Modelling Exogenous Variability in Cloud Deployments

Giuliano Casale[1]
g.casale@imperial.ac.uk

Mirco Tribastone[2]
tribastone@pst.ifi.lmu.de

[1] : Imperial College London, London, United Kingdom
[2] : Ludwig-Maximilians-Universität, Munich, Germany

ABSTRACT

Describing exogenous variability in the resources used by a cloud application leads to stochastic performance models that are difficult to solve. In this paper, we describe the *blending* algorithm, a novel approximation for queueing network models immersed in a random environment. Random environments are Markov chain-based descriptions of time-varying operational conditions that evolve independently of the system state, therefore they are natural descriptors for exogenous variability in a cloud deployment. The algorithm adopts the principle of solving a separate transient-analysis subproblem for each state of the random environment. Each subproblem is then approximated by a system of ordinary differential equations formulated according to a fluid limit theorem, making the approach scalable and computationally inexpensive. A validation study on several hundred models shows that blending can save up to two orders of magnitude of computational time compared to simulation, enabling efficient exploration of a decision space, which is useful in particular at design-time.

1. INTRODUCTION

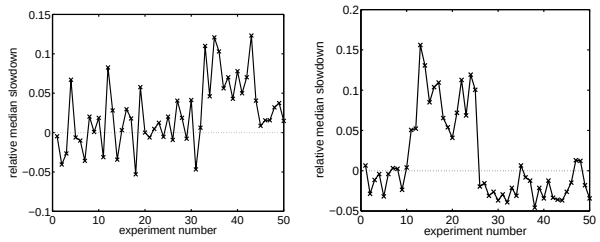
Cloud applications run in shared environments where performance and availability of computational resources may change over time due to contention from other users. These uncertainties make it difficult to reason at design-time on the performance and availability that an application will offer once deployed. Hence, a current research challenge is to identify simple modelling techniques that can effectively account for such variability in performance and availability predictions. In this paper, we propose a methodology for efficient analysis of queueing-based performance models that include a description of exogenous variability, modelled by a continuous-time Markov chain.

Consider, for instance, a decision problem for a multi-tier web application deployed (or to be deployed) on a cloud platform. A common performance modelling approach is to represent each software server or virtual machine as a queue and evaluate what-if scenarios by solving the resulting stochastic queueing network model using techniques such as approximate mean value analysis [11]. However, upon including a description of the surrounding random environment variability, it is common for the resulting model to become intractable by exact algorithms. Decomposition methods tackle this problem by assuming a marked separation between the time constants of the system (e.g., the request service times) and those of the random environment (e.g., mean time to failure of a resource) [13]. Under these assumptions, one can obtain a solution that is asymptotically exact as the random environment events become increasingly less frequent [10]. However, several classes of applications deployed on the cloud do *not* satisfy these assumptions. These include, for example, data-intensive applications (e.g., MapReduce) where request service times may last hours and thus resource fluctuations happen repeatedly throughout a request execution period [5]. Similarly, startup times of virtual machines may vary from tens of seconds to several minutes, thus they can introduce significant interference with runtime execution when scaling resources [4].

To cope with the lack of robustness and scalability of existing approximation methods, we introduce the *blending* algorithm, a simple approximation for queueing networks in random environments. Blending estimates mean performance metrics of the model by evaluating a different transient analysis subproblems for each state of the random environment. Following this principle, the stationary distribution of the original model is obtained as a mixture of the transient behaviors of each sub-model in isolation. These solutions are related to each other by averaging the initial conditions provided to the transient analysis subproblems, which is a significant difference compared to simulation as we explain in Section 6. Due to the lack of tractable exact solutions for the transient probability distribution of a queueing network, we use an approximation based on ordinary differential equations (ODEs) in the sense of Kurtz [14]. Crucially, under this formulation the size of the problem does not depend on the number of requests, thus it does not suffer from the state space explosion problem; it only grows with linear complexity in the number of stations. Furthermore, the approximation is provably more accurate for increasing job population sizes. As a result, blending can analyze problems that are prohibitive to solve with discrete-state models due to the state space explosion problem.

Below we summarize the other major contributions of the present paper, also with respect to an earlier version of the blending algorithm which was proposed in [6] for exponential queueing networks in two-stage random environments:

- We formulate a generalized blending algorithm to study random environments with an *arbitrary* number of states. In the case of a two-stage environment and exponential queues, this algorithm specializes to the one in [6].
- We formally characterize the connection between the



(a) m1.medium (b) c1.medium

Figure 1: Example of exogenous variability in response times for two *m1.medium* and *c1.medium* VMs running Apache OFBiz [7]. Each point describes a 10-minutes experiment.

blending algorithm and the exact solution of the model under study.

- We generalize the class of queueing networks supported by the method by allowing Coxian service-time distributions, which enable effective approximations of empirical service time distributions.

The paper is organized as follows. Section 2 illustrates exogenous variability in a real cloud deployment using an experimental dataset collected on Amazon EC2 and shows limitations of decomposition approaches. Our reference model is given in Section 3 and in Section 4, respectively. We define a fluid approximation for the considered queueing models in Section 5 and overview its main properties. This is used in Section 6 to develop the blending algorithm. A numerical validation of the blending algorithm is given in Section 7. Conclusions are given in Section 8.

2. MOTIVATION

2.1 Random Environment Variability

We discuss the results of an experimental campaign that we have performed on Amazon EC2 Ireland using the demo e-commerce component of Apache OFBiz, an open source enterprise resource planning framework [3]. The aim of the experimental campaign is to illustrate temporal correlation between performance degradations that may be attributed to exogenous factors arising in the cloud deployment environment. The OFBiz demo application is similar to TPC-W [8] and RUBiS [9], but the framework is more complex since it features several hundreds Java classes and relies on technologies that are common in production systems, such as Groovy dynamic scripting, a template engine, and AJAX on client side.

We have stressed the application using the OFBench suite proposed in [7]. The workload consists of a CPU-bound mix of requests. The OFBiz application is shipped in its basic form with an embedded Geronimo server [2] and an embedded Derby in-memory database [1], which we run together inside a single VM. We have deployed OFBiz on several tens of *c1.medium* and *m1.medium* VMs on Amazon EC2 Ireland in the *eu-west-1a* availability zone. Each VM runs an independent copy of the application. Simultaneously, we have run the clients, one for each application, inside VMs instantiated in the same availability zone. Each experiment uses a single Firefox client coupled with a closed-loop workload generator. The time between submission of successive requests from a client is exponentially distributed with a mean of 1 second. Since there is a single client, we use a short warmup period of 2 minutes. The benchmark discards the sessions executed by the client during the warmup phase of each experiment, since these suffer high response times due to cache misses. After the warmup, the experiment continues at steady-state for 10 minutes before the server is shut down and restarted for the following experiment. Each restart takes approximately 15 minutes to regenerate the database and boot the software. Thus, successive experiments inside each VMs are independent. Each experiment is repeated 50 times, for a total of about 15 hours of experiments. Notice that we fix in all experiments the random number generator seed to replicate the same sequence of requests, including the client think times, the sequence of request types, and the same random data.

