Cutting Latency Tails via Replication: Analysis and Experimental Validation

Journal:	*Transactions on Parallel and Distributed Systems*
Manuscript ID	Draft
Manuscript Type:	Regular
Keywords:	C.4.d Modeling techniques < C.4 Performance of Systems < C Computer Systems Organization, C.4.e Performance attributes < C.4 Performance of Systems < C Computer Systems Organization, D.2.19.c Methods for SQA and V&V < D.2.19 Software Quality/SQA < D.2 Software Engineering < D Software/Software Engineering

SCHOLARONE™
Manuscripts

For Peer Review Only

Cutting Latency Tails via Replication: Analysis and Experimental Validation

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

For Peer Review Only

Abstract—Response time variability in software applications can severely degrade the quality of 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 that completes. Most prior art focuses on the *mean* latency for systems implementing *replica cancellation*, i.e., all replicas of a request are canceled once the first one finishes. Instead, in this paper we develop models to obtain the response-time *distribution* for "fast" systems, such as web services, where replica cancellation is too expensive to be implemented. 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 (MediaWiki), the requests of which execute on the order of seconds (milliseconds). The proposed analysis enables us to derive insights into optimal replication levels 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 novel 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]. With concurrent replication, a number of replicas of each request are spawned simultaneously, and the result of the first replica that completes 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 performance, 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 on the order of milliseconds. Due to the short processing times, in these systems it is not feasible to cancel all replicas of a request upon the completion of the first one, which is a mechanism commonly used to limit the additional load introduced by replication. In fact, most existing modeling works [4]–[9] consider the adoption of canceling. This assumption has the additional benefit of simplifying the analysis thanks to the synchronization introduced by the canceling mechanism (all replicas of a request finish service at the same time). Instead, we are interested in evaluating the impact of *replication without canceling*, where such synchronization does not exist. Thus, in this paper we derive a set of stochastic models that can accurately predict the request response times when *replication without canceling* is adopted.

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 feature enables us to evaluate the impact of replication on different response-time percentiles, which, as we will demonstrate, is far from homogeneous. Furthermore, relying on this model we will show that the threshold load, i.e., the maximum load under which replication is

- *Z. Qiu and P. G. Harrison are with the Department of Computing, Imperial College London, UK. E-mail: {z.qiu11,p.harrison}@imperial.ac.uk*
- *J. F. Pérez is with the School of Mathematics and Statistics, University of Melbourne, Australia. E-mail:juan.perez@unimelb.edu.au*
- *R. Birke and L. Chen are with IBM Research Zurich, Switzerland. E-mail:{bir,yic}@zurich.ibm.com*

beneficial, can differ if the evaluation is based on the response-time mean or on a specific percentile.

In developing the model we have emphasized its ability to capture a wide range of real system scenarios. In particular, we are able to model highly varying processing times by considering 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 identified that the processing times of replicas of the same request can be **correlated**. To capture this behavior, we introduce the novel concept of correlated hyper-Erlang (CHE) distributions. We extend our model to incorporate CHE distributions when determining the response-time distribution under replication. Moreover, we design an efficient fitting method based on the Expectation-Maximization algorithm to parameterize the CHE service model from real data.

We demonstrate the models' ability to predict the tail response times through extensive experimentation on a MATLAB benchmark [10] and on a standard three-tier web application, namely, MediaWiki [11]. 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 incorporate correlated service times to accurately determine the response time distribution. Leveraging the proposed analysis, we derive insights into designing replication policies, i.e., the optimal number of replicas and the threshold load – the maximum load under which replication is beneficial. Experimentation with the MediaWiki application corroborates the capability of the proposed model to guide replication decisions in real-life applications.

1.2 Related work

Request replication has been considered in [3]–[9], [12]–[15] as an efficient way to combat the variability in latency. Joshi et. al. [9] 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. [12] 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 [4]–[6] 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]–[9] 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 because of the nonnegligible

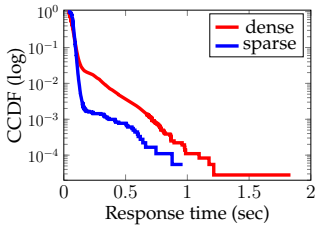


Fig. 1: Dense vs. sparse setup: response-time CCDF

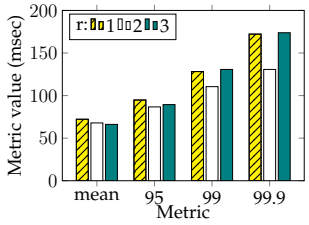


Fig. 2: Response times under replication.

time required to cancel the replicas. 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, whose requests usually take milliseconds to respond. 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 step that simplifies the system dynamics. Thus a more detailed analysis is required to capture the asynchronous behavior caused by the lack of canceling.

Also, the correlation among replicas of a request is commonly ignored [4]–[6], [8], [12], with the exception of [9], although this does not necessarily hold for many applications where the request processing time depends at least partially on the intrinsic details of the request and is therefore reflected on all its replicas. In contrast, this paper considers both cases of independent and correlated processing times among replicas of the same request, and we allow these processing times to follow fairly general (PH) distributions. In addition, introducing MAP arrivals enables us to capture bursty arrivals.

