$ remaining job types) such that $|J_{ik}| = m + 1 - n_{ik}$, and without rerouting the first time the job could possibly finish is t_{J-m} (if it is of a specific type). Note that at t_{J-m-1} we know that the job is of one of the $m+1$ remaining job types. Then rerouting *exactly at* t_{J-m-1} uses at most the same cumulative amount of processing time in each of the server pools as rerouting *between* t_{J-m-1} and t_{J-m}. In case of no rerouting, if the job does not finish at t_{J-m} we know that the job is of one of the m remaining job types, and we can apply the induction hypothesis. In case of rerouting at t_{J-m-1} the first time the job could possibly finish, say t^*_{J-m}, may differ from t_{J-m}, however we will never reroute again before t^*_{J-m}. At this time we can apply the induction hypothesis. □

Observe that Lemma 1 proves that there exists a rational rerouting policy that is better than any arbitrary policy given the initial assignment to one of the server pools. To achieve the maximum achievable stability bound we also need to find the appropriate initial assignment matrix A^x.

Remark 2. *Lemma 1 holds as well for the single-threshold rerouting policy, analyzed in Proposition 3. For the optimization we hence have* $J-1$ *candidate values for the threshold.*

5. Numerical results

In Sections 3 and 4 we derived the effective load per server for various cases. We now present numerical results to get further insight in the performance implications due to the uncertainty in job types. Throughout this section we denote the policies as follows: Rerouting (Rer), Replication (Rep), Zero redundancy (ZRed) and Full redundancy (FRed). The expected service time requirement for these four policies is given by (2), (4), (5) and (6), respectively. For known job sizes (KS) we only show the maximum of the rerouting and replication policy, where the expected service requirement is given by (11) and (13), respectively. Surprisingly, in all scenarios that we considered the maximum was achieved by the rerouting policy with known job sizes.

In Section 5.1 we examine the scenario of completely unknown job types and in Section 5.2 that of partly known job types. In both subsections we distinguish between identical and i.i.d. replicas with exponentially distributed job sizes with mean N, to ensure that the stability condition for known job types is $\lambda < 1$. We refer to Appendix Appendix A for results where the job sizes are Pareto Type I distributed with minimum possible value 1 and index $\frac{N}{N-1}$, thus again having mean N.

The scenario $I = 2$, $J = 2$, $N = 10$, $n_1 = n_2 = 5$, $(r_{11}, r_{21}) = (1, r_{\text{slow}})$ and $(r_{12}, r_{22}) = (r_{\text{slow}}, 1)$ is considered throughout. All figures show the maximum achievable stability bound denoted by $\lambda_{\max}$ as function of some relevant system parameter. For the maximization of the achievable stability bound, we used a brute-force search with a certain fine multidimensional grid of starting points.

5.1. Completely unknown job types

5.1.1. Identical replicas

In Figure 2 the maximum achievable stability bound for the various policies is depicted when varying the parameters p_1 and r_{slow}. Observe that the latter scenario is completely symmetric and therefore $q^1 = q^2 = 0.5$ is the optimal allocation.

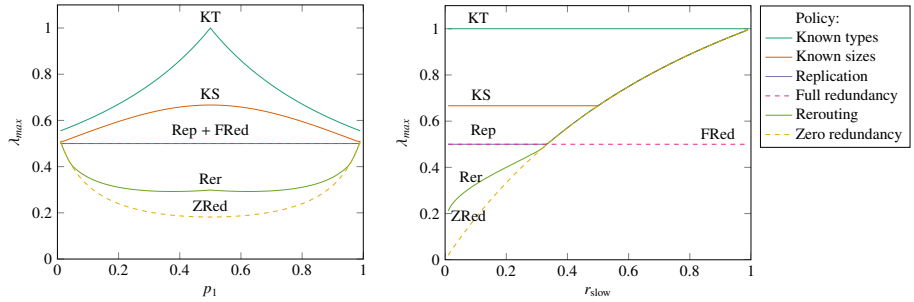

Figure 2: Achievable stability bound for identical replicas with exponentially distributed job sizes in the scenario $I = 2$, $J = 2$, $N = 10$, $n_1 = n_2 = 5$, $(r_{11}, r_{21}) = (1, r_{\text{slow}})$ and $(r_{12}, r_{22}) = (r_{\text{slow}}, 1)$ with corresponding probabilities p_1 and $p_2 = 1 - p_1$, with $r_{\text{slow}} = 0.1$ (left) and $p_1 = p_2 = 0.5$ (right). In the left sub-figure, the replication and full redundancy policy are overlapping.

The left sub-figure in Figure 2 shows that the replication policy outperforms the rerouting policy. This generally holds for scenarios where r_{slow} is relatively small (unbalanced server speeds). The reason is that the rerouting policy becomes unstable, i.e., $\lambda_{\max} \downarrow 0$, as $r_{\text{slow}} \downarrow 0$. In particular, in this scenario, the probability of rerouting the job to an (almost) incompatible server pool, after which we cannot reroute the job again, is strictly positive.

