Cutting Latency Tail: Analyzing and Validating Replication without Canceling

Zhan Qiu, *Member, IEEE,* Juan F. Pérez, *Member, IEEE,* Robert Birke, *Member, IEEE,* Lydia Chen, *Senior Member, IEEE,* and Peter G. Harrison

Abstract—Response time variability in software applications can severely degrade the quality of the user experience. To reduce this variability, request replication emerges as an effective solution by spawning multiple copies of each request and using the result of the first one to complete. Most previous studies have mainly focused on the *mean* latency for systems implementing *replica cancellation*, i.e., all replicas of a request are canceled once the first one finishes. Instead, we develop models to obtain the response-time *distribution* for systems where replica cancellation may be too expensive or infeasible to implement, as in "fast" systems, such as web services, or in legacy systems. Furthermore, we introduce a novel service model to explicitly consider correlation in the processing times of the request replicas, and design an efficient algorithm to parameterize the model from real data. Extensive evaluations on a MATLAB benchmark and a three-tier web application (MediaWiki) show remarkable accuracy, e.g., 7% (4%) average error on the 99^{th} percentile response time for the benchmark (respectively, MediaWiki), the requests of which execute in the order of seconds (respectively, milliseconds). Insights into optimal replication levels are thereby gained from this precise quantitative analysis, under a wide variety of system scenarios.

1 Introduction

The ubiquity of cloud computing has enabled many service providers to exploit almost unlimited resources in a pay-as-you-go model. While this model offers many advantages, it also poses new challenges, some of them related to the virtualized nature of the cloud offering. It has been observed that virtualization can harm performance [1], [2] owing to co-located virtual machines competing for CPU, memory bandwidth or other resources. This performance degradation leads to an increased variability in processing times, which impacts the application latency, particularly affecting those users that face the longest response times [3].

To cope with this increasing variability, concurrent or speculative request replication has been proposed [3]–[5]. With concurrent replication, a number of replicas of each request are spawned simultaneously, and the result of the first replica to complete is used. This approach can therefore benefit from resource performance variability, as two (or more) copies of a request may be submitted to resources experiencing different levels of load, thus increasing the likelihood that a request receives service from a fast server.

However, although replication has the potential to reduce *service* times, it may negatively impact the *queueing* times due to the additional load introduced by replicas, potentially leading to longer overall delays. In this paper, we aim to capture the trade-off between these two conflicting effects of replication and characterize the scenarios under which replication improves latency tails.

1.1 Contributions

Different from existing works, in this paper we are interested in evaluating replication for applications with very short processing times, such as web services, which process requests in the order of milliseconds. In these systems it is not feasible to cancel all replicas of a request upon completion of the first one, which is a mechanism commonly used to limit the additional load introduced by replication. In fact, most existing modeling studies [6]–[11] assume that canceling is performed. This assumption simplifies the analysis, thanks to the implicit synchronization introduced by the canceling mechanism (all replicas of a request finish service at the same time). Instead, we are interested here in evaluating the impact of *replication without canceling*, which lacks the synchronization mentioned above, and to this end we derive a suite of stochastic models that accurately predict request response times.

While most works in this area focus on the *mean* response time as the sole performance metric, our models are able to determine the response-time *distribution*. This enables us to evaluate the impact of replication on different response-time percentiles, which, as we will demonstrate, is far from homogeneous. Furthermore, using this model we show that the threshold load, i.e., the maximum load under which replication is beneficial, may differ depending on whether the evaluation is based on the mean response-time or on a specific percentile.

In developing the model we emphasize its ability to capture a wide range of real-world system scenarios. In particular, we are able to model highly varying processing

- *Z. Qiu and P. G. Harrison are with the Department of Computing, Imperial College London, UK. E-mail: {zhan.qiu11,p.harrison}@imperial.ac.uk*
- *J. F. Pérez is with the Department of Applied Mathematics and Computer Science, Universidad del Rosario, Colombia. E-mail: juanferna.perez@urosario.edu.co*
- *R. Birke and L. Chen are with IBM Research Zurich, Switzerland. E-mail:{bir,yic}@zurich.ibm.com*

times by utilizing phase-type (PH) distributions. We also consider Markovian arrival processes (MAP), which generalize traditional Poisson arrivals to model highly variable and correlated inter-arrival times.

Moreover, through experimentation, we have observed that the processing times of replicas of the same request can be **correlated**. To capture this behavior, we introduce a novel approach that uses correlated hyper-Erlang (CHE) distributions in an extension of our model. At the same time, we apply an efficient fitting method based on the Expectation-Maximization algorithm to parameterize the CHE service model with respect to real data.

We demonstrate the models' ability to predict the tail response times through extensive experimentation on a MATLAB benchmark [12] and on a standard three-tier web application, namely, MediaWiki [13]. The resulting average prediction errors for the 99^{th} percentile of response times are 7% and 4%, respectively. In particular, the MediaWiki experiments demonstrate the ability of the CHE service model to accommodate correlated service times effectively, evident by its accurate estimation of response time percentiles. Based on these models, we derive insights into the design of replication policies, e.g., optimal number of replicas and threshold load, for real-world applications.

1.2 Related work

Request replication has been considered in [3], [6]–[11], [14]–[18] as an efficient way to combat the variability in latency. Joshi et. al. [11] analyze the impact of replication on the mean response time and the cost of computing resources, and show that this impact depends on the processing-time distribution. Vulimiri et. al. [14] propose a queueing model to derive the mean response time as a function of system utilization and service-time distribution, and approximate the threshold load under which replication improves mean latency. Qiu and Pérez [6]–[8] evaluate the impact of replication on the response-time distribution in computing clusters, considering any number of replicas and fairly general processing and inter-arrival times.

Most existing works [3], [6]–[11], [17], [18] consider the adoption of canceling to limit the additional load introduced by the replicas. In fact, it is common to assume that redundant replicas can be canceled at no cost. In contrast, in this work we focus on cases where canceling is hard or even infeasible to implement. This is particularly relevant for *fast* systems where the canceling overhead would be comparable in magnitude to the replica processing time, such as web applications, in which requests usually take only milliseconds to respond. In addition, canceling requests on the fly may be hard to retrofit into existing systems that were not designed to provide such a feature. Also, request cancellation may introduce trust issues as servers need to allow an external entity to terminate requests in execution.

From an analytical standpoint, the case we consider in this paper is more challenging than its counterpart with canceling, where replicas of a request always finish service at the same time, adding a synchronization point that simplifies the system dynamics. In fact, canceling plays a key role in obtaining the scarce analytical solutions available for request replication [10], [11]. For instance, canceling enables [11] to reduce the problem to a single M/G/1 queue in the early-canceling case or to an approximate M/G/C queue in the late-canceling one. Similarly, canceling is key in obtaining the solution to the Markov chain model in [10]. The absence of canceling allows servers to evolve asynchronously, requiring a more complex analysis. This is also reflected in the few existing results that consider the cost of canceling, as these focus on fairly limited scenarios, e.g., 2-server systems with a centralized queue, Poisson arrivals, and exponential processing times [19]. An approximate approach to consider the operation without canceling, as in [14], is to assume that each queue operates independently, obtaining the request latency as the minimum of the response times observed in several queues. Instead, the analysis derived in this paper is exact in considering replication without canceling.

In many applications, requests are fairly homogeneous and their processing times depend largely on the server, leading to a degree of independence between replicas' service times. However, in other applications, processing times depend on the intrinsic details of the request, which impacts all of its replicas. In such cases correlation among the replicas of a request becomes significant; whilst commonly ignored [6]–[8], [10], [14], it has only recently been taken into account in [11], [20]. In contrast, this paper considers cases of both independent and correlated processing times among replicas of the same request. In addition, introducing MAP arrivals enables us to capture bursty workloads.

Request replication has a number of other aspects that have sparked attentions in recent works. For instance, Hopper [17], [18] considers the problem of replication-aware task scheduling for analytics clusters. Here replica cancellation is assumed in deriving the latency model that supports the scheduling policy. On the other hand, request replication can be performed for a subset of all requests, for instance a fixed fraction of small requests, or it can be employed only when there are idle servers. These design choices have been explored experimentally for instance in [15], [21], showing the benefits of replication. Aspects such as request scheduling and state-dependent replication are beyond the scope of this paper.

Finally, we should highlight that most existing works focus on either modeling or system experimentation. The only exception is [14], which studies the threshold load by means of approximate models, and experiments with different applications to confirm that this threshold load for one additional replica is between 25% and 50%. To the best of our knowledge, this is the first time a model for request replication is able to provide response time mean and percentiles, and its predictions are validated against system measurements.

2 Motivation and System Set-up

Employing request replication to mitigate the latency tail poses a number of challenges, which we now illustrate by means of experiments conducted on a real-life web application. We make use of MediaWiki [13], a standard three-tier web application deployed on the Amazon Web Services (AWS) public cloud. Details of the application can be found in Section 7. We deployed the MediaWiki cluster on seven

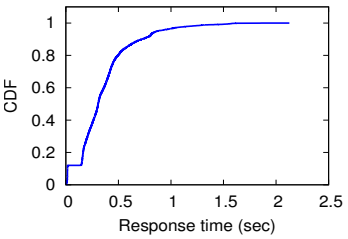


Fig. 1: Response-time CDF on AWS.

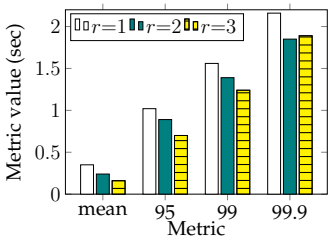


Fig. 2: Response times under replication on AWS.

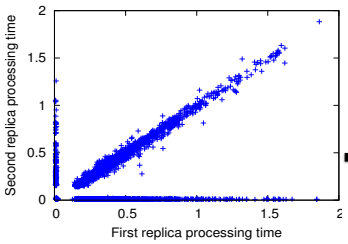


Fig. 3: Processing times of two replicas from the same request on AWS.

