LINE: Evaluating Software Applications in Unreliable Environments

Juan F. Pérez and Giuliano Casale

***Abstract*—Cloud computing has paved the way to the flexible deployment of software applications. This flexibility offers service providers a number of options to tailor their deployments to the observed and foreseen customer workloads, without incurring in large capital costs. However, cloud deployments 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 performance and reliability analysis of software applications. LINE solves Layered Queueing Network (LQN) models, a popular class of stochastic models in software performance engineering, by setting up and solving an associated system of ordinary differential equations. A key differentiator of LINE compared to existing solvers for LQNs is that LINE incorporates a model of the environment the application operates in. This enables the modeling of reliability and performance 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 support for performance and reliability modeling, and illustrates its potential by comparing LINE predictions against data obtained from a cloud deployment. We also illustrate the applicability of LINE with 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
$\mathrm{Var}(\cdot)$	Variance

I. INTRODUCTION

Recent years have seen a sustained growth in the adoption of cloud computing, a trend expected to continue in the future [1]. Advantages of cloud deployments include ample flexibility to manage resources, allowing service providers to use only those that are needed to achieve service-level objectives (SLOs), and to reduce the capital costs associated with large computing infrastructures. However, opting for a cloud-based deployment poses a number of novel challenges, related to both reliability and performance. For example, 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 costs, but that can be lost whenever their spot price exceeds the bid price [2].

In spite of the additional uncertainty introduced by the cloud offering, model-driven software engineering can help taking provisioning decisions through models that explicitly capture the inherent uncertainty in the application processing times, and in the reliability of the deployment environment. A class of models useful in capturing the complexity of modern software systems are Layered Queueing Network models (LQNs) [3]. To fully exploit LQN models, multiple efforts have been put forward to automate their generation from high-level software specifications (e.g., UML MARTE [4], [5], PCM [6], [7], CBML [8]). Further, efficient methods and tools exist to quantitatively evaluate LQN models and compute relevant performance and reliability measures, such as approximated mean-value analysis methods adopted in the LQNS tool [9].

Although reliability extensions have been proposed for LQN models [10], these focus on *mean* values of the service metrics, and are appropriate for 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 cloud resource provisioning, where, for instance, the virtualized environment can affect the application processing rates at very short time scales [11]. In addition, while existing methods and tools are effective in estimating *mean* performance metrics, the lack of support for the analytic computation of response time *percentiles* has been pointed out in the literature as a limitation of LQN models for SLO assessment [12]. As cloud applications typically serve different user classes, and support a large number of request types, metrics such as response time 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 the exploration of a large parameter space. A similar issue is faced in decision-support systems used to decide optimal

Juan F. Pérez is with the Department of Applied Mathematics and Computer Science, Universidad del Rosario, Bogotá, Colombia, e-mail:juanferna.perez@urosario.edu.co.

Giuliano Casale is with the Department of Computing, Imperial College London, London, UK, e-mail:g.casale@imperial.ac.uk.

deployments 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 demanding when solving the non-linear integer optimization problems underpinning resource provisioning.

In this paper we introduce LINE, a solver for LQN models that tackles the above limitations. The main features of LINE are as follows:

- LINE is a solver for LQN models that relies on a *fluid* representation to 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 application evolves in. This enables engineers to model the reliability and performance of complex software systems in the 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 level of individual user or request classes. 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 initial prototype of LINE was introduced in [13]. The present paper generalizes the scope and approach of the methodology by the following contributions:

- We introduce a generalized approach for reliability modeling by means of random environments. Compared to prior work [11], which focuses on random environments with exponential holding times in each environmental state, the random environments in LINE support Phase-type (PH) holding times, therefore allowing the use, for example, of complex failure distributions and rates.
- LINE supports processor-sharing servers and requests belonging to multiple classes, and different user types, as performance models are mapped onto multi-class and multi-chain queueing network models. This provides an innovation compared to current random environment theory, which is significantly more restrictive as it focuses on single-class models and first-come first-served queues only [11]. Note that in the case of single-class workloads, LINE also supports first-come first-served processors.
- LINE supports consider queueing stations with PH processing times. Such PH distributions are unrelated to those used for random environments and enable the modelling of features such as high-variability in service times, among others. Within this quite general setting, we provide a formal proof of the convergence of the LINE fluid model in the sense of Kurtz [14].
- 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 data.
- 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 compliance with response times SLOs.

Compared to recent work that also considers the fluid method to evaluate LQNs through PEPA models [15], LINE offers the ability to analyze systems operating in a random environment, supports multi-class processor-sharing resources, and computes response time percentiles, which are notoriously challenging in the presence of preemptive scheduling disciplines. At present LINE supports a subset of the features available in LQN models. More details are provided in Section III.

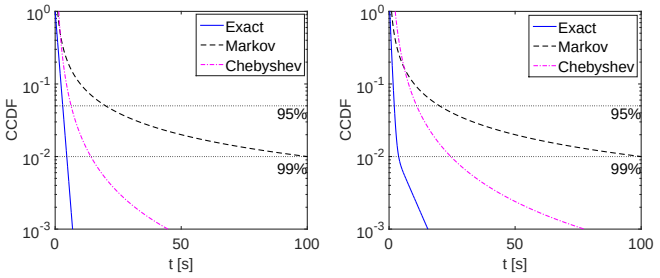
The next section motivates the importance of estimating the response time distribution by means of an example. Next, Section III provides further details on the LINE tool, together with background definitions, and Section IV describes the fluid model underlying LINE. Section V compares the response times 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, and experimental 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 [12]. While simulation is a natural method to estimate response-time distributions, it can be very time consuming, especially when estimating the distribution tail [16]. 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, such as Markov chains, are also able to estimate response-time distributions, but their solution does not scale well with the model size due to the well-known curse of dimensionality [17]. 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 [18]. A common approach [19], [20] 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

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

where $E[X^k]$ is the k-th non-centered moment of X, for integer k, and $\epsilon > 0$. The Markov inequality corresponds to the