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 [11], a standard three-tier web application deployed on a private cloud. Details of the application and its deployment can be found in Section 8. 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 complementary cumulative distribution (CCDF) of request response times for two system set-ups: dense, where the six MediaWiki VMs are co-located on three physical servers; and sparse, where each MediaWiki VM is hosted on a separate physical machine. In the densely co-located cloud, the tail can be several times larger

For Peer Review Only

than the average response time (73.3 msec). For example, the 95^{th} and 99^{th} percentiles are 117.2 and 321.7 msec, respectively, while the 99.9^{th} percentile is 713.0 msec, which is $9.73\times$ larger than the average. In the sparse case, the mean is similar to that of the co-located case, but the tail is much shorter, with the 99.9^{th} percentile being only 385.0 msec, which is $5.05\times$ 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 10 requests per second. Clearly, replication is effective in reducing the latency tail, with reductions close to 20% 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 a set of replicas of the same request are independent. Using measurements from the MediaWiki application with an arrival rate of 10 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

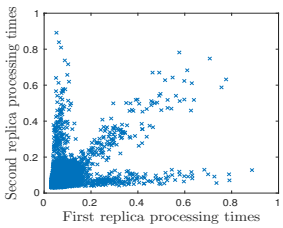


Fig. 3: Distribution of processing times of two replicas from the same request.

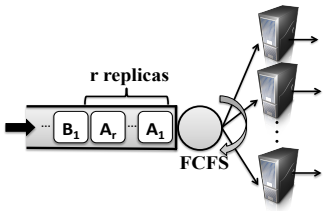


Fig. 4: Reference model.

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 [16]. 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.

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

For Peer Review Only

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 $R_{\text{all-busy}}$, and solve (13)
c. Use (17) to 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 $(\boldsymbol{s}_{\text{res}}, S_{\text{res}})$

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 [17] for the standard multi-server case, which we extend to consider requests replicated $r \geq 1$ times. Our analysis also extends the methods in [4]–[6], 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 [4].

Instead, 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. For instance, 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 therefore 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 subsequent sections, we summarize the key steps of computing the PH representation of 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 [17], 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 and its age (time-in-system) is equal to its waiting time. 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 [17] 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 [17]

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

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

For Peer Review Only

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 [18]. 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.

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 $N^{\mathrm{all}}_{\mathrm{ser}} = \{(n_1,\ldots,n_{m_R}) | 0 \le n_i \le r, \sum_{i=1}^{m_R} n_i = r\}$ covers the states where all tagged replicas are in service, and the remaining states are in $N^{\mathrm{part}}_{\mathrm{ser}} = \{(o_1,\ldots,o_{m_R}),(n_1,\ldots,n_{m_R}) | 0 \le o_i < C, 0 \le n_i < r, 1 \le \sum_{i=1}^{m_R} n_i < r, \sum_{i=1}^{m_R}(o_i+n_i) = C\}$. 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 first two rows consider tagged transitions, with and without a service completion, when all tagged replicas are in service. 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.

TABLE 2: Transition rates for S_{ser}

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

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 by Eq. (18) in Appendix A. Considering the different conditions in which a job starts service we obtain the following results.

Lemma 1 (Busy start): The initial service phase for jobs that **must wait before starting service** is given by

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

where

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

$\boldsymbol{\alpha}_{\mathrm{busy}} = -\boldsymbol{\pi}(0)T^{-1}$ 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

For Peer Review Only

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 $c_1 \boldsymbol{\alpha}_{\text{busy}} L(M \otimes D_1)$, where L is given by Eq. (4). Also, 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 Eq. (15) in Appendix A and can thus be found by solving an associated Sylvester matrix equation.

Lemma 2 (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 [17, 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 multi-nomial probability given in Eq. (12), 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. □

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}	$\boldsymbol{n}$	$(\boldsymbol{n}, \boldsymbol{n}^w)$	$p(\boldsymbol{n}^w)$	$\|\|\boldsymbol{n}\|\|+r \geq C, w=C-\|\|\boldsymbol{n}\|\|$

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 probability $p(\boldsymbol{n}^w)$ given by Eq. (12).

Lemma 3 (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, P, and $Q_{\text{not-all-busy}}$ are given by Eqs. (7), (15) and (14), respectively, and c_2 is a normalizing constant.

Proof: From the proof of Lemma 2, 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 $Q_{\text{not-all-busy}}$ in Eq. (14), 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 Eq. (15), to re-write this expression as $c_2 \boldsymbol{\pi}(0) P Q^{-1}_{\text{not-all-busy}} (R_{\text{ser}} \otimes D_1)$, 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). □

For Peer Review Only

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

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

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 can therefore extend the numerical methods in [19] to efficiently compute the matrix T, the vector $\boldsymbol{\pi}(0)$, as well as moments and percentiles of the response-time distribution. Appendix D highlights 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 multi-processor 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 [10], where each request emulates a game of blackjack that consists of a predefined number of blackjack hands. This enables us to 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)[1] 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 [20], [21] 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.

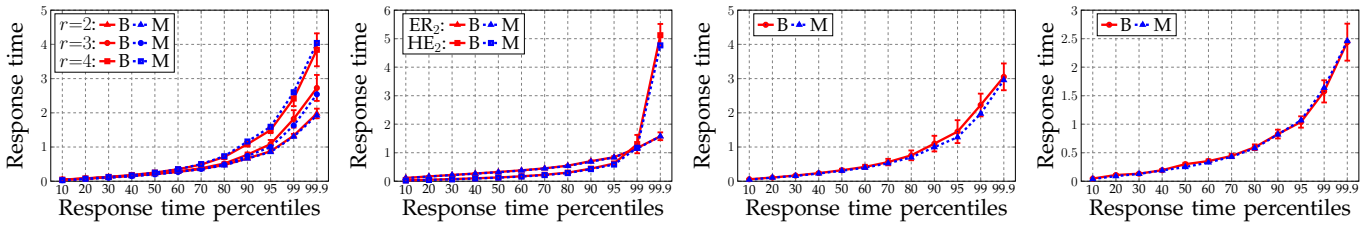
4.2 Performance Prediction

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 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, where we observe that our model predicts the entire response-time distribution well,

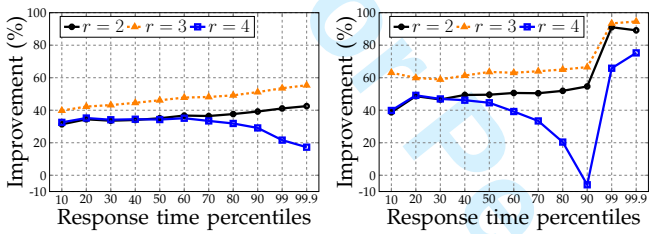
1. SCV $= Var[Y]/E[Y]^2$ for a random variable Y, where $Var[Y]$ and $E[Y]$ are the variance and expected value of Y, respectively.

