Scaling Size and Parameter Spaces in Variability-aware Software Performance Models

Matthias Kowal
TU Braunschweig
Braunschweig, Germany
m.kowal@tu-braunschweig.de

Max Tschaikowski Mirco Tribastone
IMT Institute for Advanced Studies Lucca
Lucca, Italy
{max.tschaikowski, mirco.tribastone}@imtlucca.it

Ina Schaefer
TU Braunschweig
Braunschweig, Germany
i.schaefer@tu-braunschweig.de

***Abstract*—In software performance engineering, what-if scenarios, architecture optimization, capacity planning, run-time adaptation, and uncertainty management of realistic models typically require the evaluation of many instances. Effective analysis is however hindered by two orthogonal sources of complexity. The first is the infamous problem of state space explosion—the analysis of a single model becomes intractable with its size. The second is due to massive parameter spaces to be explored, but such that computations cannot be reused across model instances. In this paper, we efficiently analyze many queuing models with the distinctive feature of more accurately capturing variability and uncertainty of execution rates by incorporating general (i.e., non-exponential) distributions. Applying product-line engineering methods, we consider a *family* of models generated by a core that evolves into concrete instances by applying simple *delta* operations affecting both the topology and the model's parameters. State explosion is tackled by turning to a scalable approximation based on ordinary differential equations. The entire model space is analyzed in a *family-based* fashion, i.e., at once using an efficient symbolic solution of a *super-model* that subsumes every concrete instance. Extensive numerical tests show that this is orders of magnitude faster than a naive instance-by-instance analysis.**

I. Introduction

In model-based software performance engineering [12], a variety of analyses involve the evaluation of many instances of a model. For instance, typical *what-if* questions investigate the effect of changes in parameters on the performance of a system; software architecture optimization methods look for the best design alternative that satisfies certain constraints [1]; later on in the software life cycle, e.g., at run-time, the violation of a given quality-of-service agreement may trigger an on-line reconfiguration process that examines a better alternative [10]; uncertainty can be cast into a problem of evaluating model instances obtained from samples of the probability distributions for the values of the unknown parameters [41].

These scenarios introduce two orthogonal sources of complexity which may make the reasoning in general very difficult, if not intractable, for complex software performance models to be evaluated over large parameter spaces. The first is the infamous problem of *state-space explosion*, arising in many techniques based on a discrete state-space representation, such Markov chains as typically adopted in the software performance engineering literature [2]. Here, the issue is that the size of the model, intended as the number of states on which it can be found, grows very rapidly (in the worst case, exponentially), with the number of components in the system (e.g., number of jobs, service centres, and so on). The second source of complexity comes from the fact that, in general, the analysis of a model instance cannot be re-used to evaluate another model drawn from the same parameter space (e.g., when a rate value or a routing probability are changed). This creates a multiplicative, combinatorial, effect on state-space explosion.

In this paper, we consider software performance models with the distinctive capability of supporting, as first-class citizens, general probability distributions for describing the durations of the activities in a system. More traditional approaches, instead, are based on the assumption of exponential distributions, which straightforwardly leads to a continuous-time Markov chain (CTMC) model. Our peculiarity is particularly relevant in practice because there is substantial evidence that in many real-world situations, durations *are not* exponentially distributed. For instance, Internet traffic has been fitted to heavy-tailed distributions [15]; web objects have been shown to be Zipf distributed [27]; events such as time-outs, with low variability in their execution times, can be modeled using Erlang distributions [32]. Unfortunately, apart from specific classes of models (e.g., [6], [32]), the modeler must resort to computationally expensive procedures such as stochastic simulations in order to handle general distributions. As a consequence, even the analysis of a single model instance—let alone the efficient computation of large parameter spaces—can be difficult under these conditions.

We present a class of software performance models which enables a scalable evaluation of a single instance as well as of the whole parameter space. Specifically, we consider models expressed as queuing networks, a formalism which has proven very popular in the software performance engineering community (e.g., [13], [14], [3], [28], [17], [31]). In particular, we study closed queuing network with a single class of users. There is a rather immediate association between the constituent elements of a queuing network and the components of a software system. A service station may either represent a software device (e.g., a web server) or a hardware resource (e.g., CPU or disk). Routing probabilities, indicating the probability with which a job enters a service station upon leaving another station, can represent the operational profile of the workload. In order to model parallelism we allow each station to be composed of multiple, independent and identical servers (e.g., to represent thread concurrency levels for software resources or multiple cores for hardware resources), with the crucial characteristic that the service times are modeled by Coxian distributions. This is a class of distributions that can be informally considered as a "composition" of exponential *stages*.

It has the advantage of being able to approximate any given general distribution arbitrarily closely [16], [32], while keeping the model as a CTMC.

The combination of parallelism in service stations and non-exponential service time distributions prevents the model to enjoy analytical solutions (e.g., product forms [4]). To solve this issue, we consider an alternative solution based on the theory of *fluid limits*, which approximates the model by a system of ordinary differential equations (ODEs) [26]. Its size depends only on the queuing network's topology and not on the number of CTMC states, which instead grow combinatorially with the number of stations and jobs in the queuing network. This is a *sound* approximation in the sense that it is shown to converge to the exact solution when the number of servers and jobs grows to infinity, with excellent accuracy in the case of realistic large-scale systems (e.g., [8], [24], [40]).

The ODE approximation tackles the size scalability problem well; for instance, the number of equations needed to encode Coxian-distributed service times is linear with the number of exponential stages. But, on its own, this does not help tackle the complexity introduced by a large parameter space. This is because the resulting ODEs are typically in a nonlinear form in the most interesting cases. Hence, analytical ODE solutions are out of reach in this context. However, the ODE representation is a decisive factor to arrive at an efficient solution technique that scales well also with large parameter spaces. In particular, we show that the ODE estimates of the network's throughputs can always be obtained as a solution of a system of *linear* equations. Remarkably, these equations can be interpreted as the ODE counterparts of the well-known *traffic equations* for queuing networks [6], [32], which basically establish that the flux of jobs incoming at a service centre must be equal to the outgoing flux of jobs (i.e., those serviced at the centre) in the steady-state.

As a main technical contribution of this paper, we show that the linear structure of these equations is somewhat preserved under certain structural and parameter changes to the original software performance model. Specifically, we provide a language where such changes can be expressed systematically. For this we are inspired by software product line engineering as a highly successful and powerful approach to designing systems exhibiting a high degree of variability, due, for instance, to the number of possible configurations which are possible [29]. Following previous work of some of this authors [25], our reference model is a *Performance Annotated Activity Diagram* (PAAD) supported by a delta-oriented (cf. [30]) language. A PAAD is a formalization of a subset of UML activity diagrams with performance annotations, similarly to those used in the literature e.g., [17], [3], [28], [39].

Parameter spaces are encoded in a variability model generated from a kernel model, the *core*, and a set of deltas. A delta is a list of basic operations such as the addition, removal, or modification of the performance annotations of a node or edge of the PAAD. Overall, a delta transforms the core model into a new model *variant* [30]. Each variant can be solved for its ODE traffic equations one-by-one, thus yielding a naive solution technique for the whole parameter space. Instead, we exploit the commonalities across variants (e.g., nodes whose parameters are never modified, or arcs that are never removed by any delta) to build a single super-model, the so-called *150% model* [34], which keeps track of the concrete variants where each PAAD element can be found with a specific performance annotation. For the 150% model, we construct a system of ODE traffic equations symbolically: all parameters that never change across all variants are treated as constant values; instead, those that are changed at least once are treated as symbols. We then prove that the symbolic solution of the 150% model, called the *family-based solution*, is equal to the solution of any variant whenever we evaluate the symbolic expression with the concrete values related to that variant. This allows for the efficient treatment of large parameter spaces, since the symbolic solution is computed only once for the whole family. Numerical tests show that this approach is one order of magnitude faster than the naive instance-by-instance analysis.

II. Related Work

There has been a considerable amount work on leveraging variability-aware models to improve the efficiency of the analysis. For *qualitative* analyses, examples are feature-aware model-checking algorithms [11] or behavioral equivalences [19], [37]. For quantitative properties, Tawhid and Petriu derive performance models from a UML software product line model [33], but commonalities across model instances are not exploited to provide an efficient family-based analysis. Exteberria *et al.* consider the situation of a feature-aware software performance design in the presence of uncertainty [18]. Each concrete variant is still analyzed in isolation, although its parametric uncertainty is handled through an assumption of monotonicity of the performance indices on the unknown values. In [22], Ghezzi and Sharifloo provide instead a symbolic solution to quantitative analysis of software product lines using a parametric probabilistic model checker [20]. This approach requires the parametric Markov chain to be built. When the focus is on energy/reliability properties as in [22], this is feasible because it involves single-user scenarios, yielding Markov chains of manageable size. For performance analysis, as discussed, the size of the CTMC will grow combinatorially with the number of jobs and servers. We avoid a CTMC representation by considering parametric traffic equations, whose size only depends on the network topology.

We are most closely related to recent previous work by some of this paper's authors [25], where PAADs were introduced for the first time. In [25], the objective was still to exploit commonalities across variants by building a symbolic 150% model solution. From a modeling viewpoint, there are two major limitations in [25], imposed by certain syntactic restrictions on the PAAD which are removed in this paper: (i) activities are assumed to take exponentially distributed service times; (ii) and they are assumed to be performed by computational units without parallelism (i.e., single servers).

