Efficient Optimization of Software Performance Models via Parameter-Space Pruning

Mirco Tribastone
Electronics and Computer Science
University of Southampton, United Kingdom
m.tribastone@soton.ac.uk

ABSTRACT

When performance characteristics are taken into account in a software design, models can be used to identify optimal configurations of the system's parameters. Unfortunately, for realistic scenarios, the cost of the optimization is typically high, leading to computational difficulties in the exploration of large parameter spaces. This paper proposes an approach to provably exact parameter-space pruning for a class of models of large-scale software systems analyzed with *fluid techniques*, efficient and scalable deterministic approximations of massively parallel stochastic models. We present a result of monotonicity of fluid solutions with respect to the model parameters, and employ it in the context of optimization programs with evolutionary algorithms by discarding candidate configurations *a priori*, i.e., without ever solving them, whenever they are proven to give lower fitness than other configurations. An extensive numerical validation shows that this approach yields an average twofold runtime speed-up compared to a baseline optimization algorithm that does not exploit monotonicity. Furthermore, we find that the optimal configuration is within a few percent from the *true* one obtained by stochastic simulation, whose solution is however orders of magnitude more expensive.

Categories and Subject Descriptors

I.6.5 [**Simulation and Modeling**]: Model Development—*Modeling methodologies*; D.2.8 [**Software Engineering**]: Metrics—*Performance measures*

Keywords

Software performance engineering; capacity planning; fluid approximations; queueing networks; monotone systems

1. INTRODUCTION

The evaluation of nonfunctional properties of software systems can be assisted by models. This is especially useful in early stages of the development process, when executable artifacts are not available but the designer wishes to obtain estimates about the behavior of the system in order to make more informed decisions about the architecture and the implementation [15]. Motivated by the ever increasing popularity of service-level agreements that concerns aspects such as performance, reliability, and availability, much research has gone into the integration of model-based software performance engineering practices with traditional development processes [9]. In this context, it is possible to identify at least three main challenges:

i) Developing mechanisms to shield the software engineer from the technical details of the underlying mathematical machinery used for the analysis.

ii) Guaranteeing that the model is a faithful representation of the real system throughout all the stages of the development process.

iii) Providing accurate and efficient evaluation techniques that scale well with increasing system sizes.

Challenge i) seems to be relatively well understood, owing to the large body of research concerned with enriching software models with suitable annotations for nonfunctional properties, the most notable case being the SPT/MARTE profiles for the UML [27]. In this paper, we assume that ii) has been also tackled—for instance by using techniques that continuously learn the model parameters as done in [35]—and study iii), with emphasis on models for software *performance*.

In a typical scenario, a performance model can be used for *capacity planning*, i.e., for estimating the amount of resources to be allocated in order to satisfy some required quality of service, or, more in generally, for *what-if analysis*, i.e., evaluating the impact of certain changes on the overall system's behavior. This introduces two orthogonal issues about the scalability of such analyses.

The first issue is related to the effectiveness with which a given instance is evaluated. Most models of software performance are based on Markov chains, which are however prone to the infamous problem of state-space explosion for increasing populations of system components. To tackle this problem, many approaches are available that consider exact aggregations (e.g., [19]) or approximate analysis (e.g., [16]) in order to reduce the computational cost. In this paper we make use of *fluid techniques* for the analysis. These are based on ordinary differential equations (ODEs) as deterministic approximations to the average path of a continuous-time Markov chain (CTMC) that models a *population process*. In such CTMC, the state descriptor gives the populations of entities that are in a particular *local state*. For

ICPE'14, March 22–26, 2014, Dublin, Ireland.

http://dx.doi.org/10.1145/2568088.2568090.

instance, in stochastic Petri nets the state lists the number of tokens in each place of the net (e.g., [4]). In queueing networks, the state may give the number of clients at each station (e.g., [5, 13]). In stochastic process algebra, each element of the state descriptor gives the copies of the components that exhibit a distinct sequential behavior [33]. In all cases, the crucial advantage of fluid techniques is that the size of the ODE system is independent from the actual population sizes, but is only dependent on the size of the state descriptor.

A fluid model is computationally much more advantageous than its stochastic CTMC counterpart, especially for large populations. However, as with CTMC analysis, the solution is generally not available in a closed form and thus it must be evaluated numerically. This implies that the results of a model with a given parameter configuration cannot be reused for solving the same model with a different one. Therefore, when fluid models are used for parameter-space explorations *each point* must be evaluated anew. This is, in essence, our second scalability issue: *The analysis is in general not scalable with the size of the parameter space.*

This paper presents a result of foundational nature that serves as the basis for tackling this very issue. Our purpose is to exploit a property of *monotonicity* of certain performance indices with respect to the model parameters. Given two vectors of parameters $\vec{v}_1$ and $\vec{v}_2$, we study under which conditions it is possible to prove that, for some partial order "$\leq$", $\vec{v}_1 \leq \vec{v}_2$ implies that $\phi(\vec{v}_1) \leq \phi(\vec{v}_2)$, where $\phi(\cdot)$ is the evaluation of the performance index for a given parametrization. This result can be readily applied. For example, when exploring the parameter space for minimizing ϕ, if $\vec{v}_1 \leq \vec{v}_2$ holds, $\phi(\vec{v}_2)$ needs not be computed because it yields a provably larger index. Importantly, checking for the inequality $\vec{v}_1 \leq \vec{v}_2$ comes at a negligible cost because it only compares model parameters, whereas the evaluation of ϕ requires solving the model, which, as discussed, may become expensive.

We study monotonicity for fluid models of closed queueing networks with arbitrary topologies, where stations serve with a generalized processor sharing (GPS) discipline. The reason for this choice is that GPS has been successfully proposed to model the dynamics of complex software systems in shared data centers and virtualized environments [14, 5, 6]. Thus, our approach is already usable for applications of practical interest. In addition, as will be discussed in more detail in Section 7, the fluid dynamics of GPS queueing networks are structurally close to those of, e.g., stochastic process algebra, stochastic Petri nets, and layered queueing networks. Thus, the approach can be easily generalizable to those techniques as well. The main theoretical contribution of this paper is to prove that the solution of the fluid model is monotone with respect to the client populations.

Armed with this result, we use it to improve the efficiency of the exploration of large parameter spaces for optimization purposes. We consider the problem of finding the best tradeoff between performance (i.e., throughput) and cost for a two-class GPS queueing network which is representative of a canonical three-layered software architecture. Similarly to [6], the network consists of a service station for each layer, and a *delay* station that models the user workload. The objective is to find the best workload mix using a genetic algorithm (GA). This represents one of the possible constraint optimization techniques, which was chosen because of its popularity in model-based software performance engineering (e.g., [20, 2]). Our approach, however, can also be used in conjunction with other search strategies such as hill-climbing and simulated annealing, where frequent comparisons are made between points in the parameter space.

We show the effectiveness of our a-priori parameter-space pruning by comparing the runtimes of a baseline GA where monotonicity is not exploited against that of a tuned version where genomes with provably worse fitness are immediately discarded. The numerical results show an average twofold speed-up. Furthermore, a comparison against the *true* optimal configurations returned by evaluating GA with stochastic simulation shows excellent accuracy across a wide range of operating conditions for our example network.

Paper outline. We review related work in Section 2. In an effort to make the paper self-contained, Section 3 gives a concise account of the fluid technique used in this paper and its relation with the CTMC that it approximates. Section 4 defines the fluid model of a closed queueing network with GPS service. Section 5 defines the optimization problem and introduces the notion of monotone ODE systems. It then proves that our fluid GPS model does enjoy monotonicity, and shows how to apply this result to the optimization. Section 6 presents the numerical results for our case study of a three-layered software system. Finally, Section 7 ends with a discussion on the methodology presented in this paper and outlines lines of future work.

2. RELATED WORK

There has been a considerable amount of research concerned with the application of optimization techniques to models of software systems. Whilst we refer to [3] for an exhaustive and up-to-date survey, in this section we focus on approaches that are most closely related to the techniques presented in this paper. In particular, we consider the literature that deals with the optimization of the parameters of a model to trade-off between performance and cost. Much effort has been devoted to the studying optimal selection of components in service-based systems. In [12] this is studied by means of linear optimization. While the resulting problem can be efficiently analyzed, the underlying model is based on the crucial assumption of the absence queueing delays, which is instead the main focus of this paper. Similar assumptions hold in frameworks that deal with optimal service compositions, e.g., [34, 7].