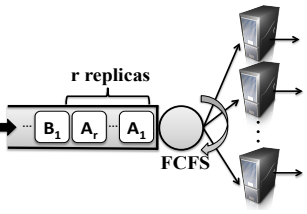


Fig. 4: Reference model.

t1.micro instances: six running the MediaWiki stack and one used as central queue to dispatch the requests. The cloud deployment exposes the application to large variations in its effective resource capacity, caused by virtual machines (VM) co-located on the same physical machine, a notorious drawback of cloud computing [1], [2]. This resource capacity variability can cause large variations on the application response times, as illustrated in Figure 1. Here we show the cumulative distribution (CDF) of request response times. The tail can be several times larger than the average response time (350.5 msec). For example, the 95^{th} and 99^{th} percentiles are 1022.1 and 1559.1 msec, respectively, while the 99.9^{th} percentile is 2163.1 msec, which is $6.17\times$ larger than the average.

To cope with this variability, we implement request replication to create r copies of each application request. However, due to the fast dynamics (on the order of milliseconds) of web applications, it is difficult to cancel the outstanding requests upon completion of the first one without incurring a high processing overhead. We thus focus on replication policies that do not cancel outstanding requests. Figure 2 depicts the response time mean and tail percentiles when implementing between 1 and 3 replicas for an arrival rate of 1.33 requests per second. Clearly, replication is effective in reducing the latency tail, with reductions close to 15% with 2 replicas. However, introducing a third replica hurts the tail even if it improves the mean latency. This highlights the importance of developing analytic models able to compute the latency *distribution* under replication, as those proposed in this paper, and not just the mean.

A common assumption when modeling replication is that the processing times of the replicas of a request are independent. Using measurements from the MediaWiki application with an arrival rate of 1.33 requests per second and 2 replicas, Figure 3 depicts a scatter plot of the processing times of the replicas of each request. Here we observe a large mass along the $45°$ line, indicating a strong correlation between the replicas' processing times. Also, two other masses close to the axes indicate that in many cases one replica has a long processing time whereas the other has a short one. Thus, the replica processing times are highly varying but not independent, key features that shall be considered in the analytical model proposed in the next sections.

2.1 Reference model

Based on the motivating case study presented earlier, our reference model, shown in Figure 4, consists of a central dispatcher and C distributed, homogeneous, and independent servers. Requests arrive at the dispatcher and join the next server that becomes available with first-come first-served (FCFS) scheduling. For each arriving request, $r \geq 1$ replicas asking for the same content are initiated simultaneously and added to the dispatcher. The request is considered complete as soon as one of its r replicas completes service. The remaining $r-1$ replicas continue their execution without being terminated. The mean replica processing time is $1/\mu$, and we consider two main types of replica processing times, namely, phase-type and correlated hyper-Erlang, to capture the key features of the processing times found in the case study. Our objective is to derive the response-time distribution of requests analytically, for any given replication factor r, number of servers C, and processing-time distribution.

Requests arrive at the system according to a Markovian arrival process (MAP) with parameters (m_a, D_0, D_1). A MAP consists of an underlying Markov chain with m_a states, which evolves according to the rates in matrices D_0 and D_1. The rates in D_1 determine the arrival rate in each state, whereas the rates in D_0 mark transitions without arrivals. We also introduce the notion of a *job*, which refers to the set of replicas belonging to the same request. The job response time is the difference between its arrival time and the completion time of the first replica, which is the same as the request response time.

The proposed model makes extensive use of phase-type (PH) distributions [22]. A PH distribution is the distribution of the time to absorption in a Markov chain with $n+1$ states, where the first n states are transient and the $(n+1)^{\text{th}}$ state is absorbing. The generator matrix of such a chain can be written as $\begin{bmatrix} B & \boldsymbol{b} \\ \mathbf{0} & 0 \end{bmatrix}$, where the matrix B holds the transition rates among the n transient states, and the exit vector $\boldsymbol{b} = -B\mathbf{1}$ holds the rates at which the chain jumps into the absorbing state. Here $\mathbf{1}$ is a column vector of ones, and $\mathbf{0}$ a row vector of zeros. We denote this distribution as PH$(\boldsymbol{\tau}, B)$, where $\boldsymbol{\tau}$ is the $1\times n$ vector holding the initial probability distribution with which the chain starts in any of the n transient states.

PH distributions serve three purposes in this paper: (i) in Section 3, they are used to model request processing times more general than the standard exponential distribution; (ii) a subset of PH distributions is generalized in Section 6 to model correlated processing times; and (iii) the model we introduce obtains a PH representation of the service-, waiting- and response-time distributions, following the steps discussed in the next section.

Algorithm 1 Computing the response-time distribution

Stage 1: Find waiting-time distribution $(\boldsymbol{s}_{\text{wait}}, S_{\text{wait}})$
a. Compute T: find S and $A^{(\text{jump})}$, and solve (2)
b. Compute $\boldsymbol{\pi}(0)$: find $S_{\text{not-all-busy}}$, $S_{\text{(all)-(not-all)}}$, $R_{\text{not-all-busy}}$ and $R_{\text{all-busy}}$
c. Find $(\boldsymbol{s}_{\text{wait}}, S_{\text{wait}})$
Stage 2: Find matrix S_{ser} and vector $\boldsymbol{s}_{\text{ser}}$ of the service-time distribution $(\boldsymbol{s}_{\text{ser}}, S_{\text{ser}})$ as in Section 3.2
Stage 3: Combine waiting and service times as in (10) to obtain the response-time distribution $(\boldsymbol{s}_{\text{res}}, S_{\text{res}})$

3 Independent Processing Times

One can view the job response time as made up of two parts: (i) a waiting time from arrival until the first replica starts processing, and (ii) a service time from the service start of the first replica until the earliest replica completes processing. Accordingly, we divide the analysis to first obtain the job waiting-time distribution and then find the job service-time distribution.

To obtain the waiting-time distribution we rely on the techniques introduced in [23] for the standard multi-server case, which we extend to consider requests replicated $r \geq 1$ times. Our analysis also extends the methods in [6]–[8], which consider the case where replicas are canceled immediately after the completion of the first one. The canceling mechanism facilitates the analysis as all replicas of the same request are terminated at the same time, freeing all resources used by the request simultaneously. In fact, when the number of servers C is a multiple of the number of replicas r, the analysis can be carried out by simply grouping all servers in C/r groups as all the servers in each group are synchronously seized and released by the r replicas of an incoming request [6].

The case without canceling lacks the synchronization mentioned above, requiring a more delicate treatment, particularly when deriving the service-time distribution. In contrast to standard queueing systems, where the service-time distribution is known beforehand, here the (stationary) *job* service-time distribution depends on the overall system state. When the system is lightly loaded, most jobs start service with all their replicas, maximizing the benefits of concurrent execution. Instead, when the system is heavily loaded, most jobs will only be able to start with a single replica, reducing the benefits of replication. Considering the different conditions in which a job can start service is thus necessary to derive the (stationary) job service-time distribution, and different from existing models that consider non-replicated services or replicated requests with canceling.

To help readers navigate the approach described in the next sections, we summarize the key steps of computing the response-time distribution in Algorithm 1. The notation used in the algorithm will be introduced in the sections treating each step. In the following, we assume that the replica service-time distribution is $\text{PH}(\boldsymbol{s}_{\text{R}}, S_{\text{R}})$ with m_R phases, and refer to a period during which all the servers are busy as an *all-busy* period and to a period where at least one server is idle as a *not-all-busy* period.

TABLE 1: Transition rates for S and $A^{(\text{jump})}$ with PH services

Matrix	Condition	From	To	Rate
S	$w \geq 0$	$(\boldsymbol{n}, w)$	$(\boldsymbol{n}+\boldsymbol{e}_j-\boldsymbol{e}_i, w)$	$n_i S_{\text{R}}(i,j)$
	$w \geq 1$	$(\boldsymbol{n}, w)$	$(\boldsymbol{n}+\boldsymbol{e}_j-\boldsymbol{e}_i, w-1)$	$n_i S^*_{\text{R}}(i)\boldsymbol{s}_{\text{R}}(j)$
$A^{(\text{jump})}$	$w=0$	$(\boldsymbol{n}, 0)$	$(\boldsymbol{n}+\boldsymbol{e}_j-\boldsymbol{e}_i, r-1)$	$n_i S^*_{\text{R}}(i)\boldsymbol{s}_{\text{R}}(j)$

3.1 The waiting-time distribution

The waiting time of a job is the time period between its arrival and the time its first replica starts service. To obtain the waiting-time distribution, we observe the queue *only during all-busy periods*, as jobs that arrive during a not-all-busy period see at least one idle server and start service without waiting. In the same vein as in [23], we define a bi-variate Markov process $\{X(t), J(t) | t \geq 0\}$, where the *age* $X(t)$ is the total time-in-system of the *youngest* job in service. Thus, the waiting time of a job is equal to the value of the age $X(t)$ at the time instant when the job starts service. More specifically, the age $X(t)$ takes values in $[0, \infty)$, increasing linearly with rate 1 if no *job* starts service, because the time-in-system of the youngest job in service increases at rate 1. Instead, *when a new job starts service*, a downward jump in $X(t)$ occurs as the new job becomes the youngest in service. When a new job starts service its age (time-in-system) is equal to its waiting time. Note that we update $X(t)$ whenever a new job starts service, but do not keep track of when each job finishes, as jobs and their replicas may finish in any order. On the other hand, the *phase* $J(t) = (D_{\text{all-busy}}(t), A(t))$ holds the phases of both the MAP arrival process $A(t)$ and the service process $D_{\text{all-busy}}(t)$. During an all-busy period, as a replica in service can be in any phase $1 \leq i \leq m_R$, we define the service state to be $D_{\text{all-busy}}(t) = (n_1(t), \ldots, n_{m_R}(t), w(t))$, where $n_i(t)$ is the number of replicas in service phase i and $w(t)$ is the number of replicas of the *youngest job* waiting in the queue, at time t. Thus $D_{\text{all-busy}}(t)$ takes values in the set $N_{\text{all-busy}} = \{(n_1, \ldots, n_i, \ldots, n_{m_R}, w) | n_i \in \{0, \ldots, C\}, \sum_{i=1}^{m_R} n_i = C, 0 \leq w < r\}$, of size $m_{\text{all-busy}}$. As the number of arrival phases is m_a, the total number of phases during the all-busy period is $m = m_a m_{\text{all-busy}}$.

