# Delta probing policies for redundancy

Youri Raaijmakers
Eindhoven University of Technology
Eindhoven, The Netherlands
y.raaijmakers@tue.nl

Sem Borst
Eindhoven University of Technology
Eindhoven, The Netherlands
Nokia Bell Labs
Murray Hill, NJ, USA
s.c.borst@tue.nl

Onno Boxma
Eindhoven University of Technology
Eindhoven, The Netherlands
o.j.boxma@tue.nl

## ABSTRACT

We consider job dispatching in systems with $N$ parallel servers. In redundancy-$d$ policies, replicas of an arriving job are assigned to $d \leq N$ servers selected uniformly at random (without replacement) with the objective to reduce the delay. We introduce a quite general workload model, in which job sizes have some probability distribution while the speeds (slowdown factors) of the various servers for a given job are allowed to be inter-dependent and non-identically distributed. This allows not only for inherent speed differences among different servers, but also for affinity relations. We further propose two novel redundancy policies, so-called delta-probe-$d$ policies, where $d$ probes of a fixed, small, size $\Delta$ are created for each incoming job, and assigned to $d$ servers selected uniformly at random. As soon as the first of these $d$ probe tasks finishes, the actual job is assigned for execution – with the same speed – to the corresponding server and the other probe tasks are abandoned. We also consider a delta-probe-$d$ policy in which the probes receive preemptive-resume priority over regular jobs. The aim of these policies is to retain the benefits of redundancy-$d$ policies while accounting for systematic speed differences and mitigating the risks of running replicas of the full job simultaneously for long periods of time. Analytical and numerical results are presented to evaluate the effect of both probing policies on the job latency, and to illustrate the potential performance improvements.

## KEYWORDS

Parallel-server system, dispatching, redundancy, probing policies, delay performance

**ACM Reference Format:**
Youri Raaijmakers, Sem Borst, and Onno Boxma. 2018. Delta probing policies for redundancy. In *Proceedings of Performance Evaluation (Toulouse'18)*. ACM, New York, NY, USA, 11 pages. https://doi.org/10.475/123_4

## 1 INTRODUCTION

Redundancy scheduling has emerged as a powerful paradigm for improving the delay performance in parallel-server systems, and has attracted strong interest in recent years [2–4, 7–10, 15, 17, 20]. The key element in redundancy scheduling is job replication where replicas of each incoming job are dispatched to, say $d$, different servers. Redundant copies are then abandoned as soon as the first of these $d$ replicas either enters service ('cancel-on-start') or finishes service ('cancel-on-completion').

Note that the 'cancel-on-start' variant is equivalent to a strategy which assigns an arriving job to the server with the currently smallest workload among $d$ servers selected uniformly at random (assuming each of the individual servers adheres to a FCFS discipline) without generating any replicas. Such a load balancing strategy is also commonly referred as a JSW($d$) policy, reflecting the connection with the ordinary Join-the-Shortest-Workload (JSW) policy which assigns each job to the server with the minimum current workload. This latter criterion (myopically) minimizes the waiting time of each incoming job, and emulates the operation of a centralized queue with a global FCFS discipline. Likewise, the 'cancel-on-completion' (c.o.c.) version shares some similarity with a strategy which assigns an incoming job to the server that will provide the minimum sojourn time among $d$ servers selected uniformly at random. Viewed from this angle, redundancy scheduling is also closely related to a JSQ($d$) policy which assigns an incoming job to the server with the currently shortest queue length among $d$ servers selected uniformly at random [13, 14, 21].

By virtue of the "power-of-$d$ choices", redundancy scheduling inherits the strong merits of JSW($d$) and JSQ($d$) policies in improving the delay performance. A major advantage of a redundancy strategy is that it does not rely on workload information or advance knowledge of service times, and does not entail any communication overhead upon arrival of a job. On the flip side, a redundancy strategy involves generating replicas and incurs a communication burden in abandoning redundant copies once the first replica has entered or finished service.

Note that in the c.o.c. version several replicas of the same job may be in service concurrently until the first replica finishes service. This unique feature of redundancy scheduling is not shared with JSW($d$) and JSQ($d$) policies, and may either yield additional performance gains or result in potential vulnerabilities. Indeed, having several replicas of the same job in service concurrently entails a wastage of service capacity in a certain sense. However, this situation also creates an opportunity to complete service of a short replica and avoid full service of a long replica if the run times of the various replicas are different, thus reducing the total amount of service capacity devoted to the job.

As it turns out, when the run times of the various replicas are independent and exponentially distributed, which is the prevalent case considered in the literature [5, 10, 16, 20], the collective amount of work spent on completing a job remains exponentially distributed, no matter how the services of different replicas may overlap in time. When the run times of the various replicas are


*Toulouse'18, December 2018, France*

ACM ISBN 123-4567-24-567/08/06…$15.00
https://doi.org/10.475/123_4

independent and highly variable (e.g., heavy-tailed), concurrent service provides some degree of immunity from extremely long run times, and even tends to reduce the total amount of service capacity devoted to a job. In contrast, when the run times are less variable or highly variable but correlated, which is the typical case observed in practice, concurrent service can cause a substantial wastage of service capacity, and at high degrees of replication it can in fact nullify the performance gains from the "power-of-$d$ choices" and adversely affect stability properties. Thus the performance gains from the c.o.c. redundancy policy sensitively depend on the characteristics of the joint distribution of the run times of the various replicas.

Gardner *et al.* [7] introduced the so-called S&X model to capture the typical correlation among the run times of various replicas as induced by the intrinsic size of a job. The analysis in [7] demonstrated that higher degrees of replication can degrade the delay performance for certain run time distributions, and even give rise to instability issues, corroborating the above observations. Gardner *et al.* [7] also proposed a refinement of the c.o.c. redundancy policy termed 'Redundant-to-Idle-Queue' (RIQ), which only assigns replicas to the idle servers among the $d$ sampled servers, if any, and otherwise simply assigns the job to just one of these $d$ servers selected uniformly at random. It was shown in [7] that the RIQ policy provides similar performance gains as standard redundancy scheduling policies, but prevents the potential repercussions and in particular instability issues at higher degrees of correlations for certain run time distributions.

While the S&X model accounts for the typical correlation among the various run times induced by the intrinsic size of a job (the '$X$'), it adheres to the assumption that the slowdown factors at the various servers (the '$S$' components) are independent and identically distributed. In particular, the special case of deterministic job sizes $X$ is equivalent to the independent run times (IR) model, which had been prevalent in the earlier literature.

The present paper aims to make two contributions to the modeling and analysis of redundancy-$d$ policies. The *first contribution* is that we examine a more general workload model which relaxes the assumption of i.i.d. slowdown factors, allowing the slowdown factors to be inter-dependent and possibly non-identically distributed. This allows not only for inherent speed differences among different servers, but also for 'affinity' relations where some types of jobs can be handled more efficiently by some of the servers than others [18].

Although redundancy scheduling policies can handle random variations in slowdown factors among statistically homogeneous servers, it turns out that they are not well-suited to deal with systematic speed differences among heterogeneous servers, let alone affinity relations. The *second contribution* of the present paper is to introduce two novel policies which are able to handle not only random fluctuations in slowdown factors, but also structural forms of heterogeneity in server speed and fundamental job-server compatibility characteristics. In these delta-probe-$d$ policies, $d$ probes of a fixed, small, size $\Delta$ are created for each incoming job, and assigned to $d$ servers selected uniformly at random. As soon as the first of these $d$ probes finishes, the actual job is assigned for execution – with the same speed – to the corresponding server and the other probe tasks are abandoned. The aim of these policies is to retain the benefits of redundancy-$d$ policies while accounting for systematic speed differences (e.g., in the scenario of job classes) and mitigating the risks of running replicas of the full job simultaneously for long periods of time. To reduce the chance that a probe on a slow server is finished before other probes on faster servers have completed their service, we also consider a delta-probe-$d$ policy in which the probes receive preemptive-resume priority over regular jobs.

Analytical and numerical results are presented to investigate the effect of both probing policies on the job latency (the time until a job is completed) and demonstrate the potential performance improvements. In particular for $\Delta \downarrow 0$ quite explicit delay results can be derived. We show that both delta-probe-$d$ policies tend to outperform the c.o.c. redundancy policy in terms of expected latency. More specifically, for $d = N$ and $\Delta$ small enough it can be proved that the delta-probe-$d$ policy with preemptive-resume priority outperforms the c.o.c. policy in terms of expected latency, and has a larger stability region. The delta-probe-$d$ policy without preemptive-resume priority usually performs better than the policy with preemptive-resume priority in the case of balanced server speed distributions, but the situation is reversed in the case of unbalanced server speed distributions.