The approach taken in [11] uses instead Markov chains and probabilistic logics for formal specification of the quality of service. Although it is possible to account for queueing effects, the resulting problem would suffer from the curse of dimensionality with increasing numbers of components.

More scalability for the analysis is offered by layered queueing networks, solved by means of approximate mean value analysis [16]. These models are featured in Litoiu *et al.*, who consider optimization for deployment in service-oriented systems, in [23], studying the optimization of the concurrency levels in a distributed system, and in [22], for cloud environments. A similar analytic model is presented in [26], which uses Menascé's two-layered extended queueing network for an optimization algorithm for service-oriented architectures. While the use of extended queues is computationally advantageous with respect to other stochastic models for the evaluation of a point in the parameter space, no results are

provided regarding a-priori pruning of entire regions with provably lower cost, as is done in this paper.

Further reduction of the computational cost of the analysis is achievable by considering simpler mathematical models. Marzolla and Mirandola identify a class of BPEL workflows for web-service compositions that can be modeled as a single-class queueing network, for which they propose rapid evaluation based on bounds on steady-state performance for bottleneck identification [25]; however, an explicit optimization problem is not formulated. In [10] a performance model is presented for load balancing in service oriented architectures based on an open queueing network with $M/M/1$ stations. While this allows for closed-form expressions of the response time, the approach cannot be generalized to other situations, in particular, when multiple classes of services are to be considered. Instead, a multi-class queueing network for a similar brokering scenario is presented in [8], where the solution is based on operational analysis. However, unlike our approach, the model cannot be generalized to arbitrary topologies.

In conclusion to this section, our use of fluid techniques offers a compromise between cost and model expressiveness, since such techniques are able to incorporate queueing effects due to contention in a scalable manner. In addition, for the objective functions herein proposed, a-priori pruning of the parameter space may be obtained, thereby further reducing the computational burden of the solution of optimization problems. From a theoretical viewpoint, the paper makes a contribution to monotonicity properties of performance models. Although these have been long understood in the stochastic setting under certain conditions (see, e.g., [29]), and assumed to hold in other cases (e.g., [23]), here they are proven for fluid multi-class models.

3. PRELIMINARIES

The purpose of this section is to provide the necessary background to fluid techniques. This will be assisted by a trivial small example of a pure-delay two-station queueing network, which will only be used to build intuition and to illustrate all the notions introduced. The general model presented in Section 4, instead, will feature multiple classes of clients, arbitrary topology, and contention for the processing capacity of the servers.

Before proceeding, we fix some notation. As usual, $\mathbb{N}$ is the set of natural numbers, including 0; $\mathbb{Z}$ is the set of integers whereas $\mathbb{R}$ denotes the set of real numbers. Scalars are lowercase Roman letters, vectors have an arrow over the symbol, whereas matrices are uppercase Roman letters. When written in matrix notation, a system of ODEs is denoted by $\frac{d}{dt}\vec{x}(t) = G(\vec{x}(t))$; alternatively, to ease layout, when it is written in components the explicit dependence on t is dropped (all our ODE systems are autonomous) and Newton's *dot* notation is used instead, therefore we write $\dot{x}_i = G_i(x_1, \ldots, x_n)$, where x_1, ..., x_n are scalars.

We begin by formally defining a *population model*.

Definition 1. *A* population model *is defined by the following elements:*

- *A vector of n variables, denoted by $\vec{x} = (x_i)_{1 \le i \le n}$;*
- *A set of m* interaction functions *$f_j : \mathbb{R}^n \to \mathbb{R}$ and associated* jump vectors *$\vec{l}_j \in \mathbb{Z}^n$, for all $1 \le j \le m$;*

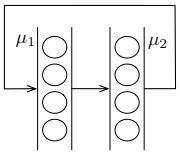

Figure 1: A pure-delay two-station queueing network.

- *An initial condition $\vec{x}_0 \in \mathbb{N}^n$.*

Intuitively, $\vec{x}$ represents the system's state whereas f_j describe the system dynamics, i.e., $f_j(\vec{x})$ describe the rate at which the state changes due to the j-th function, which may be interpreted as a force or interaction acting on the system. The impact of the interaction on each state variable is given by the jump vector. A negative (resp., positive) entry indicates a decrease (resp., increase) of the corresponding variable. Finally, $\vec{x}_0$ gives the initial state of the system. Requiring a vector of natural numbers gives the intuition behind the kinds of models considered in this paper, which are population-based, i.e., each state variable describes the evolution of a population of individuals of the same type, as discussed. The non-negativity of $\vec{x}_0$ therefore amounts to enforcing meaningfulness from a physical viewpoint.

Example 1 (Pure-delay network). *Figure 1 shows a simple tandem two-delay queueing network where a single class of clients visits two stations in sequence, with rates μ_1 and μ_2, where $\mu_1, \mu_2 > 0$. The model consists of the state vector $\vec{x} = (x_1, x_2)$, denoting the number of clients at each station (the queue length), by the interaction functions f_1 and f_2, and by the jump vectors $\vec{l}_1$ and $\vec{l}_2$ defined as*

$$f_1(x_1, x_2) = \mu_1 x_1, \qquad \vec{l}_1 = (-1, +1),$$
$$f_2(x_1, x_2) = \mu_2 x_2, \qquad \vec{l}_2 = (-1, +1).$$

The functions give the total rate of service at both stations, which simply grows linearly with the queue length. The jumps indicate that each transition moves one client from one queue to another, cyclically. The model is completed by letting $\vec{x}_0 = (N_1, N_2)$ be the initial population of clients.

A model of this kind can be used to describe a continuous-time Markov chain (CTMC), which is typically called a *Markov population process* to stress the fact that the state descriptor denotes counts of individuals.

Definition 2 (Population process). *For a population model we define a CTMC $\{X(t), t \in \mathbb{R}\}$, with state space S and transition rates denoted by $q(\vec{r}, \vec{s})$, for all $\vec{r}, \vec{s} \in S$ and $\vec{r} \neq \vec{s}$, as follows.*

1. *Let $X(0) = \vec{x}_0$ and $\vec{x}_0 \in S$.*
2. *Then S is defined to be the smallest set such that if $\vec{x} \in S$ and $f_j(\vec{x}) > 0$ then $\vec{x} + \vec{l}_j \in S$ and*

$$q(\vec{x}, \vec{x} + \vec{l}_j) = \sum_{j'=1, \vec{l}_{j'} = \vec{l}_j}^{m} f_{j'}(\vec{x}), \qquad 1 \le j \le m.$$

In our example, we have that $\vec{x}_0 = (N_1, N_2) \in S$; thus $\vec{x}' = (N_1 - 1, N_2 + 1) \in S$ with $q(\vec{x}_0, \vec{x}') = \mu_1 N_1$ and

$\vec{x}'' = (N_1 + 1, N_2 - 1) \in S$ with $q(\vec{x}_0, \vec{x}'') = \mu_2 N_2$, and so on. While in this simple case the cardinality of S is equal to $N_1 + N_2$, in general it is known to depend on the norm of $\vec{x}_0$ exponentially at worst—this is an instance of the *state-space explosion* problem. To tackle this, one can consider the following deterministic approximation, the *fluid approximation* (or fluid model).

DEFINITION 3 (FLUID APPROXIMATION). *The fluid approximation of a population model is an initial value problem with the following ODE system*

$$\frac{d}{dt}\vec{x}(t) = G(\vec{x}(t)), \qquad G(\vec{x}) = \sum_{j=1}^{n} \vec{l}_j f_j(\vec{x}),$$

and with initial condition $\vec{x}(0) = \vec{x}_0$.

In components, the fluid approximation of our example is

$$\frac{d}{dt}x_1(t) = -\mu_1 x_1(t) + \mu_2 x_2(t),$$
$$\frac{d}{dt}x_2(t) = +\mu_1 x_1(t) - \mu_2 x_2(t),$$

with $x_1(0) = N_1$ and $x_2(0) = N_2$. This shows the scalability of the fluid approximation: The ODE system size is independent from the actual populations (which only affect the initial conditions), and is only dependent on the length of the state descriptor (which in turn only depends on the network topology). In the remainder, we consider systems for which the fluid approximation enjoys existence and uniqueness of the ODE solution over some time interval $[0, T]$. All the examples herein proposed satisfy this property.

