

LINE: Evaluating Software Applications in Unreliable Environments

Juan F. Pérez and Giuliano Casale

Abstract—Cloud computing has opened the way for the flexible deployment of software applications. This flexibility offers service providers a number of options to tailor their deployments to the observed and foreseen demand, without incurring in large capital costs. Cloud deployments however pose novel challenges regarding application reliability and performance. Examples include managing the reliability of deployments that make use of spot instances, or coping with the performance variability caused by multiple tenants in a virtualized environment.

In this paper we introduce *LINE*, a tool for the performance and reliability analysis of software applications. *LINE* relies on Layered Queueing Network (LQN) models, a popular abstraction in software performance engineering. *LINE* extends the capabilities of LQNs to capture the operational environment the application operates in. This enables the modeling of reliability issues such as resource failures, server breakdowns and repairs, slow start-up times, resource interference due to multi-tenancy, among others. This paper describes the *LINE* tool, its performance and reliability models, and illustrates its potential by comparing its results against data obtained from a cloud deployment, and exploiting it for a case study on reliability-aware resource provisioning.

ABBREVIATIONS AND ACRONYMS

CCDF	Complementary CDF
CDF	Cumulative Distribution Function
LQN	Layered Queueing Network
MC	Markov chain
ODE	Ordinary differential equation
PCM	Palladio Component Model
PH	Phase-type
RE	Random environment
RT	Response time
QN	Queueing Network
SCV	Squared coefficient of variation
SLO	Service-level objective

NOTATION

$P(\cdot)$	Probability
$E[\cdot]$	Statistical expected value
$\text{Var}(\cdot)$	Variance

I. INTRODUCTION

Recent years have seen the sustained growth in the adoption of cloud computing, a trend expected to continue in the future

Juan F. Pérez and Giuliano Casale are with the Department of Computing, Imperial College London, London, UK, e-mail:{j.perez-bernal,g.casale}@imperial.ac.uk.

The research of Giuliano Casale and Juan F. Pérez leading to these results has received funding from the European Union under grant agreement FP7-318484 (MODAClouds) and H2020-644869 (DICE), and from the EPSRC grant EP/M009211/1 (OptiMAM). The authors wish to thank Weikun Wang for kindly sharing with us the OFBiz data traces used for validation.

[1]. Advantages of cloud deployments include ample flexibility to manage resources, using only those that are needed to achieve service-level objectives (SLOs), and the ability to reduce the capital costs associated with large computing infrastructures. However, opting for a cloud-based deployment also poses a number of challenges, related to both reliability and performance. As the infrastructure is no longer under the control of the service provider, the virtualization layer may introduce overheads that cannot be directly managed, but that must be considered when provisioning resources. Further, cloud computing offers attractive deployment alternatives, such as spot instances, which can be used at significantly lower prices, but that can be lost whenever the spot price surpasses the bid price.

Under such uncertainty, model-driven software engineering can help in making provisioning decisions. In particular, cloud deployments call for models that explicitly capture the inherent uncertainty in the application processing times, and in the reliability of the deployment environment. A class of models that has gained popularity are Layered Queueing Networks (LQN) [2], which offer a modeling paradigm well-suited for complex software applications, including features such as resource pooling, admission control, and synchronous calls, among others. To fully exploit LQN models, multiple efforts have been put forward to automate their generation from high-level software specifications (e.g., UML MARTE [3], [4], PCM [5], [6], CBML [7]). Further, efficient methods and tools exist to automatically solve LQN models and compute relevant performance measures, such as the approximated mean-value analysis in the LQN solver (LQNS) [8].

Although reliability extensions have been proposed for LQN models [9], these focus on *mean* values of the service metrics, and are limited to scenarios where failures and repairs occur at a much slower time scale than the application processing and response times. This therefore limits the use of LQN models to tackle reliability-aware resource provisioning problems as those envisioned in a cloud environment, where, for instance, the virtualized environment can affect the application processing rates in very short time scales [10]. In addition, while existing methods and tools are very effective to estimate *mean* performance metrics, the lack of support for the analytic computation of response time *percentiles* has been pointed out as a limitation of LQN models [11]. Moreover, cloud applications typically serve different user classes, and support a large number of request types. Metrics such as response time means and percentiles can therefore be needed at the user-type or request-type level to effectively evaluate SLOs. Although response time percentiles can be estimated by means of simulation, this approach is very intensive in computational

resources, limiting its application when exploring a large parameter space. In addition, an application model can also serve as part of a decision-support system to determine, for instance, the optimal deployment given a set of operational and service constraints. Simulation models are not well-suited for this task because of their heavy computational requirements, which are particularly taxing when solving the non-linear integer optimization problems associated with resource provisioning in unreliable environments under service constraints.

In this paper we introduce LINE, a software tool to evaluate LQN models that tackles these limitations. The main features of LINE are:

- The LINE solver is based on a *fluid* queueing model, which can efficiently analyze large-scale models.
- LINE explicitly models reliability issues by means of a *random environment*, which is a stochastic model of the operational environment the software system evolves in. This provides the ability to model the reliability and performance of complex software systems in presence of resource failures, server breakdowns and repairs, start-up delays, and interference due to multi-tenancy, an increasingly important issue for cloud applications.
- In addition to mean performance metrics, LINE also estimates the response-time *distribution* of the application requests, which can be determined at the user-type or the request-type level. This enables the evaluation of SLOs expressed as response time percentiles.
- LINE can be directly used with the Palladio Bench tool to analyze application models based on the Palladio Component Model (PCM) paradigm.

An early version of LINE was introduced in [12]. In comparison, the contributions of this paper are:

- We have integrated and generalized the approach in [10] for reliability modeling by means of random environments. The random environments in LINE support Phase-type (PH) holding times, compared to the exponential assumption in [10]. Further, the performance model in [10] was a single-class queueing network with first-come first-served processors. The LINE performance model instead supports processor-sharing servers and requests belonging to multiple classes, and different user types, as it is based on a multi-class and multi-chain queueing network.
- We generalize the performance model in [12] to consider PH processing times. Within this more general framework we provide a formal proof of the convergence of the fluid model in the sense of Kurtz [13].
- We validate the results of LINE against the response times observed from an e-commerce application deployed on the cloud, as well as against simulation.
- We illustrate the potential of LINE by incorporating it into a reliability-aware resource provisioning strategy. The reliability model of LINE is used to represent a deployment based on spot instances, which can be lost and recovered dynamically, affecting both the application availability and performance. The provisioning problem is posed as an integer optimization problem, which is solved with simulated annealing, and where LINE evaluates the

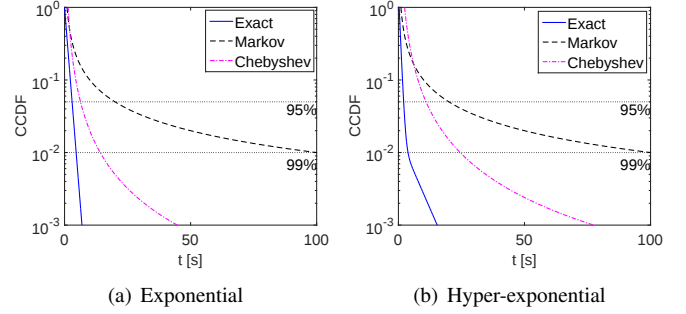


Fig. 1. Approximating percentiles. Markov and Chebyshev inequalities

deployment's ability to achieve response times SLOs.

Compared to recent work that also considers the fluid method to evaluate LQNs through PEPA models [14], LINE offers the ability to analyze systems operating in a random environment and supports multi-class processor-sharing resources.

The next section motivates the importance of estimating the response time distribution by means of an example. Next, Section III provides further details on the LINE tool, together with background definitions, and Section IV describes the fluid model underlying LINE. Section V compares the response times obtained with LINE against those observed in a cloud-deployed e-commerce application. The reliability analysis by means of random environments is introduced in Section VI, its results are illustrated in Section VII, and a report of computation times is given in Section VIII. Section IX introduces a reliability-aware resource provisioning approach based on LINE, and Section X concludes the paper.

II. MOTIVATING EXAMPLE: APPROXIMATING PERCENTILES

One of the key features of LINE is its ability to estimate the response time distribution, which can support the assessment of percentile-based SLOs. In the introduction we mentioned that the lack of support to estimate response time percentiles has been marked as a limitation of current LQN solvers [11]. While simulation is a natural method to estimate response-time distributions, it can be very time consuming, especially when estimating the distribution tail [15]. This is worsened when evaluating and optimizing the application quality at design time, which typically requires the evaluation of a large number of scenarios. Traditional analytical models, e.g. Markov chains, are also able to estimate response-time distribution, but their solution does not scale well with the model size due to the well-known curse of dimensionality [16]. Instead, more tractable and efficient models, e.g. product-form queueing networks, do not have tractable algorithms to compute percentiles in multi-class networks of processor-sharing resources [17]. A common approach [18], [19] to overcome this problem is to use approximations, such as the Markov and Chebyshev inequalities, to estimate the response-time distribution from its mean and variance. These inequalities are based on the Chebyshev bound, which, for any non-negative random variable X , states

$$P(X \geq \epsilon) \leq \frac{E[X^k]}{\epsilon^k}, \quad (1)$$

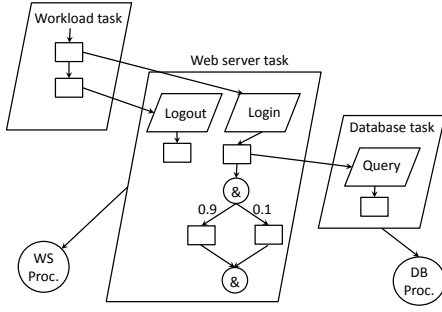


Fig. 2. LQN example

where $E[X^k]$ is the k -th non-centered moment of X , for integer k , and $\epsilon > 0$. The Markov inequality is based on the case $k = 1$, and therefore relies only on the mean $E[X]$. The Chebyshev inequality is based on the case $k = 2$, requiring knowledge of the second moment $E[X^2]$, or equivalently of the mean $E[X]$ and the variance $\text{Var}(X) = E[X^2] - E[X]^2$. Replacing the inequality in (1) by an equality thus provides an approximation of the Complementary Cumulative Distribution Function (CCDF) $P(X \geq \epsilon)$. These approximations however can be very loose, as we show in Figure 1. There we consider two probability distributions, exponential and hyper-exponential, and depict the actual CCDF and the approximations derived from the Markov and Chebyshev inequalities. While these inequalities provide effective bounds on the distribution, they can be very loose, even for simple cases such as the exponential distribution. As a result, provisioning decisions based on these approximations can be very conservative, as the predicted percentiles are much larger than the actual ones, which implies larger deployment costs for the service provider. This example highlights the importance of the analytical estimates of the response time distribution provided by LINE.

III. THE LINE TOOL

LINE is an open-source software tool for the performance and reliability analysis of software applications. It has been developed in MATLAB [20], and its source code and binaries can be downloaded from <http://line-solver.sourceforge.net>. LINE focuses on solving Layered Queueing Network (LQN) models, which have become a popular abstraction to model software applications [4]–[7], [11], [21], [22]. LINE is particularly simple to use in combination with the Palladio Bench tool [5], a software-engineering tool based on Palladio Component Models (PCM). PCMs allow the description of key characteristics of the application components and the resources where these are deployed, such as processing speeds and mean times to failure, making them well-suited for performance and reliability analysis. Once an application has been modeled as a PCM, the LINE solver can be called from Palladio Bench, using a patch that enables LINE to be selected as one of the Palladio LQN solvers. LINE then estimates relevant service

metrics such as throughput or response times¹.