Let the vector $\boldsymbol{\pi}(x)$ hold the steady-state density of $\{X(t), J(t)\}$, which has been shown [23] to have a matrix-exponential form such that

$$\boldsymbol{\pi}(x) = \boldsymbol{\pi}(0) \exp(Tx). \tag{1}$$

We thus have to obtain the vector $\boldsymbol{\pi}(0)$ and the matrix T. The $m \times m$ matrix T satisfies the nonlinear integral equation [23]

$$T = S_{\text{MAP}} + \int_0^\infty \exp(Tt) A^{(\text{jump})}_{\text{MAP}}(t) dt, \tag{2}$$

where $S_{\text{MAP}} = S \otimes I_{m_a}$, $A^{(\text{jump})}_{\text{MAP}}(t) = A^{(\text{jump})} \otimes \exp(D_0 t) D_1$, I_{m_a} is the identity matrix of size m_a, and $\otimes$ denotes the Kronecker product. S and $A^{(\text{jump})}$ are $m_{\text{all-busy}} \times m_{\text{all-busy}}$ matrices that hold the transition rates of the service process associated with transitions without and with the start of a new job, respectively. Matrix $S + A^{(\text{jump})}$ is thus the generator of the marginal service phase process during the all-busy period. Table 1 summarizes the transition rates of these two matrices, where $\boldsymbol{n} = (n_1, \ldots, n_{m_R})$ is the state of the service process before the transition, and $\boldsymbol{e}_i$ is a zero vector with a one in the i-th entry. Here we consider that any of the n_i

replicas in service phase i undergoes a transition without service completion with rate $S_{\mathrm{R}}(i,j)$ or with a service completion with rate $S^*_{\mathrm{R}}(i)$, where $S^*_{\mathrm{R}}{=}{-}S_{\mathrm{R}}\mathbf{1}$, allowing a new replica to start service. In the latter case, if no replicas of the youngest job are waiting in queue, a new job starts service and this transition is recorded by the matrix $A^{(\mathrm{jump})}$.

Matrix T can be found by iteratively solving Eq. (2), where each iteration involves the solution of a Sylvester matrix equation [24]. Once T has been found, we obtain $\boldsymbol{\pi}(0)$ to complete the matrix-exponential representation of $\boldsymbol{\pi}(x)$. After finding $\boldsymbol{\pi}(0)$ and T, we can obtain the PH representation of the waiting-time distribution $(\boldsymbol{s}_{\mathrm{wait}}, S_{\mathrm{wait}})$. We provide the details in Appendix A[1].

Notice that the vector $\boldsymbol{\pi}(x)$ exists if the process $\{X(t), J(t)\}$ is positive recurrent, which holds under the following condition, proved in Appendix B.

Lemma 1. The process $\{X(t), J(t)\}$ is positive recurrent if and only if $r\lambda < C\mu$, where $1/\mu$ is the mean replica processing time and λ is the request arrival rate.

Given the no-canceling policy, this condition simply shows that the input traffic increases r times with replication and that the queue becomes unstable if this total traffic surpasses the processing capacity $C\mu$.

3.2 The service-time distribution

We now move to the second step of the procedure, where we find a PH representation $(\boldsymbol{s}_{\mathrm{ser}}, S_{\mathrm{ser}})$ for the job service-time distribution. To determine this distribution, we follow the execution of a tagged job from the time its first replica starts service until one of its replicas completes service. In addition, when not all replicas of the tagged job are in service, we need to keep track of the non-tagged replicas in service, as their service completion marks the start of the tagged replicas in the queue. To define the service process let $n_i(t)$ be the number of tagged replicas in service phase i, and $o_i(t)$ the number of non-tagged replicas in service phase i, at time t. The service process $D_{\mathrm{ser}}(t)$ is thus $(n_1(t), \ldots, n_{m_R}(t))$ when $\sum_{i=1}^{m_R} n_i(t){=}r$ (all tagged replicas in service) and as $\{(o_1(t), \ldots, o_{m_R}(t)), (n_1(t), \ldots, n_{m_R}(t))\}$, when $\sum_{i=1}^{m_R} n_i(t) < r$ (at least one tagged replica in the queue). This process thus takes values in the set $N_{\mathrm{ser}} = N^{\mathrm{all}}_{\mathrm{ser}} \cup N^{\mathrm{part}}_{\mathrm{ser}}$, where $N^{\mathrm{all}}_{\mathrm{ser}} = \{(n_1, \ldots, n_{m_R}) | 0 \leq n_i \leq r, \sum_{i=1}^{m_R} n_i = r\}$ covers the states where all tagged replicas are in service, and the remaining states are in $N^{\mathrm{part}}_{\mathrm{ser}} = \{(o_1, \ldots, o_{m_R}), (n_1, \ldots, n_{m_R}) | 0 \leq o_i < C, 0 \leq n_i < r, 1 \leq \sum_{i=1}^{m_R} n_i < r, \sum_{i=1}^{m_R}(o_i + n_i) = C\}$. We let $m_{\mathrm{ser}} = |N_{\mathrm{ser}}|$, $m^{\mathrm{all}}_{\mathrm{ser}} = |N^{\mathrm{all}}_{\mathrm{ser}}|$, and order N_{ser} by placing the phases in the set $N^{\mathrm{all}}_{\mathrm{ser}}$ first, and then those in $N^{\mathrm{part}}_{\mathrm{ser}}$.

The job service-time distribution $\mathrm{PH}(\boldsymbol{s}_{\mathrm{ser}}, S_{\mathrm{ser}})$ is thus defined by a $1 \times m_{\mathrm{ser}}$ vector $\boldsymbol{s}_{\mathrm{ser}}$ with the distribution of the initial job service phase, and an $m_{\mathrm{ser}} \times m_{\mathrm{ser}}$ sub-generator matrix S_{ser} holding the transition rates among all service phases. The nonzero entries of S_{ser} are defined in Table 2, where the vectors $\boldsymbol{o} = (o_1, \ldots, o_{m_R})$ and $\boldsymbol{n} = (n_1, \ldots, n_{m_R})$ hold the state of all non-tagged and tagged replicas in service before the transition, respectively. The condition under which each transition is valid is included in the last column of the table, where we make use of the 1-norm

1. All appendices are provided as electronic supplementary material

TABLE 2: Transition rates for S_{ser}

From	To	Rate	Condition
$\boldsymbol{n}$	Service completion	$\sum_{i=1}^{m_R} n_i S^*_{\mathrm{R}}(i)$	$\|\|\boldsymbol{n}\|\|{=}r$
$\boldsymbol{n}$	$\boldsymbol{n} + \boldsymbol{e}_j - \boldsymbol{e}_i$	$n_i S_{\mathrm{R}}(i,j)$	$\|\|\boldsymbol{n}\|\|{=}r$
$(\boldsymbol{o}, \boldsymbol{n})$	$(\boldsymbol{o} + \boldsymbol{e}_j - \boldsymbol{e}_i, \boldsymbol{n})$	$o_i S_{\mathrm{R}}(i,j)$	$\|\|\boldsymbol{n}\|\|{\leq}r{-}1$
$(\boldsymbol{o}, \boldsymbol{n})$	$(\boldsymbol{o}, \boldsymbol{n} + \boldsymbol{e}_j - \boldsymbol{e}_i)$	$n_i S_{\mathrm{R}}(i,j)$	$\|\|\boldsymbol{n}\|\|{\leq}r{-}1$
$(\boldsymbol{o}, \boldsymbol{n})$	$(\boldsymbol{o} - \boldsymbol{e}_i, \boldsymbol{n} + \boldsymbol{e}_j)$	$o_i S^*_{\mathrm{R}}(i)\boldsymbol{s}_{\mathrm{R}}(j)$	$\|\|\boldsymbol{n}\|\|{\leq}r{-}2$
$(\boldsymbol{o}, \boldsymbol{n})$	$(\boldsymbol{n} + \boldsymbol{e}_j)$	$o_i S^*_{\mathrm{R}}(i)\boldsymbol{s}_{\mathrm{R}}(j)$	$\|\|\boldsymbol{n}\|\|{=}r{-}1$
$(\boldsymbol{o}, \boldsymbol{n})$	Service completion	$\sum_{i=1}^{m_R} n_i S^*_{\mathrm{R}}(i)$	$\|\|\boldsymbol{n}\|\|{\leq}r{-}1$

$\|\|\boldsymbol{n}\|\| = \sum_{i=1}^{m_R} n_i$, which is the total number of tagged replicas in service. The first two rows consider tagged transitions, with and without a service completion, when all tagged replicas are in service. Note that this transition is only valid when all r replicas are in service, i.e., when $\|\|\boldsymbol{n}\|\| = r$. In the remaining rows, at least one tagged replica is still in the queue. Rows three and four cover the cases of transitions without service completion for non-tagged and tagged replicas, respectively. Instead, in rows five and six a non-tagged replica completes service, letting one tagged replica start service. The difference is that in row five at least one tagged replica is left in the queue after the transition, whereas in row six the last tagged replica starts service. The last row considers the service completion of a tagged replica when there still are tagged replicas in the queue.

Having obtained S_{ser}, it remains to determine the vector $\boldsymbol{s}_{\mathrm{ser}}$, which holds the stationary probability that a job starts service in each phase. In other words, $\boldsymbol{s}_{\mathrm{ser}}$ is the distribution of the service phase of the youngest job in service immediately after its service starts. To obtain $\boldsymbol{s}_{\mathrm{ser}}$ we first define the $m_{\text{all-busy}} \times m_{\mathrm{ser}}$ matrix M, which holds the rates at which a downward jump in the age process occurs in service phase $i \in N_{\text{all-busy}}$ and causes the new job to start service in phase $j \in N_{\mathrm{ser}}$. Thus, the nonzero entries in matrix M correspond to service-completion rates from service states where no replicas of the youngest job in service are waiting in the queue ($w = 0$), allowing a new job to start service with a single replica. Therefore, the transition rate in M from state $(\boldsymbol{n}, 0)$ to state $(\boldsymbol{n} - \boldsymbol{e}_i, \boldsymbol{e}_j)$ is given by $n_i S^*_{\mathrm{R}}(i)\boldsymbol{s}_{\mathrm{R}}(j)$. Also, we define the probability γ that a job has to wait, given in Appendix A. Considering the different conditions in which a job starts service we obtain the following results.

Lemma 2 (Busy start). The initial service phase for jobs that **must wait before starting service** is given by

$$\boldsymbol{s}_{\text{busy}} = \gamma c_1 \boldsymbol{\alpha}_{\text{busy}} L(M \otimes D_1), \tag{3}$$

where

$$L = \int_0^\infty \exp(Tu)\,(I_{m_s} \otimes \exp(D_0 u))\,du, \tag{4}$$

$\boldsymbol{\alpha}_{\text{busy}} = -\boldsymbol{\pi}(0)T^{-1}$ is the stationary distribution of the phase, and c_1 is a normalizing constant.