For Peer Review Only



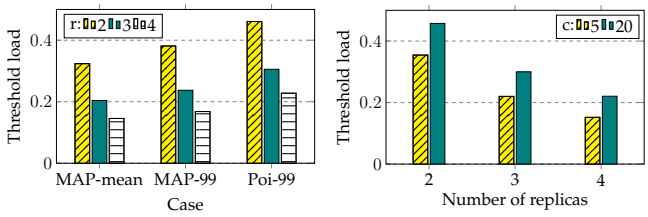
(a) (Poisson, Exp, 4) (b) (Poisson, ER_2/HE_2, 4) (c) (MAP, Exp, 4) (d) (Poisson, Exp, 20)

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.



(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.

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, ..., 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, as the improvement is -5.86%. 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

For Peer Review Only

section search method [22]. 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.

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 [23], 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 [24] 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 B.

7 Fitting Correlated Processing Times

The CHE service-time model introduced in Section 6 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 [25] and some of its sub-classes [24], [26], no such method exist for the CHE service model since this is the first time this model is proposed. In this section, we propose a maximum-likelihood method for this purpose, based on the Expectation Maximization (EM) algorithm proposed by Dempster et al. [27]. 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 [20], 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 $(B, h_1, \ldots, h_B, \alpha_1, \ldots, \alpha_B, \lambda_1, \ldots, \lambda_B)$.

For Peer Review

However, and similar to [24], 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 C. 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

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

As evidenced in this last expression, derived in Appendix C, 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 [24] for the independent hyper-Erlang case. For each possible combination, it then executes the EM steps derived in Appendix C, starting from an initial guess for the parameters and iterating until the difference between 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 8, we illustrate how this method is able to capture the correlation structure that emerges in the processing times of request replicas from a real system.

Algorithm 2 EM algorithm for the CHE service model

Require: Data: $\mathcal{D}$, Number of phases: m_R, ϵ
$\mathcal{S} = \{(B, h_1, \ldots, h_B) | \sum_{b=1}^{B} h_i = m_R\}$
bestLH $= \infty$
for $(B, h_1, \ldots, h_B) \in \mathcal{S}$ **do**
  Initial guess $\hat{\Theta} = (\hat{\alpha}_1, \ldots, \hat{\alpha}_B, \hat{\lambda}_1, \ldots, \hat{\lambda}_B)$
  diff $= 1$
  while diff $> \epsilon$ **do**
    Compute $\delta(b|x_n, \hat{\Theta})$, $b = 1, \ldots, B$, $d = 1, \ldots, D$ as in (21), (19)
    Compute α_b, λ_b, $b = 1, \ldots, B$ as in (11)
    diff $= \max |\hat{\Theta} - (\alpha_1, \ldots, \alpha_B, \lambda_1, \ldots, \lambda_B)|$
    $\hat{\Theta} = (\alpha_1, \ldots, \alpha_B, \lambda_1, \ldots, \lambda_B)$
  end while
  Compute $\log L(\hat{\Theta}|\mathcal{D})$ as in (20)
  if $\log L(\hat{\Theta}|\mathcal{D}) >$ bestLH **then**
    $\Theta^* = \hat{\Theta}$
    bestLH $= \log L(\hat{\Theta}|\mathcal{D})$
  end if
end for
return Θ^*

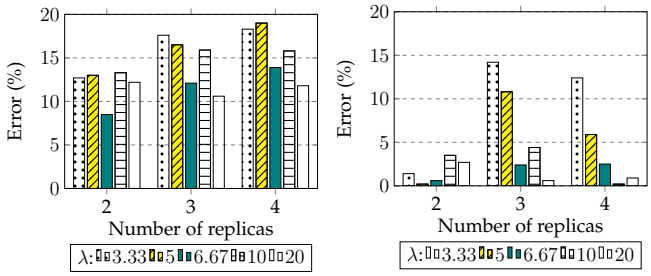
8 Evaluation on MediaWiki

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.

8.1 Set-up

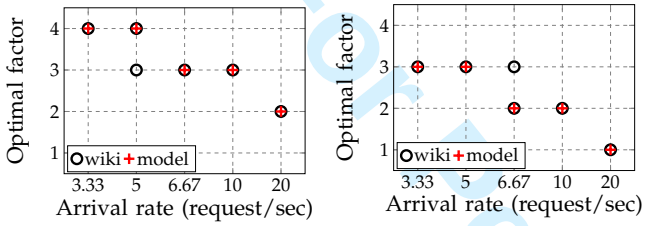
Our private cloud testbed is composed of eight identical physical servers, seven used to run the experiments and one used as experiment orchestrator and repository. Each server is equipped with 32 cores, 128 GB DDR4 RAM, six 1-TB solid state disks in RAID5, and two 10-Gigabit Ethernet adapters. We use MediaWiki, the open source platform used to run the Wikipedia website, as a representative application in the cloud [11]. 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 [28], an open-loop workload generator. Due to the space limit, we only present the results for the dense set-up in the following.