The right sub-figure in Figure 2 reveals that both the rerouting and replication policy achieve the same achievable stability bound for $r_{\text{slow}} > 0.4$, which is the same as for the zero redundancy. Therefore, the achievable stability bound can be achieved by the threshold $\tau = \infty$, see (5). So here the threshold does not increase the achievable stability bound. This generally holds in scenarios where r_{slow} is relatively large (balanced server speeds). For unbalanced server speeds, the replication policy is equivalent to the full redundancy policy and both outperform the rerouting policy.

The achievable stability bound in Figure 2 fails to give information about the processing time per job that is needed to distinguish the job types. In particular, consider the example where one job type has sizes $\epsilon << 1$ and $K >> 1$ on a server from pool 1 or 2, respectively, and another job type has sizes K and ϵ on these server pools. In the rerouting policy, after only ϵ time units we know the job type and we reroute when the service requirement has not been completely fulfilled. Thus, we can distinguish the job types really fast. However, the service time requirement of a rerouted job is in total 2ϵ, while in the case of known types the service requirement of all jobs is ϵ. Therefore, for unbalanced server speeds, the rerouting policy always has a performance loss of at least 33%, despite the fact that it can distinguish the job types fast.

5.1.2. Independent and identically distributed replicas

In Figure 3 we consider the same scenario as in Figure 2, but now for i.i.d. replicas.

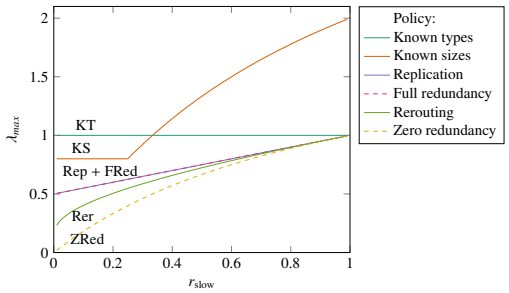

Figure 3: Achievable stability bound for i.i.d. replicas with exponentially distributed job sizes in the scenario $I = 2$, $J = 2$, $N = 10$, $n_1 = n_2 = 5$, $(r_{11}, r_{21}) = (1, r_{\text{slow}})$ and $(r_{12}, r_{22}) = (r_{\text{slow}}, 1)$ with corresponding probabilities $p_1 = p_2 = 0.5$. The replication and full redundancy policy are overlapping.

Figure 3 shows that the replication policy outperforms the rerouting policy, especially in scenarios with unbalanced server speeds. In this figure, the replication policy is equivalent with the full redundancy policy. This means that learning the job types for the replication policy does not improve the achievable stability bound. Moreover, it can be seen that observing the job sizes can significantly increase the achievable stability bound. For balanced server speeds, rerouting in the case of known sizes even outperforms the policy with known job types.

5.2. Partly known job types

In this section we present numerical results for the achievable stability bound in the case where job types are partly known, i.e., for an arriving job there is a belief that it is of a specific type.

5.2.1. Identical replicas

We consider a scenario with unbalanced server speeds and balanced server speeds (Figure 4), when varying the probability $p_{1\to1} = p_{2\to2}$.

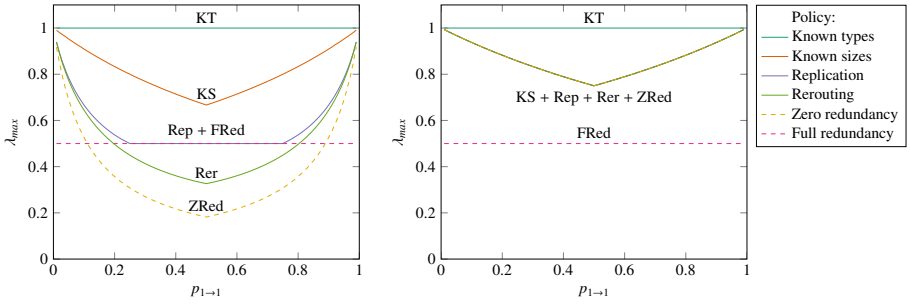

Figure 4: Achievable stability bound for identical replicas with exponentially distributed job sizes in the scenario $I = 2$, $J = 2$, $N = 10$, $n_1 = n_2 = 5$, $(r_{11}, r_{21}) = (1, r_{\text{slow}})$ and $(r_{12}, r_{22}) = (r_{\text{slow}}, 1)$ with corresponding probabilities $p_1 = p_2 = 0.5$, with $r_{\text{slow}} = 0.1$ (left) and $r_{\text{slow}} = 0.6$ (right) when varying the probability $p_{1\to1} = p_{2\to2}$.