The remainder of the paper is organized as follows. In Section 2 we present a detailed model description, along with a further specification of the various delta-probe-$d$ policies under consideration. Analytical results for the delay performance of these policies are derived in Section 3. Section 4 reports on extensive numerical experiments, in which the performance of the two delta-probe-$d$ policies is compared with that of other policies such as c.o.c. and RIQ. Section 5 contains conclusions and some suggestions for further research.

## 2 SYSTEM DESCRIPTION AND WORKLOAD MODEL

We consider a system with a single dispatcher and $N$ parallel servers, each of which has its own queue and follows a FCFS discipline. Jobs arrive according to a Poisson process of rate $\lambda$, and have sizes that are independent and identically distributed copies of some generic random variable $X$, with distribution function $X(\cdot)$. When a job arrives, the dispatcher immediately assigns possibly multiple replicas to one or more servers according to a specific policy which will be further detailed below. In case replicas are assigned to several servers, the intrinsic job size is preserved, but the service speeds $R_1, \ldots, R_N$ on the various servers, and hence the corresponding run times, may differ. For a particular job with a given size $x$, $B_i = x/R_i$ represents the run time when executed by server $i$, $i = 1, \ldots, N$.

We allow the service speeds $R_1, \ldots, R_N$ of a generic job to be governed by some joint distribution $F(r_1, \ldots, r_N)$, reflecting possible server heterogeneity and job-server compatibility characteristics, thus covering a broad range of common workload models as special cases. We can broadly distinguish between two scenarios depending on whether the service speeds $R_1, \ldots, R_N$ of a generic job are identically distributed or not, see Figure 1.

When the service speeds are identically distributed, a special case is when they are in fact exchangeable random variables. The case where the service speeds are then additionally independent

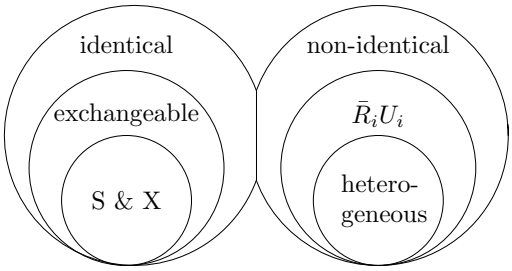


**Figure 1: Overview server speed scenarios**

corresponds to the S&X model mentioned earlier, where it is typically further assumed that the service speeds are smaller than or equal to 1. This assumption ensures that the run time can only be stretched by the service speed compared to the intrinsic job size $X$. The latter job size in our model corresponds exactly to the job size $X$ in the S&X model, while the slowdown factors $S_1, ..., S_N$ in the S&X model correspond to the reciprocal values of the service speeds $R_1, ..., R_N$.

When the service speeds are non-identically distributed, a special case is where they are still identically distributed up to rescaling, i.e., of the form $(R_1, \ldots, R_N) = (\bar{R}_1 U_1, \ldots, \bar{R}_N U_N)$, where $\bar{R}_1, \ldots, \bar{R}_N$ are non-identical coefficients and the random variables $U_1, \ldots, U_N$ are identically distributed. The case where the latter random variables are all equal to the same constant may be interpreted as a scenario with heterogeneous server speeds.

For notational convenience, we consider the case where the joint distribution $F(r_1, \ldots, r_N)$ is discrete, and has mass in a finite number of say $M$ points $(r_{1m}, ..., r_{Nm})$ with corresponding probabilities $p_m$, $m = 1, \ldots, M$. This system may equivalently be thought of as having $M$ job classes, where $r_{nm}$ is the service speed of class-$m$ jobs at the $n$-th server.

The scenario of job classes in turn subsumes the 'output-queued' flexible server model introduced in [19] as a special case. In the latter model the service time of a class-$j$ job at the $i$-th server is exponentially distributed with parameter $\mu_{ij}$, $i = 1, \ldots, N$, $j = 1, \ldots, M$. This corresponds to an exponentially distributed job size with unit mean and $r_{ij} = 1/\mu_{ij}$, $i = 1, \ldots, N$, $j = 1, \ldots, M$, in our model.

We now proceed to describe the redundancy-$d$ policies and the novel delta-probe-$d$ policies that we examine in the remainder of the paper.

## 2.1 Redundancy-$d$ policies

As stated in the introduction, in redundancy-$d$ policies replicas of an arriving job are assigned to $d \leq N$ servers selected uniformly at random (without replacement). We will focus on the cancel-on-completion (c.o.c.) variant, where redundant copies are instantaneously abandoned as soon as the first of the $d$ replicas finishes.

We will also consider the Redundant-to-Idle-Queue (RIQ) policy, which only assigns replicas to the idle servers among the $d$ randomly selected servers, if any, and otherwise simply assigns the job to one of these $d$ servers selected uniformly at random. The performance of the RIQ policy was studied in [7] in a regime with a large number of servers, under an "Asymptotically Independent Idleness" assumption, meaning that the probability that a particular server is idle does not depend on the idle-busy status of any finite subset of other servers (e.g., the other $d - 1$ randomly selected servers for an incoming job).

## 2.2 Delta-probe-$d$ policies

In delta-probe-$d$ policies, $d$ probe tasks of size $\Delta$ are created for each incoming job. Just like in other redundancy-$d$ policies, these probe tasks are then assigned to $d \leq N$ servers selected uniformly at random (without replacement). The goal is to infer how attractive these servers are to handle the job in question. We assume that the observed speeds at which the probe task is executed by the various servers are representative of what their speeds would be for the actual job, if selected for execution. As soon as the first of the $d$ probe tasks finishes, the actual job is assigned to the corresponding server, and the redundant probe tasks are instantaneously abandoned. Observe that the choice of the server for execution of a given job, depends on the combination of the speeds of the $d$ selected servers and their instantaneous workloads, similarly as in redundancy scheduling. (The workload here refers to the aggregate run time, i.e., the size divided by the speed, of both probe tasks and regular jobs at a particular server.) However, when the probe tasks are relatively small, concurrent service of the same job by several servers for an extended period of time is prevented.

We distinguish between two versions of the delta-probe-$d$ policy, depending on whether the probe tasks receive (preemptive) priority over regular jobs or not.

*2.2.1 Standard version.* In the standard version, also referred to as delta-probe policy, each server handles probe tasks and regular jobs in order of arrival. In this case, the selection of a server for a given job primarily depends on the instantaneous workloads of the $d$ selected servers, provided the probe tasks are relatively small. Indeed, as $\Delta$ tends to zero, the standard version reduces to the JSW($d$) policy which simply assigns each job to the server with the smallest workload among the $d$ sampled servers, without paying any attention to the speeds of the servers for that particular job.

*2.2.2 Preemptive version.* In the preemptive version, also referred to as the delta-probe preemption policy, probe tasks receive preemptive-resume priority over regular jobs. Thus each server operates as in a two-class preemptive-priority system with a First-Come First-Served discipline within each class. In this case, the selection of a server for a given job primarily depends on the speeds of the $d$ selected servers for that particular job, as long as the probe tasks are relatively small. Indeed, as $\Delta$ tends to zero, the preemptive version behaves as a policy which simply assigns each job to the server with the highest speed among the $d$ selected servers for that particular job, without taking the workloads into consideration.

Remark 1. *In the case $\Delta \downarrow 0$ and $d = N$ the delta-probe preemption policy always outperforms the c.o.c. redundancy policy, which intuitively can be seen as follows. In the c.o.c. policy the system is always in perfect synchronicity, i.e., the workloads are equal for all servers, and the replica with the smallest run time is always completed*

*first. In fact, the behavior of a queue under the c.o.c. redundancy policy for $d = N$ reduces to that of an $M/G/1$ queue. In the delta-probe preemption policy the replica with the smallest run time is also completed first, but here only the actual job is taken into service. The run times are thus equal under both policies, but in the c.o.c. policy the job has to wait for all the jobs in the system whereas in the delta-probe preemption policy the job has to wait only for all the jobs on the server that is fastest for this specific job. As a consequence, the waiting time under the delta-probe preemption policy is no larger than under the c.o.c. policy sample-pathwise and thus we can conclude that the delta-probe preemption policy has a larger stability region and a lower expected latency than the c.o.c. policy.*

# 3 DELAY PERFORMANCE OF THE DELTA-PROBE-$d$ POLICIES

In this section we analyze the delay performance of the delta-probe-$d$ policies. Exact analysis of both delta-probe-$d$ policies is in general extremely difficult due to the fact that the assignment of the actual job to a specific server needs to be known. It can be observed that in the delta-probe policy this assignment depends on the joint workloads of the $d$ sampled servers, which cannot be explicitly derived. For this policy, we are only able to derive exact expressions for the key performance measures, such as the stability region and expected delay, in specific scenarios. Section 3.1 is devoted to this policy.