For Peer Review Only



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

Fig. 8: Experimental validation with the MediaWiki.



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

Fig. 9: Optimal replication factor: measurement vs. model.

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. We note that this is a degenerated set-up, whereas the more general set-up immediately forwards the requests to VMs, each of which can hold multiple requests locally.

To emulate performance variability in the cloud, particularly 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 [29]. 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.

8.2 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 8 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.

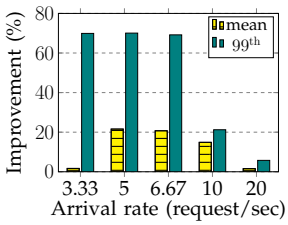


Fig. 10: Improvement using optimal factor r^*.

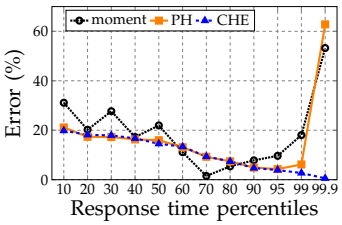
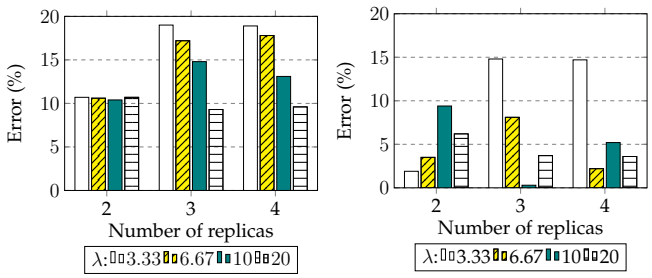


Fig. 11: Prediction errors using different fitting methods.

A key requirement for a system that implements replication is to determine the optimal replication factor r^*. Figure 9(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 9(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 10 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.

For Peer Review Only



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

Fig. 12: Prediction errors using $\lambda = 5$.

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 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 [20], an EM method for independent PH distributions [24], and the EM method for CHE distributions introduced in Section 7. We consider the case with arrival rate 6.67 and replication factor r=2. Figure 11 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 12 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.

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.

9 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. While we have devised efficient methods to obtain the response-time metrics, future work will look into alternative approaches to tackle large-scale systems that adopt replication.

Acknowledgments

The research of Juan F. Pérez is supported by the ARC Centre of Excellence for Mathematical and Statistical

For Peer Review Only

Frontiers (ACEMS).

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

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

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

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

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

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

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

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

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

[13] 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.

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

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

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

[17] 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.

[18] 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.

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

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

[21] 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.

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

[23] 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.

[24] 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.

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

[26] 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.

[27] 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.

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

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

[30] B. Sengupta, "Markov processes whose steady state distribution is matrix-exponential with an application to the GI/PH/1 queue," *AAP*, vol. 21, pp. 159–180, 1989.

[31] G. Golub, S. Nash, and C. Van Loan, "A Hessenberg-Schur method for the problem AX+XB=C," *IEEE TACON*, vol. 24, pp. 909–913, 1979.

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 a Research Fellow at the University of Melbourne, School of Mathematics and Statistics. He obtained a PhD in Computer Science from the University of Antwerp, Belgium, in 2010, and was a Research Associate in performance analysis at Imperial College London, Department of Computing. His research 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. in Operations Research and Industrial Engineering from the Pennsylvania State University in Dec 2006. Her research interests include performance evaluation for datacenters and streaming 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 co-recipient of the best paper awards at CCgrid'15 and eEnergy'15. She is a IEEE senior member.

For Peer Review Only

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.

For Peer Review Only

TABLE 4: Transition rates for $S_{\text{(all)-(not-all)}}$ and $S_{\text{not-all-busy}}$

Matrix	From	To	Rate
$S_{\text{(all)-(not-all)}}$	$(\boldsymbol{n}, 0)$	$\boldsymbol{n} - \boldsymbol{e}_i$	$n_i S_{\text{R}}^*(i)$
$S_{\text{not-all-busy}}$	$\boldsymbol{n}$	$\boldsymbol{n} - \boldsymbol{e}_i + \boldsymbol{e}_j$	$n_i S_{\text{R}}(i,j)$
	$\boldsymbol{n}$	$\boldsymbol{n} - \boldsymbol{e}_i$	$n_i S_{\text{R}}^*(i)$

APPENDIX A
INDEPENDENT PROCESSING TIMES

A.1 Computing $\boldsymbol{\pi}(0)$

To find the vector $\boldsymbol{\pi}(0)$ in Eq. (1), which is the stationary distribution of the phase at the beginning of an all-busy period, we construct a Markov chain by observing the system state every time a new all-busy period starts. To this end we need to connect the not-all-busy with the all-busy periods and follow the evolution of the system during the not-all-busy periods. We thus define the process $D_{\text{not-all-busy}}(t) = (n_1(t), \ldots, n_{m_R}(t))$, which keeps track of the number of replicas in service in each phase during the not-all-busy period and takes values in the set $N_{\text{not-all-busy}} = \{(n_1, \ldots, n_{m_R}) | n_i \in \{0, \ldots, C-1\}, \sum_{i=1}^{m_R} n_i \leq C-1\}$, which is of size $m_{\text{not-all-busy}}$.