The assumptions of [25] led to an underlying PAAD semantics based on Jackson queuing networks (e.g., [6]). The parametric analysis was directly performed on the traffic equations of the network. In this paper we are essentially inspired by an observation from [25] that parametric analysis is possible if the problem is linear with a closed-form solution (such as matrix inversion for the traffic equations). Despite using similar definitions of PAADs and 150% model construction, from a technical viewpoint the present contribution is starkly

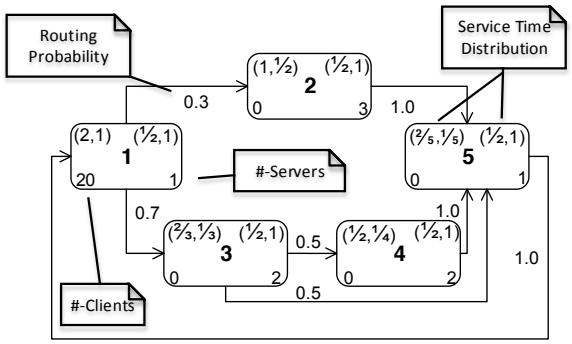

Fig. 1: A Performance Annotated Activity Diagram.

different: (i) with Coxian-distributed multi-server stations, the traffic equations cannot be reused directly for a family-based analysis; (ii) to overcome this issue we turned to an ODE representation of the system; (iii) the ODEs are nonlinear, yet we show that their traffic equations, defined here for the first time to are knowledge, are linear. Through these steps, which involve somewhat nontrivial mathematical analysis, we are finally able to provide a family-based evaluation also for our more expressive PAADs.

III. Overview of Main Results

Using a running example, in this section we present an informal overview of the main contributions of this paper, which will be made formal and precise in later sections.

Model specification using PAADs: A Performance Annotated Activity Diagram (PAAD) is our reference model that can capture UML activity diagrams with performance annotations e.g., [17], [3], [28], [39]. Here, the main restriction is the lack of support for fork/join synchronization barriers, as in [17], [3] (see also [9] for a more general discussion on this). Fig. 1 shows an example PAAD. Each node, labeled with a boldface symbol, represents a service center and is annotated (at its borders) with the following information: service time distribution (top left and right vectors), number of clients at that station in the initial condition (bottom left) and server multiplicity (bottom right). Edges are annotated with probabilities, with the usual interpretation useful to model operational profiles: a job serviced at station **1** will go to station **2** with probability 0.3, else it goes to station **3**.

The only nonstandard annotation concerns the service time distribution, which we now briefly explain. The two vectors provide a canonical representation of a CTMC that describes the service time at the station. The length of either vector gives the number of states of such CTMC (the *stages* of the distribution); the left vector lists the rate of the exponential residence time at each state; the right vector gives the probability with which the service process moves from one state to the next; if such probability is less than one, the service ends. The time between the start of the process in the first state and its exit from any state determines the non-exponential distribution of a service at the station. For instance, Fig. 2 shows a CTMC representation for the Coxian distribution of station **3**. Starting from state 1, the service will be exponentially distributed with rate $2/6 + 2/6 = 2/3$, but with probability $1/2$ it will be followed by a further exponential delay with rate $1/3$. We remark the usefulness of Coxian distributions. For instance, given a service time distribution with given mean E and variance V, if the squared coefficient of variation V/E^2 satisfies $V/E^2 \geq 1/2$ (informally indicating a distribution with high variability), a rate vector $(2/E, E/V)$ and a probability vector $(E^2/2V, 1)$ give a Coxian distribution that matches both the mean and the variance [32].

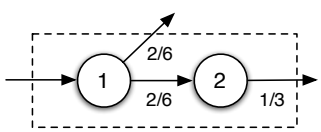

Fig. 2: Coxian CTMC for **3**.

PAAD analysis: As discussed in Section I, a PAAD is interpreted as a queuing network. Since each service centre consists of exponential stages, the whole network is a CTMC. A representative state is given by a *population vector*; each element of the vector provides information about a service centre, giving the number of jobs waiting in the queue and those in each Coxian stage. For instance, the initial state of the CTMC in Fig. 1 will give 0 jobs in stations **2**–**5**, 19 jobs in queue at station **1**, while one job is serviced in stage 1 of the two-stage Coxian distribution. We stress that, on its own, this representation is not convenient because in general such networks can be analyzed only through simulation, hindering an efficient evaluation. However, this semantics corresponds to the so-called *Markov population process* format, e.g., [7]. The most relevant consequence is that the behavior can be approximated by a compact system of ODEs [26]. In this system, there is one variable for each element of the population vector, which provides an estimate of the average number of jobs in that state. For instance, the number of jobs C_1^1 in station **1** that are in stage 1 of the Coxian distribution and the number of the server S_1^5 that are serving the clients in the first stage of the Coxian distribution in station **5** are given by the following ODEs:

$$\dot{C}_1^1 = -2 \cdot \min(C_1^1, S_1^1) + \frac{1}{5} \cdot \min(C_1^5, S_1^5) + \frac{1}{5} \cdot S_2^5 \quad (1)$$

$$\dot{S}_1^5 = -\frac{1}{5} \cdot \min(C_1^5, S_1^5) + \frac{1}{5} \cdot S_2^5 \quad (2)$$

where C_l^i denotes the number of clients in station i in Coxian stage l, whereas S_k^i is the number of servers in station i available for clients in the Coxian stage l. Notably, this ODE is independent of the number of jobs and servers, which only affect the initial condition, while the CTMC semantics grows combinatorially with them, as discussed.

We now briefly discuss how the ODEs (1-2) induce the traffic equation of station **1**. For this, we first note that the negative part of the ODE $\dot{C}_1^1$ estimates the flux-out of station 1, T^1. Consequently, (1) implies that $T^1 = 2 \cdot \min(C_1^1, S_1^1)$. Moreover, we observe that setting the ODEs to zero (requiring an equilibrium condition which must be satisfied in the steady state), and plugging (2) into (1) yields $2 \cdot \min(C_1^1, S_1^1) = \frac{2}{5} \cdot \min(C_1^5, S_1^5)$. Since the negative part of $\dot{C}_1^5$ can be shown to be $\frac{2}{5} \cdot \min(C_1^5, S_1^5)$, the last statement rewrites to $T^1 = r_{5,1} \cdot T^5$, where $r_{5,1}$ denotes the routing probability from station **5** to station **1**. Under the assumption that all derivatives are zero (that is, we are in a steady state), it can be shown that the throughputs satisfy $T^i = \sum_j r_{j,i} T^j$ for all i. These equations,

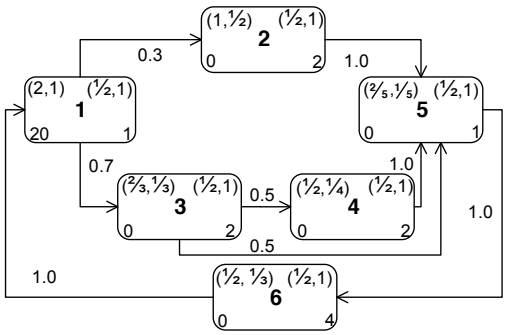

Fig. 3: PAAD variant obtained by applying δ_1.

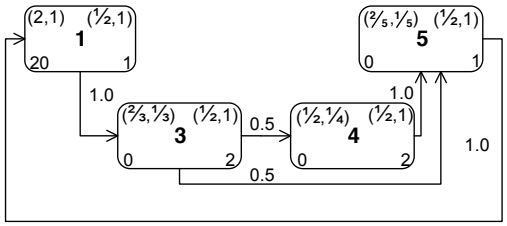

Fig. 4: PAAD variant obtained by applying δ_2.

written for the unknown throughputs, now become linear. A crucial technical result that we establish in this paper is that the throughputs admit a closed form expressions which only depend on the parameters of the queuing network. This will be instrumental to find a symbolic solution for a family of variants induced by our delta-based language, discussed next.

PAAD variability modeling: We use a delta-based language to model parameter spaces. Our formalism generates a family of models starting from a *core* PAAD to which we apply deltas, sequences of basic operations. The operation may: add a new node/edge; modify the annotation of a node/edge; remove a node/edge. We require typical well-formedness conditions that ensure that no dangling nodes are present, or that the sum of all outgoing probabilities from a node is equal to 1. For instance, let us assume that the PAAD of Fig. 1 is the core model, and let us consider the following delta δ_1:

$$\delta_1 = \{\mathsf{rem}\ (5,1,1), \mathsf{add}\ (6,0,4,(1/2,1/3),(1/2,1)),\\ \mathsf{add}\ (5,1,6), \mathsf{add}\ (6,1,1), \mathsf{mod}\ (2,0,2,(1,1/2),(1/2,1))\}.$$

Its overall intent is to introduce an extra node between station **5** and **1**, and to add 1 more server to station **2**. This is implemented by the following sequence of basic operations: removal of the arc between station **5** and **1** [rem $(5,1,1)$]; addition of a node **6** with 4 servers and a some Coxian distribution [add $(6,0,4,(1/2,1/3),(1/2,1))$]; addition of an arc between node **5** and **6** and **6** and **1**, both with probability 1 [add $(5,1,6)$ and add $(6,1,1)$, respectively]; modification of the annotations of node **2** [mod $(2,0,2,(1,1/2),(1/2,1))$]. Figure 3 shows the resulting PAAD after applying δ_1 to the core PAAD. As a further example, let us consider δ_2:

$$\delta_2 = \{\mathsf{rem}\ (1,3/10,2), \mathsf{rem}\ (2,1,5), \mathsf{rem}\ 2,\\ \mathsf{mod}\ (1,7/10,3)\ \mathsf{by}\ 1\}.$$

With similar meaning, the resulting PAAD is shown in Fig. 4.