The ODE system can be interpreted as an estimate of the average CTMC behavior because, by a suitable approximation, it can be shown that (e.g., [28])

$$\frac{d}{dt}\mathbb{E}[X(t)] \approx G(\mathbb{E}[X(t)]),$$

where $\mathbb{E}[\cdot]$ denotes the expectation operator. For a class of models, the relationship is mathematically stronger in that the solution to the ODE is shown to be the asymptotic behavior of a suitable sequence of the associated population processes [21]. For the purposes of the present paper, however, we need not be concerned with the distinction between these two interpretations. Here it suffices to say that in general, the larger the population sizes (hence, the larger the CTMCs) the more accurate the approximation, with average errors of only a few percent (e.g., [31, 32, 33]).

In the example, every initial condition will give rise to ODE solutions that are nonnegative. This is what one would expect from any model where populations are physically relevant entities. We now provide a sufficient condition for non-negativity of the ODE solution that can be checked by inspection of the vector field of the fluid approximation.

LEMMA 1 (NONNEGATIVE FLUID APPROXIMATION). *For a population model, suppose that* $G_i(x) \geq 0$ *for every* $1 \leq i \leq n$ *and for every* x *such that* $x_i = 0$ *and* $x_{i' \neq i} \geq 0$. *Then it holds that*

$$x_i(t) \geq 0\ ,$$

for every i *and any* $t > 0$ *where the solution is defined.*

PROOF. Suppose toward a contradiction that there exists at least an i and a time $t > 0$ such that $x_i(t) < 0$. By continuity, together with the non negativity of the initial conditions, this would imply the existence of a time $t_1^i \in (0, t)$ such that $x_i(t_1^i) = 0$, $x_i(t) \geq 0$ for $t < t_1^i$ and $\dot{x}_i(t_1^i) < 0$. Let $t_m \triangleq \min_i t_1^i$, then $x_m(t_m) = 0$, $\dot{x}_m(t_m) < 0$ and $x_i(t) \geq 0$ for any i and $t \leq t_m$, which contradicts the assumption $G_m(x) \geq 0$ for any x with $x_m = 0$ and $x_{i \neq m} \geq 0$. □

4. GENERAL MODEL

We now consider a population model of a closed queueing network with $M + 1$ stations, labelled by 0, 1, ..., M, and K classes, labelled by 1, ..., K. Without loss of generality, we shall assume that only station 0 is a delay station, where clients do not contend for service. (An extension with an arbitrary number of delay stations is straightforward.) This is used to model the residence of clients outside the system before successive arrivals. Let $\mu_k > 0$ be the service rate for the k-th class at the delay station. The remaining M stations, instead, all serve with GPS discipline with total capacity D_i, $1 \leq i \leq M$. Let $w_i^k > 0$ be the *weight* (or priority) for class k, i.e., how much of the total capacity is proportionally assigned to each class requiring service at station i; let $\lambda_i^k > 0$ be the service rate, respectively, for class k at station $i > 0$. The routing of clients across the network is described, as usual, by K matrices, denoted by $P^k = (p_{ij}^k)_{1 \leq i,j \leq M}$ such that $p_{ij}^k \geq 0$ and $\sum_j p_{ij}^k = 1$ for all i. This implies that every client never leaves the system. The state variables describe the queue length at station i for each class k, and are denoted by x_i^k, with $0 \leq i \leq M$ and $1 \leq k \leq K$. The interaction functions are defined as follows:

$$f_0^k(\vec{x}) = p_{0j}^k \mu_k x_0^k, \qquad \vec{l}_0^k = -\mathbb{1}_{x_0^k} + \mathbb{1}_{x_j^k},$$
$$f_i^k(\vec{x}) = p_{ij}^k \frac{\lambda_i^k w_i^k x_i^k D_i}{\sum_{l=1}^K w_i^l x_i^l}, \qquad \vec{l}_i^k = -\mathbb{1}_{x_i^k} + \mathbb{1}_{x_j^k}, \quad (1)$$

for all $1 \leq k \leq K$ and $1 \leq i \leq M$, $0 \leq j \leq M$, where $\mathbb{1}_{x_i^k}$ represents the vector of length $\|\vec{x}\|$ of all zeros except at the position for variable x_i^k, where it is equal to 1.

The fluid approximation for a GPS queueing network is given by

$$\dot{x}_0^k = -(1 - p_{00}^k)\mu_k x_0^k + \sum_{j=1}^{M} p_{j0}^k \frac{\lambda_j^k w_j^k x_j^k D_j}{\sum_{l=1}^K w_j^l x_j^l},$$
$$\dot{x}_i^k = -\frac{\lambda_i^k w_i^k x_i^k D_i}{\sum_{l=1}^K w_i^l x_i^l} + p_{0i}^k \mu_k x_0^k + \sum_{j=1}^{M} p_{ji}^k \frac{\lambda_j^k w_j^k x_j^k D_j}{\sum_{l=1}^K w_j^l x_j^l}, \quad (2)$$

for all $1 \leq k \leq K$ and $1 \leq i \leq M$.

EXAMPLE 2. *Let us consider a model with* $K = 2$ *classes and, similarly to Example 1, a tandem queueing network, i.e.,* $M = 1$, $p_{01}^k = p_{10}^k = 1$ *for* $k = 1, 2$. *In this case, the*

fluid approximation is given by:

$$\dot{x}_0^1 = +\frac{\lambda_1^1 w_1^1 x_1^1 D_1}{w_1^1 x_1^1 + w_1^2 x_1^2} - \mu_1 x_0^1,$$
$$\dot{x}_1^1 = -\frac{\lambda_1^1 w_1^1 x_1^1 D_1}{w_1^1 x_1^1 + w_1^2 x_1^2} + \mu_1 x_0^1,$$
$$\dot{x}_0^2 = +\frac{\lambda_1^2 w_1^2 x_1^2 D_1}{w_1^1 x_1^1 + w_1^2 x_1^2} - \mu_2 x_0^2,$$
$$\dot{x}_1^2 = -\frac{\lambda_1^2 w_1^2 x_1^2 D_1}{w_1^1 x_1^1 + w_1^2 x_1^2} + \mu_2 x_0^2.$$

With this representation, it is evident that the assumption on closed workloads translates into a conservation-of-mass property given by the fact that

$$\frac{d}{dt}x_0^1(t) + \frac{d}{dt}x_1^1(t) = 0$$

and, similarly,

$$\frac{d}{dt}x_0^2(t) + \frac{d}{dt}x_1^2(t) = 0.$$

This implies that

$$x_0^1(t) + x_1^1(t) = x_0^1(0) + x_1^1(0)$$

and

$$x_0^2(t) + x_1^2(t) = x_0^2(0) + x_1^2(0),$$

for all t for which the solution of the fluid approximation is defined. In other words, the population of clients of each class is constant with time, and is equal to the total initial population. In general, the following proposition holds, by inspection of (2).

PROPOSITION 1. *Any GPS queuing network model satisfies the following properties:*

i) The fluid approximation is nonnegative.

ii) Let $N^k > 0$ denote the initial population of class k-clients at time $t = 0$, i.e., $N^k = \sum_{i=0}^{M} x_i^k(0)$. Then it holds that

$$x_j^k(t) = N^k - \sum_{\substack{i=0 \\ i \neq j}}^{M} x_i^k(t), \qquad \textit{for all } t. \tag{3}$$

5. OPTIMIZATION

This section presents the main result of this paper. The aim is to show that, when the parameters of a fluid model are to be optimized, then it is possible to prune certain regions of the parameter space which yield a provably higher cost. This can be done a priori, without ever analyzing the model in those regions. We show the applicability of this result to an optimization problem solved with GA, which is formulated in Section 5.1. Then, Section 5.2 discusses the result of monotonicity, which is the fundamental property that is exploited in our parameter-pruning approach. Finally, the implications on the optimization problem are discussed in Section 5.3.