Figure 4 indicates that decreasing the uncertainty in the job types increases the achievable stability bound in a convex manner. Moreover, for unbalanced server speeds, decreasing the uncertainty about the job types at first does not have any effect on the achievable stability bound for

the replication policy. However, the replication policy still achieves a larger achievable stability bound than the rerouting policy, especially in scenarios with high uncertainty about the job types.

For balanced server speeds, it can be seen that the achievable stability bounds for the known sizes, replication, rerouting and zero redundancy policy are all equal. Hence, for balanced server speeds, thresholds do not increase the achievable stability bound even if the uncertainty in the job types is decreased.

5.2.2. Independent and identically distributed replicas

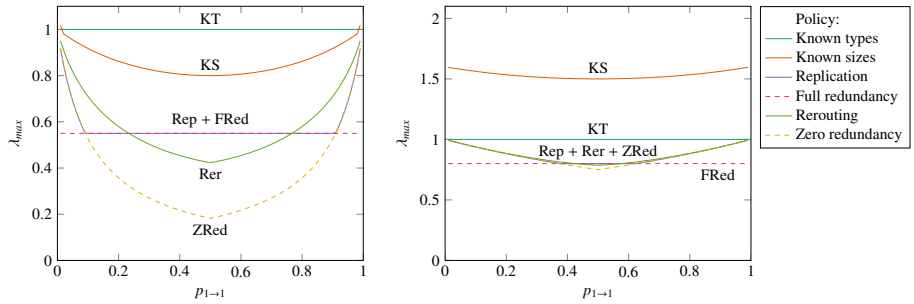

Figure 5: Achievable stability bound for i.i.d. replicas with exponentially distributed job sizes in the scenario $I = 2$, $J = 2$, $N = 10$, $n_1 = n_2 = 5$, $(r_{11}, r_{21}) = (1, r_{\text{slow}})$ and $(r_{12}, r_{22}) = (r_{\text{slow}}, 1)$ with corresponding probabilities $p_1 = p_2 = 0.5$, with $r_{\text{slow}} = 0.1$ (left) and $r_{\text{slow}} = 0.6$ (right) when varying the probability $p_{1\to1} = p_{2\to2}$.

Figure 5 reveals that the use of thresholds does not improve the achievable stability bound for the replication policy. Interestingly, for the replication policy in a scenario with (highly) unbalanced server speeds, decreasing the uncertainty has no effect on the achievable stability bound at first. Namely, in the figure it can be seen that the achievable stability bound is constant for $p_{1\to1}$ between 0.1 and 0.9, i.e., where the replication and full redundancy policy coincide.

6. Reducing the expected latency

In the previous sections we focused on the achievable stability bound as key performance metric of interest. The present section aims to take a first step towards investigating how the use of thresholds can help reduce the expected latency. Throughout this section we assume that the service discipline of the two server pools is FCFS, and that there is only one server per pool, i.e., $n_1 = n_2 = 1$.

The expected latency has been studied from a somewhat related learning perspective before. Among the numerous studies in the literature, we will highlight two. In [9] the performance of several policies is compared in the setting with heterogeneous server speeds. In particular, the JSQ-DAS (JSQ-d with Accomplishment Sampling) policy uses the idea of server accomplishment to decide which servers to poll, which can be viewed as a learning scheme. In [2] a system is introduced that improves the expected latency. The improvement is achieved by replication of small jobs, which ensures that these jobs do not have to wait for large jobs, called stragglers. Learning the optimal threshold for replication is of vital importance for the performance, i.e., the expected latency.

Consider the same model as for the analysis of the effective load per server. Again jobs are assigned to a certain server pool and can be rerouted (or replicated) to the other pool after a certain processing time. In this section we assume that the jobs arrive at the system according to a Poisson process with parameter λ. We even know what fraction of jobs is rerouted (or replicated) at server pool i, namely $p_i^\tau := \sum_{j=1}^J p_j \alpha_{ij} \mathbb{P}\left(\frac{X_i}{r_{ij}} > \tau_i\right)$. By the Poisson thinning property these arrivals follow a Poisson process with parameter λp_i^τ. However, the departure process, and thus the arrival process at the other server pool after rerouting (or replication), is a complicated process. It is not Poisson, not even a renewal process, even if the job sizes are exponentially distributed. Still, it turns out that we can approximate the mean latency of jobs quite accurately by assuming that jobs are rerouted (or replicated) according to a Poisson process. The Poisson assumption is exact in the extreme cases of $\boldsymbol{\tau} = \mathbf{0}$, in which case all jobs are immediately rerouted (or replicated) and of $\boldsymbol{\tau} = \boldsymbol{\infty}$, in which case there is no rerouting (or replication). For large $\boldsymbol{\tau}$, rerouted traffic should also be reasonably close to Poisson traffic, while its contribution to the mean latency is quite small. The Poisson assumption allows us to use the well-known Pollaczek-Khinchin formula for the mean delay in an $M/G/1$ queue.