To analyze a software application modeled as a PCM, it is first necessary to transform the PCM into an LQN model. To this end Palladio Bench implements the PCM2LQN transformation [6], which generates an LQN model of the application in the XML format used by the LQNS solver [8]. This LQN model is then parsed by LINE to generate and solve its performance model, described in the next sections. LINE however offers more than solving standard LQN models. First, LINE allows the user to model events that affect the application's reliability by means of *random environments*. Second, the request processing times can be described as PH distributions, which allow for more general behaviors than traditional exponential distributions. Third, LINE estimates the response time distributions, in addition to means, making possible the assessment of percentile-based SLOs. The next subsections provide further details into LQNs, random environments, and PH distributions.

A. Layered queueing networks

An LQN [2] is composed of *tasks*, which represent the software servers, and are deployed on *processors*. Both tasks and processors have a multiplicity attribute, which can be used to model multi-threaded applications deployed on multi-core servers. They also have a scheduling policy to define how the incoming requests are served. A task exposes a set of services, called *entries*, which can be called from other tasks. A simple example is shown in Figure 2, where the Web Server task, deployed on the Web Server Processor, exposes two entries: Logout and Login. Each entry has a set of *activities*, which are executed according to an activity execution graph. An activity executes by posing a demand on the processor it is deployed on, or by generating a call to an entry in a different task.

Activities can execute sequentially, probabilistically, or in parallel. Probabilistic execution is implemented via a *probabilistic OR* node, which has a number of outgoing vertices, each with an associated probability of executing the activities along that vertex. This is illustrated in Figure 2, where the Login entry executes by first calling the Query entry in the Database task, and then proceeds with a probabilistic OR, executing either of two activities, with probabilities 0.9 and 0.1. Similarly, parallel execution is implemented via *fork-join* nodes, where the fork node allows the parallel execution of the next activities, and the join node is executed only after all the previous activities have been completed. The call to an entry can be synchronous or asynchronous, which determines whether the caller processor is blocked or not until the called entry finishes.

The application users are modeled by a *reference task* that exposes no entries, but has a set of task-level activities that specify how the services exposed by other tasks are called by the users. The Workload task in Figure 2 is a reference task that sequentially calls the two entries in the Web Server

¹We must highlight that, to use LINE in combination with Palladio Bench, it is not necessary to have a MATLAB license, as the LINE binaries can be executed by means of the MATLAB Compiler Runtime, a royalty-free set of libraries for the execution of compiled MATLAB applications [20].

task. While these are the basic components of an LQN, a number of extensions have been added to the framework, as surveyed in [2]. These extensions include approximations for non-exponential service times, two-phase services to free the caller after a first service phase, and quorum consensus where an activity is executed after k out of the n predecessor activities have been completed. LINE focuses on probabilistic OR nodes, but not fork-and-join nodes, with the exception of fork-and-join of local activities that can be represented using superposition of phase-type distributions. The extension to support general fork-and-join models is left for future work.

B. From LQN to fluid QN models

In Section IV we introduce the LINE performance model, which is a fluid Queueing Network (QN) model to analyze LQNs. The fluid QN model is built from the LQN description by appropriately matching the LQN tasks, processors, entries, and activities, into processing stations, job classes, and job demands on the processing stations. This transformation is described in detail in [12]. As mentioned in the introduction, the fluid QN model considered in this paper is more general than the one introduced in [12] in that it considers PH-distributed processing times, and a random environment for reliability modeling. These two features however do not have any effect on the transformation introduced in [12], which remains valid for the more general case considered here.

C. PH distributions and Random environments

The LINE performance and reliability model exploits PH distributions to represent fairly general times, beyond the capabilities of the exponential distribution. In fact, PH distributions have been shown to be dense among all distributions with non-negative support [23]. A continuous-time PH distribution is the time to absorption in a continuous-time Markov chain (MC) that has m transient states and one absorbing state. The MC generator matrix can thus be written as

$$Q = \begin{bmatrix} T & t \\ 0 & 0 \end{bmatrix},$$

where T is the sub-generator matrix that corresponds to the transient states, and the vector $t = -Te$, with e a column vector of ones, holds the absorption rates in each transient state. Similarly, the initial state in this MC is selected according to the row probability vector $[\alpha \ \alpha_0]$, where α_i holds the probability that the MC starts in transient state i , and $\alpha_0 = 1 - \alpha e$. The PH distribution derived from this MC is said to have parameters (m, α, T) . Examples of PH distributions are included in Appendix C for reference.

PH distributions play two key roles in the LINE reliability and performance model. First, as discussed in Section IV, LINE uses PH distributions to characterize the request processing times. Second, as described in Section VI, LINE introduces PH distributions to model the holding times in each of the environmental stages that compose a random environment. To incorporate these features in the standard LQN model, we have developed an XML extension. This allows the definition of PH distributions, which are linked to the LQN Activities

TABLE I
QN NOTATION

Parameter	Definition
i, j	Station indexes
r, s	Job class indexes
a, b	Station indexes
e, h	Environmental stage indexes
M	Number of stations
R	Number of classes
N	Number of users
P	Routing probability matrix
$P_{i,j}^{r,s}$	Probability that a class- r job that finishes service at station i proceeds to station j as a class- s job
m_i	Number of servers at station i
Service-time distribution of class- r jobs in station i	
$m^{i,r}$	Number of phases
$\alpha^{i,r}$	Initial probability vector
$T^{i,r}$	Sub-generator matrix
$T_{a,b}^{i,r}$	Transition rate from phase a to b . Entry (a, b) of $T^{i,r}$
$x_{i,r,a}(t)$	Number of class- r jobs in service phase a in station i at time t
$x_i(t)$	Total number of jobs in station i at time t

that use such distribution as execution times. It also enables the definition of random environments, and how the value of certain LQN parameters change according to the environment. More details can be found on the LINE documentation².

IV. THE LINE FLUID PERFORMANCE MODEL

The LINE performance model is a closed queueing network (QN) model composed of processor-sharing (PS) and delay stations, where the processing times are Phase-type (PH) distributed, and the jobs can belong to multiple classes and switch their class after leaving a station. Closed models have a long history in the performance analysis of computer systems [24], [25], as they can model not only applications accessed by a set of users, but also finite resources such as database connection pools and thread pools. To analyze this QN model we introduce a fluid approximation based on a set of ordinary differential equations (ODE), which provides a deterministic approximation method to the expected sample path of the system state. Although approximate, the fluid model has the advantage of avoiding the state-space explosion encountered in Markovian models. Further, we rely on [13] to show that the fluid model becomes exact when the number of jobs in the system is large. The notation introduced in this section is summarized in Table I.

A. The QN model

We consider a closed QN with a total of N jobs circulating among M stations. Each job belongs to one of R classes at any given point in time, but it may switch its class when leaving any station. The processing time of a class- r job in station i is PH-distributed with parameters $(m^{i,r}, \alpha^{i,r}, T^{i,r})$. The state of the QN at time t is described by $X(t) = \{X_{i,r,a}(t), 1 \leq a \leq m^{i,r}, 1 \leq i \leq M, 1 \leq r \leq R\}$, where $X_{i,r,a}(t)$ is the number

²<http://svn.code.sf.net/p/line-solver/code/trunk/doc>

of class- r jobs in service phase a in station i at time t . From here on, we use the index (i, r, a) to refer to the entry of any vector that corresponds to class- r jobs in service-phase a in station i . Two types of events may modify the system state: a *job service completion*, which makes the job move to a new station j where it starts service in phase b with a possible switch to class s ; or a *service phase transition* without service completion, which only alters the processing phase from a to b . Letting $e_{i,r,a}$ be the zero vector with a one in entry (i, r, a) , we see that the first type of event causes the state to jump from state $\mathbf{x}$ to state $\mathbf{x} + e_{j,s,b} - e_{i,r,a}$, as one class- r job in phase a is removed from station i and added to station j as a class- s job in service-phase b . Similarly, the second event type causes a jump to state $\mathbf{x} + e_{i,r,b} - e_{i,r,a}$.

We assume that station i consists of m_i servers that process incoming jobs in a PS fashion. To handle the PS multi-server case we rely on the following abstraction. If at most m_i jobs are present in station i , each of them is assigned to a different processor, otherwise the jobs are assumed to share a super-processor with m_i times the capacity of a single processor. As a result, the transition rates associated to station i depend on the total number of jobs present at that station $x_i = \sum_{r=1}^R \sum_{a=1}^{m_i^r} x_{i,r,a}$. On the one hand, if $x_i \leq m_i$, the transition rates associated to class- r jobs in phase a are proportional to $x_{i,r,a}$, as each job is assigned to a separate processor. For instance, if there are $m_i = 8$ processors and $x_i = 4$ jobs in process, each job is assumed to be in a separate processor, while four processors remain idle. On the other hand, if $x_i \geq m_i$, the corresponding transition rates are proportional to $\frac{x_{i,r,a}}{x_i} m_i$, since in this case the class- r jobs in phase a receive a fraction $\frac{x_{i,r,a}}{x_i}$ of the m_i servers available. In the example above, with $m_i = 8$ processors but with $x_i = 10$ jobs, all the jobs are assumed to share the processor, each receiving the equivalent of 8 tenths of a processor. We can summarize these two alternatives into the expression

$$g(\mathbf{x}, i, a, r) = \frac{x_{i,r,a}}{x_i} \min \{m_i, x_i\}. \quad (2)$$

The service completion rates in state $\mathbf{x}$ is thus

$$f^c(\mathbf{x}, e_{j,s,b} - e_{i,r,a}) = t_a^{i,r} \alpha_b^{j,s} P_{i,j}^{r,s} \frac{x_{i,r,a}}{x_i} \min \{m_i, x_i\}, \quad (3)$$

where we capture that a class- r job in phase a finishes service in station i with rate $t_a^{i,r} g(\mathbf{x}, i, a, r)$, and it is routed to station j as a class- s job with probability $P_{i,j}^{r,s}$, where it starts service in phase b with probability $\alpha_b^{j,s}$. Similarly, the rate associated with a service phase transition is given by

$$f^n(\mathbf{x}, e_{i,r,b} - e_{i,r,a}) = T_{a,b}^{i,r} \frac{x_{i,r,a}}{x_i} \min \{m_i, x_i\}, \quad (4)$$

where it is sufficient to consider the transition rate from service phase a to phase b , $T_{a,b}^{i,r}$. Thus $f^c(\cdot)$ captures completion rates, whereas $f^n(\cdot)$ captures service rates that do not immediately lead to completion. Also notice that by assigning $m_i = N$ servers to a station, it becomes a delay node, since a server is always available for every job. For later reference, we will use $f^c(\mathbf{x}, i, j, r, s, a, b)$ and $f^n(\mathbf{x}, i, r, a, b)$ as shorthand for $f^c(\mathbf{x}, e_{j,s,b} - e_{i,r,a})$ and $f^n(\mathbf{x}, e_{i,r,b} - e_{i,r,a})$, respectively.

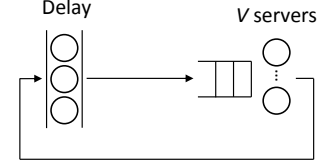


Fig. 3. Example 1 – Queueing network model. The queue adopts a processor sharing service discipline.