In the delta-probe preemption policy the assignment of an actual job to a server seems more tractable, since the job itself is typically assigned to the fastest server, when $\Delta \downarrow 0$. For arbitrary $\Delta$ there are exceptions which are discussed in Section 3.2. For this policy we obtain explicit expressions for the expected delay and, when $\Delta \downarrow 0$, for the delay distribution in general settings.

## 3.1 The delta-probe policy

An exact analysis of the delta-probe policy for arbitrary $\Delta$ seems out of reach. Even in the simple setting with $d = N = 2$ and $M = 2$ the analysis appears to be intractable. Hence, we will mostly resort to simulation to gain insight in the performance of this policy; the results are presented in Section 4.

As $\Delta$ approaches zero, this policy becomes more tractable though. Indeed, when $\Delta \downarrow 0$, the behavior reduces to that in an $M/G_1, G_2, ..., G_N/N$ system with a JSW($d$) policy, where $G_i$ denotes the generic service times of server $i$, but with one exceptional property in case of at least two idle servers. In this case the actual job is always assigned to the fastest server among these idle servers, instead of uniformly at random. From simulation it is seen that this property does not significantly influence the performance, i.e., the expected latency, at high load, since the servers are then almost always non-idle. When $d = N$ this model can equivalently be thought of as an $M/G_1, G_2, ..., G_N/N$ system with a centralized queue, again with the above-mentioned exception. The stability condition for this model is obviously given by

$$\rho := \frac{\lambda}{\sum_{n=1}^{N} \frac{1}{\mathbb{E}[G_n]}} < 1. \tag{1}$$

Remark 2. *In the setting where $d = N = 2$ and $\Delta \downarrow 0$, the delta-probe policy can be exactly analysed via the approach given in [12].*

Remark 3. *For identically distributed service speeds, $d = N$ and $\Delta \downarrow 0$, the behavior reduces to that in an $M/G/N$ queue. For the latter system there are several approximations known for the expected latency, see [6] for a simple approximation.*

## 3.2 The delta-probe preemption policy

The delta-probe preemption policy with positive $\Delta$ turns out to be more tractable than the delta-probe policy. Below we analyze its delay performance, but first two comments are in order.
(i) For notational convenience, we assume that the server speeds satisfy $r_{im} \neq r_{km}$ for all $1 \leq i \neq k \leq N$, $1 \leq m \leq M$. In Remark 4 we discuss how this assumption can be relaxed at the expense of somewhat more elaborate notation.
(ii) In the analysis we assume that each job is assigned to the fastest server for that job. This assumption holds exactly for $d = N$, and results in a very accurate approximation for $d < N$, where the following issue arises. As $\Delta \downarrow 0$, the $d$ sampled servers always start simultaneously with the service of the probe task, and therefore the fastest server completes that probe task first. However, if a new job arrives during the service of a probe task, and there is overlap between the sets $Z_{old}$ and $Z_{new}$ of $d$ sampled servers of the old job and of the new job, then the probe task of the new job does not get priority over the probe task of the old job. Hence at some servers that new probe task will not immediately be initiated, whereas it might already start on other servers that do not belong to $Z_{old}$. Thus it may happen that the new job is not assigned to the fastest server. However, this event happens with small probability for $\Delta$ sufficiently small (and it cannot happen when $d = N$). In Section 4 we corroborate the accuracy of the approximation (ignoring this event) via simulation.

Let us now determine the expected latency of an arbitrary job at an arbitrary server, say server $i$. Observe that the workload at server $i$ behaves exactly like the workload $V_i$ in a particular $M/G/1$ queue (there is dependence between the speeds of the $N$ servers, but because the slowdown factors of successive jobs are chosen independently, successive run times on an arbitrary server are independent). It has an arrival rate

$$\eta_i = \frac{\lambda d}{N},$$

and generic run time $G_i$, to be specified below, and load $\rho_i := \eta_i \mathbb{E}[G_i]$; hence, from $M/G/1$ theory (cf. [11]),

$$\mathbb{E}[V_i] = \frac{\eta_i \mathbb{E}[G_i^2]}{2(1-\rho_i)}. \tag{2}$$

By the PASTA property, $\mathbb{E}[V_i]$ also equals the expected amount of work found by an arriving job. Below we first evaluate $\mathbb{E}[V_i]$, and subsequently use that expression to obtain the expected latency $\mathbb{E}[T_i]$ of a job that is actually served at server $i$. Let $G_i^B$ denote the total run time of probe plus job, of size $\Delta + X$, given that the job is indeed executed on server $i$. Then a flow balance argument (expected latency equals $\mathbb{E}[V_i] + \mathbb{E}[G_i^B]$ plus the expected amount of high-priority work arriving at server $i$ during $T_i$) implies

$$\mathbb{E}[T_i] = \mathbb{E}[V_i] + \mathbb{E}[G_i^B] + \eta_i \mathbb{E}[D_i]\mathbb{E}[T_i],$$

where $D_i$ denotes the time that $\Delta$ is in service at server $i$ (before one of the $d$ replicas of $\Delta$ is finished). Hence

$$\mathbb{E}[T_i] = \frac{\mathbb{E}[V_i] + \mathbb{E}[G_i^B]}{1 - \eta_i \mathbb{E}[D_i]}. \tag{3}$$

We will successively determine all the components of (2) and (3). Let $\boldsymbol{y}$ denote the vector $(y_1, y_2, ..., y_d)$ and let $S_N$ denote all possible combinations of the $d$ sampled servers, which is defined as $S_N = \{\boldsymbol{y} \in \mathbb{N}^d : 1 \le y_1 < y_2 < \ldots < y_d \le N\}$. Here, $\boldsymbol{y} \in S_N$ specifies to which $d$ servers the probes are assigned. Observe that at a specific server we can distinguish three types of arrivals:

1 An arrival for which a probe is dispatched to the server of interest and also the job itself is executed by this server – so the server serves $\Delta + X$,
2 An arrival for which a probe is dispatched to the server of interest but the job itself is executed at another server – so the server only serves (part of) $\Delta$,
3 An arrival where the $d$ probes are dispatched to $d$ other servers.

Note that only job arrival types 1 and 2 contribute to the workload of this specific server.

*Arrival rates*

If we denote the arrival rates of types 1 and 2 at server $i$ by $\lambda_i^+$ and $\lambda_i^-$, respectively, then

$$\lambda_i^+ = \frac{\lambda}{\binom{N}{d}} \sum_{\boldsymbol{y} \in S_N} \sum_{m=1}^{M} p_m \mathbb{1}_{i,\boldsymbol{y}m}^+, \tag{4}$$

and

$$\lambda_i^- = \frac{\lambda}{\binom{N}{d}} \sum_{\boldsymbol{y} \in S_N} \sum_{m=1}^{M} p_m \mathbb{1}_{i,\boldsymbol{y}m}^-,$$

where $\mathbb{1}_{i,\boldsymbol{y}m}^+$ is the indicator function that server $i$ is the fastest for this specific job:

$$\mathbb{1}_{i,\boldsymbol{y}m}^+ = \mathbb{1}_{\{r_{im} = \max\{r_{y_1 m}, r_{y_2 m}, \ldots, r_{y_d m}\}\}}, \tag{5}$$

and

$$\mathbb{1}_{i,\boldsymbol{y}m}^- = \mathbb{1}_{\{r_{im} \neq \max\{r_{y_1 m}, r_{y_2 m}, \ldots, r_{y_d m}\},\ i \in \{y_1, y_2, \ldots, y_d\}\}}. \tag{6}$$

Notice that $\lambda_i^+ + \lambda_i^- = \frac{\lambda d}{N} = \eta_i$.