The expected latency of a job that is initially assigned to server pool i under the $S(A, \boldsymbol{\tau})$ rerouting policy is given by

$$\mathbb{E}\left[T_i^{\text{Rer}}(A, \boldsymbol{\tau})\right] \approx \frac{\lambda \mathbb{E}\left[\left(B_i^{\text{Rer}}(A, \boldsymbol{\tau})\right)^2\right]}{2\left(1 - \lambda \mathbb{E}\left[B_i^{\text{Rer}}(A, \boldsymbol{\tau})\right]\right)} + \mathbb{E}\left[X_i^{\text{Rer}}(\boldsymbol{\tau})\right] + \frac{\lambda \mathbb{E}\left[\left(B_l^{\text{Rer}}(A, \boldsymbol{\tau})\right)^2\right]}{2\left(1 - \lambda \mathbb{E}\left[B_l^{\text{Rer}}(A, \boldsymbol{\tau})\right]\right)} \sum_{j=1}^J p_j \mathbb{P}\left(\frac{X_i}{r_{ij}} > \tau_i\right),$$

where, for $i = 1, 2$, the expected service time requirement is given by Equation (2) and

$$\mathbb{E}\left[\left(B_i^{\text{Rer}}(A, \boldsymbol{\tau})\right)^2\right] = \sum_{j=1}^J p_j \alpha_{ij} \mathbb{E}\left[\left(\min\left\{\frac{X_i}{r_{ij}}, \tau_i\right\}\right)^2\right] + \sum_{j=1}^J p_j \alpha_{lj} \mathbb{E}\left[\left(\frac{X_i}{r_{ij}}\right)^2 \mathbb{1}\left\{\frac{X_l}{r_{lj}} > \tau_l\right\}\right],$$

with

$$\mathbb{E}\left[X_i^{\text{Rer}}(\boldsymbol{\tau})\right] = \sum_{j=1}^J p_j \left(\mathbb{E}\left[\min\left\{\frac{X_i}{r_{ij}}, \tau_i\right\}\right] + \mathbb{E}\left[\frac{X_l}{r_{lj}} \mathbb{1}\left\{\frac{X_i}{r_{ij}} > \tau_i\right\}\right]\right).$$

Indeed, an arriving job at server pool i has to wait for all the jobs present, and this mean delay is given by the Pollaczek-Khinchin formula for an $M/G/1$ queue. The expected service time requirement of the job before rerouting is equal to $\mathbb{E}\left[\min\left\{\frac{X_i}{r_{ij}}, \tau_i\right\}\right]$. With probability p_i^τ the job is rerouted to server pool l. Again the job has to wait for all the jobs present at this pool. Moreover, in this case the service time requirement of the job was τ_i on server pool i and the expected service time requirement is equal to $\mathbb{E}\left[\frac{X_l}{r_{lj}} \mathbb{1}\left\{\frac{X_i}{r_{ij}} > \tau_i\right\}\right]$ on pool l.

The expected latency of a job that is initially assigned to server pool i under the $S(A, \boldsymbol{\tau})$ replication policy is given by

$$\mathbb{E}\left[T_i^{\text{Rep}}(A, \boldsymbol{\tau})\right] \approx \frac{\lambda \mathbb{E}\left[\left(B_i^{\text{Rep}}(A, \boldsymbol{\tau})\right)^2\right]}{2\left(1 - \lambda \mathbb{E}\left[B_i^{\text{Rep}}(A, \boldsymbol{\tau})\right]\right)} + \mathbb{E}\left[X_i^{\text{Rep}}(\boldsymbol{\tau})\right],$$

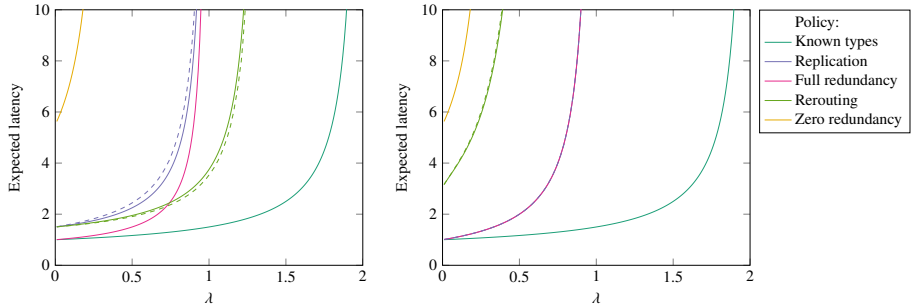

Figure 6: Expected latency for identical replicas in the scenario $I = 2$, $J = 2$, $N = 2$, $n_1 = n_2 = 1$, $(r_{11}, r_{21}) = (1, 0.1)$ and $(r_{12}, r_{22}) = (0.1, 1)$ with corresponding probabilities $p_1 = p_2 = 0.5$ when varying the arrival rate λ and degenerate (left) and exponential (right) service times. The approximations are depicted by the dashed lines. For degenerate job sizes, the replication policy is depicted with fixed $\boldsymbol{\tau} = \mathbf{1}$, while $\boldsymbol{\tau} = \mathbf{0}$ gives a lower expected latency. For exponential job sizes, the replication and full redundancy policy are overlapping.