Example 1. To illustrate the previous description consider the system in Figure 3, with $R = 2$ job classes, a delay and a processing station, the latter one consisting of $m_2 = S$ servers. Assume that class- r jobs face an exponentially-distributed holding time in the delay station, with mean $1/\mu_r$. For simplicity we assume that jobs do not switch class, thus $P_{1,2}^{1,1} = P_{2,1}^{1,1} = P_{1,2}^{2,2} = P_{2,1}^{2,2} = 1$, and all other entries in P are zero. Thus, letting the class-1 processing times follow an Erlang distribution, with parameters as in (13), the transition rate associated to a class-1 service phase transition from phase 2 to phase 3, in station 2, is

$$\begin{aligned} f^n(\mathbf{x}, e_{2,1,3} - e_{2,1,2}) &= T_{2,3}^{2,1} \frac{x_{2,1,2}}{x_2} \min \{S, x_2\}, \\ &= \gamma \frac{x_{2,1,2}}{x_2} \min \{S, x_2\}. \end{aligned}$$

Similarly, we let the processing times of class-2 jobs follow a hyper-exponential distribution with parameters as in (14). Thus, the transition rates associated with a class-2 service completion, in station 2 and service phase 1, is

$$\begin{aligned} f^c(\mathbf{x}, e_{1,2,1} - e_{2,2,1}) &= t_1^{2,2} \alpha_1^{1,2} P_{2,1}^{2,2} \frac{x_{2,2,1}}{x_2} \min \{S, x_2\}, \\ &= \gamma_1 \frac{x_{2,2,1}}{x_2} \min \{S, x_2\}, \end{aligned}$$

since $\alpha_1^{1,2} = 1$ due to the exponential assumption in the delay node. Further, assume $S = 2$ servers and $x_2 = 3$ jobs in the processing station, which correspond to one class-1 job ($x_{2,1,2} = 1$) and two class-2 jobs ($x_{2,2,1} = 2$). The transition rates above thus become

$$\begin{aligned} f^n(\mathbf{x}, e_{2,1,3} - e_{2,1,2}) &= \gamma \frac{1}{3} 2, \\ f^c(\mathbf{x}, e_{1,2,1} - e_{2,2,1}) &= \gamma_1 \frac{2}{3} 2, \end{aligned}$$

displaying how the two processors are shared among the three jobs present at station 2.

B. The fluid model

To analyze the QN model introduced in the previous section, we consider a sequence of QN models, indexed by v , such that when $v \rightarrow \infty$, their sample paths tend to that of an ODE system, which can be used to approximate the transient behavior of the original QN model. Let $\{X_v(t)\}_{v \in \mathbb{N}_+}$ be a sequence of QN models such that $X_1(t) = X(t)$ is the QN model defined in the previous section, and $X_v(t)$ for $v \geq 2$ is defined as $X_1(t)$ with a population of vN jobs and vm_i servers in station i . As a result, we have a sequence of QN models $\{X_1(t), X_2(t), \dots\}$, where the model $X_v(t)$ has v times more jobs and servers in each station than the

original model $X(t)$. By proportionally scaling both jobs and servers we keep the expected load in each station fixed, and we can rely on the results in [13] to show that, when $v \rightarrow \infty$, the asymptotic behavior of the system state can be described by an ODE system. For a recent survey on ODE approximations of this kind see [26]. The state space of $X_v(t)$ is $\{\mathbf{x} \in \mathbb{N}^{\bar{m}} : \sum_{i=1}^M \sum_{r=1}^R \sum_{a=1}^{m^{i,r}} x_{i,r,a} = vN\}$, where $\bar{m} = \prod_{i=1}^M \prod_{r=1}^R m^{i,r}$. Since we assume that the number of servers scales proportionally with the population size, the number of servers in station i , for any QN model $X_v(t)$, can be written as a fraction $0 < c_i \leq 1$ of the total number of jobs. Thus, the number of servers in station i in the QN model $X_v(t)$ is $vm_i = c_i vN = c_i \sum_{h=1}^M \sum_{r=1}^R \sum_{d=1}^{m^{h,r}} x_{h,r,a}$, for any state $\mathbf{x}$ in the state space of $X_v(t)$.

The transition rates of $X_v(t)$ in state $\mathbf{x}$ associated to service completions can thus be written as

$$\begin{aligned} f_v^c(\mathbf{x}, i, j, r, s, a, b) \\ = vt_a^{i,r} \alpha_b^{j,s} P_{i,j}^{r,s} \frac{x_{i,r,a}}{v} \min \left\{ \frac{c_i}{v} \sum_{h=1}^M \sum_{r=1}^R \sum_{d=1}^{m^{h,r}} x_{h,r,a}, \frac{x_i}{v} \right\}, \\ = vf_1^c(\mathbf{x}/v, i, j, r, s, a, b) = vf^c(\mathbf{x}/v, i, j, r, s, a, b). \end{aligned}$$

Similarly, $f_v^n(\mathbf{x}, i, r, a, b) = vf^n(\mathbf{x}/v, i, r, a, b)$. We thus obtain that the transition rates of the system $X_v(t)$ in state $\mathbf{x}$ can be written as v times the rates of the original system $X_1(t)$ in state $\mathbf{x}/v$. This property of the transition rates is called density-dependence. As a result, and provided that the rate functions $f^c(\mathbf{x}, i, j, r, s, a, b)$ and $f^n(\mathbf{x}, i, r, a, b)$ are continuous for all $\mathbf{x} \in \mathbb{R}^{\bar{m}}$, the sequence $\{X_v(t)\}_{v \in \mathbb{N}_+}$ is a *density-dependent* family of processes [13]. To show that the rate functions are Lipschitz continuous we notice that both $T_{a,b}^{i,r}$ and $t_{a,b}^{i,r}$ are bounded above by the largest exit rate in the sub-generator matrix $T^{i,r}$, which we assume to be finite, $\lambda_{i,r} = \max_a \{-T_{a,a}^{i,r}\} < \infty$. Therefore $f^c(\mathbf{x}, i, j, r, s, a, b) \leq \lambda_{i,r} g(\mathbf{x}, i, r, a)$ and $f^n(\mathbf{x}, i, r, a, b) \leq \lambda_{i,r} g(\mathbf{x}, i, r, a)$, with g as defined in (2). The Lipschitz continuity of the rate functions thus follows from the following lemma, the proof of which is provided in the Appendix.

Lemma 1. *The functions $g(\mathbf{x}, i, r, a)$ in (2) are Lipschitz continuous in $\mathbf{x} \in \mathbb{R}^{\bar{m}}$.*

Theorem 3.1 in [13] shows that, under certain conditions, the sample paths of the *normalized* sequence $\{X_v(t)/v\}_{v \in \mathbb{N}_+}$ converge in probability to a deterministic ODE system. For any $\mathbf{x} \in \mathbb{R}^{\bar{m}}$, let $F(\mathbf{x})$ be the drift of $X(t)$ in state $\mathbf{x}$, that is

$$\begin{aligned} F(\mathbf{x}) = \sum_{i,j=1}^M \sum_{r,s=1}^R \sum_{a,b=1}^{m^{i,r}} (e_{j,s,b} - e_{i,r,a}) f^c(\mathbf{x}, i, j, r, s, a, b) \\ + \sum_{i=1}^M \sum_{r=1}^R \sum_{a,b=1}^{m^{i,r}} (e_{i,r,b} - e_{i,r,a}) f^n(\mathbf{x}, i, r, a, b), \end{aligned} \quad (5)$$

and let $\mathbf{x}(t) \in \mathbb{R}^{\bar{m}}$ be the state of a deterministic system that evolves according to the ODE

$$\frac{d\mathbf{x}(t)}{dt} = F(\mathbf{x}(t)), t \geq 0, \quad (6)$$

and has initial state $\mathbf{x}_0$. Using the results in [13], we prove the following theorem in the Appendix.

Theorem 1. *The sequence of QNs $\{X_v(t)\}_{v \in \mathbb{N}_+}$ converges in the sense of (9) to the solution of the ODE (6).*

Informally, the implication of Theorem 1 is that as the parameter v grows, the MC becomes increasingly similar to the solution of the ODE (6). Since v is a parameter that scales the number of jobs and servers in the system, Theorem 1 states that a large-scale LQN with many jobs and many servers will be approximated in an increasingly accurate manner by the ODE trajectory. It is of paramount importance to notice that this scaling preserves the total server capacity per job, thus the scaling is often a reasonable approximation also for systems where v is slightly greater than unity.

C. Response time distribution

An advantage of the fluid model is that it allows us to approximate the response time distribution of requests, which is important for assessing SLOs. We adopt the approach proposed in [27] for PEPA models to the computation of the distribution in the class of models supported by LINE. To do so we assume an arbitrary initial state $\mathbf{x}(0)$ is provided and introduce a tagged class $R+1$. To estimate the response time distribution of class- r jobs in station i we set the service time distribution of the new class $R+1$ to be equal to that of class r , and define a new initial fluid $\bar{\mathbf{x}}(0)$ that is equal to $\mathbf{x}(0)$ except for $\bar{x}_{R+1,i,a}(0) = x_{r,i,a}(0)$ and $\bar{x}_{r,i,a}(0) = 0$. This means that all the fluid in class r and station i is moved to the tagged class. Next we set the routing probabilities of the tagged class as $P_{R+1,s}^{i,j} = P_{r,s}^{i,j}$, thus allowing the fluid from the tagged class $R+1$ to move as if it was class- r fluid, never returning to class $R+1$. The resulting tagged class is a transient class that has zero fluid once all the initial fluid has been drained from station i .

Let $y(t)$ keep track of the station visited by a job of the tagged class at time t , thus $y(t) \in \{1, \dots, M\}$ for $t \geq 0$. Let S_i be the random variable describing the response time of the jobs in the tagged class in station i , and let Φ_i be its CDF. For any $t \geq 0$, the event $\{y(t) = i | y(0) = i\}$ is equivalent to $\{S_i > t\}$, thus we can state

$$\Phi_i(t) = \mathbb{P}(S_i \leq t) = 1 - \mathbb{P}(y(t) = i | y(0) = i).$$

The last term can also be written as an expected value in terms of the process $X(t)$, extended with the tagged class, as

$$\mathbb{P}(y(t) = i | y(0) = i) = \frac{\mathbb{E}[X_{i,R+1}(t)] - \mathbb{E}[X_{i,R+1}(0)]}{\mathbb{E}[X_{i,R+1}(0)]}.$$

As in [27], we make use of the fluid solution $\mathbf{x}(t)$ to approximate the expected value of $X(t)$, such that

$$\begin{aligned} \Phi_i(t) &= 1 - \frac{\mathbb{E}[X_{i,R+1}(t)] - \mathbb{E}[X_{i,R+1}(0)]}{\mathbb{E}[X_{i,R+1}(0)]} \\ &\approx 1 - \frac{x_{i,R+1}(t) - \bar{x}_{i,R+1}(0)}{\bar{x}_{i,R+1}(0)}. \end{aligned}$$

TABLE II
OFBIZ REQUESTS

Request	Mean [s]	SCV	Request	Mean [s]	SCV
Main	0.193	0.065	QuickAdd	0.156	0.266
Logout	0.039	1.743	AddToCart	0.702	0.329
CheckLogin	0.121	1.751	CheckOutOpts	0.236	0.641
Login	0.049	0.483	OrderHistory	0.096	0.168

The probability density function $\phi_i(\cdot)$ can be obtained as the derivative of $\Phi_i(\cdot)$, thus

$$\phi_i(t) \approx -\frac{1}{\bar{x}_{i,R+1}(0)} \frac{dx_{i,R+1}(t)}{dt} = -\frac{F_{i,R+1}(\mathbf{x}(t))}{\bar{x}_{i,R+1}(0)},$$

where $F(\cdot)$ is the drift defined in (5).

In the previous discussion we assumed a given initial fluid $\mathbf{x}(0)$. To approximate the steady-state response time distribution we can let the ODE evolve until the rate of change vanishes and use the resulting vector as the required $\mathbf{x}(0)$. Although we have not found an instance where the ODE fails to converge to a fixed point, we do not have a proof that there is a unique attractor for the ODE, and the convergence result on which we rely is only valid for a finite T . However, for the analysis with blending, as discussed in Section VI, it is sufficient to determine the value of $\mathbf{x}(T)$ for a finite T , which is then used in the subsequent iterations of the algorithm.