Figure 1 depicts the experimental results. The metric plot is the average response time of the requests experienced by the client, hence it includes also network latency. However, the bandwidth between server and client is normally above 850-900 MBit/sec, as recorded via *iperf* at the beginning of each experiment, and it does not change significantly throughout the experimental campaign. The metric shown in the figures is defined as follows. We consider the response times of the page with the highest hit rate, i.e., the homepage. Then, we compute the median of its response times across all the experiments and use this a baseline to determine the relative median slowdown within each experiment. The results in the figure show that the deviation from this baseline normally does not exceed 10-15%, but it may last a variable amount of time, from minutes to hours. Since the experiments are independent and the network is largely overprovisioned, it seems reasonable to attribute this slowdown to the VM itself. As a further indication of this, we have studied the same metric for other pages and found nearly identical trends. Notice that our results are quite consistent with the ones reported in [5], which indicate for Amazon EC2 Ireland a CPU variability of approximately 7-8% at 1 hour timescales. However, compared to the study in [5], Figure 1 provides additional information about the temporal correlation of the degradations, suggesting that a state-based model could be useful to describe such perturbations.

2.2 Limitation of Decomposition Approaches

We now show the limitations of decomposition modelling approaches in describing fluctuations in resource capacities such as those illustrated in the previous example. For the sake of illustration, we consider a performance model of a web application represented as a network of queues. This is also the class of models that we focus on in the rest of this paper, even though we expect that the proposed approach is sufficiently general to be extended to other model classes. We consider a queueing network model for a very basic application deployment involving two VMs running in parallel, having $S_1 = S_2 = 4$ cores each, and a client population of $N = 100$ users that issues requests with an average think time of $Z = 7s$ in between successive request arrivals by the same client. A request is randomly sent to any of the two VMs. We assume a two-state random environment that produces fluctuations similar to Figure 1, i.e., for 20% of the time VM2 offers a degraded performance. Service demands of requests are exponentially distributed, operating in the

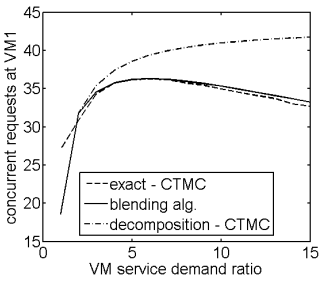


Figure 2: Limitation of classic decomposition approaches. The figure shows in the x-axis the ratio of request service demands at two VMs for a period of degraded performance lasting 20% of the observation time, similarly to Figure 2.

normal regime with the same mean as in the experiments in Figure 1, i.e., 850ms. During the performance degradation, the demand at VM2 increases by a constant factor f. Our what-if analysis consists in studying the system performance as a function of the f degradation factor.

Figure 2 illustrates the predictions. The exact solution is obtained by formulating this model as a continuous-time Markov chain (CTMCs) and solving it directly by numerical methods. The decomposition approach, instead, solves independently of each other the two CTMCs obtained by conditioning on the random environment state being the normal regime or the degraded regime. Finally, the blending algorithm anticipates the technique developed in the next section. As we see from the figure, decomposition is unable to correctly characterize this model as the degradation factor increases. This is because the larger the degradation, the more the model will exhibit large and frequent fluctuations of performance metrics over time, which is an event difficult to model by decomposition. Conversely, the method we propose in this paper is more appropriate to evaluate this what-if scenario, besides providing an overall more scalable algorithm than CTMC-based techniques. In the next sections, we provide details about our reference models and introduce formally the blending algorithm, whose computationally properties are finally evaluated in Section 7 against simulation.

3. MODEL AND NOTATION

In the sequel, we define *state* the current condition of the queueing network model that represents, e.g., a cloud application, as opposed to the term *stage*, which instead describes the condition of the random environment, e.g., the cloud deployment or a service offered by the platform.

3.1 Random Environment

The class of random environments considered in this paper evolves according to a continuous-time Markov chain, jumping between stages, and specified by the following set of parameters:

- E: number of stages;
- $e, h = 1, \ldots, E$: stage indexes;
- α_{eh}: rate of jump from stage e to stage h;
- $\alpha_e = \sum_{h \neq e} \alpha_{eh}$: total outgoing rate from stage e;
- $p_{eh} = \alpha_{eh}/\alpha_e$: jump probability from stage e to h.

Consider for example the dataset shown in Figure 1. Upon defining a queueing model for the running application, we can assume to describe request service times by the response times obtained in the single-user experiments in Figure 1. The histogram of the mean service times across the experiments can be decomposed into E bins to describe the effect of the random environment on the mean service times. The probability p_{eh} is then the probability that a period with mean service times characterized by bin e is followed by a period with mean service times characterized by bin h; the mean duration of the periods associated to bin e and h is α_e^{-1}, respectively.

3.2 System Model

We denote by M the number of queueing stations and by N the number of circulating requests in the model; both values are assumed invariant across stages. All stations have unbounded buffer sizes. Service time processes are independent and identically distributed random variables following a Coxian distribution, which is a popular parametric model including exponential, hyper-exponential, and Erlang distributions as special cases [13]. We shall refer to the states of Coxian service processes as *phases*.

When the random environment is in stage e, the queueing network is parameterized as follows:

- $i, j = 1, \ldots, M$: station indexes;
- $r^e_{i,j}$: routing probability from station i to station j;
- S^e_i: number of servers at station i;
- K^e_i: number of phases in the Coxian process at i;
- $k \equiv k(i) = 1, \ldots, K^e_i$: phase index for station i;
- $\mu^e_{i,k}$: service rate in phase k at station i;
- $\phi^e_{i,k}$: probability for requests at i to complete after service phase k.

We now further specify how stations are affected by random environment transitions. In our methodology we assume that upon stage change, each station state is altered according to a reset rule. That is, for a jump from stage e to h, we define a mapping matrix $\boldsymbol{R}_{eh}$, which is 1 in element (i, j) if and only the ith state in stage e is mapped to the jth state in stage h, 0 otherwise. Note that $\boldsymbol{R}_{eh}$ is in general rectangular and that multiple states i can jump to the same state j. Also we assume $\boldsymbol{R}_{eh}$ to be such that there is no loss or generation of mass upon stage transition and that the number of stations and servers does not change with the active stage. Throughout this paper, we illustrate this rule by considering the mapping where the Coxian service process of the station is restarted upon an environment stage change, meaning that all requests in service are instantaneously moved back to phase 1 at that station. Other reset rules are possible, however this has the advantage that the conditional distribution given the current state is always known, but it may overestimate the real service times when the stage jump rate is large.

3.3 Continuous-Time Markov Chain

From the above definitions, the queueing network model may be described by a continuous-time Markov chain (CTMC) having state vector $(\boldsymbol{n}, e)$, where $e = 1, \ldots, E$ is the current stage of the random environment. The distribution of requests across stations is described by $\boldsymbol{n} = (\boldsymbol{n}_1, \boldsymbol{n}_2, \ldots, \boldsymbol{n}_M)$,

$\boldsymbol{n}_i = (x_i, s_{i,1}, \ldots, s_{i,K_i^e})$ being the state vector of station i, where:[1]

- $s_{i,k}$ is the number of servers that are currently processing requests in phase k, thus $\sum_{k=1}^{K_i^e} s_{i,k} = S_i^e$. If a server is idle, it is counted within $s_{i,1}$ together with the servers processing requests in phase 1. Note that for a delay station $s_{i,k} = +\infty$.[2]

- $x_i = q_i + n_{i,1}$ is the sum of the number of requests q_i waiting in the queue buffer, and the number of requests $n_{i,1}$ that are receiving service in phase 1. This representation provides advantages for the fluid analysis developed in Section 5.