where, for $i = 1, 2$, the expected service time requirement is given by Equation (4) and

$$\mathbb{E}\left[\left(B_i^{\mathrm{Rep}}(A,\boldsymbol{\tau})\right)^2\right] = \sum_{j=1}^{J} p_j \alpha_{ij}\left(\mathbb{E}\left[\left(\min\left\{\frac{X_i}{r_{ij}}, \tau_i\right\}\right)^2\right] + k^2_{il,j}(\tau_i)\right) + \sum_{j=1}^{J} p_j \alpha_{lj} k^2_{li,j}(\tau_l),$$

with

$$\mathbb{E}\left[X_i^{\mathrm{Rep}}(\boldsymbol{\tau})\right] = \sum_{j=1}^{J} p_j \left(\mathbb{E}\left[\min\left\{\frac{X_i}{r_{ij}}, \tau_i\right\}\right] + k_{il,j}(\tau_i)\right),$$

for $i = 1, 2$.

Indeed, an arriving job at server pool i has to wait for all the jobs present, and this mean delay is again given by the Pollaczek-Khinchin formula for an $M/G/1$ queue. The expected service time requirement of the job before replication is equal to $\mathbb{E}\left[\min\left\{\frac{X_i}{r_{ij}}, \tau_i\right\}\right]$. After replication the (remaining) expected service time requirement is $k_{il,j}(\tau_i)$.

To obtain the expected latency of an arbitrary job we can simply sum, the expected latency for a job that is initially assigned to server pool i multiplied by the probability of assigning this job to server pool i, over i. For example, in the rerouting policy we obtain

$$\mathbb{E}\left[T^{\mathrm{Rer}}(A,\boldsymbol{\tau})\right] = \sum_{i=1}^{2}\sum_{j=1}^{J} p_j \alpha_{ij} \mathbb{E}\left[T_i^{\mathrm{Rer}}(A,\boldsymbol{\tau})\right].$$

Numerical results

We provide numerical results to give further insights in the expression derived for the expected latency.

In Figure 6 the expected latency is depicted as a function of λ, for degenerate and exponentially distributed job sizes. The approximations of the expected latency (dashed lines) appear to be reasonably good. Depending on the variability of the job size distribution either the rerouting or replication policy achieves the lowest expected latency.

7. Conclusion and suggestions for further research

We have quantified the effective load per server for a system with two server pools when job types are completely unknown or partly known, but where jobs can either be rerouted or replicated to the other server pool after receiving service for some amount of time. From the numerical results we observed that in most of the scenarios rerouting nor replication increases the achievable stability bound. Moreover, we observed that for balanced server speeds the zero redundancy policy achieves the maximum achievable stability bound. We also observed that decreasing the uncertainty in job types increases the achievable stability bound in a convex manner.

Topics for further research include:

(i) Extension to multiple server pools. This is more involved than having two server pools, since after rerouting or replication there is the extra choice to which server pool(s).

(ii) Extensions of the analysis of the expected latency in Section 6. One could, e.g., use ideas from [5, 19] to approximate the departure process of a single-server queue. Moreover, note that analytic expressions for the expected latency are lacking in the case of redundancy and thus it is not known what the optimal expected latency is for generally distributed job sizes. The question how uncertainty of job types affects the expected latency remains interesting.

(iii) Optimization of the expressions for the achievable stability bound under the $S(A, \boldsymbol{\tau})$ policy.

(iv) Extension where the service does carry over. In this case the expected service time requirement for the rerouting policy becomes

$$\mathbb{E}\left[B_i^{\mathrm{Rer}}(A,\boldsymbol{\tau})\right] = \sum_{j=1}^{J} p_j \alpha_{ij} \mathbb{E}\left[\min\left\{\frac{X}{r_{ij}}, \tau_i\right\}\right] + \sum_{j=1}^{J} p_j \alpha_{lj} \mathbb{E}\left[\frac{X - r_{lj}\tau_l}{r_{ij}} \mathbb{1}\left\{\frac{X}{r_{lj}} > \tau_l\right\}\right],$$

and for the replication policy

$$\mathbb{E}\left[B_i^{\mathrm{Rep}}(A,\boldsymbol{\tau})\right] = \sum_{j=1}^{J} p_j \alpha_{ij} \left(\mathbb{E}\left[\min\left\{\frac{X}{r_{ij}}, \tau_i\right\}\right] + k_{il,j}(\tau_i)\right) + \sum_{j=1}^{J} p_j \alpha_{lj} k_{li,j}(\tau_l).$$

where

$$k_{il,j}(y) = \mathbb{E}\left[\min\left\{\frac{X}{r_{ij}} - y, \frac{X - r_{ij}\tau_i}{r_{lj}}\right\} \mathbb{1}\left\{\frac{X}{r_{ij}} > y\right\}\right].$$

Acknowledgments

The work in this paper is supported by the Netherlands Organisation for Scientific Research (NWO) through Gravitation grant NETWORKS 024.002.003.