5.1 Problem Formulation

We study the GPS model (1) with an optimization case study that aims at maximizing throughput and minimizing some operating cost. Whilst we focus on this problem in the remainder of this paper, let us remark that other scenarios are also possible. For instance, an analogous situation—by virtue of the duality between system throughput and response time by means of Little's law [24]— may consider response-time minimization and some revenue maximization.

Following [32], throughput may be estimated as a *reward measure* over the ODE solution. As an estimate of steady-state throughput, in particular, we consider a sufficiently large time point T where the solution is numerically close to equilibrium; this can be done by verifying that the norm of the ODE derivative at time T is less than a threshold.

The per-class throughput at equilibrium is given by

$$\mu_k(1 - p_{00}^k)x_0^k(T),$$

which is the total rate at which clients move from station 0 to some other station $i \neq 0$ (since this is a delay station, its throughput is proportional to the number of clients). Operating costs, instead, are assumed to be a function of the number of users (see [1] for a real case), denoted by $C(N^1, \ldots, N^K)$. This leads to the the following objective function to be minimized:

$$\phi(\vec{x}) = -\sum_{k=1}^{K} \mu_k(1 - p_{00}^k)x_0^k(T) + C(N^1, \ldots, N^K) \tag{4}$$

subject to the constraints

$$0 < N^k \leq U^k, \qquad \text{for all } 1 \leq k \leq K. \tag{5}$$

Therefore, this optimization program has client populations as the decision variables, with upper bounds given by constants U^k. This set-up may correspond to a practical situation where the modeler has no choice with respect to the server capacity (for instance, when a third-party server farm or cloud environment are used) and when the client demands are known. Under these conditions, the modeler may be interested in finding the workload mix, given by the per-class client populations, that optimizes the system's behavior.

5.2 Monotone Systems

At the basis of this technique is the notion of *monotone* ODE systems, which is briefly overviewed in this subsection in order to make the paper self-contained. The reader may find a detailed treatment in [30] and references therein.

First, we define O to be an orthant of $\mathbb{R}^n$, i.e., $O = \{\vec{x} \in \mathbb{R}^n : (-1)^{e_i} x_i \geq 0\}$, for $e_i \in \{0, 1\}$. For any $\vec{x}, \vec{y} \in \mathbb{R}^n$, we write $\vec{x} \leq_O \vec{y}$ if and only if $\vec{y} - \vec{x} \in O$. Now, let us consider an autonomous ODE system $\frac{d}{dt}\vec{x}(t) = G(\vec{x}(t))$ defined in $\mathbb{R} \times X$, with $X \subseteq \mathbb{R}^n$ an open and convex set and G a differentiable function in X. We call such a system *monotone* if, for any two initial conditions $\vec{x}(0) \leq_O \vec{y}(0)$, it holds that the corresponding solutions, denoted by $\vec{x}(t)$ and $\vec{y}(t)$, preserve the ordering, i.e., $\vec{x}(t) \leq_O \vec{y}(t)$, for all t for which both solutions are defined. We will be concerned with the case where O is the positive orthant of $\mathbb{R}^n$, i.e., $e_i = 0$ for all $1 \leq i \leq n$. Thus $\vec{x}(0) \leq_O \vec{y}(0)$ means $x_i(0) \leq y_i(0)$ for all $1 \leq i \leq n$. In the remainder of this paper, a comparison between two vector shall always be intended in this sense.

The following result characterizes monotonicity with respect to the Jacobian of G, denoted by $DG(\vec{x})$.

PROPOSITION 2 (SEE LEMMA 2.1 IN [30]). *Let G be a function defined on an open and convex set of $\mathbb{R}^n$ where it*

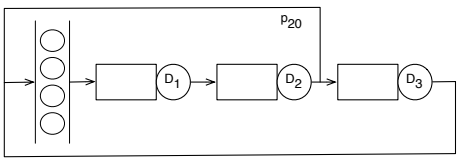

Figure 2: Queueing network model for the numerical validation of Section 6.

is differentiable. The ODE system $\frac{d}{dt}\vec{x} = G(\vec{x}(t))$ *is said to be* monotone *in the orthant* O *if and only if the matrix* $PDG(\vec{x})P$ *has nonnegative off-diagonal elements for every* $\vec{x} \in X$*, where* $P = diag((-1)^{e_1}, \ldots, (-1)^{e_n})$.

We are now ready to state the crucial proposition of this paper.

PROPOSITION 3 (MONOTONE GPS NETWORK). *Let the vector field of (2) be defined in the open and convex set such that* $x_i^k > 0$ *for all* $1 \leq k \leq K$ *and* $0 \leq i \leq M$*. Then, it holds that the ODE system (2) is monotone in the positive orthant.*

PROOF. See Appendix. □

5.3 Parameter-Space Pruning

Let $\vec{x}(0)$ and $\vec{y}(0)$ be two initial conditions for the fluid approximation (2) in the feasible region determined by constraints (5), with $\vec{x}(0) \leq \vec{y}(0)$. Let $\vec{x}(t)$ and $\vec{y}(t)$ represent the respective ODE solutions. Finally, let $N_x^k = \sum_{i=0}^{M} x_i^k(0)$, i.e., the total number of class-k clients in the system with initial conditions $\vec{x}(0)$. (N_y^k is defined analogously.) The above proposition yields that if $\vec{x}(0) \leq \vec{y}(0)$ then $\vec{x}(t) \leq \vec{y}(t)$ for all t.

We now exploit this fact to reduce the computational cost of the optimization program presented in Section 5.1. Let us consider the objective function (4) and write

$$\phi(\vec{x}) - \phi(\vec{y}) = C(N_x^1, \ldots, N_x^K) - C(N_y^1, \ldots, N_y^K) + (1 - p_{00}^k) \sum_{k=1}^{K} \mu_k y_0^k(T) - \mu_k x_0^k(T).$$

Now, the second line of the equation is nonnegative because the system is monotone. Thus, whenever the first summation is nonnegative, it holds that $\phi(\vec{x}) \geq \phi(\vec{y})$. Crucially, the sign of the first summation *can be established a-priori*, i.e., without having to solve the ODE systems, because all its terms are known parameters, i.e., the initial populations and the given cost function.

To further clarify this relation, let us consider the special case which involves setting $C \equiv 0$; this corresponds to a situation where there is no cost associated with the client populations. By inspection of the first line of the above equation, it is clear that $\phi(\vec{x}) \leq \phi(\vec{y})$: The maximum throughput is attained at the point of the feasible region with the largest populations, consistently with intuition. Hence, no computational cost for the optimization is required whatsoever.

6. NUMERICAL EXAMPLES

The purpose of this section is to study the computational advantages obtained by exploiting the results presented in the previous section. To this aim, we consider an optimization scenario based on a model of a three-tier software system modeled as a queueing network with three distinct GPS service centers (e.g., front-end, business logic, and a database-managent system) and a delay station, as illustrated in Fig. 2. In addition to being a reasonable high-level performance model of a complex distributed software system (for instance, it can be seen as a closed-workload variant of the model in [6]), the network is simple enough to keep stochastic simulation feasible, as this will be used as a baseline to assess the quality of the approximation introduced by the fluid approximation. On the other hand, the model is complicated enough to be exercised, with appropriate choice of parameters, under different operating conditions (e.g., performance bottlenecks at different stations).

The problem is solved by genetic algorithm with two approaches: a *black-box* approach (BB), which finds an optimal configuration *without* adopting parameter-space pruning; and a *grey-box* approach (GB) which does do parameter-space pruning in the following manner: Given a parameter vector $\vec{y}(0)$, of the initial populations of clients, if there exists another parameter vector $\vec{x}(0)$ with provably superior cost, then an *infinite fitness value* is assigned to it.

To show soundness of the approach, we compared the distance between the minima returned by both methods. As an index of effectiveness, we compared the runtimes of BB and GB. Finally, in order to show the overall accuracy of optimization via fluid techniques, we compared the estimated optima against those obtained by optimization via stochastic simulation of the associated CTMC, which is taken to represent the *true* behavior of the system. With this (expensive) study, we assess the *absolute quality* of fluid optimum, and we numerically evaluate the computational advantages gained by fluid analysis.