We remark that typical studies such as sensitivity analysis or what-if scenarios can be encoded in this delta-based framework. For instance, understanding the performance when the server multiplicity at station **1** varies from 1 to 10 can be encoded with 10 different deltas

$$\delta_k = \{\mathsf{mod}\ (1,20,k,(2,1),(1/2,1))\}, \text{with } 1 \leq k \leq 10.$$

Notably, our framework also allows for structural changes in addition to parametric ones.

Each PAAD variant can be solved in isolation, but this does not exploit the commonalities across variants. We now describe how an efficient family-based analysis can be done.

Family-based PAAD analysis: Following [25], we define the 150% as a super-variant that subsumes every concrete variant of the family. We omit the details in this overview; rather, we show in Fig. 5 a graphical representation of the 150% which subsumes the core model in Fig. 1 as well as the two variants originated from δ_1 and δ_2. Notationally, nodes and edges that do not occur in all variants are drawn with dashed lines. Elements that do occur in every variant are marked with solid lines. Labels shown at the bottom-middle of each node indicate the presence conditions. For instance, label c refers to the core model while label δ_1 in node **6** means that the node is only present in the variant obtained by applying δ_1. A similar labeling is used for arcs.

The main result of this paper is that ODE traffic equations can too be written for the 150% PAAD model. To incorporate variability, all elements that are added, removed, or modified by at least one delta are replaced by symbolic expressions. A soundness result guarantees that the symbolic solution of the traffic equations evaluated with the concrete values of a given delta provides the correct throughput estimates.

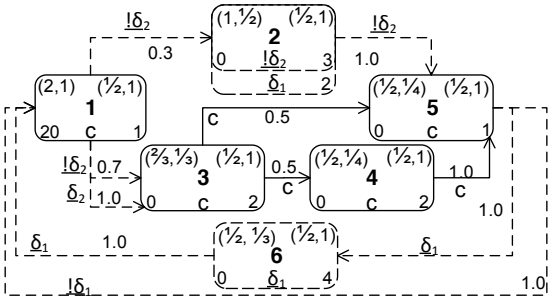

Fig. 5: 150% PAAD for the core of Fig. 1 with δ_1 and δ_2.

IV. Performance-annotated Activity Diagrams

We now begin with the formal definitions, starting from the notion of PAAD.

Definition 1 (Performance-Annotated Activity Diagram)**.** *A* performance-annotated activity diagram *(PAAD) is a tuple* $(\mathcal{V}, \mathcal{E}, \mathcal{C}, \mathcal{S}, \mu, p)$ *where:*

- $\mathcal{V} \subseteq \mathbb{N}$ *is the set of vertices; in the following we shall use* $\mathcal{V} = \{1, 2, \ldots, n\}$*;*
- $\mathcal{E} \subseteq \mathcal{V} \times [0; 1] \times \mathcal{V}$ *is the set of labeled edges;*
- $\mathcal{S} = (S^1, \ldots, S^n) \in \mathbb{N}^n_{>0}$ *is the server multiplicity vector;*
- $\mathcal{C} = (C^1, \ldots, C^n) \in \mathbb{N}^n_{\geq 0}$ *is the initial condition vector;*
- $\mu = (\mu^1, \ldots, \mu^n)$*, with* $\mu^i = (\mu^i_1, \ldots, \mu^i_{m_i}) \in \mathbb{R}^{m_i}_{>0}$ *for* $m_i > 0$ *are the Coxian rate vectors;*
- $p = (p^1, \ldots, p^n)$*, where* $p^i = (p^i_1, \ldots, p^i_{m_i}) \in \mathbb{R}^{m_i}_{\geq 0}$ *with* $p^i_{m_i} = 1$ *for* $1 \leq i \leq n$ *are the Coxian probability vectors.*

For instance, the PAAD of Fig. 1 has $S^1 = 1$, $C^2 = 0$, $m_3 = 3$ and $\mu^3 = (2/3, 1/3)$, and $p^3 = (1/2, 1)$. We remark that m_i is the number of stages of the Coxian distribution for node i; the Coxian vectors for station i are denoted by $\mu^i = (\mu^i_1, \ldots, \mu^i_{m_i})$ and $p^i = (p^i_1, \ldots, p^i_{m_i})$. With this notation, the mean service time at station i is (cf. [32])

$$\mathbb{E}^i = 1/\mu^i_1 + p^i_1/\mu^i_2 + p^i_1 p^i_2/\mu^i_3 + \ldots + p^i_1 \cdot \ldots \cdot p^i_{m_i-1}/\mu^i_{m_i}. \quad (3)$$

We now provide conditions to be enjoyed by a PAAD to yield a meaningful queuing model.

Definition 2 (Well-formedness)**.** *A PAAD is well-formed if and only if the following hold:*

i) $\sum_{(i,r,j') \in \mathcal{E}} r = 1$ *for all* $i \in \mathcal{V}$*;*

ii) Any pair $(i, r, j), (i, r', j) \in \mathcal{E}$ *yields* $r = r'$*. Hence, we let* $r_{i,j}$ *be the unique probability such that* $(i, r_{i,j}, j) \in \mathcal{E}$*.*

Condition *i)* imposes a closed topology: with probability 1 a job serviced at any station goes into some other station; condition *ii)* requires that there be only one arc between any two edges. All examples so far are clearly well formed.

A. Semantics of PAADs

As discussed, we first interpret a PAAD as a closed queuing network with multi-server queues and Coxian distributed service times. A node i is a queue with S^i servers. If the number of jobs in station i at time t, denoted by $C^i(t)$, is less or equal S^i, each job undergoes a service with a Coxian distributed service time that is specified by vectors μ_i and p_i. If, instead, $C^i(t) > S^i$, the number of jobs that are queueing for service at time t is given by $C^i(t) - S^i$. All stations have a first-come-first-served policy. Finally, $R = (r_{i,j})_{1 \leq i,j \leq n}$ is the routing probability matrix, defining with which probability a job in node i, after being serviced, moves to any node j. In the following we assume that R is irreducible, meaning that any two nodes are connected by a path with non-zero probability.

CTMC model: We provide a CTMC model where the state of station i is characterized by the vector $(\mathbf{C}^i_1, \mathbf{S}^i_1, \mathbf{S}^i_2, \ldots, \mathbf{S}^i_{m_i})$ where each element is a nonnegative integer denoting the following (we use boldface symbols for elements of the CTMC state descriptor):

- $\mathbf{C}^i_1$ is the population of jobs that are either waiting for service or which are in service in stage 1;
- $\mathbf{S}^i_1$ is the population of servers which are either idle or servicing jobs in stage 1;
- $\mathbf{S}^i_l$, with $l > 1$, is the population of servers that serve jobs in stage l.

Since the number of servers and jobs coincides at stages $l > 1$, $\mathbf{S}^i_l$ is also the number of jobs that are serviced in stage $l > 1$. Therefore, the number of jobs in station i is given by $\mathbf{C}^i = \mathbf{C}^i_1 + \mathbf{S}^i_2 + \ldots + \mathbf{S}^i_{m_i}$. (As expected, the number of servers in station i is given by the constant $\mathbf{S}^i = \mathbf{S}^i_1 + \mathbf{S}^i_2 + \ldots + \mathbf{S}^i_{m_i}$.) The network state descriptor is a vector from $\mathbb{N}_0^{\sum_i (m_i+1)}$ in the form $\mathbf{X} = (\mathbf{C}^1_1, \mathbf{S}^1_1, \ldots, \mathbf{S}^1_{m_1}, \ldots, \mathbf{C}^n_1, \mathbf{S}^n_1, \ldots, \mathbf{S}^n_{m_n})$. We note that this model can also cover infinite-server (i.e., delay) stations, by choosing the initial number at least equal to the total population of jobs. For a network of size $n \geq 1$, we assume in the following that stations $1, \ldots, \nu$, with $1 \leq \nu \leq n$ are finite-server stations, whereas stations $\nu + 1, \ldots, n$ are delay (i.e., infinite-server) stations.