Proof: If a job finds all servers busy upon its arrival, and therefore must wait, its initial phase must consider the state of all non-tagged replicas already in service. The initial phase of this job is therefore related to the stationary distribution of the service phase after a downward jump in the age process $X(t)$, as these are the time points at which new jobs start service. We know that the joint distribution of the age and the phase is $\boldsymbol{\pi}(x)$. Furthermore, because of

the matrix-exponential form, the probability that the age reaches $[x+u, x+u+du)$, given that the age is x and the phase is i, is independent of x. Thus, the probability that level x is visited after a downward jump and that this occurs by visiting service phase $j \in N_{\text{ser}}$ is given by the j^{th} entry of

$$c_1 \int_0^\infty \int_0^\infty \boldsymbol{\pi}(x) \exp(Tu) \left(M \otimes \exp(D_0 u) D_1\right) du dx. \quad (5)$$

Here c_1 is a normalizing constant, $\boldsymbol{\pi}(x)\exp(Tu)$ is the probability density of reaching an age in $[x+u, x+u+du)$, and $\exp(D_0 u)D_1$ is the probability density of observing a downward jump of size $[u, u+du)$. As stated above, the matrix M holds the service-completion rates that trigger a downward jump and the probability with which the new job starts service in each phase. With $\boldsymbol{\alpha}_{\text{busy}} = -\boldsymbol{\pi}(0)T^{-1}$, we can write Eq. (5) as $c_1 \boldsymbol{\alpha}_{\text{busy}} L(M \otimes D_1)$, with L as in Eq. (4). To ensure that (5) is stochastic, we set $c_1^{-1} = \boldsymbol{\alpha}_{\text{busy}} L(D_1 \otimes M)\mathbf{1}$. As γ is the probability that a job has to wait, such a job starts service according to $\boldsymbol{s}_{\text{busy}}$ in Eq. (3). □

Note that the matrix L has a similar form as P in Appendix A and can thus be found by solving an associated Sylvester matrix equation.

Lemma 3 (Full start). The initial service phase for jobs that start service **without waiting and without initiating an all-busy period** is given by

$$\boldsymbol{s}^{\text{all}}_{\text{not-busy}} = (1-\gamma) p_r [\boldsymbol{s}_r \quad \mathbf{0}_1^{m_{\text{ser}} - m^{\text{all}}_{\text{ser}}}], \quad (6)$$

where p_r is given by Eq. (7) and $\mathbf{0}_a^b$ is an $a \times b$ zero matrix.

Proof: A job arrives during a not-all-busy period and starts service without waiting with probability $1-\gamma$. Among the arrivals in a not-all-busy period, all but one start service with all their replicas, whereas the last arrival initiates an all-busy period. Let η_1 be the number of arrivals in a not-all-busy period, thus $E[\eta_1]-1$ is the expected number of jobs among them that find more than r idle servers upon arrival. Thus the probability that a job starts service during a not-all-busy period without initiating an all-busy period is

$$p_r = (E[\eta_1]-1)/E[\eta_1]. \quad (7)$$

$E[\eta_1]$ can be computed as in [23, Section 7.2]. When a job finds more than r idle servers, it starts service in state $\boldsymbol{n}^r = (n_1^r, \ldots, n_{m_R}^r) \in N^{\text{all}}_{\text{ser}}$, where n_i^r is the number of replicas that start service in phase i. We define the vector $\boldsymbol{s}_r$ of size $m^{\text{all}}_{\text{ser}}$, with entries $\boldsymbol{s}_r(\boldsymbol{n}^r) = p(\boldsymbol{n}^r)$, where $p(\boldsymbol{n}^r)$ is the multinomial probability given in Appendix A, for every phase vector $\boldsymbol{n}^r \in N^{\text{all}}_{\text{ser}}$. The vector $\boldsymbol{s}_r$ thus reflects the random and independent selection of the initial service phase by each of the r replicas in the job. Thus, in this case, a job starts service with probability vector $\boldsymbol{s}^{\text{all}}_{\text{not-busy}}$ in (6) as the job does not wait with probability $1-\gamma$, finds more than r idle servers with probability p_r, and its r replicas start service according to $\boldsymbol{s}_r$. Note that vector $\boldsymbol{s}_r$ is assigned to the first phases as these correspond to the set $N^{\text{all}}_{\text{ser}} \subset N_{\text{ser}}$, where the tagged job has all its replicas in service, while the remaining entries of $\boldsymbol{s}^{\text{all}}_{\text{not-busy}}$ are zero. □

Before considering the final case, we define an $m_{\text{not-all-busy}} \times m_{\text{ser}}$ matrix R_{ser}, the $(i,j)^{\text{th}}$ element of which holds the probability that a job that *initiates an all-busy period* starts service in phase $j \in N_{\text{ser}}$ given that the service phase was $i \in N_{\text{not-all-busy}}$ just before its arrival. The entries of matrix R_{ser} are shown in Table 3, illustrating the start of a new job with w replicas in phase $\boldsymbol{n}^w$ with multinomial probability $p(\boldsymbol{n}^w)$ given in the Appendix.

TABLE 3: Transition prob. for $R_{\text{not-all-busy}}$, $R_{\text{all-busy}}$, and R_{ser}

Matrix	From	To	Prob	Condition
$R_{\text{not-all-busy}}$	$\boldsymbol{n}$	$\boldsymbol{n}+\boldsymbol{n}^r$	$p(\boldsymbol{n}^r)$	$\|\|\boldsymbol{n}\|\|+r<C$
$R_{\text{all-busy}}$	$\boldsymbol{n}$	$(\boldsymbol{n}+\boldsymbol{n}^w, r-w)$	$p(\boldsymbol{n}^w)$	$\|\|\boldsymbol{n}\|\|+r\geq C, w=C-\|\|\boldsymbol{n}\|\|$
R_{ser}	$\boldsymbol{n}$	$(\boldsymbol{n}, \boldsymbol{n}^w)$	$p(\boldsymbol{n}^w)$	$\|\|\boldsymbol{n}\|\|+r\geq C, w=C-\|\|\boldsymbol{n}\|\|$

Lemma 4 (Partial start). The initial service phase for jobs that start service **without waiting and initiate an all-busy period** is given by

$$\boldsymbol{s}^{\text{part}}_{\text{not-busy}} = (1-\gamma)(1-p_r) c_2 \boldsymbol{\pi}(0) P Q^{-1}_{\text{not-all-busy}} (R_{\text{ser}} \otimes D_1), \quad (8)$$

where p_r is given in Eq. (7), and P, and $Q_{\text{not-all-busy}}$ are given in Appendix A. respectively, and c_2 is a normalizing constant.

Proof: From the proof of Lemma 3, with probability $(1-\gamma)(1-p_r)$ a job finds $1 \leq w \leq r$ idle servers, starts service with its first w replicas, and initiates an all-busy period. In this case the initial service phase of the job is equal to the distribution of the service phase at the beginning of an all-busy period, which is given by

$$c_2 \boldsymbol{\pi}(0) \int_0^\infty \exp(Tu) S^*(u) Q^{-1}_{\text{not-all-busy}} (R_{\text{ser}} \otimes D_1) du,$$

where c_2 is a normalizing constant that ensures that the vector is stochastic. This expression considers that the age during an all-busy period is u with probability density $\boldsymbol{\pi}(0)\exp(Tu)$, and a transition according to $S^*(u)$ triggers the beginning of a not-all-busy period. The matrix $S^*(u) = S_{\text{(all)-(not-all)}} \otimes \exp(D_0 u)$ captures that the next arrival occurs after u time units, such that a service completion ends the all-busy period. Next, the system evolves according to the sub-generator $Q_{\text{not-all-busy}}$ (given in Appendix A), until an arrival finds at most r idle servers, initiating an all-busy period and selecting its initial service phase according to $R_{\text{ser}} \otimes D_1$. Note that we can make use of the integral term P, defined in Appendix A, $c_2 \boldsymbol{\pi}(0) P Q^{-1}_{\text{not-all-busy}} (R_{\text{ser}} \otimes D_1)$, and $c_2^{-1} = \boldsymbol{\pi}(0) P Q^{-1}_{\text{not-all-busy}} (R_{\text{ser}} \otimes D_1)\mathbf{1}$ makes this vector stochastic. As a result, the initial service phase of a job that initiates the all-busy period is distributed according to $\boldsymbol{s}^{\text{part}}_{\text{not-busy}}$ in Eq. (8). □

From the results above, the initial distribution of the job service phase is given by the following theorem.

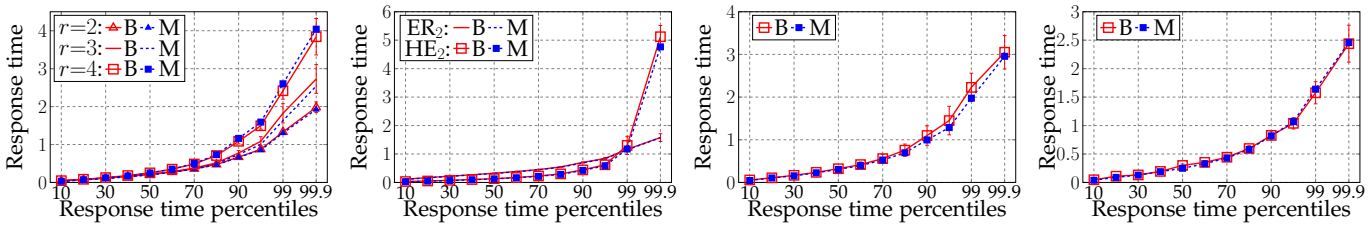
Theorem 1. The stationary distribution of the job initial initial service phase $\boldsymbol{s}_{\text{ser}}$ is given by

$$\boldsymbol{s}_{\text{ser}} = \boldsymbol{s}_{\text{busy}} + \boldsymbol{s}^{\text{all}}_{\text{not-busy}} + \boldsymbol{s}^{\text{part}}_{\text{not-busy}}. \quad (9)$$

3.3 The response-time distribution

We can now put together the PH representations of waiting and service times to obtain the PH representation $(\boldsymbol{s}_{\text{res}}, S_{\text{res}})$ of the response-time distribution as

$$\begin{aligned} \boldsymbol{s}_{\text{res}} &= [\boldsymbol{s}_{\text{wait}}, \quad \boldsymbol{s}^{\text{all}}_{\text{not-busy}} + \boldsymbol{s}^{\text{part}}_{\text{not-busy}}], \\ S_{\text{res}} &= \begin{bmatrix} S_{\text{wait}} & (-S_{\text{wait}}\mathbf{1})\boldsymbol{s}_{\text{busy}}/\gamma \\ \mathbf{0}^{m_a m_{\text{all-busy}}}_{m_{\text{ser}}} & S_{\text{ser}} \end{bmatrix}. \end{aligned} \quad (10)$$



(a) (Poisson, Exp, 4) (b) (Poisson, ER_2/HE_2, 4) (c) (MAP, Exp, 4) (d) (Poisson, Exp, 20)

Fig. 5: Experimental validation with the MATLAB benchmark on response-time percentiles. Each experiment is labeled by a tuple of (arrival process, processing-time distribution, number of servers). B and M denote results obtained from the benchmark and the model, respectively.