*Run times*

The run time of a type-1 job, i.e., a job that is actually executed by server $i$, has $j$-th moment, for $j = 1, 2, \ldots$,

$$\mathbb{E}[(G_i^B)^j] = \sum_{\boldsymbol{y} \in S_N} \sum_{m=1}^{M} p_{i,\boldsymbol{y}m}^+ \frac{\mathbb{E}[(\Delta + X)^j]}{\max\{r_{y_1 m}, r_{y_2 m}, ..., r_{y_d m}\}^j},$$

where

$$p_{i,\boldsymbol{y}m}^+ = \frac{p_m \mathbb{1}_{i,\boldsymbol{y}m}^+}{\sum_{\boldsymbol{w} \in S_N} \sum_{l=1}^{M} p_l \mathbb{1}_{i,\boldsymbol{w}l}^+}.$$

The arrival of a type-2 job only brings the additional $\Delta$ probe to server $i$. Let us denote the time spent by server $i$ on the probe of an arriving type-2 job as $D_i^-$. Then its $j$-th moment, $j = 1, 2, \ldots$, is given by

$$\mathbb{E}[(D_i^-)^j] = \sum_{\boldsymbol{y} \in S_N} \sum_{m=1}^{M} p_{i,\boldsymbol{y}m}^- \frac{\Delta^j}{\max\{r_{y_1 m}, r_{y_2 m}, ..., r_{y_d m}\}^j},$$

where

$$p_{i,\boldsymbol{y}m}^- = \frac{p_m \mathbb{1}_{i,\boldsymbol{y}m}^-}{\sum_{\boldsymbol{w} \in S_N} \sum_{l=1}^{M} p_l \mathbb{1}_{i,\boldsymbol{w}l}^-}.$$

We still need to determine $\mathbb{E}[D_i]$ in (3) and $\mathbb{E}[G_i]$ and $\mathbb{E}[G_i^2]$ for (2). Remembering that $D_i$ is the time spent by server $i$ on a probe of an arbitrary arrival of type 1 or 2 (not distinguishing between them), we have

$$\mathbb{E}[D_i] = \frac{1}{\binom{N}{d}} \sum_{\boldsymbol{y} \in S_N} \sum_{m=1}^{M} \frac{p_m \Delta \mathbb{1}_{\{i \in \{y_1, y_2, \ldots, y_d\}\}}}{\max\{r_{y_1 m}, r_{y_2 m}, ..., r_{y_d m}\}}.$$

Now consider the generic run time $G_i$ of an arbitrary arrival assigned to server $i$. It is either a type-1 job, with run time $G_i^B$, or a type-2 job, with run time $D_i^-$. Hence

$$\mathbb{E}[G_i] = \frac{\lambda_i^+}{\lambda_i^+ + \lambda_i^-} \mathbb{E}[G_i^B] + \frac{\lambda_i^-}{\lambda_i^+ + \lambda_i^-} \mathbb{E}[D_i^-],$$

and

$$\mathbb{E}[G_i^2] = \frac{\lambda_i^+}{\lambda_i^+ + \lambda_i^-} \mathbb{E}[(G_i^B)^2] + \frac{\lambda_i^-}{\lambda_i^+ + \lambda_i^-} \mathbb{E}[(D_i^-)^2].$$

We have now obtained expressions for all the ingredients of (3), and hence the expected latency $\mathbb{E}[T_i]$ follows. Finally, the expected latency of an arbitrary job is given by

$$\mathbb{E}[T] = \sum_{i=1}^{N} \frac{\lambda_i^+}{\lambda} \mathbb{E}[T_i],$$

(notice that $\sum \lambda_i^+ = \lambda$), under the stability condition

$$\rho_i < 1, \quad i = 1, 2, ..., N. \tag{7}$$

Remark 4. *When the assumption of non-equal server speeds within the realizations of the joint speed distribution is not satisfied, the maximum of the $r_{y_i m}$ in Equations* (5) *and* (6) *may no longer be unique. It is no longer true that the service of a job will only be completed on the (one) server which has the fastest speed among the d sampled servers. In the case of a non-unique maximum, the job is dispatched uniformly at random to one of the servers for which this maximum is attained. Therefore, one can simply replace $\mathbb{1}_{i,\boldsymbol{y}m}^+$ by $\phi_{i,\boldsymbol{y}m}^+$ which equals the right-hand side of Equation* (5) *divided by the number of servers for which this maximum is attained and replace $\mathbb{1}_{i,\boldsymbol{y}m}^-$ by $\phi_{i,\boldsymbol{y}m}^-$ which equals $\phi_{i,\boldsymbol{y}m}^- = \mathbb{1}_{\{i \in \{y_1, y_2, \ldots, y_d\}\}} - \phi_{i,\boldsymbol{y}m}^+$.*

**Negligible $\Delta$ probe**

When $\Delta$ approaches zero, the analysis of server $i$ reduces to that of a standard $M/G/1$ queueing model. Observe that only arrivals of type 1 introduce workload at server $i$.

The arrival intensity at server $i$ is given by Equation (4). The run time when executed by server $i$ has $j$-th moment, $j = 1, 2, \ldots$,

$$\mathbb{E}[(G_i^B)^j] = \sum_{\boldsymbol{y} \in S_N} \sum_{m=1}^{M} p_{i,\boldsymbol{y}m}^+ \frac{\mathbb{E}[X^j]}{\max\{r_{y_1 m}, r_{y_2 m}, ..., r_{y_d m}\}^j}.$$

The expected waiting time of a job at server $i$ is equal to

$$\mathbb{E}[W_i] = \frac{\rho_i \mathbb{E}[R_{G_i^B}]}{1 - \rho_i},$$

where

$$\rho_i = \eta_i \mathbb{E}[G_i] = \lambda_i^+ \mathbb{E}[G_i^B] \text{ and } \mathbb{E}[R_{G_i^B}] = \frac{\mathbb{E}[(G_i^B)^2]}{2\mathbb{E}[G_i^B]}.$$

Hence the expected latency of an arbitrary job is given by

$$\mathbb{E}[T] = \sum_{i=1}^{N} \frac{\lambda_i^+}{\lambda} \left( \mathbb{E}[W_i] + \mathbb{E}[G_i^B] \right).$$

Since in this case the analysis of server $i$ reduces to that of an $M/G/1$ queueing system, we can use classical results for the $M/G/1$ queue to obtain the Laplace-Stieltjes transform (LST) of the waiting time and the latency of a job at server $i$:

$$\tilde{W}_i(s) = \frac{(1-\rho_i)s}{\lambda_i^+ \tilde{B}_i(s) + s - \lambda_i^+}, \quad \tilde{T}_i(s) = \tilde{W}_i(s)\tilde{B}_i(s),$$

where $\tilde{B}_i(s)$ denotes the LST of the run time $G_i^B$. The LST of the waiting time of an arbitrary job is

$$\tilde{W}(s) = \sum_{i=1}^{N} \frac{\lambda_i^+}{\lambda} \tilde{W}_i(s),$$

and the LST of the latency is

$$\tilde{T}(s) = \sum_{i=1}^{N} \frac{\lambda_i^+}{\lambda} \tilde{T}_i(s).$$

## 4 NUMERICAL RESULTS

In Section 3.2 we analyzed the delay performance of the delta-probe preemption policy, both for strictly positive and negligible $\Delta$. To offer further insight in the performance of this policy, as well as the delta-probe policy, we present in this section numerical results which illustrate the advantages and the limitations of both policies. Note that in most settings only the delta-probe preemption policy can be exactly analyzed and therefore we use simulation where needed to compare the policies. The 95% confidence intervals of the simulation are within the line.

We extensively discuss the S&X model (Section 4.1), scenarios with non-identically distributed server speeds (Section 4.2), the scaling behavior (Section 4.3) and the scenario of non-exponential job size distributions (Section 4.4). Due to the general model setup and large number of parameters the range of possible scenarios is vast, and an exhaustive study is simply infeasible. In this section we focus on a representative sample of scenarios, and numerical results for some interesting extensions to the scenarios discussed here are relegated to Appendix A.

### 4.1 S&X model

As mentioned in Section 2, the S&X model introduced in [7] is subsumed within our workload model. In [7] a particular distribution of the slowdown $S_i$ (or the reciprocal of the server speed $R_i$ in our setup) is proposed, called the Dolly(1, 12) distribution (given in Table 1), which was measured empirically in [1]. Here, we compare the various policies for this specific setting.