(a) Exponential (b) Hyper-exponential

Fig. 1. Approximating percentiles. Markov and Chebyshev inequalities

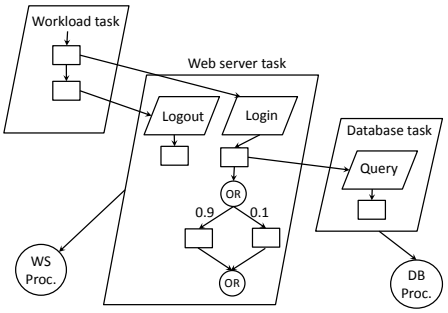


Fig. 2. LQN example

case $k = 1$, and therefore relies only on the mean $E[X]$. The Chebyshev inequality is given by 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 shown 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, since 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, and its source code and binaries can be downloaded from [21]. LINE focuses on solving LQN models, which have become a popular abstraction to model software systems [5]–[8], [12], [22], [23]. LINE is particularly simple to use in combination with the Palladio Bench tool [6], a software-engineering tool based on the PCM paradigm. 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 directly from Palladio Bench. LINE then estimates relevant service metrics such as throughput or response times[1]

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 [7], which generates an LQN model of the application in the XML format used by the LQNS solver [9].

[1]We 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.

This LQN model is then parsed by LINE to generate and solve its performance model, as described in the next sections.

A. Layered queueing networks

An LQN [3] is composed of *tasks*, which represent the software servers that 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 enables the parallel execution of the activities, and the join node is executed only after all previous activities have been completed. The call to an entry can be synchronous or asynchronous, depending on 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 task. While these are the basic components of an LQN, a number of extensions have been added to the framework, as surveyed in [3]. 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. While this transformation is described in [13], here we provide an overview. Each reference task in the LQN, which models a set of application users, is transformed into a QN *chain* of users. This chain holds a set of classes among which users in the chain can switch probabilistically, but they are not allowed to switch to classes in other chains. The number of users in the chain is given by the multiplicity of the reference task, and for each chain we also create a delay station in the QN, where the users undergo a think time, as defined by the reference task. The example in Figure 2 has a single reference task, which defines a single chain, a delay station, and an associated mean think time.

The remaining tasks are divided in two groups, depending on whether or not they hold activities that pose effective demands on the processor where the task is deployed. When such resource-demanding activities are present, the task and associated processor are mapped into a station in the QN model, with as many servers as indicated by the task multiplicity. The processor also defines a scheduling policy, which is used to define the policy at the QN station. Having defined the stations and user chains in the QN model, the transformation then creates the routing links among stations, for each user chain, by following the sequence of calls starting from each reference task.

In addition to simply routing users in one chain across the QN stations, the transformation also determines class-switching probabilities. The main purpose of the class-switching feature is to enable the QN model to capture the fact that multiple visits to the same station may be necessary to serve a request, and each of these visits may require a different execution time. For instance, in Figure 2 executing the Login service entails processing at the web server station before and after retrieving data from the database. To allow each of these executions to have a different processing time, we switch the request class just before the second visit to the web server station. While the *stationary* analysis of *mean* metrics is not affected by class-switching, the response-time *distributions* and the *transient* metrics obtained with LINE rely on this feature to provide accurate results. Finally, the fluid QN model considered in this paper is more general than the one introduced in [13] in that it considers PH-distributed processing times, and a random environment for reliability modeling, as we describe in Section III-D.

C. Current Limitations

The focus of LINE is on delivering novel performance and reliability analysis techniques for LQN analysis. These include in particular random environments, response time percentiles and PH distributions, which we show later to be useful for reliability-aware provisioning. Such features are novel in the LQN space and require more advanced solution algorithms compared to mean-value analysis, which is widespread for solving LQNs but yet does not seem amenable to analyze these features. Therefore, the initial release of LINE considers a simplified LQN model that is amenable to analysis and application of Kurtz's theorem [14], but which for ease of implementation neglects some features supported by existing LQN solvers such as LQNS. These include processors with FCFS scheduling, fork-join nodes that execute on multiple processors, and forwarding calls. Still, LINE supports key features of LQNs, for example by allowing both synchronous and asynchronous calls.

We believe that LINE could be extended to encompass most of the features that are currently missing. One potential limitation is the combination of synchronous calls with simultaneous resource contention, arising for example when a synchronous call blocks the calling processor until completion. Our initial analysis reveals that the generalization of the fluid equations to this case leads to potential violation of the Lipschitz continuity assumptions underpinning Kurtz's theorem. Such degenerate cases corresponds to situations where one or more resources are continuously blocked. While the practical relevance of such degenerate cases appears limited, additional work is needed to understand the convergence properties of the associated stochastic processes in the presence of such degeneracies.

D. 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 [24]. 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

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
n_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
$\boldsymbol{x}(t)$	Fluid solution approximating $X(t)$
$\boldsymbol{x}$, $\boldsymbol{y}$	States visited by $X(t)$ or $\boldsymbol{x}(t)$

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 enables the definition of PH distributions, which are linked to the LQN Activities 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 documentation [21].

IV. THE LINE FLUID PERFORMANCE MODEL

The LINE performance model is a closed QN model composed of multi-server processor-sharing (PS) and delay stations, where the processing times are 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 [25], [26], 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 [14] 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 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 $\boldsymbol{x}$ be an arbitrary state visited by $X(t)$ and $\boldsymbol{e}_{i,r,a}$ 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 $\boldsymbol{x}$ to state $\boldsymbol{x} + \boldsymbol{e}_{j,s,b} - \boldsymbol{e}_{i,r,a}$, 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 $\boldsymbol{x} + \boldsymbol{e}_{i,r,b} - \boldsymbol{e}_{i,r,a}$.