Note that to define vector $\boldsymbol{s}_{\text{res}}$ we make use of the $\boldsymbol{s}^{\text{all}}_{\text{not-busy}}$ and $\boldsymbol{s}^{\text{part}}_{\text{not-busy}}$ vectors in Eq. (9), which cover the cases where a job starts service without waiting. Other jobs must first undergo a waiting time, starting with vector $\boldsymbol{s}_{\text{wait}}$, after which they start service according to vector $\boldsymbol{s}_{\text{busy}}/\gamma$, where γ is used to re-normalize the vector to be stochastic. With this PH representation, we can directly compute the response-time CDF, percentiles, and moments.

Remark 1. From the previous description it is clear that the sets $N_{\text{all-busy}}$, $N_{\text{not-all-busy}}$, and N_{ser} grow large rapidly with the number of phases m_R and the number of servers C. We have extended the numerical methods in [25] to efficiently compute the matrix T, the vector $\boldsymbol{\pi}(0)$, as well as moments and percentiles of the response-time distribution. Appendix Ehighlights the key ideas behind this solution method.

4 Validation

To test the accuracy of the model introduced in the previous section, we have implemented replication for a simple MATLAB benchmark application, which is executed on a multiprocessor host where each processor plays the role of a server in the model. This section describes the experimental set-up and results.

4.1 Experimental Set-up

We have implemented replication for a MATLAB benchmark application in which requests arrive to a central dispatcher that keeps a request queue and allocates the first request in the queue to the next available worker. Workers are mapped to independent processors such that the system operation follows the reference model introduced in Section 2.1. In particular, we employ the MATLAB blackjack benchmark [12], where each request emulates a game of blackjack that consists of a pre-defined number of blackjack hands. As the execution time *per hand* is fairly deterministic, we modify the distribution of the request execution time by altering the distribution of the number of hands.

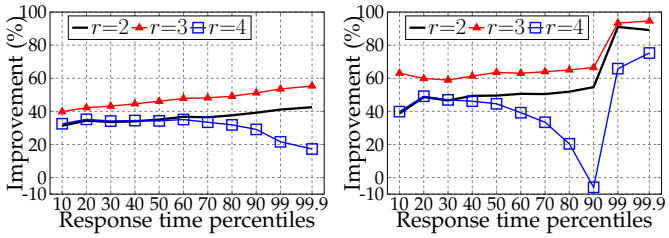
In our set-up requests are generated according to a Poisson or a MAP process. We use MAPs to consider arrivals with high variability and auto-correlation, and we set the squared coefficient of variation (SCV)[2] of the inter-arrival times to 10 and the decay rate of the auto-correlation function to 0.5. To set the request execution time we fix the *mean* number of hands to 2000 per request, and generate the number of hands according to three PH distributions, namely, exponential (Exp), 2-phase Erlang (ER_2), and hyper-exponential (HE_2). The random variates generated are rounded to obtain an integer number of hands. The SCV for ER_2, Exp, and HE_2 is 0.5, 1, and 10, respectively, showing an increasing degree of variability. We use the methods of [26], [27] to obtain PH and MAP representations with these moments. In each experiment we run the benchmark ten times, each time executing a total of 5000 requests, and compute the response time percentiles and their 95% confidence intervals.

2. SCV $= Var[Y]/E[Y]^2$ for a random variable Y, where $Var[Y]$ and $E[Y]$ are the variance and expected value of Y, respectively.

4.2 Performance Prediction

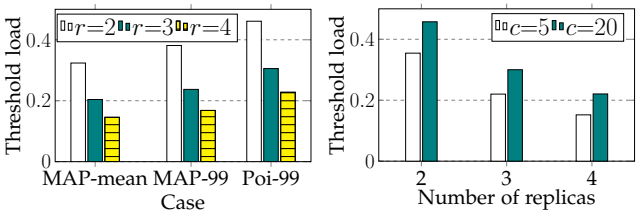
We ran the benchmark application on an Intel Core i7-3770 machine with 4 cores running at 3.4 GHz, and 16 GB of memory, launching one worker per core. Running the application without replication we obtain a mean processing time of 0.52 sec, which we use to parameterize the model. Setting the arrival rate to 1 request per second we obtain a load of 0.13. We compare the measured and the predicted response times in Figure 5, where each experiment is labeled by a tuple of (arrival process, processing-time distribution, number of servers), and we denote benchmark results by B and model results by M in the legend. Under Poisson arrivals and exponentially-distributed processing times, as in Figure 5(a), our model shows remarkably good prediction results for replication factors r=2, 3 and 4. For instance, the average error across all the percentiles depicted for the r=2 case is only 2.72%. Focusing on the r=2 case, Figure 5(b) considers more general request processing-time distributions, ER_2 and HE_2, where we observe that our model is also accurate for systems with non-exponential processing times. Similarly, replacing Poisson arrivals by MAP, Figure 5(c) shows that our model predicts results well even under highly varying and auto-correlated arrivals.

We have also deployed the application on a shared cluster made of 4 Intel Xeon E5-4650 machines, with a total of 32 cores running at 2.70 GHz, and 512 GB of memory. Here we deploy 20 workers and set the arrival rate to 4 requests per second. With a measured mean request execution time of 0.71 sec, we achieve a utilization of 0.14. Figure 5(d) displays the results for this case, under Poisson arrivals and exponentially distributed processing times. We observe that our model predicts the entire response-time distribution



(a) ER_2 services (b) HE_2 services

Fig. 6: Percentile improvement.



(a) Different arrival processes and performance metrics (b) Different number of servers

Fig. 7: Threshold load.

well, showing a good prediction capability under a high level of parallelism.

5 Insights regarding Replication

The excellent accuracy of the proposed model enables us to explore a large experimental space efficiently and derive insights regarding replication. Particularly, we aim to answer two key questions: (i) when can replication improve the performance and to what extent; and, (ii) what is the maximum baseline load for replication to improve the response times compared with the baseline.

5.1 The Impact of Replication

For the results in this section we consider a system with C=20 servers, Poisson arrivals and two processing-time distributions, namely, ER_2 and HE_2, defined in Section 4. The mean processing time is set to 1 sec, and the arrival rate is set to achieve a baseline load of 0.2. Figure 6 depicts the relative improvement for each percentile p in the range $\{10, 20, \ldots, 90, 99, 99.9\}$, comparing the response times obtained with replication levels r=2, 3 and 4 against those without replication. For r=2 and 3, the case under ER_2 services, as shown in Figure 6(a), has a relatively smooth behavior, with the improvement increasing slightly with the percentile. In contrast, for HE_2 services we observe large peaks in the tail. For instance, for r=3 the improvement on the tail is as high as 95%, while the average improvement is around 60%. We have observed a similar behavior in other set-ups, indicating that replication is effective in reducing latency, particularly the tail of the latency distribution. While the improvement increases from r=2 to 3 for all the percentiles, it decreases when r increases to 4, especially for the tail under ER_2 services, and between the 50^{th} and 90^{th} percentiles under HE_2 services. In fact, having 4 replicas is harmful for the 90^{th} percentile under HE_2 services. Thus, care must be taken in evaluating the specific percentiles of interest as the effect of replication is nonuniform over the response-time distribution, to the point that it may benefit some percentiles while hurting others. Also, the service time distribution must be adequately characterized as it has a large impact on the benefits of replication.

5.2 The threshold load

We now turn to find the *threshold load*, that is, the maximum baseline load under which replication is beneficial. Here we illustrate our main observations by means of some example set-ups, but we have observed similar behaviors in many other configurations. Note that, for a replication factor r, the baseline load is upper bounded by $1/r$, as introducing r replicas in a system with a baseline load larger than $1/r$ would make it unstable. To determine the threshold load, we rely on the golden section search method [28]. We summarize our key findings in the following.

1. The threshold load is nonuniform along the response-time distribution. Figure 7(a), where we assume 20 servers and HE_2 services with a mean processing time of 1 sec, shows how the threshold load changes when evaluating different response-time metrics. Comparing the first two cases, which assume MAP arrivals, we observe that the threshold load is always larger if we evaluate the 99^{th} percentile rather than the mean. As a result, a decision to replicate based solely on potential gains for the response time tail may actually have a negative effect on the mean. For instance, in this set-up we find that under a load of 0.4 the 99^{th} percentile *decreases* by 20% with the introduction of one replica, while the mean *increases* by over 90%. This highlights the importance of explicitly considering the response-time distribution, and the targeted percentiles, when evaluating a replication strategy.

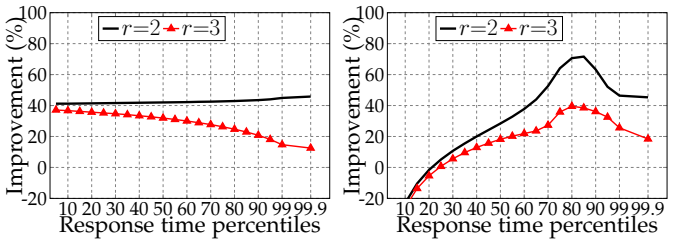
2. The threshold load decreases with the variability and auto-correlation of the arrival process. Replacing MAP arrivals by Poisson, as shown in the third case in Figure 7(a), we observe a large increase in the threshold load. For instance, for r=2, the threshold load increases from 0.38 under MAP arrivals to 0.46 under Poisson arrivals. This is caused by the high burstiness of the MAP process, as replication during bursty periods increases the probability of causing a high utilization, and thus longer delays. Thus, results based on the Poisson assumption, when the actual arrival process displays high variability and/or auto-correlation, can lead to the wrong decision of introducing replication when it is not beneficial.