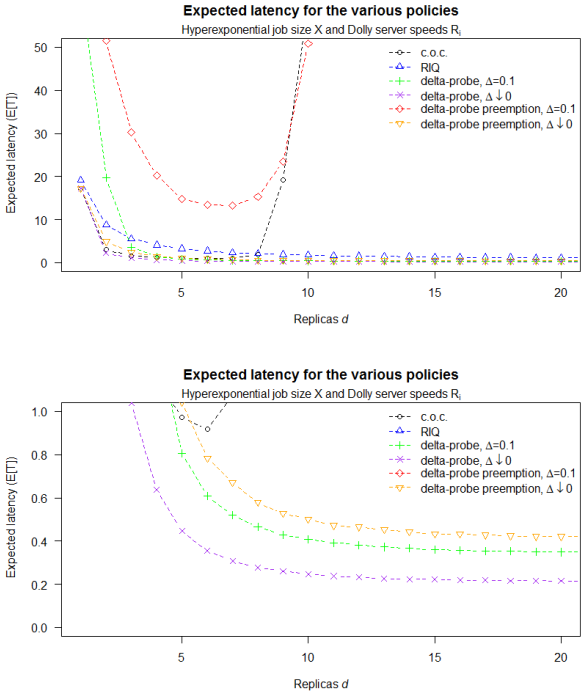


**Figure 2: Expected latency (obtained by simulation) as a function of $d$ in the S&X model, when $N = 1000$, $\lambda = 700$, $X \sim$ hyperexponential with $\mathbb{E}[X] = 1/4.7$ and squared coefficient of variation $C_X^2 = 10$, and $1/R_i \sim$ Dolly$(1, 12)$ (see Table 1) with $\mathbb{E}[1/R_i] = 4.7$ for $i = 1, \ldots, N$. Both figures relate to the same setting, but the bottom figure is zoomed in on an expected latency between 0 and 1.**

**Table 1: The Dolly$(1, 12)$ empirical server speed distribution. The speed $1/R_i$ ranges from 1 to 12, with mean 4.7**

| $1/R_i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Prob. | 0.23 | 0.14 | 0.09 | 0.03 | 0.08 | 0.10 | 0.04 | 0.14 | 0.12 | 0.021 | 0.007 | 0.002 |

Figure 2 shows the expected latency as function of the parameter $d$, and reveals some remarkable behavior. First of all, observe that in the c.o.c. redundancy policy adding more replicas to the system will eventually increase the workload, since the speed is bounded by 1, and therefore increases the expected latency. This behavior is more thoroughly discussed in [7]. Interestingly, for the delta-probe-$d$ policies with $\Delta = 0.1$ the same 'bathtub shape' can be observed. For the delta-probe policy with $\Delta = 0.1$ the increase in expected latency happens only beyond $d = 40$ (not shown in the figure). The reason for this difference is that in the delta-probe preemption policy (almost) all probe tasks are (partly) executed by the $d$ sampled servers due to the preemptive-resume priority, whereas in the delta-probe policy at least one probe task is executed by the sampled servers and it may very well be that the other probe tasks are abandoned before initiation. In this setting the delta-probe policy performs so well that even with $\Delta = 0.1$ it outperforms, for

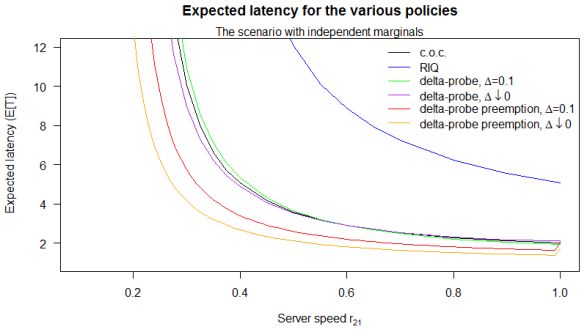


**Figure 3: Expected latency (obtained by simulation) as a function of $r_{21} = r_{23}$ when $d = N = 2$, $\lambda = 0.5$, $X \sim$ Exp(1) and where the server speeds are $r_{11} = r_{12} = 1$, $r_{13} = r_{14} = 0.1$ and $r_{22} = r_{24} = 1$, with corresponding probabilities $p_1 = p_2 = p_3 = p_4 = 0.25$.**

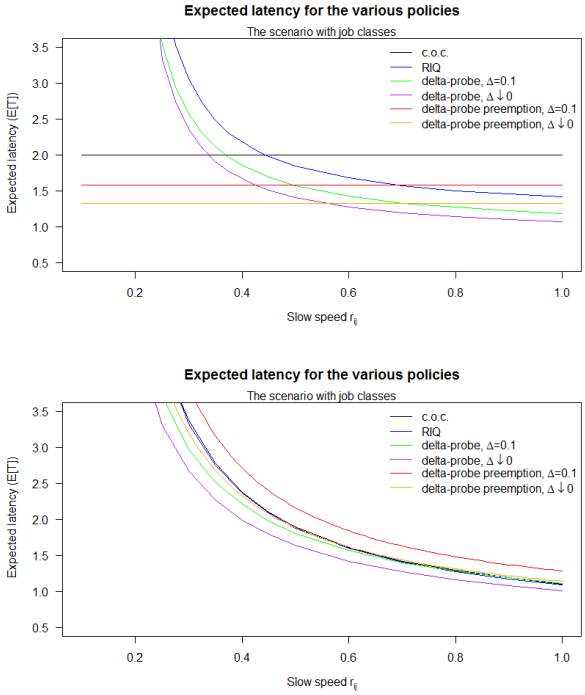


**Figure 4: Expected latency (obtained by simulation) as a function of the slow speed $r_{ij}$ with $i \neq j$, when $d = 2$, $M = N$, $\lambda = 0.5$, $X \sim$ Exp(1), $r_{ii} = 1$ for $i = 1, \ldots, N$, and $N = 2$ (top) and $N = 4$ (bottom). Here a specific job is fast on one server and slow on the other server(s).**

$d < 20$, the delta-probe preemption policy with $\Delta \downarrow 0$. Also observe that the graphs for both delta-probe-$d$ policies with $\Delta \downarrow 0$ and the RIQ policy have the same 'ski' slope, which indicates that additional replicas do not create any additional workload in the system. The reason is that both delta-probe-$d$ policies and the RIQ policy, in the case of non-idle servers, only introduce a single replica into the system. Finally, it can be observed that the delta-probe-$d$ policies clearly outperform the RIQ policy in terms of expected latency.

## 4.2 Non-identical server speeds

We now present results for scenarios where the server speeds are non-identically distributed and possibly correlated, which is not covered by the S&X model. First we look at the scenario where the marginal random variables of the joint server speed distribution are independent. Then the scenario of possibly correlated marginals is considered. Note that for the delta-probe preemption policy the analysis in Section 3.2 is used. The latency of the other policies is evaluated via simulation.

*4.2.1 Independent server speeds.* Figure 3 shows the expected latency for the various policies as a function of $r_{21}$. One can observe that for a server speed $r_{21}$ closer to zero the expected latency increases and the simulation reveals that some policies, i.e., the c.o.c. redundancy and RIQ policy, are not stable for $r_{21} = 0.2$. When comparing the stability condition of the delta-probe-$d$ policies for $\Delta \downarrow 0$, cf. Equations (1) and (7), it can be derived that the delta-probe and the delta-probe preemption policy are stable for $r_{21} > 0.189$ and $r_{21} > 0.154$, respectively. Thus the delta-probe preemption policy with $\Delta \downarrow 0$ has a larger stability region for this specific scenario. Figure 3 also depicts that the delta-probe preemption policy has a lower expected latency than the delta-probe policy with $\Delta \downarrow 0$. The performance of the RIQ policy is clearly the worst, due to the unbalanced speeds and relatively high load. (Unbalanced here means that the differences of the speeds within a realization of the joint server speed distribution $(r_{1m}, \ldots, r_{Nm})$, $m = 1, \ldots, M$, are large.) Surprisingly, the performance of the c.o.c. redundancy and delta-probe policy is (fairly) similar. Figure 6 in Appendix A shows the same scenario but with unequal probabilities.

*4.2.2 Possibly correlated server speeds.* Here, we look at a system with job classes, i.e., the scenario where the $i$-th job class is fast on server $i$ but slow on the other server(s). Jobs belong to each of the classes with equal probability. For the interested reader we refer to Figure 7 in Appendix A, which shows the scenario where jobs belong to each of the classes with unequal probabilities.