V. EVALUATING RESPONSE TIME PERCENTILES

To illustrate the response time predictions obtained with LINE, we make use of Apache OFBiz³, an open-source enterprise framework, which includes an e-commerce application that we deploy on the cloud using Amazon EC2⁴. Once deployed, we use the OFBench tool [28] to generate traffic from a set of emulated users, also running on cloud instances. The application, which is composed of an application and a database server, is deployed on a *c1.xlarge* virtual machine that features 8 cores, and serves all the traffic generated by N emulated users, chosen in the range $N = [10, 90]$. The users access the application by submitting a sequence of requests during a single session. Between the submission of consecutive requests, and between the end and the start of a session, the users undergo a think time.

To determine the processing time distributions we collect monitoring data from the OFBiz log files, which gather arrival and departures timestamps at request level. Relying on this dataset, we use the FMLPS method in [29] to estimate the *mean* processing times. In addition, we estimate the squared coefficient of variation (SCV) of the processing times using the CI method in [29]. That is, we use CI to obtain estimates for the service demand samples from response time samples. By calculating the mean and variance of these samples, we readily obtain the SCV. Notice that, although accurate, these methods remain approximate in estimating the application effective CPU consumption (see [29] for more details). As illustrated in Table II, there are 8 different request types

TABLE III
LINE RELATIVE ERRORS

N	errMean	errCDF	err95	err99
10	2%	8%	13%	14%
30	1%	13%	9%	15%
50	1%	13%	9%	15%
70	3%	9%	7%	8%
90	11%	11%	11%	13%

TABLE IV
RT 95TH PERCENTILE

N	Trace	Sim	LINE	Markov	Chebyshev
10	2.45	2.76	2.76	41.07	9.36
30	2.70	2.93	2.93	38.07	8.89
50	3.12	3.42	3.42	42.62	10.05
70	4.15	5.10	4.45	62.72	14.83
90	5.23	4.86	4.68	61.18	14.39

executed by the clients accessing the application, each of which has different statistical characteristics. In particular, we observe that the SCV is not equal to 1, as assumed by the exponential distribution, a feature that can be captured by LINE through the use of PH distributions. We thus use standard methods [30] to match the mean and SCV of the processing times with Erlang and hyper-exponential distributions, which belong to the set of PH distributions.

We have conducted experiments with a number of clients between 10 and 90, which generate an average server utilization between 6% and 56%. Notice that in a production deployment, a large number of instances are deployed and the requests are assigned according to a load balancer, such that each available resource receives a fraction of the total number of users. We compare the response times predicted by LINE against the times observed in the trace, as well as against results obtained by simulation and the Markov and Chebyshev inequalities. The simulation model considers the QN model extracted from the LQN model, described in Section IV-A, without the approximations introduced by the LINE fluid solution. This model is built and run in the JMT simulation tool [31]. Figures 4(a) to (c) depict the results for the cases with 50, 70, and 90 users, focusing on the session response time (RT) cumulative distribution function (CDF). Under 50 users, we observe a clear match between LINE and simulation, both of which closely follow the trace CDF. For 70 and 90 users, both LINE and simulation are also very close to the trace CDF, although the errors are larger. This behavior is confirmed by Table III, which includes the relative errors on the mean, the 95th and 99th percentiles, and the mean error on the CDF evaluated at all the percentiles. While the mean is almost perfectly matched for up to 70 users, the error is larger for 90 users. Overall, the errors in mean, CDF, and percentiles are at most 15%.

As mentioned in Section II, a common approach [18], [19] to approximate a CDF is to use the mean and variance of the associated random variable, together with the Markov and Chebyshev inequalities. We have thus used these inequalities

³<http://ofbiz.apache.org/>

⁴<http://aws.amazon.com/>

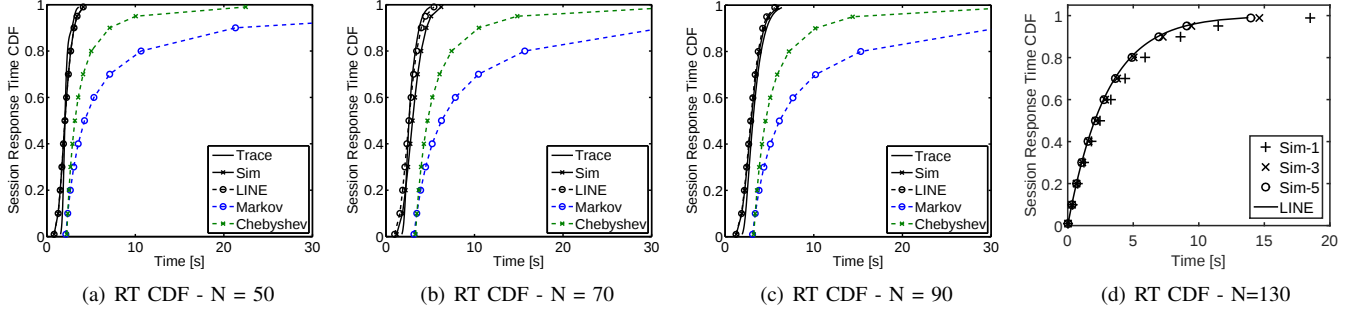


Fig. 4. Session Response Time CDF

TABLE V
RT 95TH PERCENTILE - PREDICTED FROM $N = 10, 90$

N	Trace	Sim	LINE	Markov	Chebyshev
10	2.45	2.76	2.76	41.11	9.37
30	2.70	3.53	3.52	45.79	10.69
50	3.12	4.02	4.04	50.30	11.85
70	4.15	4.51	4.52	55.14	13.07
90	5.23	4.83	4.68	61.07	14.34

to approximate the RT CDF, using the mean and variance obtained from simulation, which have relatively small estimation errors. Figures 4(a) to (c) show how these two approximations compare with the trace CDF. Clearly, both approximations are far from the actual RT CDF, differing by up to 1 and 2 orders of magnitude for the higher percentiles. Table IV shows the specific values for the 95th percentile obtained with the different methods, showing large errors for the Markov and Chebyshev approximations. For instance, the 95th percentile estimated by the Markov and Chebyshev inequalities are 12 and 3 times larger than the one observed. The results obtained with LINE offer a much better approximation of the RT CDF. Table V presents similar results, but in this case we use the mean demands for the $N = 10, 90$ cases to parameterize all the models. We have observed that the mean demands in this case increase with the number of users. Thus, we used the measurements from the $N = 10, 90$ cases to approximate the demands for the other cases by means of a simple linear interpolation, and use these demands to predict the response times for the other scenarios. While the errors are larger than in Table IV, LINE offers similar predictions as simulation, and much closer to the observed response times than the Markov and Chebyshev inequalities.

We now compare the results of LINE with those of a simulation model. Our goal here, compared to the previous experiments, is to explore a case where the system performance is not known by measurements, and we want to see if LINE can offer the same prediction quality as simulation. We first observe that the differences between the simulation and LINE are very small, below 1%, when the number of users is up to 50, as illustrated in Figure 4(a). Increasing the number of users, and thus the utilization, causes larger errors, which are however overcome as the scale of the system increases. Figure 4(d) illustrates this for the case with 130 users, which

generate a utilization of 82%. The figure depicts the CDF estimated by LINE, and the ones obtained with simulation where a scale S is indicated by the label Sim- S . This means that both the number of users and servers are scaled by a factor of S . We observe how the difference between the fluid and the simulation reduces as the scale grows. In fact, the mean relative error in the 95-th percentile is 28% for the system without scaling, and this decreases to 6% and 2% when the system is scaled by a factor of 3 and 5, respectively. LINE therefore offers better results for systems under low and medium loads, as well as for systems with a large number of users and servers, which are the conditions expected for production cloud applications.

VI. RELIABILITY ANALYSIS

While the performance model introduced in the previous sections provides an efficient method to analyze large LQN models, it lacks support for reliability analysis. We now introduce this capability by means of an environment that generates faults and reliability issues. Consider for example the classical problem of modeling a system with servers that may experience breakdowns at random times followed by repairs. LINE supports the modeling of these systems through the notion of *random environment* (RE). An RE is composed of E stages, each representing a possible condition of the environment. For example, in a model with breakdowns and repairs, stages may be used to count the number of faulty servers waiting for a repair.

A. Definitions

We assume the RE evolves as follows. The holding time in stage e follows a PH distribution with parameters (n_e, β_e, V_e) , and let $v_e = -V_e e$ be the vector of absorption rates, where e is a column vector of ones. After a visit to stage e , the environment moves to stage h with probability $d_{e,h}$, such that $\sum_{h \neq e} d_{e,h} = 1$. We can therefore describe the RE as a continuous-time MC with a state space composed of E levels, where level e has n_e states. The generator matrix Q of this MC can be partitioned according to the state-space, such that the sub-matrix $Q_{e,h}$ holds the transition rates from level e to level h , or equivalently from stage e to stage h . The sub-matrix $Q_{e,h}$, for $h \neq e$, is thus given by

$$Q_{e,h} = v_e d_{e,h} \beta_h,$$

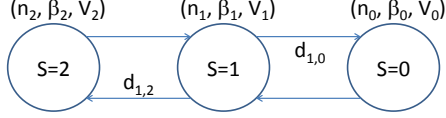


Fig. 5. Random environment example

combining the exit rates from stage e (v_e) with the probability that the next stage visited is h ($d_{e,h}$), and the initial probability distribution of the phases in stage h (β_h). Within stage e , the transition rates are given by $Q_{e,e} = V_e$. Solving the system $\pi Q = 0$, $\pi e = 1$, we obtain the stationary distribution π of the MC, which can be partitioned according to the state space into sub-vectors π_e corresponding to each stage.

LINE accepts the specification of the input parameters of the model listed in Table I as a function of the current stage e in the RE. This complicates the analysis since the system is now described by a set of E systems of ODEs (6), one for each possible stage in the RE, raising the question on how to jointly use these descriptions to analyze the system. In LINE we generalize the blending approach in [32], which has the ability to consider REs that evolve at different time-scales, including the time-scale at which requests are processed. As discussed in [32], this differs from methods based on decomposition techniques, which require the RE to evolve at a much larger or smaller rate than the requests' processing rates. The blending method makes use of a fluid model to track the evolution of the system state $x_e(t)$ when the RE is in stage e . This means that the system starts in state $x_e(0)$ and evolves with drift $F_e(\cdot)$ according to (6) until a time T when an environmental transition from stage e to stage $h \neq e$ occurs. The system then starts in state $x_h(0)$ and evolves with a different drift $F_h(\cdot)$ until a new transition occurs. The initial state $x_h(0)$ is defined from the terminal state $x_e(T)$ by applying a *Reset matrix* $R_{e,h}$, as $x_h(0) = x_e(T)R_{e,h}$. The Reset matrix is a stochastic matrix that defines how the jobs must be reallocated when an environmental transition occurs. For instance, as shown in Section VII, if the PH representation of the processing times is associated with a set of consecutive processing phases, an environmental transition can cause the current jobs in process in any phase to re-start in phase one due to a failure event associated with the environmental transition.

Example 1 (cont). We define a random environment for Example 1, described in Section IV. Let the random environment be composed of 3 stages, as in Figure 5, where in stage e there are e servers working in station 2. This random environment can be used to model potential server failures. For instance, the holding time (n_2, β_2, V_2) captures the time until the next failure of either server when both servers are running. The MC that models the random environment is therefore

$$Q = \begin{bmatrix} V_0 & v_0\beta_1 & 0 \\ d_{1,0}v_0\beta_0 & V_1 & d_{1,2}v_1\beta_2 \\ 0 & v_2\beta_1 & V_2 \end{bmatrix}.$$

Here we assumed $d_{0,1} = d_{2,1} = 1$, thus only one server fails or is repaired at any time.