To connect the all-busy period with the not-all-busy period, we introduce the matrix $S_{\text{(all)-(not-all)}}$, which holds the service transition rates (between arrivals) that trigger the initiation of a not-all-busy period. This matrix thus connects the set $N_{\text{all-busy}}$ with the set $N_{\text{not-all-busy}}$, has size $m_{\text{all-busy}} \times m_{\text{not-all-busy}}$, and transition rates as described in the first row of Table 4. Note that a transition can initiate a not-all-busy period from a state with zero replicas of the youngest job waiting in the queue, such that a service completion of any replica in service phase i, before an arrival, terminates the all-busy period. Next, we focus on the *not-all-busy period* and introduce the matrix $S_{\text{not-all-busy}}$, which holds the service transition rates between arrivals during the not-all-busy period, as summarized in Table 4. Here the second row considers phase transitions without service completions, whereas the third row covers service completions. Also, we adjust the diagonal entries of $S_{\text{not-all-busy}}$ to make it a Markov chain generator.

To track service phase changes due to arrivals during the not-all-busy period, we let $R_{\text{not-all-busy}}$ be an $m_{\text{not-all-busy}} \times m_{\text{not-all-busy}}$ matrix, the $(i,j)^{\text{th}}$ entry of which holds the probability that the service phase just after an arrival is j given that it was i just before. To define this matrix, we introduce the quantity $p(\boldsymbol{n}^w)$, which is the probability that a job starts with w replicas and in phase $\boldsymbol{n}^w = (n_1^w, \ldots, n_{m_R}^w)$, where n_i^w is the number of replicas that start in phase i, and such that $\sum_{i=1}^{m_R} n_i^w = w$. As each replica selects its initial phase independently, this probability follows a multi-nomial distribution, thus

$$p(\boldsymbol{n}^w) = w! \prod_{i=1}^{m_R} \frac{\boldsymbol{s}_{\text{R}}(i)^{n_i^w}}{n_i^w!}, \tag{12}$$

where $\boldsymbol{s}_{\text{R}}(i)$ is the probability that a replica start service in phase i. We can therefore define $R_{\text{not-all-busy}}$ as in Table 3, where the arriving job sees more than r idle servers and starts service with all its r replicas, with service phases as described by the vector $\boldsymbol{n}^r$. For clarity we make use of the 1-norm $||\boldsymbol{n}|| = \sum_{i=1}^{m_R} n_i$, which is the total number of replicas in service. This table also defines a similar matrix $R_{\text{all-busy}}$, of size $m_{\text{not-all-busy}} \times m_{\text{all-busy}}$, which covers the case in which an arrival sees at most r idle servers, triggering the start of an all-busy period. As shown in Table 3, in this case the arriving job becomes the youngest in service and starts with $w \leq r$ replicas in service, leaving the remaining $r-w$ waiting in the queue.

We can now find $\boldsymbol{\pi}(0)$ as the solution to the equation

$$\boldsymbol{\pi}(0) = \boldsymbol{\pi}(0) \int_0^\infty \exp(Tu) S^*(u) Q^{-1}_{\text{not-all-busy}} (R_{\text{all-busy}} \otimes D_1) du. \tag{13}$$

This equation considers that the age during the all-busy period is u and that a transition triggers the beginning of the not-all-busy period. This occurs according to matrix $S^*(u) = S_{\text{(all)-(not-all)}} \otimes \exp(D_0 u)$, which captures that the next arrival occurs after u time units, such that a service completion triggers the end of the all-busy period. Once the system is in the not-all-busy period, it evolves according to

$$\begin{aligned} Q_{\text{not-all-busy}} = {} & I_{m_{\text{not-all-busy}}} \otimes D_0 + S_{\text{not-all-busy}} \otimes I_{m_a} \\ & + R_{\text{not-all-busy}} \otimes D_1, \end{aligned} \tag{14}$$

which considers transitions of the arrival process without arrivals, service completions, and arrivals that do not trigger the start of an all-busy-period. Eventually, an arrival causes the process to be absorbed into an all-busy period according to $R_{\text{all-busy}} \otimes D_1$.

To compute the integral in Eq. (13), we define

$$P = \int_0^\infty \exp(Tu) S^*(u) du, \tag{15}$$

and integrate P by parts to obtain

$$TP + P(I_{m_{\text{not-all-busy}}} \otimes D_0) = -S_{\text{(all)-(not-all)}} \otimes I_{m_a}.$$

Because T is known, this is a Sylvester matrix equation that can be solved in $O(m_a^3 m_{\text{all-busy}}^2 m_{\text{not-all-busy}})$ time to find P. As a result, after finding P, determining $\boldsymbol{\pi}(0)$ reduces to solving a linear system with matrix $PQ^{-1}_{\text{not-all-busy}}(R_{\text{all-busy}} \otimes D_1)$, which can be done in $O(m_a^3(m_{\text{all-busy}}^3 + m_{\text{not-all-busy}}^3 + m_{\text{not-all-busy}}^2 m_{\text{all-busy}} + m_{\text{all-busy}}^2 m_{\text{not-all-busy}}))$ time.

Finally, we recall [30, Theorem 2.3], from which the process $\{X(t), J(t) | t \geq 0\}$ is positive recurrent if

$$\varrho = \boldsymbol{\rho} \int_0^\infty u A^{(\text{jump})}_{\text{MAP}}(u) du \mathbf{1} > 1, \tag{16}$$

where $\boldsymbol{\rho}$ is the invariant probability vector of the matrix $S_{\text{MAP}} + (A^{(\text{jump})} \otimes D_0^{-1} D_1)$. This condition guarantees [30] the existence of $\boldsymbol{\pi}(x)$ and its matrix-exponential form.