Parametrization. All tests were performed using Matlab 7.9.0, with the genetic algorithm implementation available in the *Genetic Algorithm and Direct Search* toolbox. In order to remove degrees of freedom in the set-up, unless otherwise stated the genetic algorithm was used with its default settings for both BB and GB. We considered a population of 30 individuals at each generation, and a maximum number of 20 generations.

For the evaluation of the fitness function (4) via fluid analysis we employed Matlab's `ode15s` routine by setting an absolute tolerance of 10^{-4} and a relative tolerance of 10^{-5}; all other parameters for the ODE solver were set as the default ones. The use of this solver was preferred in order to deal with potentially stiff problems, due to the randomness in the parametrization of the model, ensuring more robustness across the whole parameter space. This comes at the cost of longer execution times by `ode15s` in non-stiff models, which could be more efficiently solved by other methods, such as the well known Runge-Kutta scheme as implemented in `ode45`. However, the relative difference between these two methods is negligible compared to the difference between ODE analysis and stochastic simulation. We fixed $T = 5000.0$ for the evaluation of (4) and successfully verified that in all cases the derivatives at time T were less than 10^{-6} in norm, to ensure numerical convergence to an equilibrium.

For the evaluation of the fitness function via stochastic analysis we used Monte Carlo simulation based on Gillespie's direct method [18], which was preferred over the nu-

		FError		*SError*		*Runtimes*			*Confidence intervals*	
p_{20}^k	$\phi_{\min}^{\text{SIM}}$	*BB*	*GB*	*BB*	*GB*	*GB*	*BB (speed-up)*	*Sim (speed-up)*	*GB*	*BB*
0.85	-1.63	0.63%	0.58%	2.50%	1.65%	28.43 s	63.35 s (2.23)	37961 s (1335)	4.28%	2.97%
0.86	-2.12	0.03%	0.09%	1.18%	1.97%	28.87 s	64.52 s (2.23)	41244 s (1428)	4.97%	4.78%
0.87	-2.68	0.60%	0.51%	2.02%	1.63%	28.97 s	65.10 s (2.24)	45324 s (1564)	4.09%	4.34%
0.88	-3.28	0.75%	0.79%	2.56%	0.71%	28.46 s	63.30 s (2.22)	53240 s (1871)	4.77%	4.40%
0.89	-4.00	1.74%	1.81%	2.27%	2.04%	29.12 s	65.28 s (2.24)	53867 s (1849)	4.26%	4.20%
0.90	-5.00	0.38%	0.37%	1.62%	0.94%	31.47 s	66.05 s (2.09)	70762 s (2248)	4.60%	4.50%
0.91	-6.07	0.30%	0.27%	1.59%	1.07%	30.13 s	65.32 s (2.17)	76590 s (2541)	4.79%	4.23%
0.92	-7.42	0.80%	0.79%	1.24%	1.90%	31.21 s	65.13 s (2.09)	95300 s (3053)	4.76%	4.93%
0.93	-9.29	0.24%	0.26%	0.70%	1.00%	32.44 s	65.40 s (2.01)	125410 s (3865)	4.96%	4.96%

Table 1: Comparison between fluid optimization using a black-box approach (BB) and our approach (grey-box, GB) with parameter-space pruning.

merical CTMC solution because of the large state space sizes involved. The method of batch means was employed; the simulation was stopped when the largest confidence interval across all means at 95% confidence level was within 5%, with a maximum 8 batches of simulation, where the first was discarded for transient removal. Each batch was of length 5000.0; this choice was motivated by the fact that, as discussed, at that time interval all ODE solutions estimated numerical convergence to the equilibrium.

In all cases, we kept fixed the following parameters of the queueing network: $D_1 = 30.0$, $D_2 = 20.0$, $D_3 = 10.0$, $\mu_1 = \mu_2 = 1.5$, $\lambda_1^1 = \lambda_1^2 = \lambda_2^1 = \lambda_2^2 = 1.0$, $\lambda_3^1 = \lambda_3^2 = 0.1$, and $w_i^1 = 2.0$, $w_i^2 = 1.0$, for $i = 1, 2, 3$ (thus corresponding to a situation of two classes of clients with the same demands but different priorities/shares). The routing matrices were kept equal for both classes:

$$P^k = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ p_{20}^k & 0 & 0 & 1 - p_{20}^k \\ 1 & 0 & 0 & 0 \end{pmatrix}, \qquad k = 1, 2,$$

where we experimented with different values of p_{20}^k in order to test our approach for different operating conditions, since increasing p_{20}^k leads to less frequent visits to station 3 (which has the lowest capacity in the network). The initial conditions were set in such a way that the clients were evenly distributed across all stations.

The set-up of the optimization program was as follows. The constraints U^k were set to 2000, for $k = 1, 2$. The cost function was chosen in the form

$$C(N^1, N^2) = \alpha + \frac{(N^1 + N^2 - \beta)^\gamma}{\delta},$$

where α is interpreted as a fixed cost, β is a break-even total client population, γ gives the shape of the dependence of the cost from the client populations, and δ is a normalization factor that makes $C(N^1, N^2)$ comparable to the throughput, in order to exercise the model under conditions where changes in the parameters do affect cost sensibly. Notice that this normalization can always be done without loss of generality of this approach—for example, it could be interpreted as a change of monetary unit. In these experiments we set $\alpha = 5.0$, $\beta = 1000.0$, $\gamma = 2$, and $\delta = 2\text{E}05$.

Data analysis. The numerical tests were executed on machines equipped with an 8-way Opteron 2.6 GHz dual-core with 32 GB RAM. For each value of p_{20}^k, chosen between 0.85 and 0.93 at 0.01 steps, we ran three independent replicas. All the measurement reported in the following are the averages computed across these replicas.

As an index of effectiveness, we define the notion of percentage relative error of the optimization program as follows. Let $\phi_{\min}^{\text{BB}}$ (resp., $\phi_{\min}^{\text{BB}}$) be the minimum fitness value returned by the GA with the black-box (resp., grey-box) approach using fluid analysis; similarly, let $\phi_{\min}^{\text{SIM}}$ be the *true* minimum as returned by GA where each individual is analyzed through stochastic simulation. Then, the errors for BB and GB, denoted as *FError*, are defined as

$$\begin{aligned} \text{FError}_{\text{BB}} &= \frac{|\phi_{\min}^{\text{BB}} - \phi_{\min}^{\text{SIM}}|}{\phi_{\min}^{\text{SIM}}} \times 100, \\ \text{FError}_{\text{GB}} &= \frac{|\phi_{\min}^{\text{GB}} - \phi_{\min}^{\text{SIM}}|}{\phi_{\min}^{\text{SIM}}} \times 100. \end{aligned} \tag{6}$$

This notion alone, however, is not sufficient to fully understand the behavior of the approximation. The reason is that the evaluation of $\phi_{\min}^{\text{BB}}$ and $\phi_{\min}^{\text{GB}}$, in general, incurs two kinds of error which are due to the approximations of the stochastic process via the fluid model and of the stochastic reward (in this case, throughput) with its fluid counterpart. Thus, for instance, a relatively large FError may in fact still yield excellent accuracy when the individual with the minimum fitness obtained by fluid approximation is evaluated using stochastic simulation; that is, when one computes the true fitness of the best individual returned by the fluid GA. Using similar arguments, a small FError may turn out to be associated with a fittest individual which is away from the true optimum. In order to understand the nature of the optimal solutions returned by both BB and GB, we define the notion of *SError* as the error between $\phi_{\min}^{\text{SIM}}$ and the cost function evaluated by simulation for the fittest individual of BB and GB, which is denoted by $\phi^{\text{SIM}}(N_{\text{BB}}^1, N_{\text{BB}}^2)$ and $\phi^{\text{SIM}}(N_{\text{GB}}^1, N_{\text{GB}}^2)$, respectively:

$$\begin{aligned} \text{SError}_{\text{BB}} &= \frac{|\phi^{\text{SIM}}(N_{\text{BB}}^1, N_{\text{BB}}^2) - \phi_{\min}^{\text{SIM}}|}{\phi_{\min}^{\text{SIM}}} \times 100, \\ \text{SError}_{\text{GB}} &= \frac{|\phi^{\text{SIM}}(N_{\text{GB}}^1, N_{\text{GB}}^2) - \phi_{\min}^{\text{SIM}}|}{\phi_{\min}^{\text{SIM}}} \times 100. \end{aligned} \tag{7}$$

The computational advantage provided by our GB method is measured in terms of speed-up with respect to BB. Since both methods were applied to the same total number of individuals, the speed-up is only due to the fact that in GB some

individuals may be discarded a-priori because they yield provably worse fitness. As a general indication of the performance of fluid optimization, we also measured the speed-up with respect to stochastic simulation.