The CTMC defined on the proposed state space may be specified using a collection of *transition rate functions* which give, for a generic state $(\boldsymbol{n}, e)$, all transition rates at which the process moves to a different state $(\boldsymbol{n}', h)$. These are given in Table 1. The transitions f_1^{dep} (resp. f_k^{dep}) describe the rates of job completions at station i at the 1st (resp. kth) Coxian phase that are followed by a departure to station j; the transitions f_1^{ph} (resp. f_k^{ph}) describe the rate of requests advancement from the 1st to the 2nd Coxian phase at station i; f^{env} describe the random environment changing stage.

The infinitesimal generator of the model as a whole may be written as

$$\boldsymbol{Q} = \begin{bmatrix} \boldsymbol{Q}_1 & \cdots & \boldsymbol{Q}_{1e} & \cdots & \boldsymbol{Q}_{1E} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \boldsymbol{Q}_{e1} & \cdots & \boldsymbol{Q}_{e} & \cdots & \boldsymbol{Q}_{eE} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \boldsymbol{Q}_{E1} & \cdots & \boldsymbol{Q}_{Ee} & \cdots & \boldsymbol{Q}_{E} \end{bmatrix} \quad (1)$$

where $\boldsymbol{Q}_e = \boldsymbol{Q}_e^* - \alpha_e \boldsymbol{I}$, $\boldsymbol{Q}_e^*$ is the infinitesimal generator of the queueing network model considered in isolation with stage-e parameterization, $\boldsymbol{I}$ is the identity matrix, and for $e \neq h$, $\boldsymbol{Q}_{eh} = \alpha_{eh} \boldsymbol{R}_{eh}$.

Let $\boldsymbol{\pi} = (\boldsymbol{\pi}^1, \ldots, \boldsymbol{\pi}^e, \ldots, \boldsymbol{\pi}^E)$ be the vector of equilibrium probabilities for the model, where $\boldsymbol{\pi}^e$ refers to stage e. Assuming that the CTMC is ergodic and irreducible, the equilibrium probabilities may be obtained by direct numerical solution of the global balance equations $\boldsymbol{\pi}\boldsymbol{Q} = \boldsymbol{0}$, $\boldsymbol{\pi}\boldsymbol{1} = 1$. Unfortunately, due to the combinatorial growth of the state space, direct numerical solutions are prohibitive in most cases of practical interest. This motivates the development of the blending algorithm proposed in this paper.

4. EMBEDDED PROCESS

To cope with the state space explosion issue, we propose to characterize the equilibrium behavior of the model focusing on the network state observed immediately after stage transitions in the random environment. The associated *embedded probabilities* have stationary vector denoted by $\boldsymbol{\eta} = (\boldsymbol{\eta}^1, \ldots, \boldsymbol{\eta}^e, \ldots, \boldsymbol{\eta}^E)$, which represents the probability distribution embedded at the arrival instant of *any* random environment event. Thus, each $\boldsymbol{\eta}^e$ vector, after normalization to sum to unity, may be seen as the entry probability vector in state e at equilibrium. In this section, we first show that the $\boldsymbol{\eta}^e$ vectors are related by expressions that describe the transient evolution of the queueing network state during the sojourn time within each stage. The blending algorithm, presented in the next sections, leverages on this characterization by approximating the transient evolution using Kurtz's fluid limit theorem and then coupling this method with an outer iterative approximation aimed at approximating the random environment effects. Average steady-state performance indexes are obtained using intermediate results of the fluid analysis, as we show in Section 5.

The main contribution of this section is the following characterization of the embedded probabilities $\boldsymbol{\eta}^e$.

THEOREM 1. *In the CTMC with generator* (1)*, the embedded probabilities satisfy at equilibrium the balance*

$$\sum_{\substack{h=1 \\ h \neq e}}^{E} p_{he} \left(\int_0^{+\infty} \boldsymbol{\eta}^h e^{\boldsymbol{Q}_h^* t} \alpha_h e^{-\alpha_h t} dt \right) \boldsymbol{R}_{he} = \boldsymbol{\eta}^e \quad (2)$$

for all stages $e = 1, \ldots, E$, where $p_{he} = \alpha_{he}/\alpha_h$ is the probability of jump from stage h to stage e.

The proof is given in the Appendix.

Theorem 1 enjoys a simple probabilistic interpretation. Let us observe that:

- $\boldsymbol{\eta}^h e^{\boldsymbol{Q}_h^* t}$ is the transient probability vector at time t for the network initialized with probability $\boldsymbol{\eta}^h$ in stage h.

- $\alpha_h e^{-\alpha_h t}$ is the exponential density for the time to the next random environment event.

- $\boldsymbol{\eta}^h e^{\boldsymbol{Q}_h^* t} \boldsymbol{R}_{he}$ is the equilibrium entry probability vector in stage e after spending t time units in stage h.

Thus, the summation over all source stages h and jump times t in (2), weighted for the jump probabilities p_{eh}, simply provides the stationary entry vector $\boldsymbol{\eta}^e$ in stage e.

The importance of (2) lies in the fact that, despite not being directly computable due to state-space explosion, the balance relates equilibrium performance metrics, via the vectors $\boldsymbol{\eta}^e$, with transient analysis, via the integrand $\boldsymbol{\eta}^h e^{\boldsymbol{Q}_h^* t}$ which is a CTMC. This leads us to propose the use of fluid analysis methods, which are effective in transient analysis, for the computation of equilibrium performance metrics in random environments, as we describe next.

5. FLUID LIMITS

We now leverage on the theory of fluid limits developed by Kurtz [14] to define an approximate transient analysis method for queueing network models. Our intent is to obtain an approximation of the integrands of (2) conditional on the random environment being in stage h. Due to this conditioning, in the remainder of this section, we simplify

[1]Note that the $\boldsymbol{n}_i$ station state vector can be readily mapped to a canonical state description $(n_i, n_{i,1}, n_{i,2}, \ldots, n_{i,K_i^e})$, where $n_{i,k}$ is the number of requests receiving service in phase k and n_i is the total number of requests in the queue, according to the following transformation:

$$n_i = x_i + \sum_{k=2}^{K_i^e} s_{i,k}, \quad n_{i,k} = \begin{cases} \min(x_i, s_{i,1}), & k = 1 \\ s_{i,k}, & k = 2, \ldots, K_i^e \end{cases}$$

[2]In implementations, this value may be set equal to the total job population N.