For Peer Review Only

TABLE 5: Transition rates for S and $A^{(\text{jump})}$ - CHE services

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

TABLE 6: Transition rates for $S_{\text{(all)-(not-all)}}$ - CHE services

From	To	Rate
$(\boldsymbol{n}, 0, b)$	$(\boldsymbol{n}-\boldsymbol{e}_i)$	$n_i S^*_{\text{R}}(i)$

A.2 The PH representation

Having found $\boldsymbol{\pi}(0)$ and T, and relying on [17], we can obtain the PH representation of the waiting-time distribution, as this coincides with the distribution of the age $X(t)$ just after a downward jump, because that is the age of the job starting service, and thus its waiting time. Let the steady-state distribution of the phase $J(t)$ during the all-busy period be $\boldsymbol{\alpha}_{\text{busy}}=-\boldsymbol{\pi}(0)T^{-1}$, and define $\boldsymbol{\varphi}=(T-S_{\text{MAP}})\mathbf{1}$. The PH representation of the waiting-time distribution is thus [17]

$$\boldsymbol{s}_{\text{wait}} = \gamma\boldsymbol{\alpha}_{\text{busy}} \circ \boldsymbol{\varphi}/((\boldsymbol{\alpha}_{\text{busy}} \circ \boldsymbol{\varphi})\mathbf{1}), \quad S_{\text{wait}} = \Lambda^{-1}T'\Lambda, \quad (17)$$

where $\circ$ stands for the Hadamard product, $'$ denotes the matrix transpose, Λ is a diagonal matrix such that $\Lambda\mathbf{1} = \boldsymbol{\alpha}'_{\text{busy}}$, and γ is the probability that a job has to wait.

To obtain γ, we first define η_0 to be the number of service completions in an all-busy period and η_1 the number of arrivals in a not-all-busy period. Their expected values $E[\eta_0]$ and $E[\eta_1]$ can be obtained following a similar approach as in [17, Section 7.2]. Thus the probability that a job has to wait is

$$\gamma = (E[\eta_0]-1)/(E[\eta_0]-1+E[\eta_1]) \quad (18)$$

because, in a cycle consisting of one all-busy and one not-all-busy period, $E[\eta_0]-1$ is the expected number of jobs that have to wait and $E[\eta_0]-1+E[\eta_1]$ is the expected number of job arrivals.

Appendix B
Correlated Processing Times

In this appendix we extend the model of Section 3 to incorporate the CHE processing times introduced in Section 6.

B.1 The waiting-time distribution

As the replicas of a single job follow a CHE distribution, we let the total number of phases in the PH representation be $m_R = \sum_{i=1}^{B} h_i$. Next, we redefine the service process during the all-busy periods $D_{\text{all-busy}}(t)$ as $(n_1(t), \ldots, n_{m_R}(t), w(t), b(t))$, where $n_i(t)$ is the number of replicas in service phase i, for $1 \leq i \leq m_R$, $w(t)$ is the number of replicas of the youngest job waiting in the queue, and $b(t)$ is the service branch selected by the youngest job in service at time t. This process takes values in the set $N_{\text{all-busy}}=\{(n_1, \ldots, n_{m_R}, w, b) | n_i \in \{0, \ldots, C\}, \sum_{i=1}^{m_R} n_i=C, 0 \leq w < r, b \in \mathcal{B}\}$.

To describe the evolution of this process, we re-define the matrices S and $A^{(\text{jump})}$ as summarized in Table 5. The key difference here is that whenever a new *job* starts service, with rates recorded in matrix $A^{(\text{jump})}$, it chooses a branch j with probability $\boldsymbol{s}_{\text{R}}(j)$, and this choice is recorded in the last entry of the state descriptor. Thus, when other replicas of the same job start service, with rates in matrix S, they choose the same branch. With these matrices, we can solve Eq. (2) to determine the matrix T of the age process.

To determine $\boldsymbol{\pi}(0)$, we again need to consider the evolution of the system during the not-all-busy period. However, during the not-all-busy periods there is no need to keep track of the branch selected by the youngest job in service as this information is only needed when this job has replicas waiting in the queue. We can therefore rely on the service process $D_{\text{not-all-busy}}(t)$ as defined in Appendix A and on the $S_{\text{not-all-busy}}$ matrix in Table 4 to describe the phase process evolution due to service completions. Also, a transition that initiates a not-all-busy period ignores the branch selected by the youngest job in service, as captured by the matrix $S_{\text{(all)-(not-all)}}$ in Table 6, as this information becomes irrelevant once all replicas of the youngest job have started service. In the case of arrivals, the matrix $R_{\text{not-all-busy}}$, shown in Table 7, requires a modification to ensure that all replicas of a new job start service in the same phase i with probability $\boldsymbol{s}_{\text{R}}(i)$. When a job arrives during a not-all-busy period and finds $w \leq r$ idle servers, it initiates an all-busy period, with all its first w replicas starting service in phase i with probability $\boldsymbol{s}_{\text{R}}(i)$, as recorded in matrix $R_{\text{all-busy}}$ in Table 7. Here we also keep the selected branch in the last entry of the state descriptor so that the remaining $r-w$ replicas of the new job select the same branch when they start service. With these matrices, we can proceed as in Appendix A to obtain the waiting-time distribution.