We assume that station i consists of n_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 n_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 n_i times the capacity of a single processor. This implicitly models the re-allocation of jobs across servers to best use their capacity. 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 $x_i \leq n_i$, 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 $n_i = 8$ 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 $x_i \geq n_i$, the corresponding transition rates are proportional to $\frac{x_{i,r,a}}{x_i} n_i$, since in this case the class-r jobs in phase a receive a fraction $\frac{x_{i,r,a}}{x_i}$ of the n_i servers available. In the example above, with $n_i = 8$ 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

$$g(\boldsymbol{x}, i, a, r) = \frac{x_{i,r,a}}{x_i} \min\{n_i, x_i\}. \tag{2}$$

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

$$f^c(\boldsymbol{x}, \boldsymbol{e}_{j,s,b} - \boldsymbol{e}_{i,r,a}) = t_a^{i,r} \alpha_b^{j,s} P_{i,j}^{r,s} \frac{x_{i,r,a}}{x_i} \min\{n_i, x_i\}, \tag{3}$$

where we capture that a class-r job in phase a finishes service in station i with rate $t_a^{i,r} g(\boldsymbol{x}, i, a, r)$, 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

$$f^n(\boldsymbol{x}, \boldsymbol{e}_{i,r,b} - \boldsymbol{e}_{i,r,a}) = T_{a,b}^{i,r} \frac{x_{i,r,a}}{x_i} \min\{n_i, x_i\}, \tag{4}$$

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,

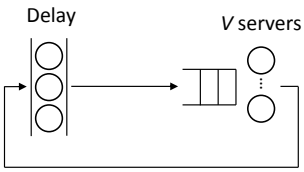


Fig. 3. Example 1 – Queueing network model. The queue adopts a processor sharing service discipline.

whereas $f^n(\cdot)$ captures service rates that do *not* immediately lead to completion. Also notice that by assigning $n_i = N$ servers to a station, it becomes a delay node, since a server is always available for every job. For later reference, we will use $f^c(\boldsymbol{x}, i, j, r, s, a, b)$ and $f^n(\boldsymbol{x}, i, r, a, b)$ as shorthand for $f^c(\boldsymbol{x}, \boldsymbol{e}_{j,s,b} - \boldsymbol{e}_{i,r,a})$ and $f^n(\boldsymbol{x}, \boldsymbol{e}_{i,r,b} - \boldsymbol{e}_{i,r,a})$, respectively.

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 $n_2 = S$ 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 $P^{1,1}_{1,2} = P^{1,1}_{2,1} = P^{2,2}_{1,2} = P^{2,2}_{2,1} = 1$, and all other entries in P are zero. Thus, letting the class-1 processing times follow an Erlang$(3, \gamma)$ distribution as in* (13)*, the transition rate associated to a class-*1 *service phase transition from phase 2 to phase 3, in station 2, is*

$$f^n(\boldsymbol{x}, \boldsymbol{e}_{2,1,3} - \boldsymbol{e}_{2,1,2}) = T^{2,1}_{2,3} \frac{x_{2,1,2}}{x_2} \min\{S, x_2\}, \\ = \gamma \frac{x_{2,1,2}}{x_2} \min\{S, x_2\}.$$

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*

$$f^c(\boldsymbol{x}, \boldsymbol{e}_{1,2,1} - \boldsymbol{e}_{2,2,1}) = t^{2,2}_1 \alpha^{1,2}_1 P^{2,2}_{2,1} \frac{x_{2,2,1}}{x_2} \min\{S, x_2\}, \\ = \gamma_1 \frac{x_{2,2,1}}{x_2} \min\{S, x_2\},$$

since $\alpha^{1,2}_1 = 1$ 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

$$f^n(\boldsymbol{x}, \boldsymbol{e}_{2,1,3} - \boldsymbol{e}_{2,1,2}) = \gamma \frac{1}{3} 2, \\ f^c(\boldsymbol{x}, \boldsymbol{e}_{1,2,1} - \boldsymbol{e}_{2,2,1}) = \gamma_1 \frac{2}{3} 2,$$

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 $v \to \infty$, 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 $X^1(t)$ but with a population of vN jobs and vn_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 [14] to show that, when $v \to \infty$, 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 [27]. The state space of $X^v(t)$ is $\{\boldsymbol{x} \in \mathbb{N}^{\bar{m}} : \sum_{i=1}^M \sum_{r=1}^R \sum_{a=1}^{m^{i,r}} x_{i,r,a} = vN\}$, 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 $vn_i = c_i vN = c_i \sum_{h=1}^M \sum_{r=1}^R \sum_{d=1}^{m^{i,r}} x_{h,r,a}$, for any state $\boldsymbol{x}$ in the state space of $X^v(t)$.

The transition rates of $X^v(t)$ in state $\boldsymbol{x}$ associated to service completions can thus be written as

$$f^c_v(\boldsymbol{x}, i, j, r, s, a, b) \\ = v t^{i,r}_a \alpha^{j,s}_b P^{r,s}_{i,j} \frac{\frac{x_{i,r,a}}{v}}{\frac{x_i}{v}} \min\left\{ \frac{c_i}{v} \sum_{h=1}^M \sum_{r=1}^R \sum_{d=1}^{m^{i,r}} x_{h,r,a}, \frac{x_i}{v} \right\}, \\ = v f^c_1(\boldsymbol{x}/v, i, j, r, s, a, b) = v f^c(\boldsymbol{x}/v, i, j, r, s, a, b).$$