References

[1] M.F. Aktas, P. Peng, and E. Soljanin. Effective straggler mitigation: When clones should attack and when? *ACM SIGMETRICS Performance Evaluation Review*, 45(2):12–14, 2017.

[2] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica. Effective straggler mitigation: Attack of the clones. *NSDI'13 Proceedings of the 10th USENIX conference on Networked Systems Design and Implementation*, 11:185–198, 2013.

[3] G. Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoica, Y. Lu, B. Saha, and E. Harris. Reining in the outliers in map-reduce clusters using mantri. *OSDI'10 Proceedings of the 9th USENIX conference on Operating Systems Design and Implementation*, pages 265–278, 2010.

[4] K. Bimpikis and M.G. Markakis. Learning and hierarchies in service systems. *Management Science*, 65(3):1268–1285, 2018.

[5] G.R. Bitran and D. Tirupati. Multiproduct queueing networks with deterministic routing: Decomposition approach and the notion of inference. *Management Science*, 34(1):75–100, 1988.

[6] A.A. Borovkov. *Stochastic Processes in Queueing Theory.* Springer, 1976.

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

[8] K. Gardner, M. Harchol-Balter, A. Scheller-Wolf, and B. Van Houdt. A better model for job redundancy: Decoupling server slowdown and job size. *IEEE ACM Transactions on Networking*, 25(6):3353–3367, 2017.

[9] K. Gardner and C. Stephens. Smart dispatching in heterogeneous systems. *ACM SIGMETRICS Performance Evaluation Review*, 47(2):12–14, 2019.

[10] G. Joshi. *Efficient Redundancy Techniques to Reduce Delay in Cloud Systems*. PhD thesis, Massachusetts Institute of Technology, 2016.

[11] G. Joshi. Boosting service capacity via adaptive task replication. *ACM SIGMETRICS Performance Evaluation Review*, 45(2):9–11, 2017.

[12] F.P. Kelly and E. Yudovina. *Stochastic Networks.* Cambridge University Press, 2014.

[13] Y. Kim, R. Righter, and R. Wolff. Job replication on multiserver systems. *Advances in Applied Probability*, 41(2):546–575, 2009.

[14] G. Koole and R. Righter. Resource allocation in grid computing. *Journal of Scheduling*, 11:163–173, 2008.

[15] Y. Raaijmakers, S.C. Borst, and O.J. Boxma. Achievable stability in redundancy scheduling. *Paper in preparation*, 2020.

[16] A.L. Stolyar. Optimal routing in output-queued flexible server systems. *Probability in the Engineering and Informational Sciences*, 19(2):141–189, 2005.

[17] A. Vulimiri, P.B. Godfrey, R. Mittal, J. Sherry, S. Ratnasamy, and S. Shenker. Low latency via redundancy. *CoNEXT'13 Proceedings of the 9th ACM conference on Emerging Networking Experiments and Technologies*, pages 283–294, 2013.

[18] J. Walrand. *An Introduction to Queueing Networks.* Prentice Hall, 1988.

[19] W. Whitt. Approximations for departure processes and queues in series. *Naval Research Logistics Quarterly*, 31(4):499–521, 1984.

Appendix A. Additional numerical results

In this appendix we present additional numerical results for the achievable stability bound in case of Pareto Type I distributed job sizes. For Figures A.7-A.10 the same scenarios as in Figures 2-5 are considered, but now for Pareto distributed job sizes, with minimum possible value 1 and index $\frac{N}{N-1}$ instead of exponentially distributed job sizes with mean N.

Figure A.7 reveals that, when comparing the policies for Pareto and exponentially distributed job sizes, the achievable stability bound for the rerouting policy performs worse for Pareto, the other policies achieve the same achievable stability bound. Only the rerouting policy performs worse for Pareto and (highly) unbalanced server speeds.

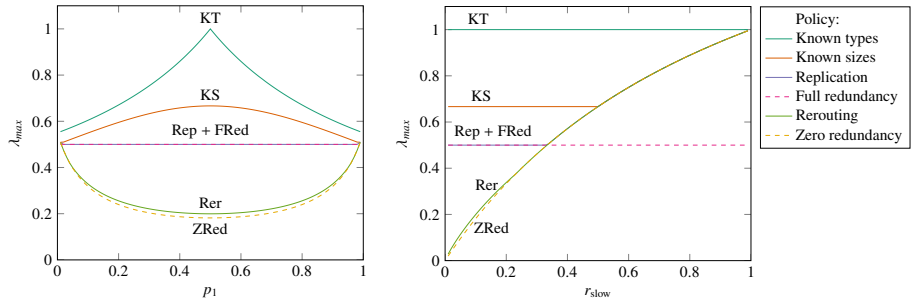

Figure A.7: Achievable stability bound for identical replicas with Pareto distributed job sizes in the same scenario as Figure 2, i.e., $I = 2$, $J = 2$, $N = 10$, $n_1 = n_2 = 5$, $(r_{11}, r_{21}) = (1, r_{\text{slow}})$ and $(r_{12}, r_{22}) = (r_{\text{slow}}, 1)$ with corresponding probabilities p_1 and $p_2 = 1 - p_1$, with $r_{\text{slow}} = 0.1$ and $p_1 = p_2 = 0.5$ (right). In the left sub-figure, the replication and full redundancy policy are overlapping.