Figure 4 shows the expected latency as function of the slower server speed. It can be concluded that, in the case of job classes and $d = N = 2$, the delta-probe preemption policy has a lower expected latency than the c.o.c. redundancy policy, which is in line with Remark 1. Furthermore, it can be seen that the delta-probe policy outperforms the delta-probe preemption policy in the case of balanced speeds and vice versa for unbalanced speeds. One of the reasons is that in the delta-probe policy the actual job is assigned to the server based on current workload, not taking into account the speed of this specific server. For balanced speeds this is not a limitation since all speeds are approximately equal. However, for unbalanced speeds this may imply that the speed of the job would be much higher if it was assigned to another server. In the delta-probe preemption policy a job is assigned to the fastest server (which can be beneficial for unbalanced speeds), but not using the second

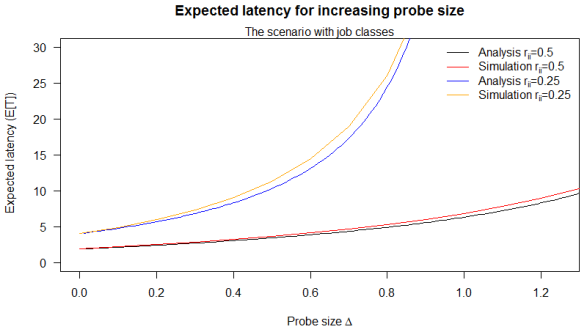


**Figure 5: Expected latency (obtained by simulation) as a function of the probe size $\Delta$ for the same scenario as Figure 4 (bottom, $N = 4$) with server speeds $r_{ij} = 0.5$ and $r_{ij} = 0.25$, $i \neq j$.**

fastest speed for balanced speeds can have an adverse influence on the latency performance. Figures 8 and 9 in Appendix A show the scenario with job classes and $N$ large, i.e., $N = 1000$.

For $d = 2$ and $N = 4$ all the policies have approximately the same expected latency. Comparing the simulation and the analysis for the delta-probe preemption policy with $\Delta = 0.1$ demonstrates that the analysis, while not exact, is within the 95% confidence intervals of the simulation.

To further investigate the difference between the analysis and the simulation for $d < N$, i.e., the setting where the analysis is not exact, we plotted both as a function of $\Delta$. Figure 5 depicts the simulation against the analytically derived expected latency value for the same setting as in Figure 4 but now for increasing $\Delta$. The accuracy of the analysis mainly depends on two factors. One is the value of $\Delta$, as the probability of overlap between $Z_{old}$ and $Z_{new}$ increases with $\Delta$ (see comment (ii) in Section 3.2). When this event occurs there is a discrepancy between the fastest and the actual realized speed. The other factor is the difference between the speeds, since a large difference means that the error made could also be large.

## 4.3 Scaling behavior of the delta-probe-$d$ policies

In this section we highlight how the performance of the delta-probe preemption policy with $\Delta \downarrow 0$ scales with growing $N$ and possibly $d$ as function of $\lambda$. We consider two scenarios: in Table 2 the value of $d$ is fixed and in Table 3 the ratio between $d$ and $N$ is fixed, both for increasing $N$.

In Table 2 the scaling behavior for the delta-probe preemption policy with $\Delta \downarrow 0$ can be seen for fixed $d$. For the other policies this table shows that for this specific setting the smaller system ($N = 10$) already gives a good approximation for the bigger system ($N = 1000$). Table 3 again shows the scaling behavior for the delta-probe preemption policy with $\Delta \downarrow 0$, but now for a fixed ratio between $d$ and $N$. Observe that for this policy, in the specific setting of Tables 2 and 3, the value of $d$ does not influence the expected latency since all servers have the same speed. Thus all in all, jobs

**Table 2: Expected latency (obtained by simulation) for the setting: $d = 4$, equal server speeds $r_{i1} = 1$, $i = 1, \ldots, N$, and $X \sim$ Exp(1).**

| policy | $N = 10$, $\lambda = 6$ | $N = 50$, $\lambda = 30$ | $N = 100$, $\lambda = 60$ | $N = 1000$, $\lambda = 600$ |
|---|---|---|---|---|
| delta-probe, $\Delta \downarrow 0$ | 1.07 | 1.04 | 1.04 | 1.04 |
| delta-probe, $\Delta = 0.1$ | 1.21 | 1.16 | 1.16 | 1.16 |
| delta-probe pre., $\Delta \downarrow 0$ | 2.50 | 2.50 | 2.50 | 2.50 |
| delta-probe pre., $\Delta = 0.1$ | 6.29 | 6.20 | 6.20 | 6.17 |
| RIQ | 1.63 | 1.50 | 1.49 | 1.47 |
| c.o.c. redundancy | not stable | | | |

**Table 3: Expected latency (obtained by simulation) for the setting: $d/N = 0.2$, equal server speeds $r_{i1} = 1$, $i = 1, \ldots, N$, and $X \sim$ Exp(1).**

| policy | $N = 10$, $\lambda = 6$ | $N = 50$, $\lambda = 30$ | $N = 100$, $\lambda = 60$ | $N = 1000$, $\lambda = 600$ |
|---|---|---|---|---|
| delta-probe, $\Delta \downarrow 0$ | 1.28 | 1.00 | 1.00 | 1.00 |
| delta-probe, $\Delta = 0.1$ | 1.47 | 1.11 | 1.10 | 1.10 |
| delta-probe pre., $\Delta \downarrow 0$ | 2.50 | 2.50 | 2.50 | 2.50 |
| delta-probe pre., $\Delta = 0.1$ | 3.84 | not stable | | |
| RIQ | 1.79 | 1.43 | 1.40 | 1.37 |
| c.o.c. redundancy | 2.09 | not stable | | |

are uniformly assigned to the servers, independent of $d$. Table 3 also illustrates a limitation of the delta-probe preemption policy, i.e., in case of $d$ large (almost) all probe tasks are (partly) executed by the $d$-sampled servers which makes the system unstable. For the interested reader we refer to Table 6 in Appendix A, which illustrates the scaling behavior in a scenario with unequal server speeds.

In almost all the numerical experiments so far the job sizes are assumed to be exponentially distributed.

## 4.4 General job sizes distributions

In this section the impact of the job size distribution is studied. Note that for the c.o.c. redundancy policy in the IR model it is known, see [8], that the expected latency decreases when the variability of the run time distribution increases. Moreover, increasing $d$ reduces the expected latency (it is even possible that for large $d$ the system becomes stable while it is not stable for small $d$).

We now present simulation results to gain further insight in the influence of the job size distribution on the expected latency of the various policies in our setup. As noted earlier, it is not feasible to exhaustively examine all possible settings because of the large number of system parameters. However, additional experiments (not reported here) indicate that the present results are representative of typical settings.

In Table 4 it can be seen that, for the same setting, all the policies perform worse, in terms of expected latency, when the job size distribution is heavier tailed. For the delta-probe preemption policy this statement is consistent with the analysis since the expected latency at server $i$, cf. Equation (3), depends (given the first moment) on the second moment of the job size, which is larger for distributions with a heavier tail.

**Table 4: Expected latency (obtained by simulation) for various job size distributions in the setting: $d = 2$, $N = 3$, $M = 2$, $r_{11} = 1$, $r_{12} = 0.5$ $r_{21} = 0.5$, $r_{22} = 0.8$, $r_{31} = 0.8$, $r_{32} = 1$, $p_1 = p_2 = 0.5$, $\mathbb{E}[X] = 1$ and $\lambda = 0.9$.**

| policy | lognormal | exponential | uniform | deterministic |
|---|---|---|---|---|
| delta-probe, $\Delta \downarrow 0$ | 1.49 | 1.46 | 1.41 | 1.38 |
| delta-probe, $\Delta = 0.1$ | 1.68 | 1.64 | 1.59 | 1.56 |
| delta-probe pre., $\Delta \downarrow 0$ | 2.05 | 1.80 | 1.56 | 1.44 |
| delta-probe pre., $\Delta = 0.1$ | 2.54 | 2.22 | 1.94 | 1.79 |
| RIQ | 2.08 | 1.87 | 1.68 | 1.58 |
| c.o.c. redundancy | 2.36 | 2.10 | 1.83 | 1.68 |

**Table 5: Expected latency (obtained by simulation) for various job size distributions in the setting: $d = N = 3$, $M = 2$, $r_{11} = 1$, $r_{12} = 0.5$ $r_{21} = 0.5$, $r_{22} = 0.8$, $r_{31} = 0.8$, $r_{32} = 1$, $p_1 = p_2 = 0.5$, $\mathbb{E}[X] = 1$ and $\lambda = 0.9$.**

| policy | lognormal | exponential | uniform | deterministic |
|---|---|---|---|---|
| delta-probe, $\Delta \downarrow 0$ | 1.27 | 1.25 | 1.23 | 1.21 |
| delta-probe, $\Delta = 0.1$ | 1.43 | 1.41 | 1.38 | 1.36 |
| delta-probe pre., $\Delta \downarrow 0$ | 2.11 | 1.82 | 1.55 | 1.41 |
| delta-probe pre., $\Delta = 0.1$ | 2.80 | 2.40 | 2.04 | 1.87 |
| RIQ | 2.27 | 1.99 | 1.74 | 1.62 |
| c.o.c. redundancy | 13.43 | 10.0 | 6.95 | 5.45 |

Comparing Tables 4 and 5 indicates that only the delta-probe policy shows performance gains, in terms of expected latency, when increasing $d$. For all the other policies the expected latency increases. For the delta-probe preemption policy this is due to the fact that, in case of $d = N$, no jobs are assigned to the slowest server and for the c.o.c. redundancy policy this behavior is consistent with the 'bathtub shape' observed in Section 4.1.