Similarly, $f^n_v(\boldsymbol{x}, i, r, a, b) = v f^n(\boldsymbol{x}/v, i, r, a, b)$. We thus obtain that the transition rates of the system $X^v(t)$ in state $\boldsymbol{x}$ can be written as v times the rates of the original system $X^1(t)$ in state $\boldsymbol{x}/v$. This property of the transition rates is called density-dependence. As a result, and provided that the rate functions $f^c(\boldsymbol{x}, i, j, r, s, a, b)$ and $f^n(\boldsymbol{x}, i, r, a, b)$ are continuous for all $\boldsymbol{x} \in \mathbb{R}^{\bar{m}}$, the sequence $\{X^v(t)\}_{v \in \mathbb{N}_+}$ is a *density-dependent* family of processes [14]. To show that these rate functions are Lipschitz continuous we notice that both $T^{i,r}_{a,b}$ and $t^{i,r}_a$ are bounded above by the largest exit rate in the sub-generator matrix $T^{i,r}$, which we assume to be finite, $\lambda_{i,r} = \max_a\{-T^{i,r}_{a,a}\} < \infty$. Therefore $f^c(\boldsymbol{x}, i, j, r, s, a, b) \leq \lambda_{i,r} g(x, i, r, a)$ and $f^n(\boldsymbol{x}, i, r, a, b) \leq \lambda_{i,r} g(x, i, r, a)$, 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 $g(\boldsymbol{x}, i, r, a)$ in* (2) *are Lipschitz continuous in $\boldsymbol{x} \in \mathbb{R}^{\bar{m}}$.*

Theorem 3.1 in [14] 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 $\boldsymbol{x} \in \mathbb{R}^{\bar{m}}$, let $F(\boldsymbol{x})$ be the drift of $X(t)$ in state $\boldsymbol{x}$, that is

$$F(\boldsymbol{x}) = \sum_{i,j=1}^{M} \sum_{r,s=1}^{R} \sum_{a,b=1}^{m^{i,r}} (\boldsymbol{e}_{j,s,b} - \boldsymbol{e}_{i,r,a}) f^c(\boldsymbol{x}, i, j, r, s, a, b) + \sum_{i=1}^{M} \sum_{r=1}^{R} \sum_{a,b=1}^{m^{i,r}} (\boldsymbol{e}_{i,r,b} - \boldsymbol{e}_{i,r,a}) f^n(\boldsymbol{x}, i, r, a, b), \quad (5)$$

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

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

and has initial state $\boldsymbol{x}_0$. Using the results in [14], 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 [28] 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 $\boldsymbol{x}(0)$ 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 $\bar{\boldsymbol{x}}(0)$ that is equal to $\boldsymbol{x}(0)$ 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 $P^{i,j}_{R+1,s} = P^{i,j}_{r,s}$, 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 $y(t) \in \{1, \ldots, M\}$ 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 cumulative distribution function (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

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

where $X_{i,R+1}(t)$ is the total number of tagged jobs in station i, i.e., $X_{i,R+1}(t) = \sum_{a=1}^{m^{i,R+1}} X_{i,R+1,a}(t)$. As in [28], we make use of the fluid solution $\boldsymbol{x}(t)$ to approximate the expected value of $X(t)$, such that

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

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

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

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

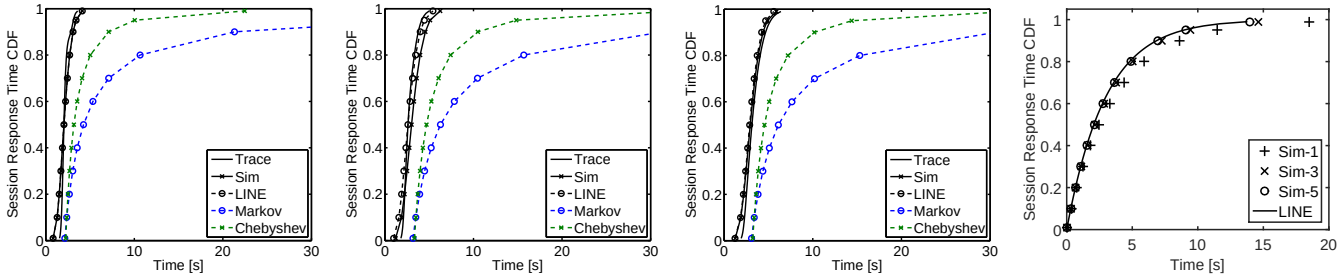
In the previous discussion we assumed a given initial fluid $\boldsymbol{x}(0)$. To approximate the steady-state response time distribution we can let the ODE evolve until the rate of change vanishes and use the resulting vector as the required $\boldsymbol{x}(0)$. Although we have not found an instance 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 $\boldsymbol{x}(T)$ 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 OFBiz[2], an open-source enterprise framework, which includes an e-commerce application that we deploy on the cloud using Amazon EC2[3]. Once deployed, we use the OFBench tool [29] 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 [30] to estimate the *mean* processing times. In addition, we estimate the squared coefficient of variation (SCV) of the processing times using the BL method in [30]. That is, we use BL to obtain estimates for the service demand samples from response time samples.

[2]http://ofbiz.apache.org/
[3]http://aws.amazon.com/



(a) RT CDF - N = 50 (b) RT CDF - N = 70 (c) RT CDF - N = 90 (d) RT CDF - N=130

Fig. 4. Session Response Time CDF

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

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

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 [30] for more details). As illustrated in Table II, there are 8 different request types 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 [31] 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 [32]. Figures 4(a) to (c) depict the results for the cases with 50, 70, and 90 users, focusing on the session response time (RT) 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 [19], [20] 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 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

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

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 $v_e = -V_e \boldsymbol{e}$ be the vector of absorption rates, where $\boldsymbol{e}$ 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

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

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 $\pi Q = 0$, $\pi \boldsymbol{e} = 1$, 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 [33], 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 [33], 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 $\boldsymbol{x}_e(t)$ when the RE is in stage e. This means that the system starts in state $\boldsymbol{x}_e(0)$ 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 $\boldsymbol{x}_h(0)$ and evolves with a different drift $F_h(\cdot)$ until a new transition occurs. The initial state $\boldsymbol{x}_h(0)$ is defined from the terminal state $\boldsymbol{x}_e(T)$ by applying a *Reset matrix* $R_{e,h}$, as $\boldsymbol{x}_h(0) = \boldsymbol{x}_e(T) R_{e,h}$. 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