B. The blending method

The blending algorithm introduced in [32] approximates the steady-state distribution of the system state by cyclically considering the evolution of the system in each environmental stage, and the transitions triggered by a change in the environmental stage. Here we extend this method to consider environmental stages with PH holding times, instead of the exponential assumption considered in [32]. We also combine it with the LINE model, which is more general than the single-class first-come first-served model in [32]. As a result, when the environment jumps from stage e to stage h , the reset matrix $R_{e,h}$ must keep track of the number of jobs of each type in each station and service phase, and how these are affected by the environmental stage transition. For instance, if service is composed of consecutive phases, all jobs of a certain class in a given station can be forced to re-start service in the first phase, reflecting a loss of information caused by a server failure. Therefore, due to the multi-class PS scheduling, the blending method considered here offers support for different reset policies depending on the job class. Let $x_e^0(0)$ be the initial system state in stage e , where the exponent $k = 0$ counts the number of environmental transitions that have occurred. We obtain $x_e^0(t)$ as the solution of (6), with drift $F_e(\cdot)$ and initial state $x_e^0(0)$, for t from zero to the time of the first environmental transition. As the holding times in the environmental stages follow a PH distribution, the probability density that a transition occurs at time t , given that the current stage is e , is $\beta_e \exp(-V_e t) v_e$. Thus the distribution of the system state just before the first environmental transition is $\int_0^\infty x_e^0(t) \beta_e \exp(-V_e t) v_e dt$. The initial state in stage h after the first environmental transition is thus given by

$$x_h^1(0) = \sum_{e=1, e \neq h}^E p_{e,h} \int_0^\infty x_e^0(t) \beta_e \exp(-V_e t) v_e dt R_{e,h},$$

where $p_{e,h}$ is the probability that, given that the current stage is h , the last stage visited was e . The Reset matrix $R_{e,h}$ is used to map the final state in stage e into the initial state in stage h . The blending method is based on this iteration, but replaces $p_{e,h}$ with a stationary version $\tilde{p}_{e,h}$ defined as

$$\tilde{p}_{e,h} = \frac{\pi_e v_e d_{e,h}}{\sum_{l=1, l \neq e}^E \pi_l v_l d_{l,h}},$$

where π_e is the stationary probability vector associated to stage e . The blending algorithm therefore iteratively computes

$$x_h^{k+1}(0) = \sum_{e=1, e \neq h}^E \tilde{p}_{e,h} \int_0^\infty x_e^k(t) \beta_e \exp(-V_e t) v_e dt R_{e,h},$$

for every stage e , until the difference between two consecutive iterations is smaller than a threshold ϵ close to zero. If the algorithm terminates after $\hat{k}$ iterations, the blending approximation of the expected system state in the long run, given it is in stage e , is $\int_0^\infty x_e^{\hat{k}}(t) \beta_e \exp(-V_e t) v_e dt$.

VII. MODELING FAILURES AS RANDOM ENVIRONMENTS

We now revisit the OFBiz application described in Section V introducing an RE to model server failures.

TABLE VI
RESET STRATEGIES

Strategy	Transition	Reset type
Full Reset (FR)	$\mathcal{N} \rightarrow \mathcal{F}$	All restart in phase 1
	$\mathcal{F} \rightarrow \mathcal{N}$	All restart in phase 1
No Reset (NR)	$\mathcal{N} \rightarrow \mathcal{F}$	No restart
	$\mathcal{F} \rightarrow \mathcal{N}$	No restart
Partial Reset (PR)	$\mathcal{N} \rightarrow \mathcal{F}$	Half restart in phase 1
	$\mathcal{F} \rightarrow \mathcal{N}$	No restart

A. Random environments: failures

The RE we consider in this section models the temporary failure of a server. The RE is made of two environmental stages, one associated to a normal operational state ($\mathcal{N}$), and one where one server fails ($\mathcal{F}$). Compared to the system considered in Section V, we duplicate the server capacity (number of VMs), and the number of users. We introduce an RE where the stage $\mathcal{N}$ has the same characteristics as the Base case, while in stage $\mathcal{F}$ half of the VMs are lost due to contention at host level. Stages $\mathcal{N}$ and $\mathcal{F}$ are assumed to last on average 600s and 60s, respectively. Figure 6(a) shows the session RT CCDF for two workload levels: 100 and 180 users. For $N = 100$ users, the simulation results obtained for each stage (*SimF* and *SimN*) coincide with the LINE results, showing that this workload can be successfully handled by the system even in its failed state. Instead, when the number of users increases to $N = 180$, a significant difference between the RTs in each stage arises. As expected, we observe that the LINE RT CCDF is located between the CCDFs of each stage obtained with simulation. The LINE CCDF is actually very close to the one corresponding to stage $\mathcal{N}$, up until the 90th percentile, but diverges from then on, as the requests experiencing the longest RTs are those that arrive during the $\mathcal{F}$ stage. The longest RTs are however smaller than those predicted for the $\mathcal{N}$ stage alone. This illustrates how the effect of a reliability issue depends on the workload.

B. Random environments: the Reset matrix

We now look into the effect of the Reset matrix described in Section VI. We assume processing times composed of ten consecutive phases, each with an exponential duration, resulting in an Erlang-distributed processing time, as in (13). Notice that the processing time in each phase could be PH distributed, as the sum of PH random variables is also PH [23]. These phases reflect an application that processes requests in ten operational phases, with a possible checkpoint after each of them. Thus, in case the application fails during any phase, its execution can be restarted from the checkpoint, without going back to the first phase. We model the failures as in the previous section, by means of a RE with two stages, $\mathcal{N}$ for normal conditions, and $\mathcal{F}$ when a failure occurs. We thus use the RE and the Reset matrix to reflect the effect of the checkpointing strategy on the offered service.

We consider three different Reset strategies, summarized in Table VI: full reset (FR), no reset (NR), and partial reset (PR). In the NR strategy, all the requests continue unaltered

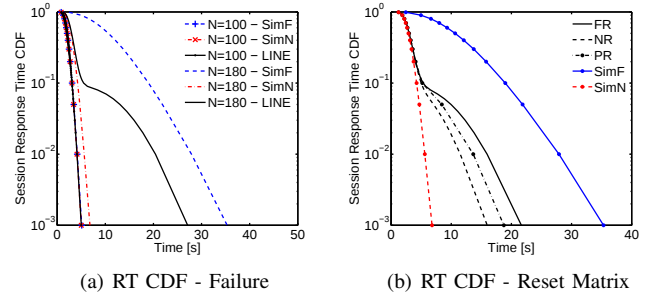


Fig. 6. Failure Random Environment and Effect of the Reset matrix

in their processing phase when an environmental transition occurs. This strategy captures the case where checkpointing is in place, as the requests in the failed server can restart in their service phase in a different server. Under the PR strategy, in a transition from stage $\mathcal{F}$ to stage $\mathcal{N}$, which occurs when the failed server is back online, all the requests are kept in their respective processing phases, as in the NR case. Instead, in a transition from stage $\mathcal{N}$ to stage $\mathcal{F}$, when a server fails, we *reset* half of the requests to the first phase. This strategy thus models the case without checkpointing, where the requests in the failed server must restart execution from phase one, in the remaining server. For comparison we also define the FR strategy, where a failure causes all the requests to restart from phase one. This strategy, also assumed in Section VII-A, marks an upper bound on the effect of the failure, as all the requests, not only those in the failed server, must restart service in case of a server failure.

The session RT CDFs obtained under the three Reset strategies are shown in Figure 6(b), together with the RT CDFs estimated via simulation for each stage. We observe that FR offers longer processing times than the other two strategies, as a result of the reset of the requests in process. The NR strategy offers the best RT since the requests are not reset by the server failure thanks to the checkpointing. The PR strategy shows how the lack of checkpointing increases the RTs, especially for the requests that experience the longer RTs. For instance, the RT 95th and 99th percentile increase by about 19% when checkpointing is removed. These results are obtained with a mean holding time of the $\mathcal{N}$ and $\mathcal{F}$ stages of 60s and 6s, respectively. We have observed that the effect of the Reset strategy diminishes as the holding times in the environmental stages increase. This is to be expected as the way the failure is handled has a smaller effect if the failures occur less frequently. The RE is therefore able to capture the effect of environmental transitions that occur at different time scales. This makes it more robust than traditional approaches, which only consider the case where the failures occur at time scales significantly slower than the processing times.

Table VII compares the results obtained with LINE against simulation for the three Reset strategies considered in the previous experiments. The difference in the mean response time (*errMean*) is negligible, while along the distribution the mean difference in percentiles (*errCDF*) is around 5%. The difference for the 95-th and 99-th percentiles, in the last two columns, is larger but it is much lower than what could

TABLE VII
LINE RELATIVE ERRORS - RANDOM ENVIRONMENTS

Reset	errMean	errCDF	err95	err99
FR	0.9%	4.5%	10.2%	11.3%
PR	0.9%	4.1%	14.7%	17.4%
NR	0.5%	4.2%	12.1%	17.6%

be obtained with approximation methods as the Markov and Chebyshev inequalities, as shown in Table IV. Moreover, in the next section we show that the times to solve the LINE model considering random environments is below 20s in the worst case. Instead, the simulations required to evaluate the above scenario took on average 1.3 hours, and a maximum of 10 hours, on the same machine and using the same development and execution environment (MATLAB). These large times are caused by the high variability introduced by the random environments, which implies that many samples are needed to accurately estimate the percentiles.

VIII. COMPUTATIONAL REQUIREMENTS

We now report on the computation times required by LINE to analyze the models in the previous sections. We focus on computation times as we have observed the memory consumption in LINE to be negligible. LINE is implemented in MATLAB 2012b and makes use of its ODE solvers. The default solver is *ode15s*, which supports stiff ODE systems, but other solvers such as *ode23s* can also be used. We have run four sets of experiments with the OFBiz application. In each set we consider 100 scenarios, varying the number of users in the set $\{10, 20, \dots, 200\}$ and the number of cores in the set $\{2, 4, 8, 16, 32\}$. For the first two sets we consider the case without RE, as in Section V. The first set, labeled PH(1), assumes exponentially-distributed processing times, which can be represented with a PH distribution with one phase. The second set, labeled PH(50), assumes lowly-varying processing times, as those observed in Table II, modeled with an Erlang distribution with 50 phases, resulting in an SCV of 0.02. We used a 3.4 GHz 4-core Intel Core i7 machine, with 4GB RAM, running Linux Ubuntu 12.04. Table VIII shows the execution times, including mean, standard deviation, maximum, minimum, and a 95% confidence interval. These times include solving the ODE system, computing mean performance measures, and computing overall RT distributions. While the times increase significantly with the larger description of the processing times, these times are still much shorter than simulations, and can be used for optimization purposes, as illustrated in the next section. In fact, the simulations of these instances required on average 6 s, and a maximum of 10.3 s, being at least one order of magnitude slower than LINE.

For the third and fourth sets of experiments we include the failure RE in Section VII. We consider two cases for the RE holding times: fast and slow. The slow RE has mean holding times of 600s and 60s in the $\mathcal{N}$ and $\mathcal{F}$ stages, respectively. The fast RE has mean holding times of only 60s and 6s in these stages, which are similar in magnitude to the mean processing times. We observe in Table VIII that the inclusion of REs

TABLE VIII
COMPUTATION TIMES (s)

Model type	Mean	Std. Dev.	Min	Max	95% CI	
PH(1)	0.09	0.01	0.08	0.12	0.09	0.09
PH(50)	0.49	0.06	0.35	0.62	0.48	0.5
RE slow	1.44	0.31	0.8	1.85	1.38	1.5
RE fast	6.45	3.74	0.93	16.74	5.71	7.2

increases the computation times, although they remain overall small, and that the increase depends on the stage holding times. This is caused by the number of events that can be observed in each stage visit. In the slow RE case, many service-completion events can be observed in a single visit to a stage, since these visits are long compared to the mean processing times. Instead, in the fast RE case, where the visits are of a magnitude similar to the processing times, only a few events can be observed in one stage visit. As a result, the blending algorithm described in Section VI requires more iterations to find the fixed point that approximates the system steady-state. As mentioned in the previous section, using simulation for these scenarios required on average 1.3 hours, and a maximum of 10 hours, which is two orders of magnitude larger than LINE. These times are clearly not feasible for optimization purposes.