3. The threshold load increases with the number of servers. Figure 7(b) shows the effect of the number of servers on the threshold load, using the 95^{th} percentile as the decision-making metric, under Poisson arrivals and Exp services with mean service rate $\mu = 1$. Clearly, the threshold of the 5-server case is smaller than that of the 20-server case. In particular, when introducing 2 replicas, the threshold load for the 5-server case is 0.35, whereas it is 0.46 for the 20-server set-up. Thus, more servers provide more flexibility to benefit from replication.

In conclusion, the threshold load decreases with the number of replicas as well as with the variability and auto-correlation of the arrival process, but increases with the number of servers.

TABLE 4: Mean errors in percentiles fitting power-law traces: comparing exponential fit and hyper-Erlang fit.

	Expo	PH (hyper-Erlang)				
Num. Phases	1	2	5	8	10	15	20
Avg. Error (%)	51.0	44.8	30.9	20.5	18.5	16.1	15.2



(a) Exponential fit (b) PH (hyper-Erlang) fit

Fig. 8: Percentile improvement for power-law traces: comparing exponential fit and hyper-Erlag fit with different replication factors.

5.3 Heavy-tailed execution times

Task execution times in production clusters have been observed to follow power-law [21] or heavy-tailed [17] distributions, which are characterized by having a right tail that decreases slower than that of an exponential distribution. We assume that execution times follow a PH distribution, which do not display a heavy tail. However, hyper-exponential distributions, a special case of PH distributions, have been used in [29], [30] to approximate the heavy-tailed behavior observed in internet traffic traces. The key to this approximation is to use each phase in the distribution to approximate the behavior of the data trace along a different scale of values.

To evaluate the impact of replication under a power-tail behavior we rely on a similar approximation, using a hyper-Erlang distribution to approximate the Bellcore trace BC-pAug89, which has been shown to display a power-law behavior [31]. We opt for the hyper-Erlang distribution as it is more flexible than the hyper-exponential used in [29], [30]. To obtain the hyper-Erlang distribution from the BC-pAug89 trace we employ the method proposed in [32], as implemented in [33]. Table 4 displays the mean relative difference between the percentiles of the fitted distribution and the trace percentiles. Specifically, we compare the percentiles $\{1, 2, \ldots, 99\}$. With the exponential fit the mean error is over 50%, which reflects a large disagreement between the observed distribution and the fitted one. Instead, with a PH fit the mean error decreases to 20% with 8 phases and to 15% with 20 phases. Note that all these distributions match the observed mean perfectly, but matching the percentiles is much harder. Even so, the PH distributions are able to approximate the trace percentiles well.

We now look into how the choice of distributions used to model the BC-pAug89 trace affects the impact of replication on the request latency. Figure 8 depicts the improvement in the response time percentiles when introducing 2 and 3 replicas, with 5 servers and a baseline load of 0.2. Figures 8(a) and (b) correspond to an exponential fit and a PH fit with 8 phases, respectively. With one additional replica (r=2) the exponential fit predicts a fairly stable improvement around 40% across all percentiles. Instead, the PH distribution shows that one additional replica actually increases the low (up to the 20-th) percentiles, but decreases by over 60% the 70-th to 90-th percentiles. This difference is caused by the ability of the PH distribution to better capture the tail behavior of the service time distribution, enabling the model to better portray the complex impact that replication has under such heavy-tailed distributions.

6 Correlated Processing Times

As illustrated in Section 2, replicas of a request display similarities in their processing times. As the model we have introduced assumes independent processing times, these similarities have not been considered. In fact, the independence of processing times is a standard assumption in queueing models, and there are few results for correlated processing times. When such a correlation is considered, as in [34], it is modeled as a general correlation between the processing times of successive requests, whereas here we are interested in correlating the processing times of the replicas of the same job. We thus introduce a model that explicitly captures the correlation among replica processing times and, importantly, fits within the analysis framework defined in Section 3.

To model correlated processing times within a Markovian model, we introduce the concept of correlated hyper-Erlang (CHE) distributions. Hyper-Erlang distributions [32] model convex combinations of Erlang distributions and are a sub-class of PH distributions. Here we extend this set of distributions to incorporate a dependence between replicas of the same job. Let a CHE distribution be characterized by a set of B branches $\mathcal{B}$, the probability α_i of choosing branch $i \in \mathcal{B}$, the number of exponential phases h_i in branch $i \in \mathcal{B}$, and the rate λ_i in each phase of branch $i \in \mathcal{B}$. These are the parameters that characterize a standard hyper-Erlang distribution. The key difference here is that, while each job is allowed to select a branch independently, all replicas of a job must select the same branch. To model this behavior, we let the first replica of a job select branch $i \in \mathcal{B}$ with probability α_i, and force all other replicas from the same job to select the same branch as the first replica. With these definitions, we extend the model of Section 3 to handle this modified service process. We provide the details in Appendix C.

***Remark* 2.** Under the CHE service model the effective service time of each replica follows an Erlang distribution. This assumption can be generalized so that these times follow hyper-Erlang distributions, which are dense in the set of all continuous distributions with non-negative support. However, we have found that this generalization has little impact on the estimation errors with the experimental data described in the following sections. This suggests that the simpler model captures well the experimental data for the set-up considered. For this reason, and in the interest of clarity, we describe here the simpler case with Erlang processing times and leave the generalization for Appendix F.

6.1 Fitting Correlated Processing Times

The CHE service-time model introduced above allows the r replicas of a job to be correlated in the sense that all of them

select the same Erlang branch of the CHE distribution. The benefits of the CHE service model can only be exploited as long as we are able to fit the model parameters from a data trace. While fitting methods exists for general PH distributions [35] and some of its sub-classes [30], [32], no such method exist for the CHE service model since this is the first time this model is proposed. We now propose a maximum-likelihood method for this purpose, based on the Expectation Maximization (EM) algorithm proposed by Dempster et al. [36]. Because our analysis method obtains the response-time *distribution*, its ability to provide accurate results depends on capturing the processing-time *distribution* well. As a maximum-likelihood approach considers the overall processing-time distribution, it appears better suited than the other widely adopted class of fitting method, moment matching [26], which captures only the first few moments. Also, the popularity of the EM method for PH distributions is due to its ability to handle mixtures of distributions, which we can also exploit for CHE distributions.

We assume a trace $\mathcal{D}=\{\boldsymbol{x}_1,\ldots,\boldsymbol{x}_D\}$, where each of the D observations is a tuple $\boldsymbol{x}_d=(x_d^1,\ldots,x_d^r)$ holding the processing times of the r replicas of a job. From this trace, we want to determine the entire set of parameters of the CHE model $(B,h_1,\ldots,h_B,\alpha_1,\ldots,\alpha_B,\lambda_1,\ldots,\lambda_B)$. However, and similar to [32], we split this problem in two. First, we assume the number of branches B, and the number of phases h_i in each branch is known. Thus, we focus on determining the parameters $\Theta=(\alpha_1,\ldots,\alpha_B,\lambda_1,\ldots,\lambda_B)$, that is, the probability α_i of choosing each branch and the corresponding Erlang rate λ_i, for $i\in\mathcal{B}$. Later in this section, we consider how to determine B and h_i.

The basic idea behind the EM algorithm is to start with an initial guess of the parameters $\hat{\Theta}$. We then iterate to obtain new values of $\hat{\Theta}$ that maximize the log-likelihood function by following the derivation detailed in Appendix D. Specifically, we compute $\delta(b|\boldsymbol{x}_d,\hat{\Theta})$, the probability that branch b is selected given data $\boldsymbol{x}_d$, and estimates $\hat{\Theta}$, which is the expectation step in the EM method. In the optimization step, we obtain the branch selection probabilities and the Erlang rates as

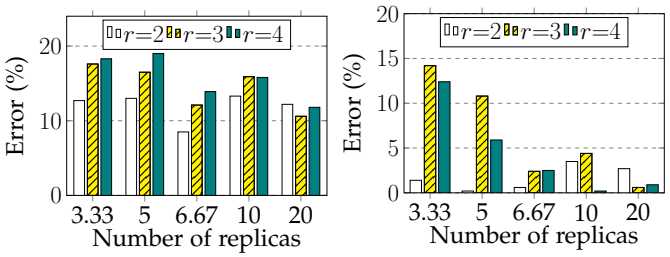
$$\alpha_b=\frac{1}{D}\sum_{d=1}^{D}\delta(b|\boldsymbol{x}_d,\hat{\Theta}),\ \lambda_b=\frac{rh_b\sum_{d=1}^{D}\delta(b|\boldsymbol{x}_d,\hat{\Theta})}{\sum_{d=1}^{D}\delta(b|\boldsymbol{x}_d,\hat{\Theta})\sum_{j=1}^{r}x_d^j}. \quad (11)$$

As evidenced in this last expression, derived in Appendix D, the key characteristic of this method, and what makes it different from existing ones, is that it considers each sample $\boldsymbol{x}_d$ as an r-tuple, and these r processing-times samples are tied by the selection of the same Erlang branch, in agreement with the CHE model.