We now define the transition rates of the CTMC, in the customary form of jump vectors and associated transition functions from a generic state $\mathbf{X}$ [7]. We denote by $q(\mathbf{X}, \mathbf{X}')$ the transition rate from state $\mathbf{X}$ to state $\mathbf{X}'$. Consider some station $1 \leq i \leq n$ and define, for all $1 \leq j \leq n$,

$$\mathbf{X} + h^{i,i}_1 = (\mathbf{C}^i_1 - 1, \mathbf{S}^i_1 - 1, \mathbf{S}^i_2 + 1, \ldots), \quad (4)$$
$$\mathbf{X} + h^{i,j}_1 = (\mathbf{C}^i_1 - 1, \mathbf{C}^j_1 + 1, \ldots), \quad \forall j, j \neq i, \quad (5)$$
$$\mathbf{X} + h^{i,i}_l = (\mathbf{S}^i_l - 1, \mathbf{S}^i_{l+1} + 1, \ldots), \ \forall 2 \leq l \leq m_i - 1, \quad (6)$$
$$\mathbf{X} + h^{i,j}_l = (\mathbf{S}^i_l - 1, \mathbf{S}^i_1 + 1, \mathbf{C}^j_1 + 1, \ldots), \ \forall 2 \leq l \leq m_i,$$

where we use ellipsis to denote all elements of $\mathbf{X}$ which are not affected by a jump. Jump (4) describes a job in stage 1 which moves into stage 2, also causing a server element $\mathbf{S}^i_1$ to decrease and become a server element $\mathbf{S}^i_2$; jump (5) describes a job in stage 1 which completes service and moves into another station j; jump (6) denotes a job in stage l which moves to the next stage, whereas the last equation describes a job that completes service at stage l and moves into station j.

According to this description and notation, the transition rates from any state $\mathbf{X}$ are, for all $1 \leq i, j \leq n$, as follows:

$$q(\mathbf{X}, \mathbf{X} + h^{i,i}_1) = p^i_1 \mu^i_1 \min(\mathbf{C}^i_1, \mathbf{S}^i_1),$$
$$q(\mathbf{X}, \mathbf{X} + h^{i,j}_1) = r_{i,j}(1 - p^i_1)\mu^i_1 \min(\mathbf{C}^i_1, \mathbf{S}^i_1) \ , \forall j \neq i,$$
$$q(\mathbf{X}, \mathbf{X} + h^{i,i}_l) = p^i_l \mu^i_l \mathbf{S}^i_l, \ \forall 2 \leq l \leq m_i - 1,$$
$$q(\mathbf{X}, \mathbf{X} + h^{i,j}_l) = r_{i,j}(1 - p^i_l)\mu^i_l \mathbf{S}^i_l, \ \forall 2 \leq l \leq m_i$$

The CTMC is completely characterized by these transitions, together with the initial condition given by $\mathcal{C}$. We denote the CTMC by $(\mathbf{X}(t))_{t \geq 0}$. Thus, for instance, $\mathbf{C}^i_1(t)$ is the stochastic process describing the number of jobs that are either waiting or in service at stage 1 in station i.

Let us define the stochastic process

$$\Lambda^i(t) := \mu^i_1 \min\left(\mathbf{C}^i_1(t), \mathbf{S}^i_1(t)\right), \quad 1 \leq i \leq n. \quad (7)$$

Informally, each $\Lambda^i(t)$ represents a *contribution* given by the first stage to the throughput at station i at time t. This is because each copy of a server in stage 1 will serve at an

exponential rate μ_1^i, and the total number of jobs that are simultaneously served is the minimum between the number of clients waiting and the the number of servers available. The total throughput at time t is given by summing across the throughputs at all stages. Now, the crucial observation that we make here is that in the steady state, the throughput at stage 1 will equal the *total station throughput*. This is because in the canonical Coxian representation herein used all jobs start service at each station from stage 1. Thus the throughput at stage 1 coincides with the total arrival rate of jobs into station i. So, by definition of steady-state this must coincide with the total exit rate of jobs from station i, which is in turn the definition of throughput.

We make the following observations that summarize and motivate the developments of this paragraph.

- A PAAD can be interpreted as queuing network with CTMC semantics.
- The state space of such CTMC will grow combinatorially with the number of jobs and servers in the network, given by $\mathcal{C}$ and $\mathcal{S}$, because we must enumerate all possible ways in which a fixed number of jobs can be placed across all queues and service stages.
- However, the CTMC is in the so-called Markov population process format (e.g., [7]): that is, each element of the state descriptor vector denotes a population of entities.

ODE model: The last observation is crucial, as we can drop state-space complexity by defining a compact system of ODEs, one for each element of the state descriptor, that provide an estimate of the expected value of the stochastic process. The procedure is based on a fundamental result by Kurtz [26] and is well-known in the literature (see, e.g., [7] and references therein). Here, we limit ourselves to providing the resulting ODEs and briefly explaining their meaning and significance.

Definition 3 (ODE Semantics)**.** *The ODE semantics of a PAAD $(\mathcal{V}, \mathcal{E}, \mathcal{C}, \mathcal{S}, \mu, p)$ is the set of ODEs $\Xi_1 \cup \ldots \cup \Xi_n$, where Ξ_i is defined by*

$$\dot{C}_1^i = -\mu_1^i \min(C_1^i, S_1^i) + \sum_{j=1}^{n} r_{j,i} \Big[(1-p_1^j)\mu_1^j \min(C_1^j, S_1^j) + \\ + \sum_{l=2}^{m_j - 1} (1-p_l^j)\mu_l^j S_l^j + \mu_{m_j}^j S_{m_j}^j \Big] \tag{8}$$

$$\dot{S}_1^i = -p_1^i \mu_1^i \min(C_1^i, S_1^i) + \sum_{l=2}^{m_i - 1} (1-p_l^i)\mu_l^i S_l^i + \mu_{m_i}^i S_{m_i}^i \tag{9}$$

$$\begin{aligned} \dot{S}_2^i &= -\mu_2^i S_2^i + p_1^i \mu_1^i \min(C_1^i, S_1^i) \\ \dot{S}_3^i &= -\mu_3^i S_3^i + p_2^i \mu_2^i S_2^i \\ &\vdots \\ \dot{S}_{m_i}^i &= -\mu_{m_i}^i S_{m_i}^i + p_{m_i-1}^i \mu_{m_i-1}^i S_{m_i-1}^i \end{aligned}$$

and the initial conditions are given by $C_1^i(0) = C^1$, $S_1^i(0) = S^1$, $S_2^i(0) = 0, \ldots, S_{m_i}^i(0) = 0$.

Each set of ODEs Ξ_i refers to the equations for station i. The left-hand sides denote time derivates using the dot notation. Each variable can be interpreted as an estimate of the corresponding stochastic process in the CTMC semantics. For instance $C_1^i(t)$, the solution to the first equation, is an approximation to $\mathbb{E}[\mathbf{C}_1^i(t)]$, the expected value of $\mathbf{C}_1^i(t)$. The results in [26] guarantee that this approximation is sound, in the sense that when the initial population of jobs and servers is large enough, the ODE solution and the expectation of the stochastic process become indistinguishable. The same relationship holds for well-behaved functions of these stochastic processes. In particular, the partial throughputs in (7) satisfy

$$\mathbb{E}\big[\mu_1^i \min\big(\mathbf{C}_1^i(t), \mathbf{S}_1^i(t)\big)\big] \approx \mu_1^i \min\big(C_1^i(t), S_1^i(t)\big). \tag{10}$$

See for instance [40], [38] for a study of these results in software performance models expressed in a process algebra [23], and [35], [36] for an application to layered queuing networks [21].

B. Product-Based Evaluation

We now provide a closed-form solution for the steady-state ODE throughput of a single PAAD. This will be instrumental for the family-based analysis, since we will show that the same computations carry over (symbolically) to the 150% model.

Using (10) and the fact that the stage-1 throughput equals the total station throughput in the steady state, we define the steady-state ODE throughput by $T^i := \mu_1^i \min(C_1^i(\infty), S_1^i(\infty))$, computed by the solutions when $t \to \infty$. In the following, we shall drop (∞) whenever its usage is obvious from the context. We proceed with our analysis by noting that any ODE steady state of Ξ solves the (nonlinear) system of equations obtained by setting each ODE of Ξ to zero. In particular, by setting all derivatives to zero, we infer by plugging (9) into (8) that $T^i = \sum_{1 \le j \le n} r_{j,i} T^j$ for all $1 \le i \le n$. This becomes, in matrix notation with $(\cdot)^T$ denoting matrix transposition,

$$R^T T = T, \qquad T = (T^1, \ldots, T^n)^T. \tag{11}$$

This system is the ODE counterpart of the well-known traffic equations [6], which relate station throughputs through the routing probability matrix. Note that the steady state throughputs T^i cannot be recovered from these equations. This is because, by irreducibility of R, they are known only up to a scaling factor [6]. Therefore, from (11) we can only deduce the *throughput ratios*, i.e. the values $T^1/T^1, \ldots, T^n/T^1$. Fortunately, by exploiting the closedness of the system and the nature of the ODE semantics, it is possible to show that (11) determines the ODE throughputs T of a PAAD.

Theorem 1. *Fix a PAAD $(\mathcal{V}, \mathcal{E}, \mathcal{C}, \mathcal{S}, \mu, p)$ and consider its underlying ODE semantics Ξ. Further, let $(1, \zeta^2, \ldots, \zeta^n)^T$ denote the unique solution of (11), and assume that the solution to Ξ converges towards an equilibrium. Then, whenever it holds that*

$$S^i > \frac{\mathbb{E}^i \zeta^i}{\sum_{j=1}^n \mathbb{E}^j \zeta^j} \sum_{j=1}^{n} C^j, \quad \textit{for all} \quad 1 \le i \le \nu, \tag{12}$$

the ODE steady state throughputs are given by $T^i = \zeta^i (\sum_{j=1}^n \mathbb{E}^j \zeta^j)^{-1} \sum_{j=1}^n C^j$ for all $1 \le i \le n$. If, instead, (12) is violated, the throughputs are given by $T^i = \zeta^i \min\Big\{ S^j (\zeta^j \mathbb{E}^j)^{-1} \;\Big|\; 1 \le j \le \nu \Big\}$, where $1 \le i \le n$.