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

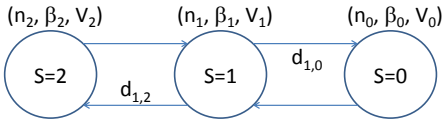


Fig. 5. Random environment example

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 [33] 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 [33]. We also combine it with the LINE model, which is more general than the single-class first-come first-served model in [33]. A 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 $\boldsymbol{x}_e^0(0)$ be the initial system state in stage e, where the exponent $k = 0$ counts the number of environmental transitions that have occurred. We obtain $\boldsymbol{x}_e^0(t)$ as the solution of (6), with drift $F_e(\cdot)$ and initial state $\boldsymbol{x}_e^0(0)$, 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 $\int_0^\infty \boldsymbol{x}_e^0(t)\beta_e \exp(-V_e t) v_e dt$. The initial state in stage h after the first environmental transition is thus given by

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

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

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

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 $\int_0^\infty \boldsymbol{x}_e^{\hat{k}}(t)\beta_e \exp(V_e t) v_e dt$.

VII. Capturing Cloud Variability with Random Environments

Cloud computing offers great flexibility to deploy software applications, as the user pays only for the resources used. This flexibility is achieved thanks to virtualized environments, where applications run in virtual machines (VMs) that share the same hardware. Although many efforts are made to prevent interference across VMs, it is unavoidable that high resource consumption by one VM affects neighboring VMs. In this section we make use of the RE model in LINE to capture this variability and its impact on the response times offered by an application deployed on Cloud resources. We now describe the experimental setup and compare the response times predicted by LINE against measurements and alternative methods.

A. Experimental Setup

To emulate the variability observed in the Cloud we make use of the OFBiz application described in Section V, which we deploy on our private Cloud, running Open Nebula version 4.10 [34]. As in Section V we deploy both the OFBiz application and database servers in a single VM (4 virtual cores, 8GB memory) running under a KVM hypervisor. Each virtual core is pinned on a physical core of an Intel(R) Xeon(R) CPU E5-2470 running at 2.30GHz. The workload is generated with the OFBench tool [29] using clients running on separate VMs, deployed on separate servers and connected via a Gigabit Ethernet switch to the OFBiz VM. To emulate Cloud variability we employ the stress utility at the hypervisor level, configured to saturate all hypervisor CPUs for a controlled amount of time. During this period, the OFBiz VM therefore experiences significant CPU steal due to time-sharing at the hyper-visor level with the workload generated by stress. We activate stress during periods of 2 minutes, which are followed by 5-minute periods without contention. This pattern repeats during the whole duration of each experiment, which we set to one hour.

B. Results

We execute the above experiments with a number of $N = 5, 10, 15, 20$ clients. Table VI depicts the average CPU utilization generated by these clients under low and high contention separately. Note that this is the average utilization, whereas the maximum instantaneous utilization can reach 99% under high contention. Also, this setup is slightly different from the one in Section V, as we use a smaller VM for the server, and the

TABLE VI
OBSERVED UTILIZATION (%) AND CHECKLOGIN RT (SEC)

Contention	N	5	10	15	20
Low	Avg. Util.	17.4	31.8	16.5	47.5
	Mean RT	0.187	0.203	0.245	0.266
	RT95	0.342	0.389	0.526	0.595
High	Avg. Util.	34.6	51.8	67.1	69.3
	Mean RT	0.199	0.215	0.295	0.297
	RT95	0.363	0.434	0.671	0.647

client generates more requests per session, causing a high CPU utilization with fewer clients. To identify periods of contention we use the *K-means* clustering method [35] on readings of the CPU utilization, which are clustered in two groups representing high and low contention. Based on this clustering and the timestamps of the CPU utilization measurements, we determine the time periods of low and high contention. To illustrate the impact of the contention introduced, Table VI depicts the measured RT for one request type, CheckLogin, during the identified periods of low and high contention. We observe a significant increase, between 6% and 28%, in both mean RT and RT95, when contention is introduced. As in Section V, we use the BL method [30] to obtain samples of the service demand, which we then use to obtain PH representations of the processing-time distributions. From the execution times sampled during the different contention periods we estimate the processing time distribution for each of the request classes in Table II, separately for periods of low and high contention. Using these distributions we introduce a LINE model similar to the one in Section V, but with an RE made of two stages that represent periods of low and high contention, respectively. The model uses the processing time distribution estimated for each stage and request type.

Tables VII and VIII depict response-time mean and tail percentiles measured and obtained with LINE, for the two extreme cases with $N = 5$ and 20 clients. Note that we focus on the *overall* session response time, which is obtained by putting together all observations, with and without contention. As in the case without RE, we compare against the percentiles obtained with the Markov and Chebyshev inequalities. In this case we use the *measured* mean and variance of the response times to obtain these percentiles, thus the only source of error is the use of the inequality. Clearly, LINE offers response times that closely follow the measurements, with relative errors in the 5%-25% range, that typically underestimate the observed values. Instead, both inequalities produce results that largely overestimate (by at least one order of magnitude) the response time percentiles. Similar results are observed in the other setups considered.

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,

TABLE VII
SESSION RT MEAN AND PERCENTILES (SEC) - $N = 5$

	Mean	RT90	RT95	RT99
Trace	3.83	4.70	5.12	6.34
LINE	3.61	4.45	4.74	5.34
Markov	-	38.25	76.50	382.51
Chebyshev	-	13.26	18.76	41.95

TABLE VIII
SESSION RT MEAN AND PERCENTILES (SEC) - $N = 20$

	Mean	RT90	RT95	RT99
Trace	5.07	6.53	7.18	8.89
LINE	4.03	4.98	5.40	6.43
Markov	-	50.66	101.31	506.57
Chebyshev	-	16.53	23.37	52.26

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 $\{10, 20, \ldots, 200\}$ 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 IX 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 add an RE with two stages, similar to the one in Section VII. We consider two cases for the RE holding times: fast and slow. The slow RE has mean holding times of 600*s* and 60*s* for the two stages, while the fast RE has mean holding times of only 60*s* and 6*s* in these stages. We observe in Table IX that the inclusion of REs increases the computation times, although they remain overall small, and that the increase depends on the stage holding times.