Transition rate	Destination state
$f_1^{dep}(\boldsymbol{n},e,i,j) = \phi_{i1}^e r_{ij}^e \mu_{i1}^e \min(x_i, s_{i1})$	$\boldsymbol{n}' = (\boldsymbol{n}_1, \ldots, \boldsymbol{n}'_i, \ldots, \boldsymbol{n}'_j, \ldots, \boldsymbol{n}_M, e)$ $\boldsymbol{n}'_i = (x_i - 1, s_{i1}, s_{i2}, \ldots, s_{iK_i^e})$ $\boldsymbol{n}'_j = (x_j + 1, s_{j1}, s_{j2}, \ldots, s_{jK_i^e})$
$f_k^{dep}(\boldsymbol{n},e,i,j) = \phi_{ik}^e r_{ij}^e \mu_{ik}^e s_{ik}$	$\boldsymbol{n}' = (\boldsymbol{n}_1, \ldots \boldsymbol{n}'_i, \ldots, \boldsymbol{n}'_j, \ldots, \boldsymbol{n}_M, e)$ $\boldsymbol{n}'_i = (x_i, s_{i1} + 1, \ldots, s_{ik} - 1, \ldots, s_{iK_i^e})$ $\boldsymbol{n}'_j = (x_j + 1, s_{j1}, s_{j2}, \ldots, s_{jK_i^e})$
$f_1^{ph}(\boldsymbol{n},e,i) = (1 - \phi_{i1}^e)\mu_{i1}^e \min(x_i, s_{i1})$	$\boldsymbol{n}' = (\boldsymbol{n}_1, \ldots \boldsymbol{n}'_i, \ldots, \boldsymbol{n}_M, e)$ $\boldsymbol{n}'_i = (x_i - 1, s_{i1} - 1, s_{i2} + 1, \ldots, s_{iK_i^e})$
$f_k^{ph}(\boldsymbol{n},e,i) = (1 - \phi_{ik}^e)\mu_{ik}^e s_{ik}$	$\boldsymbol{n}' = (\boldsymbol{n}_1, \ldots \boldsymbol{n}'_i, \ldots, \boldsymbol{n}_M, e)$ $\boldsymbol{n}'_i = (x_i, s_{i1}, \ldots, s_{ik} - 1, s_{i(k+1)} + 1, \ldots, s_{iK_i^e})$
$f^{env}(\boldsymbol{n},e) = \alpha_{eh}$	$\boldsymbol{n}' = (\boldsymbol{n}'_1, \ldots, \boldsymbol{n}'_M, h)$ $\boldsymbol{n}'_i = \left(x_i + \sum_{k=2}^{K_i^e} s_{ik}, s_{i1} + \sum_{k=2}^{K_i^e} s_{ik}, \underbrace{0, \ldots, 0}_{K_i^e - 1}\right)$

Table 1: Transition rate functions. Source state is $(\boldsymbol{n}, e) = (\boldsymbol{n}_1, \ldots \boldsymbol{n}_i, \ldots, \boldsymbol{n}_M, e)$.

notation by omitting stage indexes.[3] We also consider arbitrary initial conditions, postponing to Section 6 the definition of the initial conditions used to approximate (2).

For queueing networks, fluid limits describe the behavior of the underlying Markov process as the population of requests N and the number of servers S_i at each queue i grow simultaneously to infinity. Such growth is assumed to be proportional to a scale parameter V that preserves the ratios of servers to requests and which has the interpretation of the system size. Hence, since capacity grows at the same rate of the request population growth, this avoids saturation in the asymptotic regime. In the limit regime, the network also maintains the same topology, routing matrix, and service rates of the original model.

Formally, a fluid limit with the above properties is obtained by taking a family of CTMCs $\{\boldsymbol{Y}_V(t), V \in \mathbb{N}\}$ indexed by the scale parameter V. Each vector $\boldsymbol{Y}_V(t)$ describes the network state at time t according to the state descriptor $\boldsymbol{n}$, defined as in Section 3, and with transition rates as in Table 1. Let $\boldsymbol{Y}_0$ be the initial state of the CTMC at time $t = 0$. The CTMC $\boldsymbol{Y}_1(t)$, where $V = 1$, is then the one with states reachable from $\boldsymbol{Y}_0$ through the rates in Table 1. Similarly, $\boldsymbol{Y}_2(t)$, where $V = 2$, is the CTMC with states reachable from the initial state $2\boldsymbol{Y}_0$ through the same rates. Generalizing, $\boldsymbol{Y}_V(t)$ is the CTMC with state space reachable from $V\boldsymbol{Y}_0$, for all V.

The fluid approximation of interest here consists in the limit behavior of the *normalized* CTMC $\boldsymbol{Y}_V(t)/V$ when V grows asymptotically large, with initial state $\boldsymbol{Y}_0$. A necessary condition to apply the limit theorem is that all transition rates enjoy the so-called *density-dependent* form, which informally states that all CTMC transition rates are proportional to V once rewritten in terms of the normalized state descriptor $\boldsymbol{n}/V$. For instance, making explicit the normalized state descriptor $\boldsymbol{n}/V$ in the f_1^{dep} rates yields

$$\begin{aligned} f_1^{dep}(\boldsymbol{n},i,j) &= \phi_{i1}^e r_{ij}^e \mu_{i1}^e \min(N_i, s_{i1}) \\ &= V\phi_{i1}^e r_{ij}^e \mu_{i1}^e \min(N_i/V, s_{i1}/V) \\ &= V f_1^{dep}(\boldsymbol{n}/V, i, j). \end{aligned} \tag{3}$$

[3]Note that our transient analysis applies also to Coxian queueing networks in the special case of a (non-random) environment with a single stage.

which satisfies the density-dependent form $f_1^{dep}(\boldsymbol{n},i,j) = V f_1^{dep}(\boldsymbol{n}/V, i, j)$ for all stations i, j. Similarly, it can be shown that all the other transition rates of Table 1 can be expressed in such a form, except the last one that is approximated directly by the blending algorithm without resorting to fluid approximations. In Kurtz's theory, the density-dependent form implies that a sample path of the normalised CTMC takes increasingly small steps, of order $O(1/V)$, at increasingly large rates, of order $O(V)$. In the limit $V \to +\infty$, this allows us to relate a sample path to a continuous trajectory, solution of a system of ordinary differential equations defined in terms of the density-dependent transition rates of the CTMC, as discussed in the next subsection.

5.1 Fluid Approximation

Let us first associate a *jump vector* to each transition rate function, in such a way to record the component-wise difference between source and destination states in the CTMC. For instance, the jump vector for $f_1^{dep}(\boldsymbol{n},e,i,j)$ is

$$\boldsymbol{\delta}_1^{dep}(i,j,e) = (\mathbf{0}, \ldots, \mathbf{0}, \boldsymbol{n}'_i - \boldsymbol{n}_i, \mathbf{0}, \ldots, \mathbf{0}, \boldsymbol{n}'_j - \boldsymbol{n}_j, \mathbf{0}, \ldots, \mathbf{0}, 0)^T$$

and when the stage e is omitted it is similarly defined but with the last element removed. Stemming from the ideas described above, the fluid approximation essentially involves replacing the state descriptor $\boldsymbol{n}$ with a time-varying deterministic vector $\boldsymbol{y}(t)$ and investigating the nonlinear ODE system

$$\begin{aligned} \frac{d\boldsymbol{y}(t)}{dt} = &\sum_{i,j=1}^{M} \sum_{k=1}^{K_i} f_k^{dep}(\boldsymbol{y}(t), i, j) \boldsymbol{\delta}_k^{dep}(i,j) \\ &+ \sum_{i=1}^{M} \sum_{k=1}^{K_i} f_k^{ph}(\boldsymbol{y}(t), i) \boldsymbol{\delta}_k^{ph}(i) \end{aligned} \tag{4}$$

with initial condition $\boldsymbol{Y}_0$. Using Kurtz's theorem [14, Theorem 3.1] and the fact that the transition rates are Lipschitz continuous everywhere, which implies global existence and uniqueness of the initial value problem (4), it can be shown that, for any finite T, it holds

$$\lim_{V \to \infty} \mathbb{P}\left\{\sup_{t \le T} |\boldsymbol{Y}_V(t)/V - \boldsymbol{y}(t)|\right\} = 0. \tag{5}$$

This implies that a sample path (of finite length) is asymp-

totically undistinguishable in probability, from the unique solution to the ODE system (4). Furthermore, in our model the fluid limit can be shown to imply *convergence in mean* [12]:

$$\lim_{V\to\infty} \mathbb{E}\left\{|\boldsymbol{Y}_V(t)/V - \boldsymbol{y}(t)|\right\} = 0, \quad t \geq 0$$

which provides the motivation for considering the approximation $\mathbb{E}\{\boldsymbol{Y}_V(t)\} \approx V\boldsymbol{y}(t)$ that relates the expected queue length to the re-scaled differential trajectory. This is the interpretation herein used and the ODE solution will thus be compared against mean performance indices of the queueing network.