LQNS, which is a solver for LQN models based on approximate mean-value analysis, typically features similar or shorter execution times than those of LINE. This is to be expected as LINE relies on numerical methods to solve the ODE system associated to the fluid QN model. Further, obtaining the RT distributions requires the solution of one ODE system for each request class. The additional execution time is however justified as LINE supports PH-distributed processing times and computes the RT distribution for each request class. LQNS instead provides average performance metrics only. Further, LINE integrates REs to model reliability aspects such as failures or high-contention events.

This comparison is based on the sequential solution of single LQN models. LINE also features two *parallel* execution modes, exploiting the parallel capabilities of MATLAB. The first mode is based on the *parfor* method in MATLAB, which enables the parallel evaluation of multiple LQN models. The second mode operates as a batch processing engine that allows the asynchronous submission of multiple jobs, each of which can consist of several LQN models. Both modes provide substantial gains, especially when many LQN models are solved, as for what-if analyses and optimization.

IX. RELIABILITY-AWARE RESOURCE PROVISIONING

In this section we illustrate how LINE can be used to formulate design-time decision problems based on the reliability and performance of a cloud application. In particular, we consider an application running on spot instances, which can be obtained by bidding a price above the spot price, but that are lost when the spot price surpasses the offered bid. Spot instances are attractive as their prices can be well below those of on-demand instances, but the uncertainty in their availability makes their management challenging.

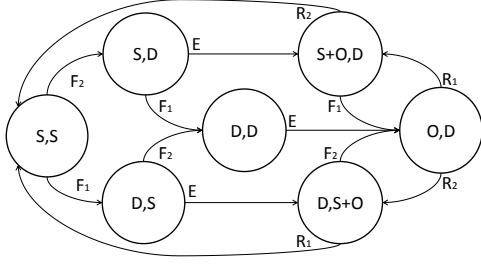


Fig. 7. Random environment for application deployed on two zones

We consider an application deployed on two availability zones, such that, when the spot instances in one zone are lost, those in the other zone can keep serving requests, while a new bid is submitted to recover the instances lost. As start-up times of spot instances can be significant [33], in case the instances in one zone are lost, on-demand instances can be started-up. Once the spot instances lost are recovered, the on-demand instances can be stopped, returning to a fully spot-based deployment. Thus, in case the instances in both zones are lost, the on-demand instances can keep the application running, avoiding prolonged down times. To model this deployment scenario, we use the RE abstraction in LINE to capture the different deployment states, caused by the evolution of the spot prices. The RE, depicted in Figure 7, has 7 stages, each of them described with the tuple (Z_1, Z_2) , where Z_i holds the state of the instances in zone i , which can be one of the following:

- S , if *only* the spot instances are up and running;
- D , if the spot instances in this zone are down or lost;
- $S + O$, if *both* the spot and on-demand instances are running;
- O , if only the on-demand instances are running in this zone.

Accordingly, three types of events may occur:

- F_i , the spot instances in zone i are lost;
- R_i , the spot instances in zone i are recovered;
- E , the emergency on-demand instances are brought up, an event triggered when the spot instances in either zone are lost.

These and other definitions are summarized in Table IX. Notice that all the spot instances in each zone are assumed to be of the same type, and therefore fail together. However, this can be generalized, to consider instances of different types, each with a different price.

As an example, consider stage (S, S) , where the spot instances in both zones are running. If event F_2 occurs, the spot instances in zone 2 are lost and the new stage is (S, D) , where only the spot instances in the first zone remain up. In this stage, event E starts the on-demand VMs, and the RE switches to stage $(S + O, D)$. Notice that the application availability is $1 - \pi_{(D,D)}$, where $\pi_{(D,D)}$ is the stationary probability of finding the application in stage (D, D) , where the instances in both zones have been lost before the on-demand instances could take over.

TABLE IX
NOTATION FOR THE SPOT-INSTANCE MODEL

State of the instances in each zone	
S	Spot instances running
D	Spot instances lost (down)
$S + O$	Both spot and on-demand instances running
O	On-demand instances running
Events	
F_i	Spot instances in zone i are lost
R_i	Spot instances in zone i are recovered
E	On-demand instances are started-up
Inter-event times	
T_F	Mean time to lose a spot instance
T_R	Mean time to recover a spot instance
T_E	Mean time to start-up an on-demand instances
Number of instances	
$m_{(i,j)}^S$	Number of spot instances in zone i for component j
m_j^O	Number of on-demand instances for component j

TABLE X
PROVISIONING AND FAILURE TIMES IN THE SPOT INSTANCE MODEL [33]

T_R	60, 300, 600s
T_E	60, 300, 600s
T_F	600, 1200, 2400, 3600s

A. What-if analysis

We illustrate this deployment scenario with a simple model of the open-source OFBiz e-commerce application [28]. Different from the setup in Section V, here we assume the application and the database servers are deployed on separate VMs, and are replicated on two availability zones. A model of the spot-instance-based deployment of this application is depicted in Figure 8. Here a request is served by either zone, visiting both servers, multiple times if necessary. Once the request is completed, the user receives the response and undergoes a think time before submitting a new request. Notice that a spot instance provisioning model has also been recently considered in [34], which is a preliminary follow-up application of the present work relying on the LINE tool to setup a model for the availability and optimal topology of a single instance.

In order to consider realistic scenarios, we have parametrized our spot instance model using statistics obtained in a recent workload characterization study [33]. The specific times considered are summarized in Table X. While the mean times to recover spot and on-demand instances T_E and T_R are defined according to [33], the mean time to loose an instance T_F is defined to consider different instance availability levels, as this affects the overall application availability. A given instance availability can be achieved by setting the bid price according to historical records as in [34]. The authors of [33] observe that the startup times of VMs in real clouds, especially spot instances, are highly variable depending on the network condition and the VM image size. Therefore, the loss of a VM introduces uncertainty on the time before the system can return to full capacity. This uncertainty can easily be modeled in LINE by means of the RE abstraction. This illustrates a

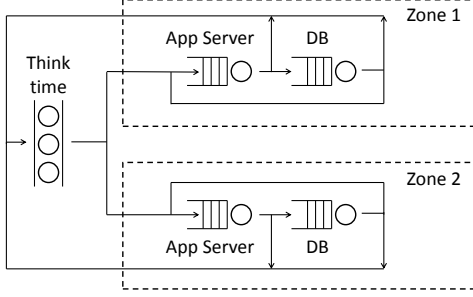


Fig. 8. Model of the 2-zone deployment

practical advantage of the features offered by the LINE solver, which are not available in ordinary queueing network solvers.

Figure 9 depicts the RT Complementary Cumulative Distribution Function (CCDF) for 3 different values of the mean time to recover the spot instances in a zone, T_R , between 60s and 600s. We also set the mean time to lose the spot instance in one zone T_F to be one hour, and the mean time to obtain an on-demand instance T_E to 60s. With this setup the spot instance availability is between 85% and 99%, which can be achieved by setting the bid price according to historic prices as in [34]. For instance, a bid price of \$ 0.534 can achieve a 90% availability for *m1.xlarge* spot instances [34]. In Figure 9(a) we allocate $m_{(i,j)}^S = 2$ spot instances in each zone for each component, and $m_j^O = 1$ on-demand instances for each component. We observe how the three cases depicted show a staircase behavior, where the RT distribution shows different “regimes”. For instance, for $T_R = 60$ a first regime covers up to the 90th percentile with an exponential tail that terminates T around 1 s, and a second regime covers the 10% of the jobs with the longest delays. These different regimes are the result of the RE, showing how the failures affect the longest RTs. For $T_R = 60$, over 90% of the requests face RTs below 1s, but as T_R increases, the first regime becomes less prevalent, leading many more jobs to face the longer RTs offered under the second regime.

As depicted in Figure 9(b), increasing the number of spot instances $m_{(i,j)}^S$ to 3, leads to a similar behavior as above, but in this case at least three regimes are apparent. For $T_R = 60$, the proportion of jobs with RTs below 1 s, is larger than in the previous case, while the second regime offers shorter RTs, going as far as 5s, and covering 99.9% of the requests. The remaining requests face a different regime that offers longer RTs. The prevalence and exact times offered by these regimes varies with T_R in a non-trivial fashion, showing the interactions between reliability and performance captured by LINE. Figure 9(b) also depicts the approximate RT CCDF for $T_R = 600$, obtained with the Chebyshev inequality, described in Section II, which relies on the mean and the variance of the RT. Not only the approximate distribution can be very far from the actual CCDF (e.g., the difference for the 95th percentile is over one order of magnitude), but the approximate CCDF completely ignores the different regimes caused by the RE.

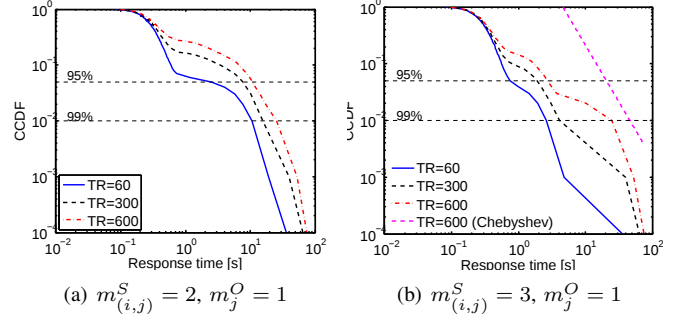


Fig. 9. Response time CCDF under changes of T_R , the time to recover spot instances

B. Optimal provisioning with LINE

Based on LINE, we can setup an optimal-provisioning problem to determine the number of instances $m_{(i,j)}^S$ and m_j^O required to achieve a certain SLO at the minimum cost. Let the vectors m^S and m^O hold the $m_{(i,j)}^S$ and m_j^O variables, and define C_i^S and C^O as the leasing costs of spot and on-demand instances, commonly given on an hourly basis. We consider two metrics to set the SLOs: the mean RT; or a percentile of the RT distribution.