Figure A.8 shows that, when comparing the policies for Pareto and exponentially distributed job sizes, the known sizes, replication, full redundancy and rerouting policy all perform better for Pareto distributed job sizes. Similar to Figure 3, the replication and full redundancy policy both outperform the rerouting policy.

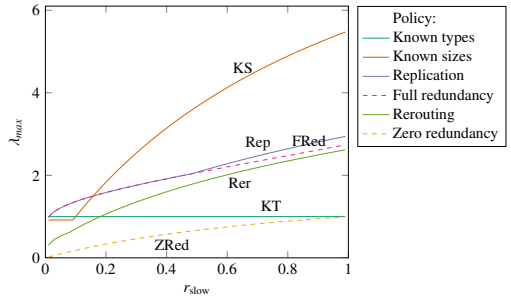

Figure A.8: Achievable stability bound for i.i.d. replicas with Pareto distributed job sizes in the same scenario as Figure 3, i.e., $I = 2$, $J = 2$, $N = 10$, $n_1 = n_2 = 5$, $(r_{11}, r_{21}) = (1, r_{\text{slow}})$ and $(r_{12}, r_{22}) = (r_{\text{slow}}, 1)$ with corresponding probabilities $p_1 = p_2 = 0.5$.

Figure A.9 indicates that, when comparing the policies for Pareto and exponentially distributed job sizes, for unbalanced server speeds, both the replication and rerouting policy perform worse for Pareto distributed job sizes. For unbalanced server speeds Figure A.9 shows that the

known sizes, replication, rerouting and zero redundancy policy all achieve the same achievable stability bound. The numerical results for the exponential and Pareto job sizes suggest that these policies are insensitive to the job size distribution, given its mean.

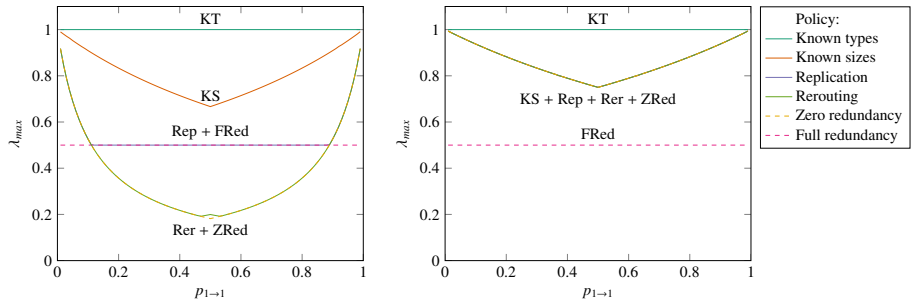

Figure A.9: Achievable stability bound for identical replicas with Pareto distributed job sizes in the same scenario as Figure 4, i.e., $I = 2$, $J = 2$, $N = 10$, $n_1 = n_2 = 5$, $(r_{11}, r_{21}) = (1, r_{\text{slow}})$ and $(r_{12}, r_{22}) = (r_{\text{slow}}, 1)$ with corresponding probabilities $p_1 = p_2 = 0.5$, with $r_{\text{slow}} = 0.1$ (left) and $r_{\text{slow}} = 0.6$ (right) when varying the probability $p_{1\to1} = p_{2\to2}$.

Figure A.10 reveals that, when comparing the policies for Pareto and exponentially distributed job sizes, the known sizes, replication, full redundancy and rerouting policy all perform better for Pareto distributed job sizes. This is due to the assumption of i.i.d. replicas and the heavy tail of the Pareto distribution.