5.2 Example

Let us consider a two-station queueing network with N circulating requests, without a random environment. Station 1 has S_1 servers with exponential service times ($K_1 = 1$ phase, rate $\mu_{1,1}$), station 2 has S_2 servers with a Coxian distribution ($K_2 = 2$ phases, rates $\mu_{2,1}$ and $\mu_{2,2}$, completion probabilities $\phi_{2,1}$ and $\phi_{2,2} = 1$). Station 1 has self-routing probability $r_{1,1} = 1 - p$, for $0 < p < 1$, whereas the other routing probabilities are $r_{1,2} = p$, $r_{2,1} = 1$, $r_{2,2} = 0$.

The state descriptor in the CTMC that models this system is $\boldsymbol{n} = (x_1, S_{1,1}, x_2, S_{2,1}, S_{2,2})$, and we set the initial condition to $\boldsymbol{Y}_0 = (N, S_1, 0, S_2, 0)$. As a result of these assumptions, the definitions in Section 3 specialize to:

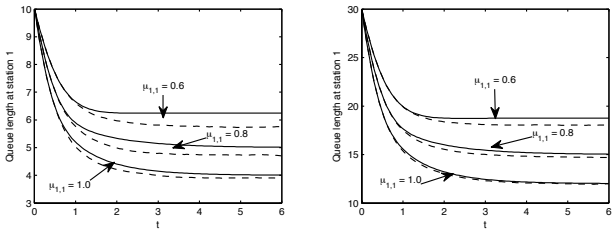
$$\begin{aligned}
f_1^{dep}(\boldsymbol{n}, 1, 1) &= (1-p)\mu_{1,1}\min(x_1, S_{1,1}),\\
f_1^{dep}(\boldsymbol{n}, 1, 2) &= p\mu_{1,1}\min(x_1, S_{1,1}),\\
f_1^{dep}(\boldsymbol{n}, 2, 1) &= \phi_{2,1}\mu_{2,1}\min(x_2, S_{2,1}),\\
f_2^{dep}(\boldsymbol{n}, 2, 1) &= \mu_{2,2}S_{2,2},\\
f_1^{ph}(\boldsymbol{n}, 2) &= (1-\phi_{2,1})\mu_{2,1}\min(x_2, S_{2,1}),
\end{aligned}$$

having jump vectors

$$\begin{aligned}
\boldsymbol{\delta}_1^{dep}(1,1) &= (0,0,0,0,0)^T,\\
\boldsymbol{\delta}_1^{dep}(1,2) &= (-1,0,+1,0,0)^T,\\
\boldsymbol{\delta}_1^{dep}(2,1) &= (+1,0,-1,0,0)^T,\\
\boldsymbol{\delta}_2^{dep}(2,1) &= (+1,0,0,+1,-1)^T\\
\boldsymbol{\delta}_1^{ph}(2) &= (0,0,-1,-1,+1)^T.
\end{aligned}$$

To develop the fluid approximation, we define the deterministic state vector $(x_1(t), S_{1,1}(t), x_2(t), S_{2,1}(t), S_{2,2}(t))$ which replaces $\boldsymbol{n}$ in the above transition rate functions expressions. Then, (4) provides the nonlinear ODE system

$$\begin{aligned}
\frac{dx_1(t)}{dt} &= -p\mu_{1,1}\min(x_1(t), S_{1,1}(t)) + \mu_{2,2}S_{2,2}(t)\\
&\quad + \phi_{2,1}\mu_{2,1}\min(x_2(t), S_{2,1}(t)); \qquad (6)\\
\frac{dS_{1,1}(t)}{dt} &= 0; \qquad (7)\\
\frac{dx_2(t)}{dt} &= -\mu_{2,1}\min(x_2(t), S_{2,1}(t))\\
&\quad + p\mu_{1,1}\min(x_1(t), S_{1,1}(t)); \qquad (8)\\
\frac{dS_{2,1}(t)}{dt} &= -(1-\phi_{2,1})\mu_{2,1}\min(x_2(t), S_{2,1}(t))\\
&\quad + \mu_{2,2}S_{2,2}(t); \qquad (9)\\
\frac{dS_{2,2}(t)}{dt} &= -\mu_{2,2}S_{2,2}(t)\\
&\quad + (1-\phi_{2,1})\mu_{2,1}\min(x_2(t), S_{2,1}(t)) \qquad (10)
\end{aligned}$$



(a) Plot of $\boldsymbol{Y}_1(t)$. (b) Plot of $\boldsymbol{Y}_3(t)$.

Figure 3: Comparisons between the expected values and the fluid approximation of the queue length at station 1 for the example of Section 5.2 with $p = 1.0$, $\phi_{2,1} = 0$, $\phi_{2,2} = 1$, $\mu_{2,1} = \mu_{2,2} = 2.0$, and $\mu_{1,1} \in \{0.6, 0.8, 1.0\}$. $\boldsymbol{Y}_0 = (10, \infty, 0, 4, 0)$. (a) CTMC $\boldsymbol{Y}_1(t)$; (b) CTMC $\boldsymbol{Y}_3(t)$. Solid lines: ODE solutions; dashed lines: CTMC expected values.

with the initial condition $\boldsymbol{Y}_0 = (N, S_1, 0, S_2, 0)$, where we choose arbitrarily to initialize the system with all requests in station 1. The interpretation of the individual ODE terms is similar to the discrete model. For example, (6)–(7) model the state of station 1. Negative fluxes model job departures, while positive fluxes model job arrivals. Null ODEs such as (7) are associated only to stations with exponential servers and indicate that the number of servers available to serve in phase 1 remains constant and, in this example, equal to the initial value S_1. Note that the sum of all derivatives (6)–(10) is zero, which preserves the property of conservation of mass for closed networks.

To numerically illustrate the accuracy of the fluid approximation for transient analysis of queueing networks, let us consider again the example in Section 5.2 and assume the following parameters: $p = 1.0$, $\phi_{2,1} = 0$, $\phi_{2,2} = 1$, $\mu_{2,1} = \mu_{2,2} = 2.0$. That is, we consider a tandem network with no self-routing probabilities where station 2 has Erlang-2 distributed service times with mean 1.0. Three different instances are studied by varying the mean service times at station 1, which is an exponential delay station in this example since the initial state vector is $\boldsymbol{Y}_0 = (10, \infty, 0, 4, 0)$.

Figure 3 shows some representative plots of the CTMC expected values and their fluid approximations during the transient regimes for $\mu_{1,1}$ equal to 0.6, 0.8, and 1.0. Figure 3(a) plots the queue-length processes at station 1 for the first CTMC, $\boldsymbol{Y}_1(t)$, of the family induced by the initial condition $\boldsymbol{Y}_0$. The ODE solutions (solid lines) are solved by standard numerical integration whereas the CTMCs were simulated with tight convergence criteria to obtain smooth trajectories for the expectations (dashed lines). The figure shows a generally acceptable quality of the approximation which here tends to increase with $\mu_{1,1}$. The case $\mu_{1,1} = 0.6$ has a maximum error of about 8% (at $t \approx 4.0$) although the trend of the transient dynamics is still captured well. It is interesting to note that these conditions are quite far away from the many requests/many servers parameterization considered by the limit theorem, nevertheless the fluid approximation is still usable for transient analysis. This is further backed up by Figure 3(b) which considers the CTMC $\boldsymbol{Y}_3(t)$, i.e., a total population of 30 requests and 12 servers in station 2. Here, the maximum error is reduced to about

4%, and generally behaves well in all the cases considered.