TABLE IX
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

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 enables 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.

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 [36], 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 6, 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 X. 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.

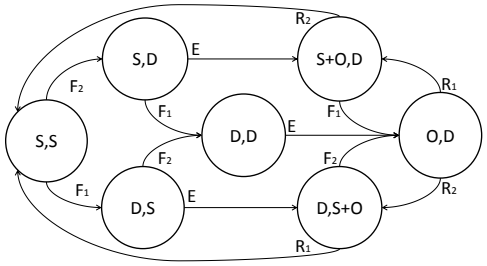


Fig. 6. Random environment for application deployed on two zones

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.

A. What-if analysis

We illustrate this deployment scenario with a simple model of the open-source OFBiz e-commerce application [29]. 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 7. 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.

TABLE X
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	
$m^S_{(i,j)}$	Number of spot instances in zone i for component j
m^O_j	Number of on-demand instances for component j

TABLE XI
PROVISIONING AND FAILURE TIMES IN THE SPOT INSTANCE MODEL [36]

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

Notice that a spot instance provisioning model has also been recently considered in [37], which is a preliminary follow-up application of the present work relying on the LINE tool to set up 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 [36]. The specific times considered are summarized in Table XI. While the mean times to recover spot and on-demand instances T_E and T_R are defined according to [36], the mean time to lose 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 [37]. The authors of [36] 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 practical advantage of the features offered by the LINE solver, which are not available in ordinary queueing network solvers.

Figure 8 depicts the RT 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 [37]. For instance, a bid price of $ 0.534 can achieve a 90% availability for *m1.xlarge* spot instances [37].

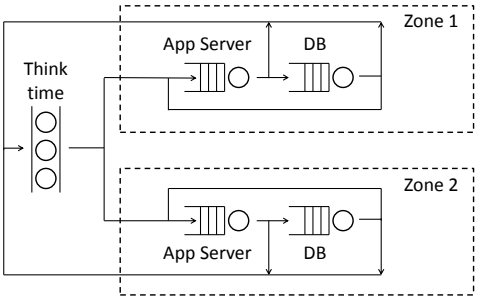
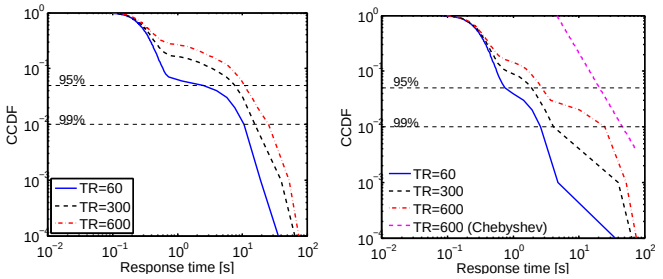


Fig. 7. Model of the 2-zone deployment



(a) $m^S_{(i,j)}=2$, $m^O_j=1$ (b) $m^S_{(i,j)}=3$, $m^O_j=1$

Fig. 8. Response time CCDF under changes of T_R, the time to recover spot instances

In Figure 8(a) we allocate $m^S_{(i,j)}=2$ spot instances in each zone for each component, and $m^O_j=1$ 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 8(b), increasing the number of spot instances $m^S_{(i,j)}$ 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 8(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.

B. Optimal provisioning with LINE

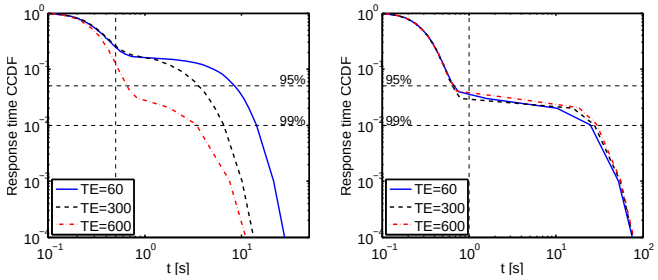
Based on LINE, we can set up an optimal-provisioning problem to determine the number of instances $m^S_{(i,j)}$ and m^O_j required to achieve a certain SLO at the minimum cost. Let the vectors m^S and m^O hold the $m^S_{(i,j)}$ and m^O_j variables, and define C^S_i and C as the leasing costs of spot 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

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

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

Figure 9(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 0.5 s 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.



(a) Mean (b) 95th percentile

Fig. 9. Response times under optimal provisioning. Various T_E

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 $R^{\max}_x$. 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

$$R_x = \text{LINE}(m^S, m^O), \quad R_x \le R^{\max}_x. \tag{8}$$

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 9(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 where requested. Thus, to ensure the constraint of 1 s 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 9(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 1 s 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 and 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

Before introducing the proof we recall that a function $f : Y \to Z$ is Lipschitz continuous on $I \subset Y$ if for every pair of points $x, y \in I$ there exists a non-negative constant L such that $d_Z(f(x), f(y)) \leq L d_Y(x, y)$, where Y and Z are metric spaces and d_Y and d_Z their corresponding metrics. We are interested in the function $g : \mathbb{R}^m \to \mathbb{R}$ defined in (2) and use the 1-norm as metric, which in the case of $\mathbb{R}$ is simply the absolute value. For two points $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^{\bar{m}}$ we consider four cases.