1) *Provisioning based on the mean response time:* Given m^S and m^O , we can obtain the mean RT $E[R]$ using LINE, which we write as $E[R] = \text{LINE}(m^S, m^O)$. Let $R^{\max}$ be the maximum *mean* RT set as an SLO. The reliability-aware optimal provisioning problem can then be stated as

$$\begin{aligned} \min \quad & \sum_{j=1}^2 \left(\sum_{i=1}^2 C_i^S m_{(i,j)}^S + C^O m_j^O \right) \\ \text{s.t.} \quad & E[R] = \text{LINE}(m^S, m^O), \\ & E[R] \leq R^{\max}, \\ & m_{(i,j)}^S \in \mathbb{N}_+, \quad i = 1, 2, \quad j = 1, 2, \\ & m_j^O \in \mathbb{N}_+, \quad j = 1, 2, \end{aligned} \quad (7)$$

where $\mathbb{N}_+$ is the set of strictly-positive natural numbers, thus one spot instance in each zone and one on-demand instance must be available for each component. This ensures that the application is able to process requests in every stage except (D, D) . This is a nonlinear non-convex integer problem, for which meta-heuristic methods are well-suited to find close-to-optimal solutions. We have implemented a solution procedure based on simulated annealing [35], which is able to quickly provide very good solutions, although these are not guaranteed to be global optima. To implement this method we define a solution to be a vector with the number of instances of each type for each component $[m^S, m^O]$, and the neighbors of this solution are found by increasing or decreasing one entry of this vector by one. The method randomly selects a neighbor and decides to accept it as the new current solution if the objective function improves. It can also accept a new solution if the objective function degrades, according to a probability that decreases as the temperature in the simulated annealing process decreases. For the instances considered in the paper,

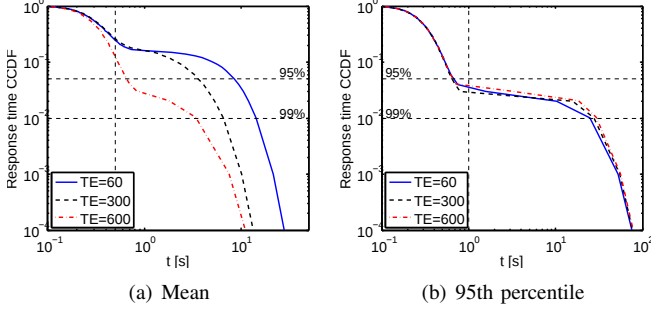


Fig. 10. Response times under optimal provisioning. Various T_E

the optimization method requires between 5 and 25 minutes to complete in the standard PC described in Section VIII.

Figure 10(a) depicts the RT distribution with the optimal provisioning found by setting the maximum *mean* response time $RT^{\max}$ to be 0.5s, and considering three different values for the time to retrieve an on-demand instance T_E . We observe that the RT distribution varies significantly, although in all the cases the maximum of 0.5s on average holds. The optimal provisioning asks for 9, 12, and 14 spot instances in total, for T_E equal to 60, 300, and 600s, respectively. At the same time, the deployment availability decreases from 99.99% to 99.97% and 97.22% in the three cases mentioned. Thus, as the availability decreases, due to the increase in the time to retrieve the on-demand instances, more spot instances are necessary to achieve the desired SLO. The increase in spot instances results in an improvement of the RT high percentiles, although at a higher deployment cost.

2) *Provisioning based on response time percentiles:* We now consider setting an SLO on the response time x -th percentile, which can be at most $R_x^{\max}$. We obtain the response-time x -th percentile R_x as $R_x = \text{LINE}(m^S, m^O)$, as it depends on m^S and m^O . This can be incorporated in the optimal provisioning problem (7) by replacing the first two constraints by

$$R_x = \text{LINE}(m^S, m^O), \quad R_x \leq R_x^{\max}. \quad (8)$$

We perform the same experiment as in the previous section, now setting an SLO of 1s for the response-time 95th percentile. The results are depicted in Figure 10(b), where we observe smaller differences in the RT distributions among the three scenarios considered. In all of them the provisioning ensures that the constraint on the 95th percentile holds, which is achieved by requesting more spot instances. In the three scenarios the optimal provisioning requests 12, 14, and 14 spot instances, respectively, an increase over the numbers in the previous experiments. However, in all three scenarios only 2 on-demand instances are required, which is a decrease compared to the previous experiment, where between 5 and 6 on-demand instances were requested. Thus, to ensure the constraint of 1s on the 95th percentile, the optimal provisioning attempts to have enough capacity in any of the stages where the spot instances in at least one availability zone are running. To reduce costs, the on-demand instances are kept at a minimum, providing little quality of service during the

periods when the application is only running on on-demand instances. This is apparent in the large difference between the two regimes in Figure 10(b), where the RTs grow significantly after the 95th percentile. By comparison, if we use the Markov inequality to estimate the RT 95th percentile, we find that the 1s SLO is not achievable even with 1000 servers. To find a feasible provisioning, according to the Markov inequality, we need to increase the SLO to 10s. For the experiment above, 13, 13, and 14 spot instances and 4, 4, and 5 on-demand instances are required, respectively. As a result, the use of the Markov inequality requires more resources than the provisioning using LINE, even though the SLO is 10 times larger. A similar result arises with the Chebyshev inequality, where the SLO needs to be doubled, and the number of required on-demand instances increases to 7. Neither of these inequalities is therefore able to provide results for tight SLOs.

X. CONCLUSION

We have introduced LINE, a tool for the performance and reliability analysis of LQN models that relies on a fluid QN model and a random environment description. We have shown how LINE is able to capture the interactions between reliability and performance, and to determine the ability of an application to achieve given SLOs. Validation against monitoring data from a cloud application, as well as simulation, show the ability of LINE to estimate performance metrics, particularly the response time distribution. While the previous sections focused on the specific scenarios of resource failures and a spot-instance-based deployment, LINE can be used to model many other scenarios of software applications that must cope with substantial uncertainty, affecting both their reliability and performance.

APPENDIX

A. Proof of Lemma 1

Using the 1-norm, for two points $\mathbf{x}, \mathbf{y} \in \mathbb{R}^{\bar{m}}$ we consider four cases.

Case 1: $x_i > m_i$ and $y_i > m_i$.

$$\begin{aligned} & |g(\mathbf{y}, i, r, a) - g(\mathbf{x}, i, r, a)| \\ &= m_i \left| \frac{y_{i,r,a}}{y_i} - \frac{x_{i,r,a}}{x_i} \right| < \frac{1}{m_i} |y_{i,r,a}x_i - x_{i,r,a}y_i| \\ &= \frac{1}{m_i} |y_{i,r,a}(x_i - y_i) - (x_{i,r,a} - y_{i,r,a})y_i| \\ &\leq \frac{N}{m_i} (|x_i - y_i| + |x_{i,r,a} - y_{i,r,a}|) \leq \frac{2N}{m_i} |y - x|. \end{aligned}$$

Case 2: $x_i \leq m_i$ and $y_i > m_i$.

$$\begin{aligned} & |g(\mathbf{y}, i, r, a) - g(\mathbf{x}, i, r, a)| \\ &= \left| \frac{y_{i,r,a}}{y_i} m_i - x_{i,r,a} \right| < \frac{1}{m_i} |m_i y_{i,r,a} - x_{i,r,a} y_i| \\ &< \frac{1}{m_i} |m_i y_{i,r,a} - m_i x_{i,r,a}| \leq |y - x|. \end{aligned}$$

Case 3: $x_i > m_i$ and $y_i \leq m_i$.

$$\begin{aligned} & |g(\mathbf{y}, i, r, a) - g(\mathbf{x}, i, r, a)| \\ &= \left| y_{i,r,a} - \frac{x_{i,r,a}}{x_i} m_i \right| < \frac{1}{m_i} |y_{i,r,a} x_i - m_i x_{i,r,a}| \\ &= \frac{1}{m_i} |y_{i,r,a} (x_i - y_i) + y_{i,r,a} y_i - m_i x_{i,r,a}| \\ &\leq \frac{1}{m_i} |m_i (x_i - y_i) + m_i (y_{i,r,a} - x_{i,r,a})| \leq 2 |y - x|. \end{aligned}$$

Case 4: $x_i \leq m_i$ and $y_i \leq m_i$.

$$|g(\mathbf{y}, i, r, a) - g(\mathbf{x}, i, r, a)| = |y_{i,r,a} - x_{i,r,a}| \leq |y - x|.$$

As a result $|g(\mathbf{y}, i, r, a) - g(\mathbf{x}, i, r, a)| \leq 2N|\mathbf{y} - \mathbf{x}|$, as $1 \leq m_i \leq N$.

B. Proof of Theorem 1

[13, Theorem 3.1] states that the sample paths of the sequence $\{X_v(t)/v\}_{v \in \mathbb{N}_+}$ converge to the deterministic limit $\mathbf{x}(t)$ as $v \rightarrow \infty$, i.e., that $\lim_{v \rightarrow \infty} X_v(0)/v = \mathbf{x}_0$ implies that for every $\delta > 0$

$$\lim_{v \rightarrow \infty} \mathbf{P} \left(\sup_{s \leq t} \left| \frac{1}{v} X_v(s) - \mathbf{x}(s) \right| > \delta \right) = 0. \quad (9)$$

This holds for every finite t if $\mathbf{x}(s) \in E$ for $0 \leq s \leq t$, where E is an open set $E \subset \mathbb{R}^m$ such that

$$|F(\mathbf{y}) - F(\mathbf{x})| < M_E |\mathbf{y} - \mathbf{x}|, \quad \mathbf{x}, \mathbf{y} \in E, \quad (10)$$

$$\sup_{\mathbf{x} \in E} \left\{ \sum_{i,j=1}^M \sum_{r,s=1}^R \sum_{a,b=1}^{m^{i,r}} |e_{j,s,b} - e_{i,r,a}| f^c(\mathbf{x}, i, j, r, s, a, b) + \sum_{i=1}^M \sum_{r=1}^R \sum_{a,b=1}^{m^{i,r}} |e_{i,r,b} - e_{i,r,a}| f^n(\mathbf{x}, i, r, a, b) \right\} < \infty, \quad (11)$$

$$\begin{aligned} & \lim_{d \rightarrow \infty} \sup_{\mathbf{x} \in E} \left\{ \sum_{(i,j,r,s,a,b) \in S^1(d)} |e_{j,s,b} - e_{i,r,a}| f^c(\mathbf{x}, i, j, r, s, a, b) \right. \\ & \left. + \sum_{(i,r,a,b) \in S^2(d)} |e_{i,r,b} - e_{i,r,a}| f^n(\mathbf{x}, i, r, a, b) \right\} = 0, \quad (12) \end{aligned}$$

where $S^1(d)$ is the set $\{(i, j, r, s, a, b) : |e_{j,s,b} - e_{i,r,a}| > d\}$, $S^2(d)$ is defined similarly, and M_E is a constant. Notice that condition (10) implies the Lipschitz continuity of $F(\cdot)$ which ensures the existence of a unique solution to the ODE (6).

Theorem 1 follows from [13, Theorem 3.1] by showing that the sequence $\{X_v(t)/v\}_{v \in \mathbb{N}_+}$ verifies the conditions (10), (11), and (12). As the QN model $X(t)$ is closed, the entries $X_{i,r}(t)$ are bounded above by N , a condition that also holds for every $X_v(t)/v$ as well as for $\mathbf{x}(t)$. As a result the set E can be chosen as the smallest open set that contains the set $\{\mathbf{x} \in \mathbb{R}^m : \mathbf{x} \geq 0, \sum_{i=1}^M \sum_{r=1}^R \sum_{a=1}^{m^{i,r}} x_{i,r,a} = N\}$, as the sample paths of $\mathbf{x}(t)$ never leave this set.