B.2 The service-time distribution

The service-time process requires a similar adaptation to keep track of the service branch selected by the tagged job when some of its replicas are waiting. We thus define the service process $D_{\text{ser}}(t)$ as $(n_1(t), \ldots, n_{m_R}(t))$ when $\sum_{i=1}^{m_R} n_i(t)=r$ (all tagged replicas in service), and as $\{(o_1(t), \ldots, o_{m_R}(t)), (n_1(t), \ldots, n_{m_R}(t), b(t))\}$ when $\sum_{i=1}^{m_R} n_i(t)<r$ (at least one tagged replica in the queue). Note that the only difference with the service process in the independent case is that we keep track of the branch $b(t)$ selected by the tagged job when at least one tagged replica is in the queue. As this process is very similar to that in the fully-independent case, Table 8 summarizes the main differences in S_{ser} with respect to the one in Table 2. In this case, when at least one tagged replica is in the queue and a non-tagged replica completes service, a

For Peer Review Only

TABLE 7: Transition probabilities for $R_{\text{not-all-busy}}$, $R_{\text{all-busy}}$, and R_{ser} - CHE services

Matrix	From	To	Probability	Condition
$R_{\text{not-all-busy}}$	$\boldsymbol{n}$	$\boldsymbol{n}+r\boldsymbol{e}_i$	$\boldsymbol{s}_{\text{R}}(i)$	$\|\|\boldsymbol{n}\|\|+r<C$
$R_{\text{all-busy}}$	$\boldsymbol{n}$	$(\boldsymbol{n}+w\boldsymbol{e}_i, r-w, i)$	$\boldsymbol{s}_{\text{R}}(i)$	$\|\|\boldsymbol{n}\|\|+r\geq C, w{=}\|\|\boldsymbol{n}\|\|+r-C$
R_{ser}	$\boldsymbol{n}$	$(\boldsymbol{n}, w\boldsymbol{e}_i, i)$	$\boldsymbol{s}_{\text{R}}(i)$	$\|\|\boldsymbol{n}\|\|+r\geq C, w{=}\|\|\boldsymbol{n}\|\|+r-C$

TABLE 8: Transition rates for S_{ser} - CHE services. Main differences to Table 2

From	To	Rate	Condition
$(\boldsymbol{o},\boldsymbol{n},b)$	$(\boldsymbol{o}-\boldsymbol{e}_i,\boldsymbol{n}{+}\boldsymbol{e}_b,b)$	$o_i S^*_{\text{R}}(i)$	$\|\|\boldsymbol{n}\|\|\leq r-2$

tagged replica starts service in the same phase b selected by all its siblings.

As in Section 3.2, to determine the stationary probability vector $\boldsymbol{s}_{\text{ser}}$ according to which a job starts service in each service phase, we consider three different scenarios and obtain this vector as in Eq. (9). The key differences are:

Busy start In this case, a job starts service with its first replica, which selects its initial service phase according to $\boldsymbol{s}_{\text{R}}$, just as in the fully independent case, thus a job starts service according to Eq. (3).

Full start When a job finds more than r idle servers, it starts service with all its replicas without initiating an all-busy period. The main difference here is that the replicas select the same phase. Thus, a job selects phase $r\boldsymbol{e}_i$ with probability $\boldsymbol{s}_{\text{R}}(i)$. We therefore define a vector $\hat{\boldsymbol{s}}_{\boldsymbol{r}}$ of size $|N^{\text{all}}_{\text{ser}}|$, with entries $\hat{\boldsymbol{s}}_{\boldsymbol{r}}(\boldsymbol{n}^r)=\boldsymbol{s}_{\text{R}}(i)$, when $\boldsymbol{n}^r=r\boldsymbol{e}_i$, and zero otherwise. We then find $\boldsymbol{s}^{\text{all}}_{\text{not-busy}}$, the initial probability distribution of the service phase in this case, by replacing $\boldsymbol{s}_{\boldsymbol{r}}$ with $\hat{\boldsymbol{s}}_{\boldsymbol{r}}$ in Eq. (6).

Partial start Finally, if a job finds $1\leq w\leq r$ servers idle, it starts service with its first w replicas, and initiates an all-busy period. The difference with the fully-independent case is that all w replicas select the same phase. We can thus find the distribution of the initial service phase $\boldsymbol{s}^{\text{part}}_{\text{not-busy}}$ as in Eq. (8), but adjust the matrix R_{ser}. Table 7 shows this modification, where all w replicas start service in phase i with probability $\boldsymbol{s}_{\text{R}}(i)$, and the selected phase is recorded in the state descriptor to be used when the remaining replicas of the same job start service.

APPENDIX C
EM ESTIMATION FOR CHE TIMES

The key to the EM algorithm is to consider the dataset $\mathcal{D}$ as incomplete data, where the branch z_d selected in sample d is missing. If this information, summarized in the vector $\boldsymbol{z}$ of size D, was known we could write the log-likelihood of the parameter set Θ as

$$\log L(\Theta|\mathcal{D},\boldsymbol{z})=\sum_{d=1}^{D}\log\left(\alpha_{z_d}p_{z_d}(\boldsymbol{x}_d|\lambda_{z_d})\right),$$

where $p_b(\boldsymbol{x}_d|\lambda_b)$ is the likelihood of observing the response times $\boldsymbol{x}_d$ when branch b is selected and has rate λ_b. Given the CHE service model, this quantity is

$$p_b(\boldsymbol{x}_d|\lambda_b)=\frac{\left(\lambda_b^r\prod_{j=1}^{r}x_d^j\right)^{h_b-1}}{((h_b-1)!)^r}\lambda_b^r\exp\left\{-\lambda_b\sum_{j=1}^{r}x_d^j\right\},\quad(19)$$

because, given that branch b has been selected for sample d, the processing times of the r replicas follow independent Erlang distributions with parameters (h_b,λ_b). Note that the log-likelihood without considering $\boldsymbol{z}$ is given by

$$\log L(\Theta|\mathcal{D})=\sum_{d=1}^{D}\log\left(\sum_{b=1}^{B}\alpha_b p_b(\boldsymbol{x}_d|\lambda_b)\right),\quad(20)$$

which considers all possible branches that could be selected in each sample.