# 5 CONCLUSION AND SUGGESTIONS FOR FURTHER RESEARCH

In this paper we proposed two delta-probe-$d$ policies, i.e., the delta-probe policy and delta-probe preemption policy. The main idea behind these policies is that an additional probe task is launched to find the server that is the fastest for a particular job or has the smallest current workload, depending on the specific policy. In contrast to the c.o.c. redundancy policy, the use of probes ensures that there is only one server working on the actual job. For the delta-probe preemption policy explicit expressions for the expected latency, the stability condition and the delay distribution can be obtained, for generally distributed job sizes. For the delta-probe policy the problem of finding explicit expressions for the key performance measures, in general scenarios, seems out of reach.

Both delta-probe-$d$ policies tend to outperform the c.o.c. redundancy policy with the same value of $d$ in terms of expected latency. More specifically, this statement is proved for the delta-probe preemption policy for $d = N$ and $\Delta$ negligible and it is proved that this policy has a larger stability region. Our results show that the delta-probe preemption policy not always outperforms the delta-probe policy or vice versa. The relative performance mainly depends on two factors: how balanced the server speeds are and how high the load is. Usually, for (nearly) balanced server speeds the delta-probe policy outperforms the delta-probe preemption policy while for (highly) unbalanced speeds it is the other way around. The reason is that in the delta-probe policy the actual job is assigned to the server based on current workload, without paying attention to the speeds. For balanced speeds this is not a limitation since all speeds are approximately equal but for unbalanced speeds this may imply that the speed of the job would be much faster on another server. In the delta-probe preemption policy a job is assigned to the fastest server (which can be beneficial for unbalanced speeds), but not using the second fastest server for balanced speeds can have an adverse influence on the latency performance. However, the load plays a significant role as well, since in case of unbalanced speeds and high load it may be beneficial to use the slower servers and not only the fastest server for a specific job. Hence, in this regime the stability region of the delta-probe policy is larger than for the delta-probe preemption policy. Thus, there are regions where only one of the two policies is stable, or none, even though there exist alternative policies that do achieve stability.

For both delta-probe-$d$ policies the server speed distribution is of significant importance and observing this server speed, indirectly via the probes, is the key idea behind these policies. However, this also raises the question whether it is possible to learn the server speed distribution, e.g., in scenarios where some servers are always faster than others, so that the $\Delta$ probes are superfluous after a certain time. Another interesting extension of the delta-probe-$d$ policies is to allow for a random $\Delta$. In case the probes are identically distributed over the $d$ replicas we expect that the analysis still holds. However, in case the $\Delta$ are independent and non-identically distributed over the $d$ replicas, the analysis should be adapted. This extension is closely related to a model that involves using the probe task as a noisy estimate for the server speed. This would relax the assumption that the probe task experiences the same server speed as the actual job and could be more realistic in practice.

The server speeds for a particular job in our workload model are allowed to be inter-dependent and this in turn allows for affinity relations between jobs and different servers. In our future research we could consider a workload model that is even more general, i.e., a workload model that also allows for dependency between the speeds for consecutive jobs at the same server.

## 5.1 Combination of both delta-probe-$d$ policies

As mentioned above, the delta-probe policy performs best in some settings whereas the delta-probe preemption policy performs better in other settings. Ultimately, one wants to combine the benefits of both delta-probe-$d$ policies to have a policy that performs well across all situations and possibly even achieves maximum stability. The idea behind such a combined delta-probe policy is that the first probe task $\Delta_1$ is used to infer the speeds of the $d$ sampled servers and that the second probe task $\Delta_2$ is used to gauge the workload on each of the $d$ sampled servers. Here, the $\Delta_1$ has preemptive-resume priority both over the actual jobs and the $\Delta_2$, whereas the $\Delta_2$ does not have priority over the actual job.

Such a combined policy can potentially also extend the stability regions of the individual delta-probe-$d$ policies. Closely related to this is the *Mindrift* routing rule introduced in [19], which assigns

an arriving job to a server

$$i \in \arg\min_{i \in \boldsymbol{y}} r_{ij} V_i^A,$$

where $V_i^A$ is the current workload at server $i$ upon arrival and $\boldsymbol{y} \in S_N$ are the sampled servers. In [19] it is proved that, if all servers are sampled, this routing rule achieves maximum stability and is optimal in the sense that it minimizes the workload in heavy traffic. This means that the *Mindrift* routing policy achieves stability whenever the necessary conditions for that are satisfied. With the combined delta-probe-$d$ policy the aim is to somehow mimic this routing policy with the knowledge of the server speeds and workload (obtained by $\Delta_1$ and $\Delta_2$, respectively) and thus extend the stability region with respect to both individual delta-probe-$d$ policies.

## ACKNOWLEDGMENTS

This research was partly funded by the NWO Gravitation Programme NETWORKS, Grant Number 024.002.003

## REFERENCES

[1] G. Ananthanarayanan, A. Ghodsi, S. Shenker, I Stoica (2013). Effective straggler mitigation: Attack of the clones. *Proc. NSDI 2013* 185-198.
[2] M.F. Aktas, P. Peng, E. Sojanin (2017). Effective straggler mitigation: Which clones should attack and when? *ACM SIGMETRICS Perf. Eval. Rev.* **45 (2)** 12-14.
[3] M.F. Aktas, P. Peng, E. Sojanin (2017). Straggler mitigation by delayed relaunch of tasks. *ACM SIGMETRICS Perf. Eval. Rev.* **45 (3)** 224-331.
[4] U. Ayesta, T. Bodas, I. Verloop (2018). On a unifying product form framework for redundancy models. Technical report HAL-01713937.
[5] T. Bonald, C. Comte (2017). Balanced fair resource sharing in computer clusters. *Perf. Eval.* **116** 70-83.
[6] G.P. Cosmetatos (1976). Some approximate equilibrium results for the multi-server queue. *Oper. Res. Quart.* **27** 615-620.
[7] K. Gardner, M. Harchol-Balter, A. Scheller-Wolf, B. Van Houdt (2017). A better model for job redundancy: Decoupling server slowdown and job size. *IEEE/ACM Trans. Netw.*, **25 (6)** 3353-3367.
[8] K. Gardner, M. Harchol-Balter, A. Scheller-Wolf, M. Velednitsky, S. Zbarsky (2017). Redundancy-$d$: The power of $d$ choices for redundancy. *Oper. Res.* **65 (4)** 1078-1094.
[9] K.S. Gardner, S. Zbarsky, S. Doroudi, M. Harchol-Balter, E. Hyytia, A. Scheller-Wolf (2015). Reducing latency via redundant requests: Exact analysis. *ACM SIGMETRICS Perf. Eval. Rev.* **43 (1)** 347-360.
[10] K.S. Gardner, S. Zbarsky, S. Doroudi, M. Harchol-Balter, E. Hyytia, A. Scheller-Wolf (2016). Queueing with redundant requests: Exact analysis. *Queueing Systems* **83 (3-4)** 227-259.
[11] L. Kleinrock (1965). A conservation law for a wide class of queueing disciplines. *Naval Res. Logist. Quart.* **12** 181-192.
[12] C. Knessl, B.J. Matkowsky, Z. Schuss, C. Tier (1990). An integral equation approach to the M/G/2 queue. *Oper. Res.* **38 (3)** 506-518.
[13] M. Mitzenmacher (2001). The power of two choices in randomized load balancing. *IEEE Trans. Parallel Distrib. Systems* **12** 1094-1104.
[14] D. Mukherjee, S. Borst, J. van Leeuwaarden, P. Whiting (2016). Universality of power-of-$d$ load balancing schemes. *ACM SIGMETRICS Perf. Eval. Rev.* **44 (2)** 36-38.
[15] F. Poloczek, F. Ciucu (2016). Contrasting effects of replication in parallel systems: From overload to underload and back. *ACM SIGMETRICS Perf. Eval. Rev.* **44 (1)** 375-376.
[16] N.B. Shah, K. Lee, K. Ramchandran (2017). The MSD queue: Analysing the latency performance of erasure codes. *IEEE Trans. on Infor. Theory* **63 (5)** 2822-2842.
[17] N.B. Shah, K. Lee, K. Ramchandran (2016). When do redundant requests reduce latency? *IEEE Trans. Commun.* **64 (2)** 715-722.
[18] M.S. Squillante, C.H. Xia, D.D. Yao, L. Zhang (2001). Threshold-based priority policies for parallel-server systems with affinity scheduling. *Proc. American Control Conf.* **4** 2992-2999.
[19] A.L. Stolyar (2005). Optimal routing in output-queued flexible server systems. *Prob. Eng. Inf. Sc.* **19** 141-189.
[20] A. Vulimiri, P.B. Godfrey, R. Mittal, J. Sherry, S. Ratnasamy, S. Shenker (2013). Low latency via redundancy. *Proc. ACM CoNEXT 2013* 283-294.
[21] N.D. Vvedenskaya, R.L. Dobrushin, F.I. Karpelevich (1996). Queueing system with selection of the shortest of two queues: An asymptotic approach. *Problemy Peredachi Informatsii* **32** 15-27.