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

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

where the first inequality follows as both $1/x_i$ and $1/y_i$ are smaller than $1/n_i$, such that $1/(x_i y_i) < 1/n_i^2$. The second inequality follows from the triangle inequality and the fact that both y_i and $y_{i,r,a}$ are bounded above by the number of jobs N. The last inequality follows as with the 1-norm both $|x_i - y_i|$ and $|y_{i,r,a} - x_{i,r,a}|$ are smaller than $|\boldsymbol{y} - \boldsymbol{x}|$. For $|y_{i,r,a} - x_{i,r,a}|$ this is immediate as this is one of the differences considered in $|\boldsymbol{y} - \boldsymbol{x}|$. For $|x_i - y_i|$ we write it as

$$\begin{aligned}
&\left| \sum_{r=1}^{R} \sum_{a=1}^{m^{i,r}} x_{i,r,a} - \sum_{r=1}^{R} \sum_{a=1}^{m^{i,r}} y_{i,r,a} \right| = \left| \sum_{r=1}^{R} \sum_{a=1}^{m^{i,r}} (x_{i,r,a} - y_{i,r,a}) \right| \\
&\leq \sum_{r=1}^{R} \sum_{a=1}^{m^{i,r}} |x_{i,r,a} - y_{i,r,a}| \leq \sum_{i=1}^{M} \sum_{r=1}^{R} \sum_{a=1}^{m^{i,r}} |x_{i,r,a} - y_{i,r,a}|,
\end{aligned}$$

where the first inequality follows from the triangle inequality and the last expression is equal to $|\boldsymbol{y} - \boldsymbol{x}|$, thus showing that $|x_i - y_i| < |\boldsymbol{y} - \boldsymbol{x}|$. The remaining cases follow similar arguments, which we omit for brevity.

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

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

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

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

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

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

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

B. Proof of Theorem 1

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

$$\lim_{v \to \infty} \mathrm{P} \left(\sup_{s \leq t} \left| \frac{1}{v} X^v(s) - \boldsymbol{x}(s) \right| > \delta \right) = 0. \tag{9}$$

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

$$|F(\boldsymbol{y}) - F(\boldsymbol{x})| < M_E \left| \boldsymbol{y} - \boldsymbol{x} \right|, \ \boldsymbol{x}, \boldsymbol{y} \in E, \tag{10}$$

$$\begin{aligned}
&\sup_{\boldsymbol{x} \in E} \left\{ \sum_{i,j=1}^{M} \sum_{r,s=1}^{R} \sum_{a,b=1}^{m^{i,r}} |\boldsymbol{e}_{j,s,b} - \boldsymbol{e}_{i,r,a}| \, f^c(\boldsymbol{x}, i, j, r, s, a, b) \right. \\
&\left. + \sum_{i=1}^{M} \sum_{r=1}^{R} \sum_{a,b=1}^{m^{i,r}} |\boldsymbol{e}_{i,r,b} - \boldsymbol{e}_{i,r,a}| \, f^n(\boldsymbol{x}, i, r, a, b) \right\} < \infty,
\end{aligned} \tag{11}$$

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

where $S^1(d)$ is the set $\{(i, j, r, s, a, b) : |\boldsymbol{e}_{j,s,b} - \boldsymbol{e}_{i,r,a}| > d\}$, $S^2(d)$ is similarly defined, 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 [14, Theorem 3.1] by showing that the sequence $\{X^v(t)\}_{v \in \mathbb{N}_+}$ verifies the conditions conditions (10), (11), and (12). As the QN model $X(t)$ is closed, the

entries $X_{i,r,a}(t)$ are bounded above by N, a condition that also holds for every $X^v(t)/v$ as well as for $\boldsymbol{x}(t)$. As a result the set E can be chosen as the smallest open set that contains the set $\{\boldsymbol{x} \in \mathbb{R}^{\bar{m}} : \boldsymbol{x} \geq 0, \sum_{i=1}^{M}\sum_{r=1}^{R}\sum_{a=1}^{m^{i,r}} x_{i,r,a} = N\}$, as the sample paths of $\boldsymbol{x}(t)$ never leave this set.

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

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

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

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

where $\hat{m}$ summarizes the summation terms, and therefore condition (10) is satisfied.

Condition (11) can be readily verified by noticing that $|\boldsymbol{e}_{j,s,b} - \boldsymbol{e}_{i,r,a}| \leq 2$, $|\boldsymbol{e}_{i,r,b} - \boldsymbol{e}_{i,r,a}| \leq 2$, $f^1(\boldsymbol{x}, i, j, r, s, a, b) \leq \lambda^{i,r} n_i$, and $f^n(\boldsymbol{x}, i, r, a, b) \leq \lambda^{i,r} n_i$. 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$(n,\ \gamma)$ 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$(3,\ \gamma)$ distribution can be represented as $m = 3$,

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

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 $m = 2$,

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

The flexibility of PH distributions is also reflected in a number of fitting algorithms [39]–[42] to obtain a PH representation from empirical data. These algorithms are available as part of software tools such as [43]–[45], among others.

ACKNOWLEDGEMENT

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 from the EPSRC grant EP/M009211/1 (OptiMAM). This publication reflects only the authors' view and the European Commission is not responsible for any use that may be made of the information it contains. The research of Juan F. Pérez has been supported by the ARC Centre of Excellence for Mathematical and Statistical Frontiers (ACEMS), and was partly conducted while the first author was at Imperial College London and The University of Melbourne, and finalized while at Universidad del Rosario. Data and software referenced in this paper are available at https://doi.org/10.5281/zenodo.203014 and https://doi.org/10.5281/zenodo.203029, released under the CC-BY 4.0 and BSD 3-Clause licenses, respectively. The authors wish to thank Weikun Wang for processing the OFBiz data traces used for validation.

REFERENCES

[1] 451 Research, "Cloud computing 'as a service' marketplace forecast to grow 3x through 2019," https://451research.com/report-short?entityId=87624&referrer=marketing, 2015, accessed in Dec 2016.
[2] "Amazon EC2 Spot Instances," https://aws.amazon.com/ec2/spot/.
[3] 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, pp. 148–161, 2009.
[4] 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.
[5] M. Tribastone, P. Mayer, and M. Wirsing, "Performance prediction of service-oriented systems with layered queueing networks," in *ISoLA*, 2010.
[6] S. Becker, H. Koziolek, and R. Reussner, "Model-based performance prediction with the palladio component model," in *WOSP*, 2007.
[7] H. Koziolek and R. Reussner, "A model transformation from the palladio component model to layered queueing networks," in *Performance Evaluation: Metrics, Models and Benchmarks*. Springer, 2008.
[8] X. Wu and M. Woodside, "Performance modeling from software components," in *WOSP*, 2004.
[9] 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.
[10] O. Das and C. M. Woodside, *Architecting Dependable Systems II*. Springer-Verlag, 2004, ch. Dependability Modeling of Self-Healing Client-Server Applications.