We are now ready to state the EM algorithm for the CHE service model, summarized in Algorithm 2. The algorithm requires the total number of phases m_R, the data $\mathcal{D}$, and a stopping criterion ϵ. As m_R is equal to the sum of the number of stages in all branches, $\sum_{i\in\mathcal{B}}h_i$, the algorithm determines all possible combinations for the number of branches and number of stages in each branch $(B,h_1,\ldots,h_B)$, as in [32] for the independent hyper-Erlang case. For each possible combination, it then executes the EM steps derived in Appendix D, starting from an initial guess for the parameters and iterating until the difference between

Algorithm 2 EM algorithm for the CHE service model

Require: Data: $\mathcal{D}$, Number of phases: m_R, ϵ
1: $\mathcal{S}=\{(B,h_1,\ldots,h_B)|\sum_{b=1}^{B}h_i=m_R\}$
2: bestLH $=\infty$
3: **for** $(B,h_1,\ldots,h_B)\in\mathcal{S}$ **do**
4: Initial guess $\hat{\Theta}=(\hat{\alpha}_1,\ldots,\hat{\alpha}_B,\hat{\lambda}_1,\ldots,\hat{\lambda}_B)$
5: diff $=1$
6: **while** diff $>\epsilon$ **do**
7: Compute $\delta(b|x_n,\hat{\Theta})$, $b=1,\ldots,B$, $d=1,\ldots,D$ as in (26), (24)
8: Compute α_b, λ_b, $b=1,\ldots,B$ as in (11)
9: diff $=\max|\hat{\Theta}-(\alpha_1,\ldots,\alpha_B,\lambda_1,\ldots,\lambda_B)|$
10: $\hat{\Theta}=(\alpha_1,\ldots,\alpha_B,\lambda_1,\ldots,\lambda_B)$
11: **end while**
12: Compute $\log L(\hat{\Theta}|\mathcal{D})$ as in (25)
13: **if** $\log L(\hat{\Theta}|\mathcal{D})>$ bestLH **then**
14: $\Theta^*=\hat{\Theta}$
15: bestLH $=\log L(\hat{\Theta}|\mathcal{D})$
16: **end if**
17: **end for**
18: **return** Θ^*



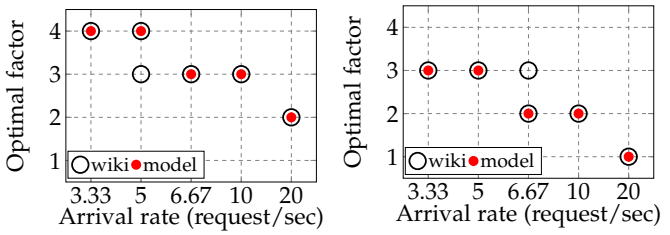
(a) Mean response time (b) 99^{th} percentile

Fig. 9: Experimental validation with MediaWiki.

two successive sets of estimates in Eq. (11) is less than the pre-defined limit ϵ. Once the estimates have been found, it computes the log-likelihood and compares it against the best one found so far, keeping the set of estimates Θ^* with the highest log-likelihood, which is returned at the end of execution. Note that considering all possible combinations $(B,h_1,\ldots,h_B)$ for a given number of phases m_R is feasible as long as this number is moderate. Given that the CHE service model is to be used for analysis as described in Section 6, we are actually interested in small to moderate values for m_R, and the full enumeration is thus feasible. In Section 7, we illustrate how this method is able to capture the correlation structure in the processing times of request replicas from a real system.

7 Evaluation on MediaWiki

In this section, we evaluate the capability of the proposed model to predict the response-time distribution of a real-life application, MediaWiki, and show how our proposed analysis can guide the selection of optimal replication factors. We first present the experimental set-up, followed by extensive experiments.



(a) Mean response time (b) 99^{th} percentile

Fig. 10: Optimal replication factor: measurement vs. model.

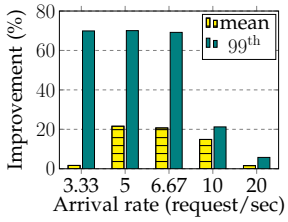


Fig. 11: Improvement using optimal factor r^*.

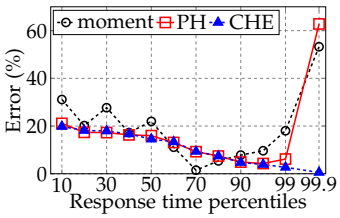


Fig. 12: Prediction errors using different fitting methods.

7.1 Set-up

Our private cloud testbed is composed of eight identical physical servers, seven used to run the experiments and one used as experiment orchestrator and repository. Each server is equipped with 32 cores, 128 GB DDR4 RAM, six 1-TB solid state disks in RAID5, and two 10-Gigabit Ethernet adapters. We use MediaWiki, the open source platform used to run the Wikipedia website, as a representative application in the cloud [13]. MediaWiki is a latency-sensitive three-tier web application composed of Apache (v2.4.7) plus PHP (v5.5.9) as application server frontend, Memcached (v1.4.14) as in-memory key-value store and MySQL (v5.5.40) as database backend. In addition, we use an in-house dispatcher written in Go, which replicates and distributes HTTP requests. We deploy a MediaWiki cluster consisting of seven VMs configured with two virtual CPUs and 4 GB of RAM. Six VMs run a complete stack of all three tiers, and one VM is used as the dispatcher. We deploy the cluster in two set-ups: (i) dense, six VMs on three physical servers, each server holding two VMs; and (ii) sparse, each VM hosted on a separate physical server. The average processing time per request for each MediaWiki VM is $1/\mu = 72.98$ msec. Requests are generated with httperf [37], an open-loop workload generator. Due to the space limit, we only present the results for the dense set-up in the following.

Upon receiving a request, the dispatcher immediately initiates r replicas of the same request and dispatches the replicas to any available VM, with FCFS scheduling. The dispatcher ensures that the maximum number of replicas at each VM is one and that the outstanding requests all wait at the dispatcher. This set-up is used in distributed systems, such as Spark, due to its ability to inherently adjust the request dispatching rate to the variable speed of each VM.

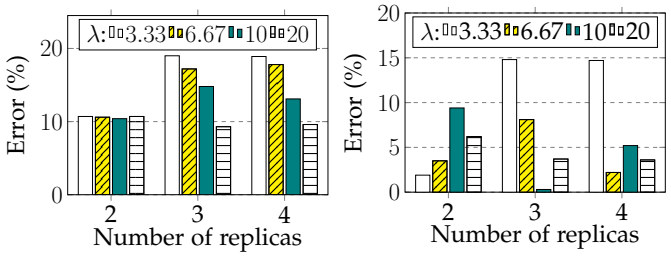
To emulate performance variability in the public cloud, we artificially spawn neighboring workloads following Poisson arrivals and exponential run times. The specific neighboring workload used is fluidanimate, a CPU-intensive benchmark from PARSEC 3.0 [38]. Each MediaWiki is co-located with such a neighbor, and we keep the average active time of neighboring workloads around 50% of the MediaWiki experiment time using a mean inter-arrival time of 60 sec and a mean runtime of 30 sec.

7.2 Results

We validate our proposed analysis against seven load scenarios, considering Poisson arrivals with mean arrival rates λ of 3.33, 5, 6.67, 10, 20, 30 and 40 requests per second, thus achieving a baseline load of 0.04, 0.06, 0.08, 0.12, 0.24, 0.36 and 0.48 without replication, respectively. For each combination of arrival rate and replication factor, we collect mean and 99^{th} percentile across 50000 requests resulting in over 100 hours of experiment time. For each arrival rate and replication factor, we parameterize the model from the trace using a CHE distribution with 10 phases and 2 branches. Figure 9 summarizes the relative percentage error between the analytical results and the measurements, focusing on the response time mean and 99^{th} percentile. Here we omit the results for λ=30 and 40 as they do not benefit from replication. The errors for these two cases are 11.5% and 9.9% for the mean, and 1.2% and 0.1% for the 99^{th} percentile, when r=2 for λ=30 and 40, respectively. Our model slightly underestimates the mean response time, although the errors are all below 20%. In contrast, we observe that our model performs very well on the 99^{th} percentile, with most errors under 5%. We note that such fitting results are remarkable given the complexity of the application considered.

A key requirement for a system that implements replication is to determine the optimal replication factor r^*. Figure 10(a) and (b) show the empirical and analytical replication factors that minimize the mean and the 99^{th} percentile, respectively. We find that in certain cases the testbed offers very similar response times under two replication factors. In these cases we allow multiple optimal replication factors if the response times differ by less than 5%. For instance, for an arrival rate of 5, the mean response time for r=2 and 3 is 62.93 and 63.76 msec, respectively, a difference of just 1.32%, thus we consider both replication factors as optimal. From Figure 10(a) and (b) we observe that our model identifies the optimal replication factor in all cases considered. As expected, the optimal replication factor decreases with increasing arrival rate, owing to the extra loads introduced by replication. A key observation here is that the optimal replication factors for the 99^{th} percentile are lower than those for the mean. This confirms our observations in Section 5 and highlights the importance of considering the targeted percentiles when choosing the replication factor.

When the optimal number of replicas is adopted, Figure 11 depicts the improvement in the mean and the 99^{th} percentile against the set-up without replication. Clearly, replication improves the response times significantly, and has a stronger impact on the tail than on the mean. For instance, when the arrival rate is 5, the improvement is 21.57% on the mean and 70.05% on the 99^{th} percentile. Moreover, for both the mean and the 99^{th} percentile, the improvement is much stronger for low arrival rates as it is more likely for several replicas to execute in parallel, thus the system benefits from selecting the replica that completes



(a) Mean response time (b) 99^{th} percentile

Fig. 13: Prediction errors using $\lambda = 5$.

first. Instead, under high loads, the extra load introduced by replication leads to longer queueing times, diminishing the benefit.

We now evaluate the introduction of correlation among the replicas' processing times and compare the predictions obtained from three different fitting methods: a moment-matching method [26], an EM method for independent PH distributions [32], and the EM method for CHE distributions introduced in Section 6.1. We consider the case with arrival rate 6.67 and replication factor r=2. Figure 12 shows the prediction errors on the response time percentiles, $\{10, \ldots, 90, 95, 99, 99.9\}$, compared with the testbed measures. Clearly, the CHE distribution achieves the lowest errors, especially at the tail, where the error on the 99.9^{th} percentile is as low as 0.55%. The independent PH distribution performs well up to the 95^{th} percentile, but fails to capture the tail, which is a key performance metric. The moment-matching method, in contrast, behaves erratically, with a large error on the tail. Further, we compare the ability of the CHE and PH models to capture the 99.9^{th} percentile under different replication factors. With r=2, the 99.9^{th} percentile predicted with the CHE model is 0.3452, very close to the measurement of 0.3471, while the PH model predicts 0.1291. This difference however diminishes as the replication factor increases, since more replicas benefit more from the resource diversity, weakening the pair-wise correlation. Thus, increasing r to 3, the measured 99.9^{th} percentile is 0.1707, very close to the 0.1667 predicted by the CHE model, while the PH model estimates 0.1415. In terms of optimal replication factor, the CHE model predicts 3 replicas – in agreement with the measurement, whereas the PH model chooses 2 replicas. Again, this observation highlights the advantage of considering the correlation explicitly by fitting the processing times using the CHE model.