Results. The results of our experimental campaign are collectively reported in Table 1. Column labelled with $\phi^{\text{SIM}}_{\min}$ gives the minimum value of the fitness function returned by running the GA with stochastic simulation, to demonstrate that the chosen values of p^k_{20} do exercise the network under different steady-state conditions. Indeed, the fitness function is not explicitly dependent on p^k_{20}, hence the changes must be attributed to the different workload mixes that optimize the system's behaviour. Columns labelled with *FError* and *SError* show the accuracy indices as defined in (6) and (7), respectively. The runtime results are given as the average wall-clock execution time (in seconds) of the overall optimization program when using GB, BB, and simulation; for convenience, the speed-ups with respect to GB are also reported between brackets. Overall, the analyses required a total of 237 hours of computation time, of which 99.9% was devoted to the stochastic simulations. While the cost ODE analysis did not vary significantly across all tests, a significant increase of the simulation runtimes can be noticed as a function of p^k_{20}. This is due to the fact that the optimal configurations for larger p^k_{20} lead to increasingly saturated networks, which are notoriously more difficult to simulate. Using selected configurations that yield near-saturation conditions for $p_{20} > 0.93$, we estimated that the simulation runtimes to achieve confidence intervals within 5% would have been at least 32 times longer than for the case $p_{20} = 0.85$. This made the analysis of such models unfeasible under our given computational constraints, since the whole optimization problem might be aborted as a result of the expiration of the total time limit for a single job (i.e., 48 h) in the computer cluster where the experiments were conducted.

An analysis of the confidence intervals across all tests showed that, using the stopping criteria mentioned above, 82% of the simulations returned confidence intervals within the desired 5% level. Instead, for the other 18% of simulations (which were stopped because the maximum number of 8 batches had been reached), the median confidence interval was 6%. However, a manual analysis of the models with the highest confidence intervals showed that those were related to genomes with a significantly poor fitness. Indeed, in a typical case this was due to the exploration of regions of the parameter space with very low populations (recall that the constraints have a lower bound of 1) where the system dynamics are slower due to less frequent service requests. Instead, the average optimal workload mixes across all values of p^k_{20} were 159 and 189 class-1 and class-2 clients, respectively. For the same reason of computational feasibility as discussed above (time expiration), we could not adjust the stopping criteria for simulation in order to decrease the proportion of results with confidence intervals greater than 5%. However, to improve the precision of the optimal configuration returned by stochastic simulation, we ran longer simulations for $\phi^{\text{SIM}}(N^1_{\text{GB}}, N^2_{\text{GB}})$ and $\phi^{\text{SIM}}(N^1_{\text{BB}}, N^2_{\text{BB}})$ by doubling the batch length to 10000.0 time units. The last columns show the confidence intervals for the simulations under these modified stopping criteria.

Overall, this validation demonstrates that provable a-priori pruning of the parameter space can significantly reduce the cost of the exploration (by at least a factor of 2 across all the experiments), whilst returning estimates that differ less than 2% from those returned by the baseline GA implementation. We also confirm the suitability of fluid models for optimization purposes: The estimated optima are in all cases at most 3% away from the real optima computed by simulation. However the computational cost of simulation is excessively high, even for a small network with relatively few clients, and consistently separated from ODE-based optimization by three orders of magnitude.

7. CONCLUSION

Summary of findings. Many analysis techniques are available that can efficiently evaluate a model of software performance. However, in general, from the solution of a model with a given parametrization it is not possible to infer the behavior of the same model with a different parametrization. This paper has provided a contribution in this direction in the context of software performance models with fluid techniques. We have shown a general result of monotonicity whereby fluid solutions preserve the ordering of their parameters. As an application, here we discussed a case study of minimization via genetic algorithms, whereby some genomes can be shown to yield a provably superior fitness value a priori, i.e., without evaluating the fitness function, by virtue of monotonicity. Suitably equipping the genetic algorithm to exploit this property has shown a speed-up factor of over 2 on average with respect to a baseline version of the algorithm that does not implement a-priori pruning. Furthermore, the fluid approximation consistently yielded excellent accuracy with respect to the *true* optimal configurations returned by evaluating the fitness function with simulation.

Although we focused on optimization via evolutionary algorithms in this paper, we wish to stress that monotonicity can be exploited in a much broader context, i.e., whenever the modeler wishes to analyze different configurations, and use the evaluation of one configuration in order to infer the behavioral trend of others.

Scope of validity and generalization. There are two main issues that may hinder a wider applicability of this technique. The first one is that monotonicity does not hold for every parameter of the model under consideration. For instance, it was not possible to prove it for the server capacities D_i. Let us notice that Proposition 2 gives a sufficient condition, therefore, in principle, it could be established via other routes also for D_i. This will be the subject of future work. Nevertheless, we argue that the scope of applicability made available in this paper is rather significant, as it covers monotonicity with respect to initial populations (i.e., the system's concurrency levels) which can be directly related to throughput—hence response time—as discussed in Section 5.

The second limitation might be the focus on queueing networks with GPS service discipline. Keeping in mind that these models can already be used in virtualized and cloud environments, as discussed in Section 1, here we also wish to stress that an analogous result of monotonicity may be proven for other models of software performance, using the same arguments presented in this paper. For example, the GPS interaction functions (2) are surprisingly similar to

those used for the stochastic process algebra PEPA [33]. Indeed, in a typical situation, PEPA-like interaction functions can be written in the form

$$f_i(\vec{x}) = \mu_i \frac{x_i}{\sum_{i' \in I} x_{i'}} \min \left\{ \sum_{i' \in I} x_{i'}, x_j \right\}, \qquad (8)$$

where I is an index set such that $i \in I$ and $j \notin I$. Now, depending on the behavior of the minimum function, the ODE can be rewritten in terms of piece-wise differentiable functions. The case where $\min\left\{\sum_{i' \in I} x_{i'}, x_j\right\} = \sum_{i' \in I} x_{i'}$ becomes trivial because the function reduces to a delay-type interaction $\mu_i x_i$. Instead, the case $\min\left\{\sum_{i' \in I} x_{i'}, x_j\right\} = x_j$ can be handled in the same way as the GPS service case. Since (8) is essentially used in the PEPA encoding of layered queueing networks [31] and of stochastic Petri nets [17], monotonicity can be extended to the fluid approximations of these two other modeling techniques. A precise formalization of this extension is however beyond the scope of this paper and will be presented in future reports.

Acknowledgement

This work is supported by the EU project QUANTICOL, 600708, and by the DFG SPP-1593 project DAPS. The author wishes to thank Max Tschaikowski for discussions, and LRZ Munich for helpful support for the computing facilities.

8. REFERENCES

[1] The biggest cost of Facebook's growth. `http://www.technologyreview.com/news/427941/the-biggest-cost-of-facebooks-growth/`.

[2] A. Aleti, S. Björnander, L. Grunske, and I. Meedeniya. ArcheOpterix: An extendable tool for architecture optimization of AADL models. In *MOMPES*, pages 61–71, 2009.

[3] A. Aleti, B. Buhnova, L. Grunske, A. Koziolek, and I. Meedeniya. Software architecture optimization methods: A systematic literature review. *IEEE Trans. Softw. Eng.*, 39(5):658–683, 2013.

[4] H. Alla and R. David. Continuous and Hybrid Petri Nets. *Journal of Circuits, Systems, and Computers*, 8(1):159–188, 1998.

[5] J. Anselmi and I. Verloop. Energy-aware capacity scaling in virtualized environments with performance guarantees. *Perf. Eval.*, 68(11):1207–1221, 2011.

[6] D. Ardagna, B. Panicucci, M. Trubian, and L. Zhang. Energy-aware autonomic resource allocation in multitier virtualized environments. *IEEE Trans. Serv. Comput.*, 5(1):2–19, Jan. 2012.

[7] D. Ardagna and B. Pernici. Adaptive service composition in flexible processes. *IEEE Trans. Softw. Eng.*, 33(6):369–384, june 2007.