## A ADDITIONAL NUMERICAL RESULTS

In this appendix we present additional results for the performance of the various policies in specific scenarios. These scenarios are mostly extensions of the scenarios that have already been discussed in Section 4. For example, the scenario in Figure 6 is similar to that in Figure 3 but now the server speeds are not equally probable. The scenario in Figure 7 is similar to that in Figure 4, but now the jobs belong to the various classes with unequal probabilities. Observe that the extension of unequal probabilities does not significantly change the performance of the various policies in Figures 6 and 7.

Figures 8 and 9 study the performance in the scenario with job classes and $N$ large, i.e., where a specific job is fast on half of the $N$ servers and slow on the other servers, and $N = 1000$. This scenario is not presented in Section 4, but notice that the performance of the various policies, especially in Figure 9, is similar to that in Figure 2.

Last, we show with Table 6 that the scaling behavior, discussed in Section 4.3, is not limited to equal server speeds. The essence is that each server still has the same generic run time, compared to the smaller system.

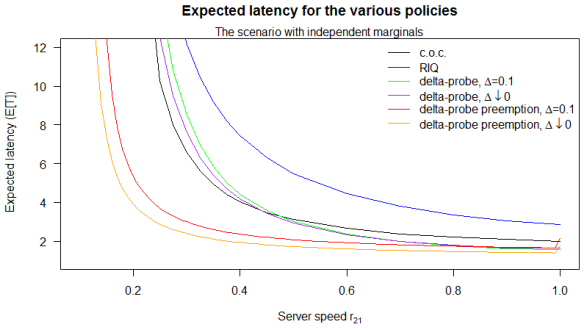


**Figure 6: Expected latency (obtained by simulation) as a function of $r_{21}$ in the setting as Figure 3 but with corresponding probabilities $p_1 = 0.5625$, $p_2 = p_3 = 0.1875$ and $p_4 = 0.0625$.**

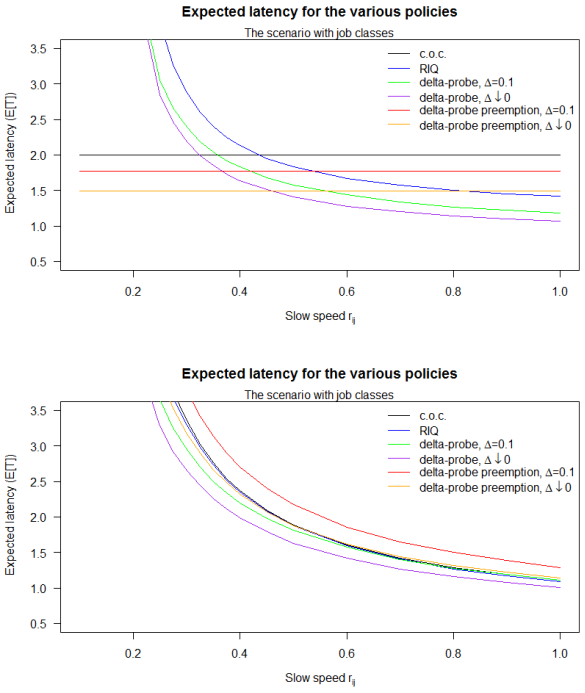


**Figure 7: Expected latency (obtained by simulation) as a function of the slow speed $r_{ij}$ with $i \neq j$ in the setting as Figure 4 but with probabilities $p_1 = 0.75$ and $p_2 = 0.25$ for $N = 2$ (top), and $p_1 = 0.5$, $p_2 = 0.25$, $p_3 = 0.125$ and $p_4 = 0.125$ for $N = 4$ (bottom). Here a specific job is fast on one server and slow on the other server(s).**

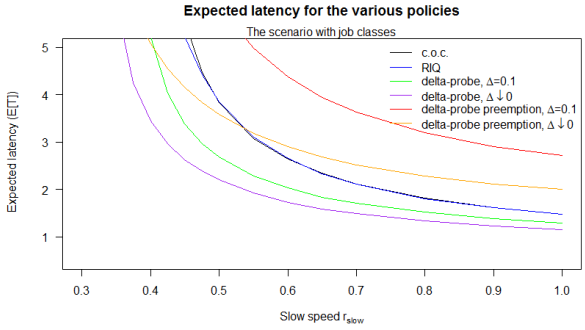


**Figure 8: Expected latency (obtained by simulation) as a function of the slow speed $r_{\text{slow}}$ when $d = 2$, $N = 1000$, $\lambda = 500$, $X \sim \text{Exp}(1)$, $r_{i1} = 1$, $r_{i2} = r_{\text{slow}}$ for $i = 1, \ldots, N/2$ and $r_{j1} = r_{\text{slow}}$, $r_{j2} = 1$ for $j = N/2 + 1, \ldots, N$. Here a specific job is fast on half the servers and slow on the other servers.**

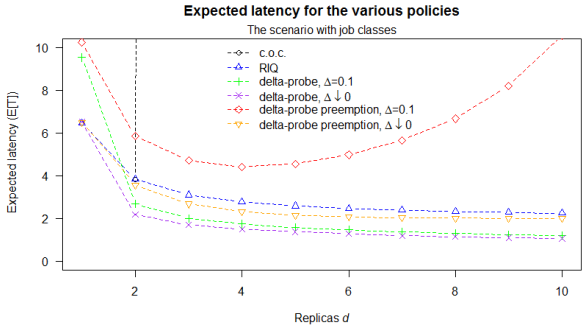


**Figure 9: Expected latency (obtained by simulation) as a function of $d$, when $N = 1000$, $\lambda = 500$, $X \sim \text{Exp}(1)$, $r_{i1} = 1$, $r_{i2} = 0.5$ for $i = 1, \ldots, N/2$ and $r_{j1} = 0.5$, $r_{j2} = 1$ for $j = N/2 + 1, \ldots, N$. Here a specific job is fast on half the servers and slow on the other servers.**

**Table 6: Expected latency (obtained by simulation) for the setting: $d = 2$, $r_{i1} = r_{i2} = 1$, $r_{i3} = r_{i4} = 0.5$ for $i = 1, \ldots, N/2$, $r_{j1} = r_{j3} = 0.5$, $r_{j2} = r_{j4} = 1$ for $j = N/2 + 1, \ldots, N$ and corresponding probabilities $p_1 = p_2 = p_3 = p_4 = 0.25$.**

| policy | $N = 10$, $\lambda = 3$ | $N = 50$, $\lambda = 15$ | $N = 100$, $\lambda = 30$ | $N = 1000$, $\lambda = 300$ |
|---|---|---|---|---|
| delta-probe, $\Delta \downarrow 0$ | 1.58 | 1.58 | 1.57 | 1.57 |
| delta-probe, $\Delta = 0.1$ | 1.79 | 1.77 | 1.77 | 1.76 |
| delta-probe pre., $\Delta \downarrow 0$ | 2.09 | 2.09 | 2.09 | 2.09 |
| delta-probe pre., $\Delta = 0.1$ | 2.62 | 2.61 | 2.61 | 2.61 |
| RIQ | 2.05 | 2.01 | 2.00 | 2.00 |
| c.o.c. redundancy | 1.99 | 1.86 | 1.85 | 1.84 |