Finally, we show that our model can be used to predict the response times under different loads and replication factors. Note that the characteristics of the replica processing-time distribution, in particular, its variability and correlation, differ under different replication factors. Thus, to predict the response times for a given replication factor r, we first fit the replica processing time from a historical trace with the same r, under any load. We then use the fitted distribution to predict the response times for different arrival rates and the same r. As an example, we fit the replica processing times observed for the case with λ=5 and r=2, and use them to predict the response times for cases with arrival rates of 3.33, 6.67, 10, and 20, all with r=2. Figure 13 shows the errors obtained for the mean and 99^{th} percentile, considering replication factors r=2, 3 and 4. We observe a good prediction performance for all arrival rates, with an average error of 12.61% for the mean and 4.89% for the 99^{th} percentile, similar to those observed in the validation setup. The model accuracy is thus resilient to being parameterized under one arrival rate and used under any other arrival rate.

All in all, our analysis can guide the choice of optimal replicas for any response time percentile and a wide range of load scenarios, including different arrival patterns and highly varying and dependent processing times.

8 Concluding Remarks

The results in the previous sections show that the models proposed in this paper are able to capture the effect of replication without canceling on the response-time distribution, accurately estimating the significant impact of the processing-time distribution and the arrival process, particularly pertaining to their variability and auto-correlation. The remarkable accuracy on predicting response-time distributions at a wide range of load scenarios enables us to derive insights on adopting replication. In particular, we observe that the impact of replication is not homogeneous across the response-time percentiles and that the threshold load, i.e., the maximum load under which replication offers latency gains, can differ if the evaluation is based on the response-time mean or on a specific percentile. Also, the introduction of the CHE service model enables us to incorporate the observed correlation among the processing times of replicas of the same request. Further, the model results, extensively validated on a three-tier web application (MediaWiki) and a MATLAB benchmark, is effective in identifying the optimal replication factors for different latency metrics, i.e., mean vs. tail percentiles, highlighting the importance of analyzing response-time distribution when designing replication policies.

A limitation of the method lies in that it operates on matrices that can grow very fast in size with the number of servers, especially if the processing-time distribution has many phases. There is therefore a trade-off between the accuracy gained by including more phases in the processing-time distribution and the scalability of the models. For instance, by exploiting the efficient methods devised in Appendix E, with the number of phases equal to two we are able to solve models with up to 50 servers in less than 2 seconds on a commodity desktop. Future work will look into alternative approaches to tackle large-scale systems that adopt replication.

References

[1] B. Farley, A. Juels, V. Varadarajan, T. Ristenpart, K. D. Bowers, and M. M. Swift, "More for your money: exploiting performance heterogeneity in public clouds," in *ACM SOCC*, 2012.

[2] J. Schad, J. Dittrich, and J.-A. Quiané-Ruiz, "Runtime measurements in the cloud: observing, analyzing, and reducing variance," *VLDB Endow.*, vol. 3, pp. 460–471, 2010.

[3] J. Dean and L. A. Barroso, "The tail at scale," *CACM*, vol. 56, pp. 74–80, 2013.

[4] 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," in *USENIX OSDI*, 2010.

[5] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica, "Improving mapreduce performance in heterogeneous environments," in *USENIX OSDI*, 2008.

[6] Z. Qiu and J. F. Pérez, "Evaluating the effectiveness of replication for tail-tolerance," in *IEEE CCGRID*, 2015.

[7] ——, "Enhancing reliability and response times via replication in computing clusters," in *IEEE INFOCOM*, 2015.

[8] ——, "Assessing the impact of concurrent replication with canceling in parallel jobs," in *IEEE MASCOTS*, 2014.

[9] N. B. Shah, K. Lee, and K. Ramchandran, "When do redundant requests reduce latency?" in *Allerton*, 2013.

[10] K. Gardner, S. Zbarsky, S. Doroudi, M. Harchol-Balter, and E. Hyytia, "Reducing latency via redundant requests: Exact analysis," in *ACM SIGMETRICS*, 2015.

[11] G. Joshi, E. Soljanin, and G. Wornell, "Efficient replication of queued tasks for latency reduction in cloud systems," in *Allerton*, 2015.

[12] "Simple benchmarking of parfor using blackjack," http://uk.mathworks.com/help/distcomp/examples/simple-benchmarking-of-parfor-using-blackjack.html, 2015.

[13] "Mediawiki," https://www.mediawiki.org/wiki/MediaWiki, 2015.

[14] A. Vulimiri, P. Godfrey, R. Mittal, J. Sherry, S. Ratnasamy, and S. Shenker, "Low latency via redundancy," in *CoNEXT*, 2013.

[15] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica, "Why let resources idle? aggressive cloning of jobs with Dolly," *Memory*, vol. 40, p. 80, 2012.

[16] B. Snyder, "Server virtualization has stalled, despite the hype," http://www.infoworld.com/print/146901, 2010.

[17] X. Ren, G. Ananthanarayanan, A. Wierman, and M. Yu, "Hopper: Decentralized speculation-aware cluster scheduling at scale," in *ACM SIGCOMM*, 2015.

[18] ——, "Speculation-aware cluster scheduling," *SIGMETRICS Perform. Eval. Rev.*, vol. 43, no. 2, 2015.

[19] K. Lee, R. Pedarsani, and K. Ramchandran, "On scheduling redundant requests with cancellation overheads," in *Allerton*, 2015.

[20] K. Gardner, M. Harchol-Balter, and A. Scheller-Wolf, "A better model for job redundancy: Decoupling server slowdown and job size," in *MASCOTS*, 2016.

[21] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica, "Effective straggler mitigation: Attack of the clones," in *USENIX NSDI*, 2013.

[22] G. Latouche and V. Ramaswami, *Introduction to matrix analytic methods in stochastic modeling*. SIAM, 1999.

[23] S. Asmussen and J. R. Møller, "Calculation of the steady state waiting time distribution in GI/PH/c and MAP/PH/c queues," *Queueing Syst.*, vol. 37, pp. 9–29, 2001.

[24] Q. He, "Analysis of a continuous time SM[K]/PH[K]/1/FCFS queue: Age process, sojourn times, and queue lengths," *JSSC*, vol. 25, pp. 133–155, 2012.

[25] Z. Qiu and J. F. Pérez, "Evaluating replication for parallel jobs: An efficient approach," accepted in IEEE TPDS.

[26] W. Whitt, "Approximating a point process by a renewal process, I: Two basic methods," *Oper. Res.*, vol. 30, pp. 125–147, 1982.

[27] J. E. Diamond and A. S. Alfa, "On approximating higher order MAPs with MAPs of order two," *Queueing Syst.*, vol. 34, pp. 269–288, 2000.

[28] J. Kiefer, "Sequential minimax search for a maximum," *Proc. Amer. Math. Soc.*, vol. 4, pp. 502–506, 1953.

[29] A. Feldmann and W. Whitt, "Fitting mixtures of exponentials to long-tail distributions to analyze network performance models," *Perform. Eval.*, vol. 31, pp. 245–279, 1998.

[30] R. E. A. Khayari, R. Sadre, and B. Haverkort, "Fitting world-wide web request traces with the EM-algorithm," *Perform. Eval.*, vol. 52, pp. 175–191, 2003.

[31] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson, "On the self-similar nature of ethernet traffic (extended version)," *IEEE ToN*, vol. 2, pp. 1–15, 1994.

[32] A. Thummler, P. Buchholz, and M. Telek, "A novel approach for phase-type fitting with the EM algorithm," *IEEE TDSC*, vol. 3, pp. 245–258, 2006.

[33] J. F. Pérez, D. F. Silva, J. C. Góez, A. Sarmiento, A. Sarmiento-Romero, R. Akhavan-Tabatabaei, and G. Riaño, "Algorithm 972: jMarkov: An integrated framework for Markov chain modeling," *ACM Trans. Math. Softw.*, vol. 43, no. 3, pp. 29:1–29:22, 2017.

[34] V. Gupta, M. Burroughs, and M. Harchol-Balter, "Analysis of scheduling policies under correlated job sizes," *Perform. Eval.*, vol. 67, no. 11, pp. 996–1013, 2010.

[35] S. Asmussen, O. Nerman, and M. Olsson, "Fitting phase-type distributions via the EM algorithm," *Scand. J. Statist.*, vol. 23, pp. 419–441, 1996.

[36] A. Dempster, N. Laird, and D. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *J. Roy. Statist. Soc.*, vol. 39, pp. 1–38, 1977.

[37] "httperf," https://github.com/httperf/httperf, 2015.

[38] C. Bienia, "Benchmarking modern multiprocessors," Ph.D. dissertation, Princeton University, January 2011.

Zhan Qiu is a PhD student in computer science at Imperial College London. She received a MSc in Computer Science from Imperial College London in Oct 2012. Her research interests include data-driven performance engineering and evaluation of computer and communication systems, parallel processing, performance optimization and resource provisioning. She is a member of the IEEE, and the IEEE Communications Society.

Juan F. Pérez is Assistant Professor at Universidad del Rosario, Colombia, Department of Applied Mathematics and Computer Science. He obtained a PhD in Computer Science from the University of Antwerp, Belgium, in 2010. He was a Research Associate in performance analysis at Imperial College London, UK, Department of Computing, and a Research Fellow in stochastic modeling at the University of Melbourne, Australia, School of Mathematics and Statistics. His interests center around the performance analysis of computer systems, especially on cloud and cluster computing and optical networking.

Robert Birke received his Ph.D. in 2009 from the Politecnico di Torino, Italy. He is with IBM Research Zurich in Cloud & Computing Infrastructure department. His research interests center on virtual resource management for large scale data centers, aiming to optimize the application latency, system throughput, and resource efficiency, particularly energy. He is a IEEE member.

Lydia Y. Chen is a research staff member at the IBM Zurich Research Lab, Zurich, Switzerland. She received a Ph.D. from the Pennsylvania State University. Her research interests include performance evaluation for datacenters and big data systems. She has served on several technical program committees in various performance and network conferences, including DSN, INFOCOM, ICDCS, ICAC, and Middleware. She is a IEEE senior member.

Peter G. Harrison is Professor of Mathematical Modelling in the Department of Computing at Imperial College London. He obtained his Ph.D.in Computing Science at Imperial College in 1979. He has researched into stochastic performance modelling and algebraic program transformation for some thirty five years, visiting IBM Research Centers during two summers. He has written two books, had over 200 research papers published and held a series of research grants, both national and international. Currently, his main research interests are in stochastic modelling, where he has developed the RCAT methodology for finding separable solutions, Hidden Markov Models, response time analysis and modulated fluid models, together with applications such as storage systems, resource virtualization and energy-saving.