For instance, in the PAAD model of Fig. 1 we get that $(T^1, T^2, T^3, T^4, T^5) = (0.20, 0.59, 0.14, 0.07, 0.20)$.

The above theorem is our first major result because it provides a closed form expression for the ODE steady state throughputs. It allows to propose the following definition, which permits the analysis of a (well-formed) PAAD. We call this *product-based (PB) evaluation* as it concerns a given, concrete PAAD, unlike the *family-based (FB) evaluation*, which will be discussed later.

Definition 4 (Product-based evaluation)**.** *The product-based (PB) evaluation of a PAAD is given by the expressions for T derived in Theorem 1.*

Note that the above definition is known to coincide with the steady state throughputs of a PAAD only if the underlying ODE system Ξ converges towards an equilibrium. While a formal proof of this is out of reach (the common approach via Lyapunov stability theory fails essentially because Ξ admits several equilibrium points in general), we conjecture that an ODE systems induced by a PAAD fulfills always this condition.

V. Variability of PAADs

Analyzing PAAD involves solving a system of linear equations (11). Here we wish to exploit commonalities in a variability model in order to perform this calculation only once on the 150% model.

We start by formally defining the notion of delta for a PAAD. To this end, we first recall that any vector can be seen as a function on natural numbers. For instance, an $x \in \mathbb{R}^2$ is a function $x : \{1, 2\} \to \mathbb{R}$ such that $x = (x(1), x(2))$. This view becomes usefull whenever the nodes $\mathcal{V}$ of a PAAD form not $\{1, \ldots, n\}$ but an arbitrary finite set of $\mathbb{N}$. This can easily happen, for instance, when nodes are removed or replaced by other nodes. With this in mind, we interpret in the following vectors as functions and identify S^i, C^i, μ^i and p^i by $S(i)$, $C(i)$, $\mu(i)$ and $p(i)$, respectively.

Definition 5 (PAAD deltas)**.** *A PAAD delta is a set of delta operations $\delta \subseteq Op$, where*

$$
\begin{aligned}
Op = \{&\mathsf{add}\,(i, \mathcal{C}(i), \mathcal{S}(i), \mu(i), p(i)) \mid i \in \mathbb{N}, \mathcal{C}(i) \in \mathbb{N}_{\geq 0},\\
&\quad \mathcal{S}(i) \in \mathbb{N}_{>0} \textit{ and } \mu(i), p(i) \textit{ define a Coxian distribution}\}\\
&\cup \{\mathsf{rem}\; e \mid e \in \mathbb{N} \times (0; 1] \times \mathbb{N}\}\\
&\cup \{\mathsf{add}\,(i, r_{ij}, j) \mid i, j \in \mathbb{N}, r_{ij} > 0\} \cup \{\mathsf{rem}\; i \mid i \in \mathbb{N}\}\\
&\cup \{\mathsf{mod}\,(i, r_{ij}, j)\ \mathsf{by}\ \tilde{r}_{ij} \mid (i, r_{ij}, j) \in \mathcal{E}, \tilde{r}_{ij} > 0\}\\
&\cup \{\mathsf{mod}\,(i, \mathcal{C}(i), \mathcal{S}(i), \mu(i), p(i))\ \mathsf{by}\\
&\qquad (\tilde{\mathcal{C}}(i), \tilde{\mathcal{S}}(i), \tilde{\mu}(i), \tilde{p}(i)) \mid \tilde{\mathcal{C}}(i) \in \mathbb{N}_{\geq 0}, \tilde{\mathcal{S}}(i) \in \mathbb{N}_{>0}\\
&\qquad\quad \textit{and } \tilde{\mu}(i), \tilde{p}(i) \textit{ define a Coxian distribution}\}
\end{aligned}
$$

For simplicity, in this paper we assume that each variant is induced by a single delta operation. This is without loss of generality, since a sequence of deltas can be combined into a single composite delta through the definition of an appropriate delta composition operation.

Up to this point, it is not ensured that the application of such a delta also leads to a well-formed PAAD variant. To this end, we require that any delta must be *applicable* and *consistent*. A delta is applicable to a PAAD if all elements (nodes including all associated values and edges with probabilities), which should be removed or modified in this delta, already exist in the variant. Similarly, it is forbidden to add elements that already exist in a PAAD. Finally, the underlying routing matrix has to remain irreducible, meaning that the probabilities of the outgoing edges of a node have to sum up to one. A delta is consistent if it adds, removes or modifies each element at most once [30]. Moreover, we require that a delta does not lead to loose edges in the resulting variant. That is, a delta that removes a node must also remove the edges connected to the node at some point. Edges are never added between nodes that are removed in the delta. If a node of an added edge does not exist in the core PAAD, the necessary source and/or target edges are also added in the delta.

The following definition formalizes how to obtain an arbitrary variant through application of a delta to a PAAD; see, e.g., Figs. 3–4 in Section III.

Definition 6 (PAAD delta application)**.** *The application of an applicable and consistent delta $\delta \subseteq Op$ to a PAAD $= (\mathcal{V}, \mathcal{E}, \mathcal{C}, \mathcal{S}, \mu, p)$ is defined by the function PAAD$' =$ apply(PAAD, δ), where PAAD$' = (\mathcal{V}', \mathcal{E}', \mathcal{C}', \mathcal{S}', \mu', p')$. It is recursively defined as follows.*

1) *Case $\delta = \emptyset$: PAAD$' =$ PAAD.*
2) *Case: $\delta = \delta' \cup \delta'' \wedge \delta', \delta'' \in Op$: PAAD$' =$ apply(apply(PAAD, δ'), δ'').*
3) *Case: $\delta = \mathsf{add}\,(i, \mathcal{C}(i), \mathcal{S}(i), \mu(i), p(i))$:*

$$
\begin{aligned}
&\mathcal{C}' = \mathcal{C} \cup \{(i, \mathcal{C}(i))\} \quad \mathcal{S}' = \mathcal{S} \cup \{(i, \mathcal{S}(i))\} \quad \mathcal{V}' = \mathcal{V} \cup \{i\}\\
&\mu' = \mu \cup \{(i, \mu(i))\} \quad p' = p \cup \{(i, p(i))\}
\end{aligned}
$$

4) *Case: $\delta = \mathsf{add}\,(i, r_{ij}, j)$: $\mathcal{E}' = \mathcal{E} \cup \{(i, r_{ij}, j)\}$.*
5) *Case: $\delta = \mathsf{rem}\; i$: $\mathcal{V}' = \mathcal{V} \setminus \{i\}$.*
6) *Case: $\delta = \mathsf{rem}\; e$: $\mathcal{E}' = \mathcal{E} \setminus \{e\}$.*
7) *Case: $\delta = \mathsf{mod}\,(i, r_{ij}, j)\ \mathsf{by}\ \tilde{r}_{ij}$: $\mathcal{E}' = \big(\mathcal{E} \setminus \{(i, r_{ij}, j)\}\big) \cup \{(i, \tilde{r}_{ij}, j)\}$.*
8) *Case: $\delta = \mathsf{mod}\,(i, \mathcal{C}(i), \mathcal{S}(i), \mu(i), p(i))$ $\mathsf{by}\ (\tilde{\mathcal{C}}(i), \tilde{\mathcal{S}}(i), \tilde{\mu}(i), \tilde{p}(i))$:*

$$
\begin{aligned}
\mathcal{C}' &= \big(\mathcal{C} \setminus \{(i, \mathcal{C}(i))\}\big) \cup \{(i, \tilde{\mathcal{C}}(i))\}\\
\mathcal{S}' &= \big(\mathcal{S} \setminus \{(i, \mathcal{S}(i))\}\big) \cup \{(i, \tilde{\mathcal{S}}(i))\}\\
\mu' &= \big(\mu \setminus \{(i, \mu(i))\}\big) \cup \{(i, \tilde{\mu}(i))\}\\
p' &= \big(p \setminus \{(i, p(i))\}\big) \cup \{(i, \tilde{p}(i))\}
\end{aligned}
$$

Note that Case 3) in the above definition exploits the fact that a (total) function $f : A \to B$ is a subset of $A \times B$, i.e. $f = \{(x, f(x)) \mid x \in A\}$. Also, since we have assumed that our deltas are applicable, we know that adding, for instance, $(i, \mathcal{C}(i))$ to $\mathcal{C}$ will give rise to a well-defined function as $i \notin \mathcal{V}$. The total application order of a delta set can be obtained by, e.g., adding an ordered list to a delta definition including the names of all previously required deltas as it is done in [5].

Let us consider a *core* PAAD with a set of deltas Δ. As stated earlier, the 150%-model is an over-saturated PAAD

consisting of all nodes and transitions that are added or modified by some $\delta \in \Delta$. Although, it is not a valid PAAD variant in general, we have the necessary information to derive a specific variant of the considered system. This information is stored in the 150%-model with the help of a labeling function $\mathcal{L}$. The definition is inspired by [25].