As the vector $\boldsymbol{z}$ is in fact a realization of a random variable Z, the next step is to assume that a guess $\hat{\Theta}$ of the parameters is known. With this, we can determine the expected value of the log-likelihood with respect to the random variable Z given the observations $\mathcal{D}$ and the guessed parameters $\hat{\Theta}$, which we write as $Q(\Theta,\hat{\Theta}){=}E_Z\left[\log L(\Theta|\mathcal{D},\boldsymbol{z})|\mathcal{D},\hat{\Theta}\right]$. Since Z lives in the space $\zeta{=}(1,\ldots,B)^D$, we can write this expected value as

$$Q(\Theta,\hat{\Theta})=\sum_{\boldsymbol{z}\in\zeta}\sum_{d=1}^{D}\log\left(\alpha_{z_d}p_{z_d}(\boldsymbol{x}_d|\lambda_{z_d})\right)\prod_{i=1}^{D}\delta(z_i|\boldsymbol{x}_i,\hat{\Theta}),$$

where the first term in the sum is the likelihood of observing the processing time trace $(\boldsymbol{x}_1,\ldots,\boldsymbol{x}_D)$ given that the branches selected were $\boldsymbol{z}$, and the second term holds the probability that branches were selected as in $\boldsymbol{z}$. If we consider each possible branch selected in each sample, we can re-write this expression as

$$Q(\Theta,\hat{\Theta})=\sum_{d=1}^{D}\sum_{b=1}^{B}\log\left(\alpha_b p_b(x_d|\lambda_b)\right)\delta(b|\boldsymbol{x}_d,\hat{\Theta}).$$

A detailed derivation of this expected value for the hyper-Erlang case can be found in [24, Appendix A.2]. In fact, we rely on [24] to find $\delta(b|\boldsymbol{x}_n,\hat{\Theta})$ by means of Bayes' rule as

$$\delta(b|\boldsymbol{x}_d,\hat{\Theta})=\frac{\delta(b|\hat{\Theta})p(\boldsymbol{x}_d|b,\hat{\Theta})}{p(\boldsymbol{x}_d|\hat{\Theta})}=\frac{\hat{\alpha}_b p_b(\boldsymbol{x}_d|\hat{\lambda}_b)}{\sum_{j=1}^{B}\hat{\alpha}_j p_j(\boldsymbol{x}_d|\hat{\lambda}_j)},\quad(21)$$

which gives us the likelihood of choosing branch b for sample d as a ratio that compares it with all branches that

For Peer Review Only

could be selected. Computing these quantities constitutes the expectation step of the EM method.

For the maximization step we find the values of Θ that maximize $Q(\Theta, \hat{\Theta})$. As the CHE service model is a mixture of densities, it has been shown in [27] that we can obtain α_b as in Eq. (11). To obtain λ_b we derivate $Q(\Theta, \hat{\Theta})$ with respect to λ_b,

$$\frac{\partial Q}{\partial \lambda_b} = \sum_{d=1}^{D} \delta(b|\boldsymbol{x}_d, \hat{\Theta}) \frac{\partial}{\partial \lambda_b} \log \left(\frac{\left(\lambda_b^r \prod_{j=1}^{r} x_d^j\right)^{h_b-1}}{((h_b-1)!)^r} \lambda_b^r \exp\left\{-\lambda_b \sum_{j=1}^{r} x_d^j\right\} \right)$$

and equate to zero to obtain Eq. (11).

Appendix D
Efficient Computation

To determine the matrix T we extend the methods in [19]. First, we note that (2) can be written as $T{=}S_{\text{MAP}}{+}L(A^{(\text{jump})}{\otimes}D_1)$, with L as in (4). Also, integrating L by parts we find

$$TL + L(I_{m_s} \otimes D_0) = -I_m. \tag{22}$$

We thus apply the following iterative scheme [30]: (i) start with $T_0{=}S_{\text{MAP}}$; (ii) solve (22), replacing T with T_0, to find L_0; (iii) obtain a new iterate as $T_1{=}S_{\text{MAP}}{+}L_0(A^{(\text{jump})} \otimes D_1)$, and repeat. The key step here is the solution of (22), which is normally performed in $O(m^3)$ time with the standard Hessenberg-Schur method [31]. We propose a more efficient method relying on the following two observations.

(i) The matrix $A^{(\text{jump})}$ is of low rank, thus $(A^{(\text{jump})} \otimes I_{m_a})$ is also of low rank, i.e., $\text{rank}(A^{(\text{jump})} \otimes I_{m_a}){=}f \ll m$. Thus letting $(A^{(\text{jump})} \otimes I_{m_a}) = \Gamma\Phi$ we can re-write (22) as

$$TY + Y(I_{m_s} \otimes D_0) = -\Gamma, \tag{23}$$

with $Y = L\Gamma$, where Y has only f columns.

(ii) If we order the service phase space descendingly by w and then lexicographically by the other entries, from Table 1 we see that the matrix S is upper triangular. From the low-rank observation above we have that $T_{n+1} = S \otimes I_{m_a} + Y_n\Phi(I_{m_s} \otimes D_1)$, thus T_{n+1} is a low-rank correction of the upper-triangular matrix $S \otimes I_{m_a}$.

As a result, in each iteration we solve (23) in lieu of (22), and instead of applying the standard Hessenberg decomposition [31] to T, which takes $O(m^3)$ time, we solve a low-rank system in $O(f^3)$ and find each column of Y_n by solving an upper-triangular system in $O(m^2)$. Each step in the computation of T is thus reduced from $O(m^3)$ to $O(f^3 + fm^2)$. The **key difference** with [19] lies in that the matrix $A^{(\text{jump})}$ in [19] is assumed to have a few nonzero columns, whereas here we consider the more general case where this matrix is of low rank. These observations lead to similar gains when computing $\boldsymbol{\pi}(0)$ and the percentiles of the response-time distribution.

For Peer Review Only