[8] E. Badidi, L. Esmahi, and M. A. Serhani. A queuing model for service selection of multi-classes QoS-aware web services. In *ECOWS*, pages 204–213, 2005.

[9] S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni. Model-based performance prediction in software development: A survey. *IEEE Trans. Softw. Eng.*, 30(5):295–310, 2004.

[10] B. Boone, S. V. Hoecke, G. V. Seghbroeck, N. Joncheere, V. Jonckers, F. D. Turck, C. Develder, and B. Dhoedt. SALSA: QoS-aware load balancing for autonomous service brokering. *Journal of Systems and Software*, 83(3):446–456, 2010.

[11] R. Calinescu, L. Grunske, M. Kwiatkowska, R. Mirandola, and G. Tamburrelli. Dynamic QoS management and optimization in service-based systems. *IEEE Trans. Softw. Eng.*, 37(3):387–409, 2011.

[12] V. Cardellini, E. Casalicchio, V. Grassi, F. Lo Presti, and R. Mirandola. QoS-driven runtime adaptation of service oriented architectures. In *ESEC/FSE*, pages 131–140. ACM, 2009.

[13] G. Casale and M. Tribastone. Fluid analysis of queueing in two-stage random environments. In *QEST*, pages 21–30, Aachen, Germany, September 2011. IEEE Computer Society Press.

[14] A. Chandra, W. Gong, and P. Shenoy. Dynamic resource allocation for shared data centers using online measurements. *SIGMETRICS Perform. Eval. Rev.*, 31(1):300–301, June 2003.

[15] V. Cortellessa, A. Di Marco, and P. Inverardi. *Model-Based Software Performance Analysis*. Springer, 2011.

[16] G. Franks, T. Al-Omari, M. Woodside, O. Das, and S. Derisavi. Enhanced modeling and solution of layered queueing networks. *IEEE Trans. Softw. Eng.*, 35(2):148–161, 2009.

[17] V. Galpin. Continuous approximation of PEPA models and Petri nets. *International Journal of Computer Aided Engineering and Technology*, 2:324–339, 2010.

[18] D. Gillespie. Exact stochastic simulation of coupled chemical reactions. *Journal of Physical Chemistry*, 81(25):2340–2361, December 1977.

[19] S. Gilmore, J. Hillston, and M. Ribaudo. An efficient algorithm for aggregating PEPA models. *IEEE Trans. Softw. Eng.*, 27(5):449–464, 2001.

[20] A. Koziolek, H. Koziolek, and R. Reussner. Peropteryx: automated application of tactics in multi-objective software architecture optimization. In *QoSA/ISARCS*, pages 33–42, 2011.

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

[22] J. Li, J. Chinneck, M. Woodside, M. Litoiu, and G. Iszlai. Performance model driven QoS guarantees and optimization in clouds. In *Proceedings of the Workshop on Software Engineering Challenges of Cloud Computing*, pages 15–22, 2009.

[23] M. Litoiu, J. Rolia, and G. Serazzi. Designing process replication and activation: A quantitative approach. *IEEE Trans. Softw. Eng.*, 26(12):1168–1178, 2000.

[24] J. Little. A Proof of the Queuing Formula: $L = \lambda W$. *Operations Research*, 9(3):383–387, 1961.

[25] M. Marzolla and R. Mirandola. Performance prediction of web service workflows. In *Software Architectures, Components, and Applications*, volume 4880 of *Lecture Notes in Computer Science*, pages 127–144. Springer, 2007.

[26] D. Menasce and V. Dubey. Utility-based QoS brokering in service oriented architectures. In *IEEE International Conference on Web Services*, pages 422–430, 2007.

[27] Object Management Group. *UML Profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE). Beta 1.* OMG, 2007. OMG document number ptc/07-08-04.

[28] P. Pollett, A. Dooley, and J. Ross. Modelling population processes with random initial conditions. *Mathematical Biosciences*, 223(2):142–150, 2010.

[29] J. G. Shanthikumar and D. D. Yao. Stochastic monotonicity in general queueing networks. *Journal of Applied Probability*, 26(2):413–417, 1989.

[30] H. L. Smith. Systems of Ordinary Differential Equations Which Generate an Order Preserving Flow. A Survey of Results. *SIAM Review*, 30(1):87–113, 1988.

[31] M. Tribastone. A fluid model for layered queueing networks. *IEEE Trans. Softw. Eng.*, 39(6):744–756, 2013.

[32] M. Tribastone, J. Ding, S. Gilmore, and J. Hillston. Fluid rewards for a stochastic process algebra. *IEEE Trans. Softw. Eng.*, 38:861–874, 2012.

[33] M. Tribastone, S. Gilmore, and J. Hillston. Scalable differential analysis of process algebra models. *IEEE Trans. Softw. Eng.*, 38(1):205–219, 2012.

[34] L. Zeng, B. Benatallah, A. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. QoS-aware middleware for web services composition. *IEEE Trans. Softw. Eng.*, 30(5):311–327, 2004.

[35] T. Zheng, C. M. Woodside, and M. Litoiu. Performance model estimation and tracking using optimal filters. *IEEE Trans. Software Eng.*, 34(3):391–406, 2008.

APPENDIX

Here we give the proof of Proposition 3.

PROOF. First, let us consider the ODE system

$$\dot{x}_0^k = -(1-p_{00}^k)\mu_k x_0^k + \sum_{j=1}^{M} p_{j0}^k \frac{\lambda_j^k w_j^k x_j^k D_j}{\sum_{l=1}^{K} w_j^l \left(N^l - \sum_{j'\neq j} x_{j'}^l\right)},$$

$$\dot{x}_i^k = -\frac{\lambda_i^k w_i^k x_i^k D_i}{\sum_{l=1}^{K} w_i^l x_i^l} + p_{0i}^k \mu_k x_0^k + \sum_{j=1}^{M} p_{ji}^k \frac{\lambda_j^k w_j^k x_j^k D_j}{\sum_{l=1}^{K} w_j^l \left(N^l - \sum_{j'\neq j} x_{j'}^l\right)}, \tag{9}$$

$$\dot{N}^k = 0,$$

for all $1 \leq k \leq K$ and $1 \leq i \leq I$. This ODE system arises from (2) by replacing each x_j^l with $N^l - \sum_{j'\neq j} x_{j'}^l$, and by adding *slack* variables N^k as trivial ODEs that yield a constant solution with respect to time t. Due to the property *ii*) of conservation of mass, each solution to the original ODE system (2) is also a solution to (9), whenever the initial conditions $N^k(0)$ for the slack variables are set as

$$N^k(0) = \sum_{i=0}^{M} x_i^k(0).$$

Thus, monotonicity may be equivalently proven on the modified ODE system (9). Its Jacobian $DG(\vec{x})$ can be written as

$$\left(\begin{array}{ccc|ccc} \frac{\partial g_0^1}{\partial x_0^1} & \cdots & \frac{\partial g_0^1}{\partial x_M^1} & \cdots & \frac{\partial g_0^1}{\partial x_M^K} & \frac{\partial g_0^1}{\partial N^1} & \cdots & \frac{\partial g_0^1}{\partial N^K} \\ \vdots & & \vdots & & \vdots & \vdots & & \vdots \\ \frac{\partial g_M^K}{\partial x_0^1} & & \cdots & & \frac{\partial g_M^K}{\partial x_M^K} & \frac{\partial g_M^K}{\partial N^1} & \cdots & \frac{\partial g_M^K}{\partial N^K} \\ \hline \frac{\partial N^1}{\partial x_0^1} & & \cdots & & \frac{\partial N^1}{\partial x_K^M} & \frac{\partial N^1}{\partial N^1} & \cdots & \frac{\partial N^1}{\partial N^K} \\ \vdots & & \vdots & & \vdots & \vdots & & \vdots \\ \frac{\partial N^K}{\partial x_0^1} & & \cdots & & \frac{\partial N^K}{\partial x_K^M} & \frac{\partial N^K}{\partial N^1} & \cdots & \frac{\partial N^K}{\partial N^K} \end{array}\right)$$

where g_i^k denotes the component of the vector field for the variable x_i^k, for all $0 \leq i \leq M$ and $1 \leq k \leq K$. With this block structure, it is possible to show monotonicity of the system by using Proposition 2 and Remark 1 in [30], whereby a sufficient condition is that: i) the off-diagonal elements of the top-left and bottom-right blocks be non-negative; and ii) the elements of the top-right and bottom-left blocks be non-positive.