[11] G. Casale, M. Tribastone, and P. G. Harrison, "Blending randomness in closed queueing network models," *Perf. Eval.*, vol. 82, pp. 15–38, 2014.

[12] 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 queuing and historical performance models," *Simulation Modelling Practice and Theory*, vol. 19, pp. 1479–1495, 2011.

[13] J. F. Pérez and G. Casale, "Assessing SLA compliance from palladio component models," in *SYNASC*, 2013.

[14] T. G. Kurtz, "Solutions of ordinary differential equations as limits of pure jump Markov processes," *J. Appl. Probab.*, vol. 7, pp. 49–58, 1970.

[15] M. Tribastone, "A fluid model for layered queueing networks," *IEEE Trans. Softw. Eng.*, vol. 39, pp. 744–756, 2013.

[16] P. Harrison and W. Knottenbelt, "Passage time distributions in large markov chains," *ACM Perf. Eval. Rev.*, vol. 30, pp. 77–85, 2002.

[17] R. Bellman, *Dynamic programming*. Princeton University, 1957.

[18] G. Casale, "Approximating passage time distributions in queueing models by bayesian expansion," *Perf. Eval.*, vol. 67, pp. 1076–1091, 2010.

[19] D. Ardagna, C. Ghezzi, and R. Mirandola, "Rethinking the use of models in software architecture," in *QoSA*, 2008.

[20] I. Cunha, J. Almeida, V. Almeida, and M. Santos, "Self-adaptive capacity management for multi-tier virtualized environments," in *IFIP/IEEE IM*, 2007.

[21] "LINE website," http://line-solver.sourceforge.net, 2016.

[22] A. Faisal, D. Petriu, and M. Woodside, "Network latency impact on performance of software deployed across multiple clouds," in *CASCON*, 2013.

[23] 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 *ICAC*, 2008.

[24] S. Asmussen, *Applied probability and queues*. Springer, 2003.

[25] 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.

[26] M. Reiser, "A queueing network analysis of computer communication networks with window flow control," *IEEE Trans. Comm.*, vol. 27, pp. 1199–1209, 1979.

[27] L. Bortolussi, J. Hillston, D. Latella, and M. Massinkc, "Continuous approximation of collective system behaviour: A tutorial," *Perf Eval*, vol. 70, pp. 317–349, 2013.

[28] R. A. Hayden, A. Stefanek, and J. T. Bradley, "Fluid computation of passage-time distributions in large Markov models," *Theor. Comput. Sci.*, vol. 413, pp. 106–141, 2012.

[29] J. Moschetta and G.Casale., "OFBench: an Enterprise Application Benchmark for Cloud Resource Management Studies," in *MICAS*, 2012.

[30] 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.

[31] W. Whitt, "Approximating a point process by a renewal process, i: Two basic methods," *Operations Research*, vol. 30, pp. 125–147, 1982.

[32] M. Bertoli, G. Casale, and G. Serazzi, "JMT: performance engineering tools for system modeling," *ACM Perf. Eval. Rev.*, vol. 36, pp. 10–15, 2009.

[33] G. Casale and M. Tribastone, "Fluid analysis of queueing in two-stage random environments," in *QEST*, 2011.

[34] "Open Nebula," http://opennebula.org/.

[35] R. Johnson and D. Wichern, *Applied Multivariate Statistical Analysis*. Prentice-Hall, 1998.

[36] M. Mao and M. Humphrey, "A performance study on the VM startup time in the cloud," in *IEEE CLOUD*, 2012.

[37] D. Dubois and G. Casale, "Autonomic provisioning and application mapping on spot cloud resources," in *IEEE ICCAC*, 2015.

[38] S. Kirkpatrick, C. D. Gelatt, Jr, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671–680, 1983.

[39] S. Asmussen, O. Nerman, and M. Olsson, "Fitting Phase Type distributions via the EM algorithm," *Scan. J. Stat.*, vol. 23, pp. 419,441, 1996.

[40] A. Bobbio, A. Horvath, and M. Telek, "Matching three moments with minimal acyclic Phase Type distributions," *Stoch. Model.*, vol. 21, pp. 303–326, 2005.

[41] H. Okamura, T. Dohi, and K. S. Trivedi, "A refined EM algorithm for PH distributions," *Perf. Eval.*, vol. 68, pp. 938–954, 2011.

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

[43] P. Reinecke, T. Krauss, and K. Wolter, "Hyperstar: Phase-type fitting made easy," in *QEST*, 2012.

[44] A. Bobbio, A. Horvath, and M. Telek, "PhFit: a general phase-type fitting tool," in *DSN*, 2002.

[45] J. F. Pérez and G. Riaño, "jPhase: An object-oriented tool for modeling phase-type distributions," in *SMCTools*, 2006.

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

Giuliano Casale received the Ph.D. degree in Information Technology from Politecnico di Milano, Italy, in 2006. In 2010 he joined the Department of Computing at Imperial College London, UK, where is currently a Senior Lecturer. Previously, he worked as a research staff member at SAP Research UK. He teaches and does research in performance engineering, cloud computing, and operations research. He has served as co-chair for several conferences in the area of performance engineering, including SIGMETRICS/Performance, MASCOTS, ICAC and ICPE. He is member of the IFIP WG 7.3 group on Computer Performance Analysis and serves in the ACM SIGMETRICS Board of Directors.