Definition 7 (150%-model)**.** *Let* $PAAD_c = (\mathcal{V}_c, \mathcal{E}_c, \mathcal{C}_c, \mathcal{S}_c, \mu_c, p_c)$ *be the* core model *and* Δ *be a set of consistent and applicable deltas. Let* $L = \{c\} \cup \{\underline{\delta}, \underline{!\delta} \mid \delta \in \Delta\}$ *be the set of labels. The* 150%-model *is* $PAAD_{150} = (\mathcal{V}_{150}, \mathcal{E}_{150}, \mathcal{C}_{150}, \mathcal{S}_{150}, \mu_{150}, p_{150}, \mathcal{L})$*, where:*

$$\mathcal{V}_{150} = \mathcal{V}_c \cup \{i \mid \exists \delta \in \Delta : \textit{add}\,(i, \mathcal{C}(i), \mathcal{S}(i), \mu(i), p(i)) \in \delta\},$$
$$\mathcal{E}_{150} = \mathcal{E}_c \cup \{(i, r_{ij}, j) \mid \exists \delta : \textit{add}\,(i, r_{ij}, j) \in \delta \vee \textit{mod}\,(i, r_{ij}, j)\ \textit{by}\ r'_{ij} \in \delta\}.$$

For any $X \in \{\mathcal{C}, \mathcal{S}, \mu, p\}$*, we define the partial functions*

$$X_{150} : \mathcal{V}_{150} \times L \to \mathbb{R}_{\geq 0}, X_{150}(i, l) = \begin{cases} X_c(i) & \text{if } l = c \wedge i \in \mathcal{V}_c, \\ X(i) & \text{if } l = \underline{\delta} \wedge \textit{add}\,(i, \mathcal{C}(i), \mathcal{S}(i), \mu(i), p(i)) \in \delta, \\ X'(i) & \text{if } l = \underline{\delta} \wedge \textit{mod}\,(i, \mathcal{C}(i), \mathcal{S}(i), \mu(i), p(i)) \\ & \quad \textit{by}\,(\mathcal{C}'(i), \mathcal{S}'(i), \mu'(i), p'(i)) \in \delta, \\ 0 & \text{if } l = \underline{!\delta} \wedge \textit{rem}\ i \in \delta, \end{cases}$$

At last, $\mathcal{L}$ *is the* labeling function *defined as*

$$\mathcal{L} : \mathcal{V}_{150} \cup \mathcal{E}_{150} \to 2^L,$$
$$\mathcal{L}(i) = \begin{cases} c & \text{if } i \in \mathcal{V}_c, \\ \emptyset & \text{otherwise,} \end{cases} \cup \{\underline{\delta} \mid \textit{add}\,(i, \mathcal{C}(i), \mathcal{S}(i), \mu(i), p(i)) \in \delta\} \cup \{\underline{!\delta} \mid \textit{rem}\ i \in \delta\}.$$
$$\mathcal{L}(e) = \begin{cases} c & \text{if } e \in \mathcal{E}_c, \\ \emptyset & \text{otherwise,} \end{cases} \cup \{\underline{\delta} \mid \textit{add}\ e \in \delta\} \cup \{\underline{!\delta} \mid \textit{rem}\ e \in \delta\} \cup \{\underline{\delta} \mid \textit{mod}\,(i, r_{ij}, j)\ \textit{by}\ r'_{ij} \in \delta \wedge e = (i, r'_{ij}, j)\} \cup \{\underline{!\delta} \mid \textit{mod}\,(i, r_{ij}, j)\ \textit{by}\ r'_{ij} \in \delta \wedge e = (i, r_{ij}, j)\}.$$

The 150%-model is based on all nodes $\mathcal{V}_{150}$ and edges $\mathcal{E}_{150}$ that are either part of the core model or added in delta. Delta modeling allows us to change the probabilities on existing edges as well. For each modified probability, we have to add an additional edge into the 150%-model to ensure that no information is lost. As a result, we have an edge with the old probability and an edge with the modified one. The domain of the partial function X_{150}, with $X \in \{\mathcal{C}, \mathcal{S}, \mu, p\}$, is a pair, where the first element indicates the node or edge that is labeled and the second parameter specifies a delta label. In addition, the labeling function $\mathcal{L}$ keeps track of the deltas that affect a node or an edge. For instance, $\mathcal{L}(e) = \{c, \underline{!\delta}\}$ says that the edge e is present in the core model and is removed by δ. Instead, $\mathcal{L}(e) = \{\delta\}$ expresses the fact that e is not present in the core model and is added by δ. The complexity of constructing a 150%-model is linear with the number of deltas. Given our labeling function, we can iterate through all deltas in a sequential order and add the information about their specific operations to the core model, which ultimately results in the 150%-model. A prerequisite is that the set of deltas is consistent, which should be ensured beforehand. However, the full construction process can be automated and does not necessarily require expert knowledge.

VI. Family-based Evaluation

We now discuss the symbolic evaluation of the previously constructed 150% model in detail. The notion of *concretization* allows to obtain a specific variant from the 150% model.

Definition 8. *Let* $PAAD_{150}$ *be a 150% model. The concretization* $PAAD_\delta = (\mathcal{V}_\delta, \mathcal{E}_\delta, \mathcal{C}_\delta, \mathcal{S}_\delta, \mu_\delta, p_\delta)$ *of* $PAAD_{150}$ *for* $\delta \in \Delta$ *is given by*

$$\mathcal{V}_\delta = \{i \in \mathcal{V}_{150} : (c \in \mathcal{L}(i) \wedge \underline{!\delta} \notin \mathcal{L}(i)) \vee \underline{\delta} \in \mathcal{L}(i)\}$$
$$\mathcal{E}_\delta = \{e \in \mathcal{E}_{150} : (c \in \mathcal{L}(e) \wedge \underline{!\delta} \notin \mathcal{L}(e)) \vee \underline{\delta} \in \mathcal{L}(e)\}$$
$$X_\delta(i) = \begin{cases} X_{150}(i, \underline{\delta}) & \textit{if defined,} \\ X_{150}(i, c) & \textit{if defined and } X_{150}(i, \underline{\delta}) \textit{ is not defined,} \\ 0 & \textit{otherwise,} \end{cases}$$

where $X \in \{\mathcal{C}, \mathcal{S}, \mu, p\}$ *and* $i \in \mathcal{V}_\delta$*.*

The next theorem states that the concretization of a 150%-model with respect to a given delta coincides with the application of the delta to the core model. This statement will be needed later to relate the PB evaluation to the FB evaluation.

Theorem 2. *Let* $PAAD_{150}$ *be a 150% model and* $\delta \in \Delta$*. Then, it holds that* $PAAD_\delta = apply(PAAD_c, \delta)$*. Let* R_δ^T *be the routing probability matrix of* $PAAD_\delta$ *and* $R_\delta^T T = T$ *be its traffic equations. Denote by* ζ_δ *the unique solution of the traffic equations when* $\zeta_\delta^1 = 1$*. Then, the PB evaluation of* $apply(PAAD_c, \delta)$ *is given by* T_δ^i*, for* $i \in \mathcal{V}_\delta$*, where*

$$T_\delta^i = \begin{cases} \zeta_\delta^i (\sum_{j=1}^n \mathbb{E}_\delta^j \zeta_\delta^j)^{-1} \sum_{j=1}^n \mathcal{C}_\delta(j) & \textit{if (12)} \\ \zeta_\delta^i \min\{\mathcal{S}_\delta(j)(\zeta^j \mathbb{E}_\delta^j)^{-1} \mid \\ \quad \textit{with } j \in \mathcal{V}_{150} \textit{ such that } \mathcal{S}_\delta(j) < \infty\} & \textit{else} \end{cases},$$

Instead of solving the traffic equations $R_\delta^T \zeta_\delta = \zeta_\delta$ for each variant PAAD_δ separately, we provide a closed form solution for all traffic equations of the family. The key idea relies on the introduction of a routing matrix R_s on the set $\mathcal{V}_{150}$ with entries that are either constants or symbolic variables.

Theorem 3. *Let* $PAAD_{150}$ *be a 150% model and* $\delta \in \Delta$*, and assume without loss of generality that* $1 \in \mathcal{V}_c$*. Let* $R_s = (r_{i,j}^s)_{i,j \in \mathcal{V}_{150}}$ *denote the 150% routing matrix, where*

$$r_{i,j}^s = \begin{cases} r & \text{if } \exists e = (i, r, j) \in \mathcal{E}_{150} \wedge \mathcal{L}(e) = \{c\}, \\ 0 & \text{if } \nexists e = (i, r, j) \in \mathcal{E}_{150}, \\ r_{i,j}^* & \text{otherwise} \end{cases}$$

Let r^* *denote the vector of symbolic variables present in* R_s*. Let* ζ_s *denote the* symbolic *solution of* $R_s^T \zeta_s = \zeta_s$ *with* $\zeta_s^1 = 1$*. Then, it holds that* $\zeta_s^i = \zeta_\delta^i$ *for all* $i \in \mathcal{V}_\delta$ *if* r^* *is such that*

$$r_{i,j}^s = \begin{cases} (R_\delta)_{i,j} & \text{if } i, j \in \mathcal{V}_\delta \\ 0 & \text{otherwise.} \end{cases}$$

Let us illustrate Theorem 3 using the 150% model of Fig. 5.