Summarizing, the examples illustrate that fluid limit approximations provide an inexpensive way of evaluating the transient of a queueing network. This feature is at the basis of the blending approximation we introduce in Section 6 for evaluating queueing models with random environments.

5.3 Continuous Mapping

We now turn back to motivating the fluid limit as an approximation for the right-hand side of (2). Let us denote by $\mathbf{L}^h(t)$ the stochastic process defined by $\boldsymbol{\eta}^h e^{\boldsymbol{Q}^*_h t}$. Each integral in the inner summation of (2) may be rewritten as

$$\int_0^{+\infty} \mathbf{L}^h(t)\alpha_h e^{-\alpha_h t}dt. \tag{11}$$

where $\mathbf{L}^h(t) = \boldsymbol{\eta}_h e^{\boldsymbol{Q}^*_h t}$. Since this does not enjoy a closed-form solution in general, it requires numerical integration over a finite range $[0, T]$, where T is a finite threshold for numerical convergence. Our goal is to show how transient analysis over any range $[0, T]$ may be approximated in terms of the fluid model presented in the previous subsections.

Since the process $\mathbf{L}^h(t)$ describes the behaviour of the network in stage h when no stage transitions are possible, it has an asymptotic behaviour according to Section 5.1. Therefore, we may conclude that $\mathbf{L}^h(t)/V$ converges in probability, for all $0 \leq t \leq T$, to $\boldsymbol{y}^h(t)$, where V is the scaling factor and $\boldsymbol{y}^h(t)$ is the fluid model associated with the network parametrization at the h-th stage of the random environment. We write such a convergence compactly as $\mathbf{L}^h(t)/V \to \boldsymbol{y}^h(t)$. By the continuous mapping theorem [12], it holds that

$$\mathbf{L}^h(t)\alpha_h e^{-\alpha_h t}/V \to \boldsymbol{y}^h(t)\alpha_h e^{-\alpha_h t} \tag{12}$$

for all $0 \leq t \leq T$ and for any α_h. Thus, the fluid approximation $\boldsymbol{y}^h(t)$ allows us to evaluate $V \cdot \boldsymbol{y}^h(t)\alpha_h e^{-\alpha_h t}$ as an approximate integrand of (2). This provides the key computational advantage, over the original expression, of requiring only the solution of a small set of ODEs in place of the transient of an intractably large CTMC.

6. THE BLENDING ALGORITHM

The blending algorithm is an iterative approximation technique that tries to address the state space explosion issue of an exact CTMC analysis. The approach approximates (2) by solving in a cyclic fashion a set of E ODE systems similar to (4). Each ODE system represents the transient behavior of the queueing network model conditional on the random environment being in a given stage. At each cycle, a new set of initial conditions, based on the results at the previous cycle, is provided to the E ODE systems. The algorithm is stopped after C iterations if either a user-specified maximum number of iterations is reached or if the initial conditions assigned to the ODEs converge to a constant value within a tolerance.

The pseudocode is summarized in Algorithm 1. For each cycle $c = 1, \ldots, C$ and stage $e = 1, \ldots, E$, the blending algorithm solves the ODE system (4) written as

$$\frac{d\boldsymbol{y}^e_c(t)}{dt} = \sum_{i,j=1}^{M}\sum_{k=1}^{K_i} f^{dep}_k(\boldsymbol{y}^e_c(t), i, j)\boldsymbol{\delta}^{dep}_k(i,j) + \sum_{i=1}^{M}\sum_{k=1}^{K_i} f^{ph}_k(\boldsymbol{y}^e_c(t), i)\boldsymbol{\delta}^{ph}_k(i). \tag{13}$$

Each trajectory $\boldsymbol{y}^e_c(t)$ describes the transient behavior of the queueing network after t time units since last entering stage e, subject to initial conditions $\boldsymbol{y}^e_c(0)$, $e = 1, \ldots, E$. At the first iteration, $c = 1$, $\boldsymbol{y}^e_c(0)$ is initialized similarly to the example in Section 5.2. That is, the entire job population N is split in a balanced manner across stations and all server state variables are set to $s_{i,1} = S_i$ and to $s_{i,k} = 0$, for phases $k > 1$. At the following iterations the blending algorithm uses directly (2) to update the initial conditions as

$$\boldsymbol{y}^e_{c+1}(0) = \sum_{h \neq e} p_{he} \int_0^{+\infty} k_{he}\boldsymbol{y}^h_c(t)\alpha_h e^{-\alpha_h t}\boldsymbol{R}_{he}dt \tag{14}$$

for $c = 2, \ldots, C$, where $k_{he} = \sum_{i \neq h}\omega_i\alpha_{i,h} / \sum_{j \neq e}\omega_j\alpha_{j,e}$ is a normalizing constant, and ω_e denotes the equilibrium probability of stage e in the random environment. This normalizing constant accounts for the fact that $\boldsymbol{y}^h_c$ vectors are probability distributions, whereas the $\boldsymbol{\eta}_h$ vectors in (2) are not normalized to sum to one. The ratio that defines k_{he} is simply the ratio of normalizing constants for $y^e_{c+1}(0)$ and $y^h_c(0)$, which only depend on the random environment CTMC and therefore are independent of c. The algorithm terminates at the first iteration c where the maximum improvement over the previous iteration is less than a user-provided convergence tolerance δ_{max} (e.g., $\delta_{max} = 10^{-6}$) or when c exceeds the maximum number of iterations C.

Remarks. The blending algorithm differs from techniques such as stochastic simulation in the following sense. In simulation, the system state evolves continuously and the effect of random environment effects is mainly to change the rate of the different events in the simulation and possibly to instantaneously change one or more state variables. In order to produce a more efficient evaluation of the model, blending first considers with (13) all the possible sample paths conditional on the model being in stage e. Then, the information from all these trajectories is *averaged* in the initial conditions (14) provided in input to the transient subproblems at the next iteration. Thus, instead of preserving the state upon stage jump, blending effectively averages all such states into the initial state variable. This allows to simplify the model evaluation at the expense of some approximation error.