To verify condition (10) we consider $\mathbf{x}, \mathbf{y} \in E$ and the 1-norm to write

$$\begin{aligned} & |F(\mathbf{y}) - F(\mathbf{x})| \\ &= \left| \sum_{i,j=1}^M \sum_{r,s=1}^R \sum_{a,b=1}^{m^{i,r}} (e_{j,s,b} - e_{i,r,a}) (f^c(\mathbf{y}, i, j, r, s, a, b) - f^c(\mathbf{x}, i, j, r, s, a, b)) \right. \\ & \quad \left. + \sum_{i=1}^M \sum_{r=1}^R \sum_{a,b=1}^{m^{i,r}} (e_{i,r,b} - e_{i,r,a}) (f^n(\mathbf{y}, i, r, a, b) - f^n(\mathbf{x}, i, r, a, b)) \right| \\ &\leq 2 \sum_{i,j=1}^M \sum_{r,s=1}^R \sum_{a,b=1}^{m^{i,r}} |f^c(\mathbf{y}, i, j, r, s, a, b) - f^c(\mathbf{x}, i, j, r, s, a, b)| \\ & \quad + 2 \sum_{i=1}^M \sum_{r=1}^R \sum_{a,b=1}^{m^{i,r}} |f^n(\mathbf{y}, i, r, a, b) - f^n(\mathbf{x}, i, r, a, b)| \end{aligned}$$

since $|e_{j,s,b} - e_{i,r,a}| \leq 2$. From Lemma 1, (3), (4), and letting $\lambda = \max_{i,r} \{\lambda_{i,r}\}$, we get

$$\begin{aligned} & |F(\mathbf{y}) - F(\mathbf{x})| \leq 2 \sum_{i,j=1}^M \sum_{r,s=1}^R \sum_{a,b=1}^{m^{i,r}} 2N\lambda^{i,r} |\mathbf{y} - \mathbf{x}| \\ & \quad + 2 \sum_{i=1}^M \sum_{r=1}^R \sum_{a,b=1}^{m^{i,r}} 2N\lambda^{i,r} |\mathbf{y} - \mathbf{x}| \\ & \leq 4\lambda N |\mathbf{y} - \mathbf{x}| \prod_{i=1}^M \prod_{r=1}^R m^{i,r} \prod_{j=1}^M \prod_{s=1}^R m^{j,s} \\ & \quad + 4\lambda N |\mathbf{y} - \mathbf{x}| \prod_{i=1}^M \prod_{r=1}^R (m^{i,r})^2 \\ & \leq 8\lambda N \bar{m}^2 |\mathbf{y} - \mathbf{x}|, \end{aligned}$$

and therefore condition (10) is satisfied. Notice that we replace the summations by equivalent products, as no terms are left in the summations. The first term reduces to counting the total number of pairs $((i, r, a), (j, s, b))$ that can be obtained, while the second term counts the pairs $((i, r, a), (i, r, b))$.

Condition (11) can be readily verified by noticing that $|e_{j,s,b} - e_{i,r,a}| \leq 2$, $|e_{i,r,b} - e_{i,r,a}| \leq 2$, $f^1(\mathbf{x}, i, j, r, s, a, b) \leq \lambda^{i,r} m_i$, and $f^n(\mathbf{x}, i, r, a, b) \leq \lambda^{i,r} m_i$. Finally, condition (12) is verified by observing that the sets $S^1(d)$ and $S^2(d)$ are empty for every $d > 2$.

C. PH Examples

Examples of PH distributions include the exponential, Erlang, and hyper-exponential distributions. The simple exponential distribution with rate λ has a PH representation with $m = 1$ phase, $\alpha = 1$, $T = -\lambda$. The Erlang(n , γ) distribution, which is the sum of n independent and identically distributed exponential distributions with parameter γ , has a PH representation with n phases. For instance, the Erlang(3, γ)

distribution can be represented as $m = 3$,

$$\alpha = [1 \ 0 \ 0], \quad T = \begin{bmatrix} -\gamma & \gamma & 0 \\ 0 & -\gamma & \gamma \\ 0 & 0 & -\gamma \end{bmatrix}. \quad (13)$$

Another common example is the hyper-exponential distribution, which is the convex combination of n exponential distributions, where the i -th distribution has rate γ_i and is selected with probability β_i . For instance, a hyper-exponential distribution with two phases can be represented as $m = 2$,

$$\alpha = [\beta_1 \ \beta_2], \quad T = \begin{bmatrix} -\gamma_1 & 0 \\ 0 & -\gamma_2 \end{bmatrix}. \quad (14)$$

The flexibility of PH distributions is also reflected in a number of fitting algorithms [36]–[39] to obtain a PH representation from empirical data. These algorithms are available as part of software tools such as [40]–[42], among others.

REFERENCES

- [1] 451 Research, “Enterprise cloud computing poised for explosive growth during next two years,” 2013.
- [2] G. Franks, T. Al-Omari, M. Woodside, O. Das, and S. Derisavi, “Enhanced modeling and solution of layered queueing networks,” *Soft. Eng., IEEE Trans.*, vol. 35, no. 2, pp. 148–161, 2009.
- [3] Object Management Group (OMG), “A UML profile for MARTE: Modeling and analysis of real-time embedded systems, beta 2,” OMG, Tech. Rep. OMG Document Number: ptc/2008-06-09, 2008.
- [4] M. Tribastone, P. Mayer, and M. Wirsing, “Performance prediction of service-oriented systems with layered queueing networks,” in *Proc. ISoLA 2010*, 2010, pp. 51–65.
- [5] S. Becker, H. Koziol, and R. Reussner, “Model-based performance prediction with the palladio component model,” in *Proc. 6th WOSP*, 2007, pp. 54–65.
- [6] H. Koziol and R. Reussner, “A model transformation from the palladio component model to layered queueing networks,” in *Performance Evaluation: Metrics, Models and Benchmarks*. Springer, 2008.
- [7] X. Wu and M. Woodside, “Performance modeling from software components,” in *Proc. 4th WOSP*, 2004.
- [8] G. Franks, P. Maly, M. Woodside, D. C. Petriu, A. Hubbard, and M. Mroz, *Layered Queueing Network Solver and Simulator User Manual*, Carleton University, January 2013.
- [9] O. Das and C. M. Woodside, *Architecting Dependable Systems II*. Springer-Verlag, 2004, ch. Dependability Modeling of Self-Healing Client-Server Applications.
- [10] G. Casale and M. Tribastone, “Modelling exogenous variability in cloud deployments,” *ACM Perf. Eval. Rev.*, vol. 40, no. 4, pp. 73–82, 2013.
- [11] D. Bacigalupo, J. van Hemert, X. Chen, A. Usmani, A. Chester, L. He, D. Dillenberger, G. Wills, L. Gilbert, and S. Jarvis, “Managing dynamic enterprise and urgent workloads on clouds using layered queueing and historical performance models,” *Simulation Modelling Practice and Theory*, vol. 19, pp. 1479–1495, 2011.
- [12] J. F. Pérez and G. Casale, “Assessing SLA compliance from palladio component models,” in *Proc. 15th SYNASC*, 2013.
- [13] T. G. Kurtz, “Solutions of ordinary differential equations as limits of pure jump Markov processes,” *Journal of App Prob.*, vol. 7, pp. 49–58, 1970.
- [14] M. Tribastone, “A fluid model for layered queueing networks,” *IEEE Trans. Softw. Eng.*, vol. 39, no. 6, pp. 744–756, Jun. 2013.
- [15] P. Harrison and W. Knottenbelt, “Passage time distributions in large markov chains,” *ACM Perf. Eval. Rev.*, vol. 30, pp. 77–85, 2002.
- [16] R. Bellman, *Dynamic programming*. Princeton University, 1957.
- [17] G. Casale, “Approximating passage time distributions in queueing models by bayesian expansion,” *Perf. Eval.*, vol. 67, pp. 1076–1091, 2010.
- [18] D. Ardagna, C. Ghezzi, and R. Mirandola, “Rethinking the use of models in software architecture,” in *Proc. of the 4th International Conference on Quality of Software-Architectures: Models and Architectures*, 2008.
- [19] I. Cunha, J. Almeida, V. Almeida, and M. Santos, “Self-adaptive capacity management for multi-tier virtualized environments,” in *10th IFIP/IEEE Symposium on Integrated Network Management*, 2007.
- [20] I. The MathWorks, “Matlab release 2012b,” Natick, Massachusetts, United States.
- [21] A. Faisal, D. Petriu, and M. Woodside, “Network latency impact on performance of software deployed across multiple clouds,” in *Proc. CASCON '13*, 2013.
- [22] G. Jung, K. R. Joshi, M. A. Hiltunen, R. D. Schlichting, and C. Pu, “Generating adaptation policies for multi-tier applications in consolidated server environments,” in *Proc. of the 5th ICAC*, 2008.
- [23] S. Asmussen, *Applied probability and queues*. Springer, 2003.
- [24] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik, *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*. Prentice-Hall, Inc., 1984.
- [25] M. Reiser, “A queueing network analysis of computer communication networks with window flow control,” *IEEE Trans. Comm.*, vol. 27, no. 8, pp. 1199–1209, 1979.
- [26] L. Bortolussi, J. Hillston, D. Latella, and M. Massink, “Continuous approximation of collective system behaviour: A tutorial,” *Perf Eval.*, vol. 70, pp. 317–349, 2013.
- [27] R. A. Hayden, A. Stefanek, and J. T. Bradley, “Fluid computation of passage-time distributions in large Markov models,” *Theor. Comput. Sci.*, vol. 413, no. 1, pp. 106–141, Jan. 2012.
- [28] J. Moschetta and G. Casale, “OFBench: an Enterprise Application Benchmark for Cloud Resource Management Studies,” in *MICAS*, 2012.
- [29] J. F. Pérez, G. Casale, and S. Pacheco-Sanchez, “Estimating computational requirements in multi-threaded applications,” *IEEE Trans. Softw. Eng.*, vol. 41, pp. 264 – 278, 2015.
- [30] W. Whitt, “Approximating a point process by a renewal process, i: Two basic methods,” *Operations Research*, vol. 30, pp. 125–147, 1982.
- [31] M. Bertoli, G. Casale, and G. Serazzi, “JMT: performance engineering tools for system modeling,” *ACM Perf. Eval. Rev.*, vol. 36, pp. 10–15, 2009.
- [32] G. Casale and M. Tribastone, “Fluid analysis of queueing in two-stage random environments,” in *Proc. QEST 2011*, 2011.
- [33] M. Mao and M. Humphrey, “A performance study on the VM startup time in the cloud,” in *Proc. 5th IEEE CLOUD*, 2012.
- [34] D. Dubois and G. Casale, “Autonomic provisioning and application mapping on spot cloud resources,” in *Proc of IEEE ICCAC*, 2015.
- [35] S. Kirkpatrick, C. D. Gelatt, Jr, and M. P. Vecchi, “Optimization by simulated annealing,” *Science*, vol. 220, pp. 671–680, 1983.
- [36] S. Asmussen, O. Nerman, and M. Olsson, “Fitting Phase Type distributions via the EM algorithm,” *Scandinavian Journal of Statistics*, vol. 23, pp. 419, 441, 1996.
- [37] A. Bobbio, A. Horvath, and M. Telek, “Matching three moments with minimal acyclic Phase Type distributions,” *Stochastic Models*, vol. 21, pp. 303–326, 2005.
- [38] H. Okamura, T. Dohi, and K. S. Trivedi, “A refined EM algorithm for PH distributions,” *Perform. Eval.*, vol. 68, pp. 938–954, 2011.
- [39] A. Thummler, P. Buchholz, and M. Telek, “A novel approach for phase-type fitting with the EM algorithm,” *Dep. Sec. Comp., IEEE Trans.*, vol. 3, no. 3, pp. 245–258, 2006.
- [40] P. Reinecke, T. Krauss, and K. Wolter, “Hyperstar: Phase-type fitting made easy,” in *Proc. 9th QEST*, Sept 2012, pp. 201–202.
- [41] A. Bobbio, A. Horvath, and M. Telek, “PhFit: a general phase-type fitting tool,” in *Proc. 2002 DSN*, 2002.
- [42] J. F. Pérez and G. Riaño, “jPhase: An object-oriented tool for modeling phase-type distributions,” in *Proc. 2006 SMCTools*, 2006.

Juan F. Pérez is a Research Fellow at the University of Melbourne, Department 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.

Giuliano Casale is a Lecturer in performance analysis and operations research at Imperial College London, Department of Computing, since January 2013. Prior to this, he was a full-time researcher at SAP Research (UK), a postdoctoral research associate at the College of William & Mary (US). He obtained a PhD from Politecnico di Milano (Italy) in 2006. He has served as program co-chair for ACM SIGMETRICS/Performance 2012 QEST 2012, and ICAC 2014, and as general co-chair for ACM/SPEC ICPE 2013, and as technical program committee member for more than 50 conferences and workshops.