The symbolic routing matrix R_s is given by:

$$R_s = \begin{pmatrix} 0.0 & r^*_{1,2} & r^*_{1,3} & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & r^*_{2,5} & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.5 & 0.5 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 \\ r^*_{5,1} & 0.0 & 0.0 & 0.0 & 0.0 & r^*_{5,6} \\ r^*_{6,1} & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{pmatrix}$$

Its unique solution ζ_s when the first component is equal to one, is given by

$$\zeta_s = \big(1, r^*_{1,2}, r^*_{1,3}, r^*_{1,3}/2, r^*_{2,5}r^*_{1,2}+r^*_{1,3}, r^*_{5,6}(r^*_{2,5}r^*_{1,2}+r^*_{1,3})\big).$$

The theorem states that evaluating ζ_s with the probability values of a delta δ coincides with the product-based solution of the concrete variant obtained by applying δ to the core model. For instance, instantiating R_s with the values of δ_2 yields

$$R_s(r_{\delta_2}) = \begin{pmatrix} 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.5 & 0.5 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 \\ 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{pmatrix}$$

while applying δ_2 to the core give the routing matrix

$$R_{\delta_2} = \begin{pmatrix} 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.5 & 0.5 \\ 0.0 & 0.0 & 0.0 & 1.0 \\ 1.0 & 0.0 & 0.0 & 0.0 \end{pmatrix}$$

We can observe that the elimination of rows 2 and 5 and columns 2 and 5 from $R_s(r_{\delta_2})$ gives rise to R_{δ_2}. More importantly, eliminating the coordinates 2 and 5 of $\zeta_s(r_{\delta_2})$ will give us the unique solution of the traffic equations $R^T_{\delta_2}\zeta_{\delta_2} = \zeta_{\delta_2}$. A similar result can be obtained for δ_1.

Overall, by solving symbolically $R^T_s\zeta_s = \zeta_s$, we have a closed form solution for the whole family. This is due to Theorem 2, which gives the throughput of each variant directly in terms of ζ_δ and the model parameters. This underpins the importance of our approach in the case where the number of variants is large, see also Section VII. For instance, the steady state throughput for node 1 will be given by either of the following symbolic expressions:

- If (12) holds, T_1 is calculated with:

$$T_1 = 20\big(10r^*_{1,3} + \mathbb{E}^*_2 r^*_{1,2} + 5r^*_{1,2}r^*_{2,5} + \mathbb{E}^*_6 r^*_{5,6}(r^*_{1,3} + r^*_{1,2}r^*_{2,5}) + 1\big)^{-1},$$

  where $\mathbb{E}^*_i$ is the symbolic expression corresponding to the average service time of node i, computed according to (3).

- Otherwise, if (12) does not hold, T_1 is the minimum of

$$\left\{1, \frac{S^*_2}{\mathbb{E}^*_2 r^*_{1,2}}, \frac{2}{3r^*_{1,3}}, \frac{1}{r^*_{1,3}}, \frac{1}{5r^*_{1,3} + 5r^*_{1,2} * r^*_{2,5}}, \frac{S^*_2}{\mathbb{E}^*_6 r^*_{5,6}(r^*_{1,3} + r^*_{1,2}r^*_{2,5})}\right\} \quad (13)$$

Notice that also condition (12) can be treated symbolically. Hence, for instance, a performance analysis of the core PAAD will result in a violation of (12), and we will look for the minimum in (13). It turns out that the minimum is given by the fifth element of the set in (13), which gives us 0.20. This is identical to the product-based solution in Section IV. The last element is not relevant for the core PAAD, since it refers to node 6 which is introduced by δ_1.

VII. Numerical Evaluation

We compared the PB approach, where each variant is solved numerically for a given set of concrete parameter values, against the FB analysis in terms of execution times. The experimental set-up was as follows. We considered randomly generated 150%-models of PAADs with sizes between 16 and 430 nodes. Additionally, we varied the number of symbolic variables for each network size to observe the impact on the symbolic evaluation time. Each symbol represents a parameter that may take any value in the reals. Hence, even a single symbol in the system can produce an infinite number of variants. We wish to verify the hypothesis that a FB analysis is increasingly more efficient considering larger networks.

In particular, let n be the number of nodes (equivalent to the network size) in the evaluation, r denotes the number of symbols in the routing probability matrix, E represents the number of symbols in the mean service times and S stands for the number of symbols in the servers. For any given selection of (n, r, E, S), all other parameters were randomly generated given the following boundaries. The mean service time at each node ranged from $[1.0; 10.0]$, the servers were chosen from $[1; 20]$ for each node and the jobs were set to 50 circulating simultaneously through the network. For instance, the symbolic evaluation of the running example corresponds to a configuration with $(n, r, E, S) = (6, 6, 2, 2)$.

For each tuple (n, r, E, S), we then calculated the steady state throughputs for 200 randomly generated variants using the both FB and the PB analysis. In detail, we randomly generated 200 vector of parameters, each of size $r + E + S$, representing a specific realization of the symbolic parameters in the 150% model. In the FB analysis, each vector was used to evaluate the symbolic expressions of the steady state throughputs as in Theorem 3. Instead, the PB approach uses the concrete parameters to numerically solve the system of equations (11) and calculate the steady state throughputs as in Theorem 1 for each variant in isolation. We measured the execution runtimes for both the FB and the PB analysis methods. The experiment was repeated three times to reduce noise in the results. The case study was executed in Matlab R2014a, using the *Symbolic Math Toolbox* for the FB analysis. The results were computed on an Intel Core i7 4.50 GHz with 16 GB RAM. For replicability, the Matlab code is available at https://www.isf.cs.tu-bs.de/data/Matlab_Experiment.zip.

Table 1 reports the resulting execution times for the analysis of the 200 different variants, averaged across the three repetitions of the experiment for both the FB and PB analyses. We calculated the speed-up ratio PB/FB for each tuple configuration and the time taken to symbolically solve the system of linear equations (last column, *SS*). We were able to make the following observations based on these results.

- The FB analysis is always more effective compared to the PB approach, with speed-ups up to one order of magnitude

TABLE I: Numerical results.

Variables				*Runtimes (s)*			
n	r	E	S	*FB*	*PB*	*PB/FB*	*SS*
16	1	0	0	0.008	0.021	2.625	0.442
16	0	1	0	0.008	0.021	2.625	0.240
16	0	0	1	0.008	0.021	2.625	0.209
16	1	1	1	0.008	0.021	2.625	0.261
16	2	2	2	0.009	0.021	2.333	0.350
16	0	6	0	0.008	0.021	2.625	0.222
16	8	8	8	0.009	0.022	2.444	0.776
16	16	16	16	0.009	0.022	2.444	9.131
32	1	0	0	0.015	0.035	2.333	0.554
32	0	1	0	0.015	0.032	2.133	0.315
32	0	0	1	0.014	0.032	2.285	0.319
32	1	1	1	0.015	0.032	2.133	0.307
32	2	2	2	0.015	0.033	2.200	0.790
32	0	6	0	0.014	0.032	2.285	0.297
32	8	8	8	0.014	0.032	2.285	0.416
32	16	16	16	0.015	0.033	2.200	6.813
142	1	0	0	0.125	0.207	1.656	1.334
142	0	1	0	0.116	0.203	1.750	1.007
142	0	0	1	0.100	0.195	1.950	1.042
142	1	1	1	0.085	0.187	2.200	1.018
142	2	2	2	0.087	0.179	2.057	1.039
142	0	6	0	0.092	0.174	1.891	1.029
142	8	8	8	0.096	0.204	2.125	1.172
142	16	16	16	0.089	0.161	1.808	1.149
430	1	0	0	0.078	0.844	10.82	4.304
430	0	1	0	0.078	0.848	10.87	4.278
430	0	0	1	0.081	0.864	10.66	4.359
430	1	1	1	0.078	0.884	11.33	4.445
430	2	2	2	0.079	0.864	10.93	4.408
430	0	6	0	0.084	0.869	10.34	4.380
430	8	8	8	0.081	0.874	10.79	4.449
430	16	16	16	0.080	0.885	11.06	4.436

that increase with the size of the 150% model n. The symbolic expressions are only computed once for the whole family; thus, the time needed to compute them (column *SS*) can be neglected with an increasing number of variants. In addition, in two cases we calculated the break-even point intended as the number of variants have to be analyzed numerically to match the same cumulative analysis time of the symbolic analysis. For the first row, we have to analyze 6802 variants, while for the last row the break-even point is at 1104 variants. We remark that these many evaluations may be done in large optimization problems, for instance.

- For a fixed n, the impact of varying the number parameters r, E, and S seems to be quite negligible on the FB and PB execution runtimes. This is because theese parameters do not affect the PB runtimes, since all values are concrete. Instead, the small range of FB execution runtimes indicates that, although the symbolic expressions become larger, the symbolic engine is quite effective at evaluating them.

- On the other hand, varying the number parameters r, E, and S for a fixed value of n seem to have a large impact on the symbolic process of solving the family based model once (column *SS*). In particular, the numerical results suggest that it depends on the ratio between the size of the 150% model and the number of symbolic parameters. For instance, the configuration $(16, 16, 16, 16)$ takes almost 12 times longer to compute compared to the configuration $(16, 8, 8, 8)$; however, for network sizes of 142 and 430 the difference between the last two rows becomes negligible.

- Speed-ups of up to two orders of magnitude were reported in [25], using a similar experimental set-up for Jackson-type networks. We explain the lower effectiveness found here by the more complex formulae needed for the ODE approximation, see Theorem 1. Similarly to [25], however, we observe the general trend that the FB analysis outperforms the PB analysis in the case of large product lines.