6.1 Mean Performance Indexes

The approximate computation of mean performance indexes is based on the convergence in mean of Kurtz's fluid limit. Denote by π_e the equilibrium probability of stage e obtained by solving the random environment CTMC defined by the rates α_{eh}. Let $(x^e(t)_j, s^e_{j1}(t), s^e_{j2}(t), \ldots, s^e_{jK^e_j}(t))$ be the trajectory of the state of station j as determined during the final cycle of the blending algorithm. Then the mean queue-length is readily computed as

$$E[Q_j] = \sum_{e=1}^{E}\pi_e \int_0^{+\infty}\left(\min(x^e(t)_j, s^e_{j1}(t)) + \sum_{k=2}^{K^e_j} s^e_{jk}(t)\right)dt.$$

Algorithm 1 Blending algorithm pseudo-code

Input:

- model parameters
- ϵ: numerical tolerance
- δ_{max}: convergence tolerance
- $C > 1$: maximum number of iterations

Output: approximate $\boldsymbol{y}^e(t), \forall e$
Algorithm:
Initialize iteration counter: $c = 1$
Initialize $\boldsymbol{y}^e_c(0)$ for each stage $e = 1, \ldots, E$.
repeat
  $c = c + 1$
  for $e = 1, \ldots, E$ **do**
    Solve the ODE system (13) for $\boldsymbol{y}^e_{c+1}(t)$, $0 \leq t \leq T$, with initial values $\boldsymbol{y}^e_c(0)$ and tolerance ϵ.
  end for
  for $e = 1, \ldots, E$ **do**
    Compute $\boldsymbol{y}^e_{c+1}(0)$ by (14).
  end for
until $\max_e ||\boldsymbol{y}^e_{c+1}(0) - \boldsymbol{y}^e_c(0)||_1 < \delta_{\max}$ **and** $c \leq C$

	$M=2, E=2, S=1$			$M=2, E=2, S=5$		
		Runtime [s]			*Runtime [s]*	
Dist.	*Error*	*SIM*	*BLE*	*Error*	*SIM*	*BLE*
Erl	0.062	806	75	0.062	366	47
Exp	0.078	99	6	0.020	27	6
Cox	0.075	398	11	0.024	204	10

Table 2: Validation results: networks with 2 stations and 2 stages.

The mean throughput is obtained as

$$E[X_j] = \sum_{e=1}^{E} \pi_e \int_0^{+\infty} \mu^e_{j,1} \min(x^e(t)_j, s^e_{j1}(t)) \phi^e_{i,1} dt.$$

Both estimates are obtained simply by time-averaging the occupancy measures in the station state vector. In the case of the throughput, these are scaled by the departure rate from station j. Since in the steady-state the departure rate must be equal in all stages, the throughput in the above expression is given as the departure rate from phase 1.

From these metrics, the mean response time per visit at station j is readily obtained as

$$E[T_j] = E[Q_j]/E[X_j].$$

Finally, note that in case of delay servers where $S_j = +\infty$, the expressions above continue to be finite thanks to our assumption that all servers are initialized in phase 1, thus $\min(x^e(t)_j, s^e_{j1}(t)) = x^e(t)_j < +\infty$ and also $\sum_{k=2}^{K^e_j} s^e_{jk}(t) < +\infty$ since the job population is assumed finite and hence the number of active servers never exceeds N.

7. NUMERICAL VALIDATION

To study the error behavior of blending, we have performed an experimental campaign combining a large set of networks with randomly generated configurations, organized into groups where a subset of their parameters was kept fixed in order to stress crucial parts of the algorithm.

	$M=2, E=10, S=1$			$M=2, E=10, S=5$		
		Runtime [s]			*Runtime [s]*	
Dist.	*Error*	*SIM*	*BLE*	*Error*	*SIM*	*BLE*
Erl	0.208	6188	15	0.127	5322	18
Exp	0.152	609	4	0.058	166	5
Cox	0.139	1150	19	0.039	1284	23

Table 3: Validation results: networks with 2 stations and 10 stages.

	$M=10, E=2, S=1$			$M=10, E=2, S=5$		
		Runtime [s]			*Runtime [s]*	
Dist.	*Error*	*SIM*	*BLE*	*Error*	*SIM*	*BLE*
Erl	0.105	64923	17359	0.024	29284	8673
Exp	0.116	5923	1555	0.025	1873	434
Cox	0.158	18238	4127	0.028	14504	1141

Table 4: Validation results: networks with 10 stations and 2 stages.

Overall, we have considered 6 groups of networks with randomly generated parameters with different number of queues (M), stages in the random environment (E), and number of servers in each queue (S). The model parameters are drawn from uniform distributions with ranges $[0.0, 50.0]$ for the rates of the random environment. The population is equal to $N = 50$ jobs. Service-time distributions were varied in a controlled form. For each network configuration, we have considered 3 models with an exponential, Erlang-3 (squared coefficient of variation $SCV = 1/3$), and two-stage Coxian distribution with balanced phase probabilities ($SCV = 10$), all with the same given means sampled at random from a uniform distribution with range $[0.001, 10.0]$. Varying the service-time distributions allows us to study the impact of a parameter which cannot be captured by an ordinary fluid model of a network without random environment.

For each group we have analyzed 300 models, thus 1800 models were considered overall. Each model was analyzed using Gillespie's stochastic simulation algorithm. Both simulation and the blending algorithm were implemented in MATLAB, in order to use the same environment to compare their runtimes. The implementation of blending alternates execution of a non-stiff solver (`ode45`) and a stiff solver (`ode15s`) until both methods cannot further improve the (unconditional) mean queue-lengths of the random environment model. We define the minimal improvement in Algorithm 1 to be $\delta_{max} = 0.01$. The maximum number of iterations is set to $C = 100$. Furthermore, to speed-up convergence, the first iteration of the algorithm terminates by setting $\boldsymbol{y}^e_2(0) = \boldsymbol{y}^e_1(0)$ such that the blending algorithm performs at the first iteration a decomposition approximation computed with the fluid ODEs and initialized with a balanced job distribution across queues. This is advantageous particularly for stages with long holding times, since it allows the system to immediately accumulate mass on the bottleneck station conditional on the current stage. This would be slower using blending since several iterations would be

needed to reallocate the balanced distribution on the bottleneck stations.

The results are summarized in Tables 2-4. For each model with a population of N jobs, the accuracy error is expressed as follows:

$$\text{Error} = \max_{1 \leq i \leq N} \frac{|Q_i^{\text{BLE}} - Q_i^{\text{SIM}}|}{N}$$

where Q_i^{BLE} and Q_i^{SIM} are the estimates of the steady state queue length at station i as computed by blending and by simulation, respectively. Each table shows the average errors and execution times for a set of 100 randomly generated networks, with controlled service time distributions and server multiplicities.

The following observations arise from the results:

- Blending is systematically faster than simulation. In the networks with two stations, the difference in runtimes is two order of magnitudes on average, but it tends to decrease with larger network topologies. In the groups of networks with 10 stations (see Table 4), an analysis of the behavior of blending reveals that increased time of the analysis is mainly caused by a larger runtime per single iteration; this is due to the fact that the system of ODEs increases linearly with the number of stations and that larger problems may lead to stiffness for which equations are computationally more difficult to solve. The number of iterations, not reported in the table, are found to be quite similar in all groups.
- The accuracy of blending is adequate in all the cases considered, with an expected tendency to decrease with bigger network topologies and more stages of the random environment.
- By comparing the errors across the same row in every table, we find that increasing the number of servers at each station, while keeping all the other parameters the same, leads to an appreciable increase in the quality of the approximation. This is important for practical purposes, since modern multi-core machines often require to consider queues with multiple servers. For such models, we therefore expect our approximation to perform accurately.
- The service-time distributions have an impact on the accuracy error, but this is model-dependent and not easy to interpret. For instance, the cases with Erlang-distributed service times lead to the largest errors in Table 3, whereas they enjoy the smallest errors in the groups of networks of Table 4.

Summarizing, our validation campaign reveals that blending is remarkably faster than simulation, an important property in particular for optimization-based studies where thousands of models need to be evaluated in order to determine a good local optimum. We have performed initial experiments in this direction, suggesting that global optimization problems used to solve load-balancing problems that cannot be solved in 2 days using simulation at each iteration of the solver are instead solved in less than 2-3 hours using blending. Approximation errors are acceptable in all cases and importantly they improve sharply as the number of servers in each station increases by a few units, a situation that is typical in applications.