In order to show this, we proceed by case distinction, recalling that the GPS queueing network fluid model is non-negative, all rates λ_i^k, weights w_i^k, and server capacities D_i are positive reals, and that the routing probabilities p_{ij}^k are nonnegative reals. First, we observe that the bottom blocks of the Jacobian are all trivially zero, thus we focus on the top blocks only.

For the top-left block:

i) Case $\dfrac{\partial g_0^k}{\partial x_i^k}$, with $i \neq 0$:

$$\begin{aligned}
\frac{\partial g_0^k}{\partial x_i^k} &= \frac{\partial}{\partial x_i^k}\left\{\sum_{j=1}^{M} p_{j0}^k \frac{\lambda_j^k w_j^k x_j^k D_j}{\sum_{l=1}^{K} w_j^l \left(N^l - \sum_{j'\neq j} x_{j'}^l\right)}\right\} \\
&= \frac{\partial}{\partial x_i^k}\left\{\sum_{j\neq i} p_{j0}^k \frac{\lambda_j^k w_j^k x_j^k D_j}{\sum_{l=1}^{K} w_j^l \left(N^l - \sum_{j'\neq j} x_{j'}^l\right)} + \right. \\
&= \left. + p_{i0}^k \frac{\lambda_i^k w_i^k x_i^k D_i}{\sum_{l=1}^{K} w_i^l \left(N^l - \sum_{j'\neq i} x_{j'}^l\right)}\right\} \\
&= -\sum_{j\neq i} p_{j0}^k \lambda_j^k w_j^k x_j^k D_j \frac{\partial}{\partial x_i^k}\left\{\sum_{l=1}^{K} w_j^l \left(N^l - \sum_{j'\neq j} x_{j'}^l\right)\right\} \\
&\quad + p_{i0}^k \frac{\lambda_i^k w_i^k D_i}{\sum_{l=1}^{K} w_i^l \left(N^l - \sum_{j'\neq i} x_{j'}^l\right)} \\
&= \sum_{j\neq i} p_{j0}^k \lambda_j^k w_j^k x_j^k D_j w_i^l + \\
&\quad + p_{i0}^k \frac{\lambda_i^k w_i^k D_i}{\sum_{l=1}^{K} w_i^l \left(N^l - \sum_{j'\neq i} x_{j'}^l\right)} \geq 0.
\end{aligned}$$

ii) Case $\frac{\partial g_0^k}{\partial x_0^{\hat{l}}}$, with $\hat{l} \neq k$:

$$\begin{aligned}\frac{\partial g_0^k}{\partial x_0^{\hat{l}}} &= \sum_{j=1}^{M} p_{j0}^k \frac{\partial}{\partial x_0^{\hat{l}}} \left\{ \frac{\lambda_j^k w_j^k x_j^k D_j}{\sum_{l=1}^K w_j^l \left(N^l - \sum_{j' \neq j} x_{j'}^l \right)} \right\} \\ &= -\sum_{j=1}^{M} p_{j0}^k \lambda_j^k w_j^k x_j^k D_j \frac{\partial}{\partial x_0^{\hat{l}}} \left\{ \sum_{l=1}^{K} w_j^l \left(N^l - \sum_{j' \neq j} x_{j'}^l \right) \right\} \\ &= \sum_{j=1}^{M} p_{j0}^k \lambda_j^k w_j^k x_j^k D_j w_j^{\hat{l}} \geq 0.\end{aligned}$$

iii) Case $\frac{\partial g_0^k}{\partial x_i^{\hat{l}}}$, with $i > 0$ and $\hat{l} \neq k$: similarly to ii),

$$\begin{aligned}\frac{\partial g_0^k}{\partial x_i^{\hat{l}}} &= -\sum_{j=1}^{M} p_{j0}^k \lambda_j^k w_j^k x_j^k D_j \frac{\partial}{\partial x_i^{\hat{l}}} \left\{ \sum_{l=1}^{K} w_j^l \left(N^l - \sum_{j' \neq j} x_{j'}^l \right) \right\} \\ &= \sum_{j=1}^{M} p_{j0}^k \lambda_j^k w_j^k x_j^k D_j w_j^{\hat{l}} \geq 0.\end{aligned}$$

iv) Case $\frac{\partial g_i^k}{\partial x_0^k}$, with $i \neq 0$:

$$\begin{aligned}\frac{\partial g_i^k}{\partial x_0^k} &= -\sum_{j=1}^{M} p_{ji}^k \lambda_j^k w_j^k x_j^k D_j \frac{\partial}{\partial x_0^k} \left\{ \sum_{l=1}^{K} w_j^l \left(N^l - \sum_{j' \neq j} x_{j'}^l \right) \right\} \\ &\quad + p_{0i}^k \mu_k \\ &= \sum_{j=1}^{M} p_{ji}^k \lambda_j^k w_j^k x_j^k D_j w_j^k + p_{0i}^k \mu_k \geq 0.\end{aligned}$$

v) Case $\frac{\partial g_i^k}{\partial x_0^{\hat{l}}}$, with $\hat{l} \neq k$: similarly to ii),

$$\frac{\partial g_i^k}{\partial x_0^{\hat{l}}} = \sum_{j=1}^{M} p_{ji}^k \lambda_j^k w_j^k x_j^k D_j w_j^{\hat{l}} \geq 0$$

vi) Case $\frac{\partial g_i^k}{\partial x_{\hat{j}}^k}$, with $\hat{j} \neq i$, and $i, \hat{j} > 0$: similarly to i),

$$\begin{aligned}\frac{\partial g_i^k}{\partial x_{\hat{j}}^k} &= \frac{\partial}{\partial x_{\hat{j}}^k} \left\{ \sum_{j=1}^{M} p_{ji}^k \frac{\lambda_j^k w_j^k x_j^k D_j}{\sum_{l=1}^K w_j^l \left(N^l - \sum_{j' \neq j} x_{j'}^l \right)} \right\} \\ &= \frac{\partial}{\partial x_{\hat{j}}^k} \left\{ \sum_{j \neq \hat{j}} p_{ji}^k \frac{\lambda_j^k w_j^k x_j^k D_j}{\sum_{l=1}^K w_j^l \left(N^l - \sum_{j' \neq j} x_{j'}^l \right)} + \right. \\ &\qquad \left. + p_{\hat{j}i}^k \frac{\lambda_{\hat{j}}^k w_{\hat{j}}^k x_{\hat{j}}^k D_{\hat{j}}}{\sum_{l=1}^K w_{\hat{j}}^l \left(N^l - \sum_{j' \neq \hat{j}} x_{j'}^l \right)} \right\} \\ &= \sum_{j \neq \hat{j}} p_{ji}^k \lambda_j^k w_j^k x_j^k D_j w_j^k + \\ &\qquad + p_{\hat{j}i}^k \frac{\lambda_{\hat{j}}^k w_{\hat{j}}^k D_{\hat{j}}}{\sum_{l=1}^K w_{\hat{j}}^l \left(N^l - \sum_{j' \neq \hat{j}} x_{j'}^l \right)} \geq 0.\end{aligned}$$

vii) Case $\frac{\partial g_i^k}{\partial x_{\hat{j}}^{\hat{l}}}$, with $i, \hat{j} > 0$ and $k \neq \hat{l}$:

$$\frac{\partial g_i^k}{\partial x_{\hat{j}}^{\hat{l}}} = \sum_{j \neq \hat{j}} p_{ji}^k \lambda_j^k w_j^k x_j^k D_j w_j^{\hat{l}} \geq 0.$$

For the top-right block:

i) Case $\frac{\partial g_i^k}{\partial N^{\hat{l}}}$, with $0 \leq \hat{l} \leq K$, $0 \leq i \leq M$:

$$\frac{\partial g_i^k}{\partial N^{\hat{l}}} = -\sum_{j=1}^{M} p_{ji}^k \lambda_j^k w_j^k x_j^k D_j w_j^{\hat{l}} \leq 0.$$

□