VIII. Conclusion

We studied the combined problem of solving complex software performance models, multiple times. Complexity is caused by an expressive model that features two important characteristics: support for general (i.e., non-exponential) distributions, in order to more appropriately capture more realistic durations of activities; and the ability to express concurrency levels. This makes the analysis of a single model difficult, preventing an effective use when a large number of distinct evaluations are needed. This is relevant because such a situation presents itself in a number of different occasions in model-based software performance engineering, ranging from what-if scenarios, capacity planning, and uncertainty.

We have tackled this problem in the case of so-called performance-annotated activity diagrams, understood as closed single-class queuing networks with multiple-server stations and Coxian distributed service times, which can capture the two aforementioned desired modeling features. Exploiting software product line ideas and techniques, our *family-based* analysis leverages the commonalities across variants in terms of both parametric changes, which only affect the values of certain performance annotations, and structural changes, which may affect the topology of the model by addition or removal of nodes and arcs. The numerical tests have demonstrated that this analysis is increasingly convenient for larger-scale models.

Our approach represents a significant advance with respect to the state of the art [25], which was limited to queuing networks with exponential single-server stations. More important, the techniques used here are entirely novel and seem to provide a more general route to finding family-based solutions. This involves first casting the analysis question into a linear problem of "manageable" size, i.e., independent from the state-space size of the model; then, it has to be proven that linearity is preserved by the structural modifications made by deltas. In this more general view, the work of [25] is a rather straightforward application of this strategy, the analysis already being a linear problem to start with.

In future work we wish to test the hypothesis that this route is applicable to more expressive formalisms. For instance, we would like to remove the limitation of the lack of support for fork/join synchronization barrier, and extend family-based analyzed to extended queuing networks such as layered queues.

Acknowledgement: This work was partially supported by the DFG (German Research Foundation) under the Priority Programme SPP1593: Design For Future — Managed Software Evolution, and by the EU project QUANTICOL, 600708.

REFERENCES

[1] A. Aleti, B. Buhnova, L. Grunske, A. Koziolek, and I. Meedeniya, "Software architecture optimization methods: A systematic literature review," *IEEE Trans. Software Eng.*, vol. 39, no. 5, pp. 658–683, 2013.

[2] S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni, "Model-based performance prediction in software development: A survey," *IEEE Trans. Software Eng.*, vol. 30, no. 5, pp. 295–310, 2004.

[3] S. Balsamo and M. Marzolla, "Performance evaluation of UML software architectures with multiclass queueing network models," in *WOSP*, 2005, pp. 37–42.

[4] F. Baskett, K. M. Chandy, R. R. Muntz, and F. G. Palacios, "Open, Closed, and Mixed Networks of Queues with Different Classes of Customers," *J. ACM*, vol. 22, no. 2, pp. 248–260, 1975.

[5] L. Bettini, F. Damiani, and I. Schaefer, "Compositional type checking of delta-oriented software product lines," *Acta Informatica*, vol. 50, no. 2, pp. 77–122, 2013.

[6] G. Bolch, S. Greiner, H. de Meer, and K. Trivedi, *Queueing networks and Markov chains: modeling and performance evaluation with computer science applications*. Wiley, 2005.

[7] L. Bortolussi, J. Hillston, D. Latella, and M. Massink, "Continuous approximation of collective system behaviour: A tutorial," *Performance Evaluation*, vol. 70, no. 5, pp. 317–349, 2013.

[8] M. Bravetti, S. Gilmore, C. Guidi, and M. Tribastone, "Replicating Web Services for Scalability," in *TGC*, ser. LNCS, vol. 4912. Springer, 2007.

[9] F. Brosig, P. Meier, S. Becker, A. Koziolek, H. Koziolek, and S. Kounev, "Quantitative evaluation of model-driven performance analysis and simulation of component-based architectures," *IEEE Trans. Software Eng.*, vol. 41, no. 2, pp. 157–175, Feb 2015.

[10] M. Caporuscio, A. D. Marco, and P. Inverardi, "Model-based system reconfiguration for dynamic performance management," *Journal of Systems and Software*, vol. 80, no. 4, pp. 455–473, 2007.

[11] A. Classen, P. Heymans, P.-Y. Schobbens, A. Legay, and J.-F. Raskin, "Model checking lots of systems: Efficient verification of temporal properties in software product lines," in *ICSE*. IEEE, 2010.

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

[13] V. Cortellessa and R. Mirandola, "Deriving a queueing network based performance model from uml diagrams," in *WOSP*, 2000, pp. 58–70.

[14] ——, "PRIMA-UML: a performance validation incremental methodology on early UML diagrams," *Science of Computer Programming*, vol. 44, no. 1, pp. 101–129, 2002.

[15] M. E. Crovella and A. Bestavros, "Self-similarity in world wide web traffic: Evidence and possible causes," *IEEE/ACM Trans. Netw.*, vol. 5, no. 6, pp. 835–846, Dec. 1997.

[16] A. Cumani, "On the canonical representation of homogeneous Markov processes modelling failure-time distributions," *Microelectronics Reliability*, vol. 22, no. 3, pp. 583–602, 1982.

[17] A. Di Marco and P. Inverardi, "Compositional generation of software architecture performance QN models," in *WICSA 2004*, June 2004, pp. 37–46.

[18] L. Etxeberria, C. Trubiani, V. Cortellessa, and G. Sagardui, "Performance-based selection of software and hardware features under parameter uncertainty," in *QoSA*, 2014, pp. 23–32.

[19] A. Fantechi and S. Gnesi, "Formal modeling for product families engineering," in *SPLC*, 2008, pp. 193–202.

[20] A. Filieri, C. Ghezzi, and G. Tamburrelli, "Run-time efficient probabilistic model checking," in *ICSE*, 2011, pp. 341–350.

[21] G. Franks, T. Al-Omari, M. Woodside, O. Das, and S. Derisavi, "Enhanced modeling and solution of layered queueing networks," *IEEE Trans. Software Eng.*, vol. 35, no. 2, pp. 148–161, 2009.

[22] C. Ghezzi and A. M. Sharifloo, "Verifying non-functional properties of software product lines: Towards an efficient approach using parametric model checking," in *SPLC*, 2011, pp. 170–174.

[23] J. Hillston, *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.

[24] J. Hillston, M. Tribastone, and S. Gilmore, "Stochastic process algebras: From individuals to populations," *The Computer Journal*, 2011.

[25] M. Kowal, I. Schaefer, and M. Tribastone, "Family-based performance analysis of variant-rich software systems," in *FASE 2014*, 2014, pp. 94–108.

[26] T. G. Kurtz, "Solutions of ordinary differential equations as limits of pure Markov processes," *J. Appl. Prob.*, vol. 7, no. 1, pp. 49–58, April 1970.

[27] A. Mahanti, N. Carlsson, A. Mahanti, M. Arlitt, and C. Williamson, "A tale of the tails: Power-laws in internet measurements," *Network, IEEE*, vol. 27, no. 1, pp. 59–64, January 2013.

[28] M. Marzolla and R. Mirandola, "Performance prediction of web service workflows," in *Software Architectures, Components, and Applications*, 2007, vol. 4880, pp. 127–144.

[29] K. Pohl, G. Böckle, and F. J. v. d. Linden, *Software Product Line Engineering: Foundations, Principles and Techniques*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005.

[30] I. Schaefer, "Variability modelling for model-driven development of software product lines," in *VaMoS*, 2010, pp. 85–92.

[31] C. U. Smith, C. M. Lladó, and R. Puigjaner, "Performance Model Interchange Format (PMIF 2): A comprehensive approach to queueing network model interoperability," *Performance Evaluation*, vol. 67, no. 7, pp. 548–568, 2010.

[32] W. J. Stewart, *Probability, Markov Chains, Queues, and Simulation*. Princeton University Press, 2009.

[33] R. Tawhid and D. C. Petriu, "Automatic derivation of a product performance model from a software product line model," in *SPLC*, 2011, pp. 80–89.

[34] T. Thüm, S. Apel, C. Kästner, I. Schaefer, and G. Saake, "A classification and survey of analysis strategies for software product lines," *ACM Comput. Surv.*, vol. 47, no. 1, pp. 6:1–6:45, Jun. 2014. [Online]. Available: http://doi.acm.org/10.1145/2580950

[35] M. Tribastone, "Relating layered queueing networks and process algebra models," in *WOSP/SIPEW*, 2010, pp. 183–194.

[36] ——, "A fluid model for layered queueing networks," *IEEE Transactions on Software Engineering*, vol. 39, no. 6, pp. 744–756, 2013.

[37] ——, "Behavioral relations in a process algebra for variants," in *SPLC*, Florence, Italy, September 2014, pp. 82–91.

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

[39] M. Tribastone and S. Gilmore, "Automatic extraction of PEPA performance models from UML activity diagrams annotated with the MARTE profile," in *WOSP*, 2008.

[40] M. Tribastone, S. Gilmore, and J. Hillston, "Scalable differential analysis of process algebra models," *IEEE Transactions on Software Engineering*, vol. 38, no. 1, pp. 205–219, 2012.

[41] C. Trubiani, I. Meedeniya, V. Cortellessa, A. Aleti, and L. Grunske, "Model-based performance analysis of software architectures under uncertainty," in *QoSA*, 2013, pp. 69–78.