8. CONCLUSION

Blending is a new technique for the analysis of queueing network models in Markov random environments. This class of models can easily find application in describing the performance of cloud software systems, where uncertainties about the deployment environment can be easily modelled in terms of a Markov random environment. A numerical evaluation confirms that the this approach captures trends in performance indexes that these other techniques do not capture.A very attractive feature of our approach is scalability, the analysis being based on approximations obtained with ordinary differential equations which are insensitive to the job populations. We have found numerically that the accuracy improves with increasing system sizes, which makes our technique particularly appropriate for the analysis of massively parallel systems. Due to the good computational properties, it is readily applicable in decision problems that require the evaluation of the model over large parameter spaces. Several possible extensions of this work may be considered, for instance extensions to multi-class workloads and open models.

Acknowledgement

The work of Giuliano Casale is partially supported by the European project MODAClouds (FP7-318484) and by an EPSRC Small Equipment Funding grant. Mirco Tribastone is supported by the European project ASCENS, 257414.

9. REFERENCES

[1] Apache Derby project. http://db.apache.org.
[2] Apache Geronimo project. http://geronimo.apache.org.
[3] Apache OFBiz project. http://ofbiz.apache.org.
[4] M. Mao, M. Humphrey. A Performance Study on the VM Startup Time in the Cloud. In *Proc. IEEE Cloud*, 423–430, 2012.
[5] J. Schad, J. Dittrich, J.-A. Quiané-Ruiz. Runtime Measurements in the Cloud: Observing, Analyzing, and Reducing Variance. In *Proc. of the VLDB Endowment, Vol. 3, No. 1*, 2010.
[6] G. Casale, M. Tribastone. Fluid Analysis of Queueing in Two-Stage Random Environments In *QEST*, pages 21–30. IEEE Computer Society, 2011.
[7] J. Moschetta, G. Casale. OFBench: an Enterprise Application Benchmark for Cloud Resource Management Studies. In *Proc. of MICAS workshop - Management of Resources and Services in Cloud and Sky Computing*, Sep 2012.
[8] D. F. García and J. García. TPC-W E-commerce benchmark evaluation. *Computer*, 36(2):42–48, Feb. 2003.
[9] C. Amza, A. Ch, A. L. Cox, S. Elnikety, R. Gil, K. Rajamani, E. Cecchet, and J. Marguerite. Specification and implementation of dynamic Web site benchmarks. In *Proc. of IEEE 5th Annual Workshop on Workload Characterization*, Oct. 2002.
[10] P. Courtois. Decomposability: Queueing and Computer System Applications Academic Press, New York, 1977.
[11] D. A. Menascé and V. A. F. Almeida. *Scaling for E-Business: Technologies, Models, Performance, and Capacity Planning*. Prenctice Hall, 2000.

[12] P. Billingsley. *Probability and Measure*. Wiley, 3rd edition, 1995.

[13] G. Bolch, S. Greiner, H. de Meer, and K. S. Trivedi. *Queueing Networks and Markov Chains*. 2nd ed., John Wiley and Sons, 2006.

[14] T. G. Kurtz. Solutions of ordinary differential equations as limits of pure Markov processes. *J. Appl. Prob.*, 7(1):49–58, April 1970.

[15] G. Horváth, P. Buchholz, and M. Telek. A MAP fitting approach with independent approximation of the inter-arrival time distribution and the lag correlation. In *Proc. of the 2nd Conf. on Quantitative Evaluation of Systems (QEST)*, pages 124–133, 2005.

[16] M. F. Neuts. *Structured Stochastic Matrices of M/G/1 Type and Their Applications*. Marcel Dekker, New York, 1989.

APPENDIX

Proof of Theorem 1. From the CTMC generator $\boldsymbol{Q}$ in (1), we define two rate matrices

$$\boldsymbol{H} = diag(\boldsymbol{Q}_1, \ldots, \boldsymbol{Q}_e, \ldots, \boldsymbol{Q}_E), \qquad \boldsymbol{M} = \boldsymbol{Q} - \boldsymbol{H},$$

with the following interpretations. $\boldsymbol{H}$ is a sub-generator including rates of jump *not* associated with a random environment transition, whereas the remaining rates are included in matrix $\boldsymbol{M}$ and effectively marked for observation. That is, $(\boldsymbol{H}, \boldsymbol{M})$ defines a marked point process upon the CTMC generator where arrivals correspond to stage transitions at equilibrium.

Since all transitions are Markovian, the process $(\boldsymbol{H}, \boldsymbol{M})$ may be seen as an instance of Neuts's Markovian arrival process [16], which is a hidden Markov model with phase-type distributed inter-observation times. From basic properties of Markovian arrival processes (e.g., [15, Sec. 2]), the embedded distribution at stage transition instants satisfies

$$\boldsymbol{\eta} = \boldsymbol{\eta}(-\boldsymbol{H})^{-1}\boldsymbol{M}, \qquad \boldsymbol{\eta}\mathbf{1} = 1, \tag{15}$$

where $(-\boldsymbol{H})^{-1}\boldsymbol{M}$ is a stochastic matrix representing the discrete-time Markov chain (DTMC) embedded at stage transition instants.

We now relate $\boldsymbol{\eta}$ with the equilibrium distribution $\boldsymbol{\pi}$ of $\boldsymbol{Q}$ as follows. It is established that the relation between time-stationary distribution and equilibrium distribution of the embedded process is

$$\boldsymbol{\pi} = \gamma^{-1}\boldsymbol{\eta}(-\boldsymbol{H})^{-1}, \qquad \gamma = \boldsymbol{\eta}(-\boldsymbol{H})^{-1}\mathbf{1}, \tag{16}$$

where the normalizing constant γ has the probabilistic interpretation of mean inter-observation time in the marked process. Exploiting the block diagonal form of $\boldsymbol{H}$, we can then simplify (16) as

$$\boldsymbol{\pi}_e = \gamma^{-1}\boldsymbol{\eta}^e(\alpha_e\boldsymbol{I} - \boldsymbol{Q}_e^*)^{-1} \tag{17}$$

for all stages $e = 1, \ldots, E$. We now make the observation that the matrix inverse in (17) is simply the Laplace transform $\mathcal{L}(s)$ of the matrix exponential $e^{\boldsymbol{Q}_e^* t}$ evaluated at $s = \alpha_e$. We can therefore write

$$\boldsymbol{\pi}_e = \gamma^{-1}\alpha_e^{-1}\int_0^{+\infty} \boldsymbol{\eta}^e e^{\boldsymbol{Q}_e^* t}\alpha_e e^{-\alpha_e t}dt \tag{18}$$

Observe now that, by (16) and (15)

$$\boldsymbol{\pi}\boldsymbol{M} = \gamma^{-1}\boldsymbol{\eta}(-\boldsymbol{H})^{-1}\boldsymbol{M} = \gamma^{-1}\boldsymbol{\eta}$$

but this readily implies (2) by rewriting the formula for block $e = 1, \ldots, E$ as

$$\sum_{h=1}^{H} \boldsymbol{\pi}_h \alpha_{he} \boldsymbol{R}_{he} = \gamma^{-1}\boldsymbol{\eta}^e$$

and using (18) for the $\boldsymbol{\pi}_h$ vectors.