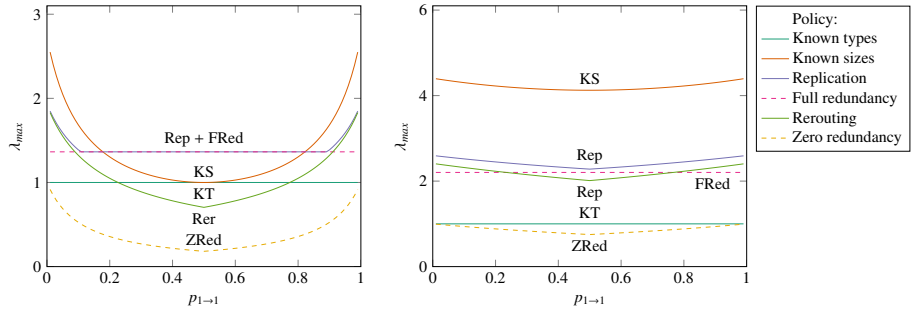

Figure A.10: Achievable stability bound for i.i.d. replicas with Pareto distributed job sizes in the same scenario as Figure 5, i.e., $I = 2$, $J = 2$, $N = 10$, $n_1 = n_2 = 5$, $(r_{11}, r_{21}) = (1, r_{\text{slow}})$ and $(r_{12}, r_{22}) = (r_{\text{slow}}, 1)$ with corresponding probabilities $p_1 = p_2 = 0.5$, with $r_{\text{slow}} = 0.1$ (left) and $r_{\text{slow}} = 0.6$ (right) when varying the probability $p_{1\to1} = p_{2\to2}$.

Appendix B. Load of the system: *n*-threshold policy

Unknown job sizes

In this appendix we extend the expressions for the effective load per server, in the case of unknown job sizes, of the rerouting policy, given by Proposition 1, by allowing for multiple reroutings (see Definition 1). For the rerouting policy this means that after assigning a job to server pool i, we can reroute the job to the other pool. In contrast to the threshold policy we can reroute the job back to the pool it was initially assigned to. In the n-threshold policy we can in

total reroute n times from one pool to the other.

As before, let A denote the $2 \times J$ stochastic assignment matrix with elements α_{ij}. Let T be the $2 \times n$ rerouting matrix, with rows $(\tau_{i1}, \tau_{i2}, \ldots, \tau_{in})$ denoting the rerouting vector for the job initially assigned to server pool i, for $i = 1, 2$. We define $\tau_{i0} = 0$ for $i = 1, 2$.

Proposition 5. *The effective load per server in pool i, for $i = 1, 2$, in the system with rerouting under the $S(A, T)$ policy is*

$$\rho^{Rer}_{i,S(A,T)} = \frac{\lambda \mathbb{E}\left[B^{Rer}_i(A,T)\right]}{n_i},$$

where,

$$\begin{aligned}\mathbb{E}\left[B^{Rer}_i(A,T)\right] = &\sum_{j=1}^{J} p_j \alpha_{ij} \mathbb{E}\left[\min\left\{\frac{X_i}{r_{ij}}, \tau_{i1}\right\}\right] \\ &+ \sum_{m=1}^{n-1}\sum_{j=1}^{J} p_j \alpha_{I(m)j} \mathbb{E}\left[\min\left\{\frac{X_i}{r_{ij}}, \tau_{I(m)m+1}\right\} \cdot \mathbb{1}\left\{\tau_{I(m)m-1} < \frac{X_i}{r_{ij}}, \tau_{I(m)m} < \frac{X_l}{r_{lj}}\right\}\right] \\ &+ \sum_{j=1}^{J} p_j \alpha_{I(n)j} \mathbb{E}\left[\frac{X_i}{r_{ij}} \mathbb{1}\left\{\tau_{I(n)n-1} < \frac{X_i}{r_{ij}}, \tau_{I(n)n} < \frac{X_l}{r_{lj}}\right\}\right],\end{aligned}$$

where $I(m) = l$ if m is odd, and $I(m) = i$ if m is even.

Known job sizes

Proposition 6. *The effective load per server in pool i, for $i = 1, 2$, in the system with rerouting under the $S(A^x, T^x)$ policy, in case of known job sizes, is*

$$\rho^{Rer,KS}_{i,S(A^x,T^x)} = \frac{\lambda \mathbb{E}\left[B^{Rer,KS}_i(A^x,T^x)\right]}{n_i},$$

where,

$$\begin{aligned}\mathbb{E}\left[B^{Rer,KS}_i(A^x,T^x)\right] = \iint_{\mathbb{R}^2} \Bigg(&\sum_{j=1}^{J} p_j \alpha^x_{ij} \min\left\{\frac{x_i}{r_{ij}}, \tau^x_{i1}\right\} \\ &+ \sum_{m=1}^{n-1}\sum_{j=1}^{J} p_j \alpha^x_{I(m)j} \min\left\{\frac{x_i}{r_{ij}}, \tau^x_{I(m)m+1}\right\} \cdot \mathbb{1}\left\{\tau^x_{I(m)m-1} < \frac{x_i}{r_{ij}}, \tau^x_{I(m)m} < \frac{x_l}{r_{lj}}\right\} \\ &+ \sum_{j=1}^{J} p_j \alpha^x_{I(n)j} \frac{x_i}{r_{ij}} \mathbb{1}\left\{\tau^x_{I(n)n-1} < \frac{x_i}{r_{ij}}, \tau^x_{I(n)n} < \frac{x_l}{r_{lj}}\right\}\Bigg) f_X(x_1, x_2) dx_1 dx_2.\end{aligned}$$