A Proactive Approach for Runtime Self-Adaptation Based on Queueing Network Fluid Analysis

Emilio Incerto
Gran Sasso Science Institute
Viale Francesco Crispi, 7
L'Aquila, Italy
emilio.incerto@gssi.infn.it

Mirco Tribastone
IMT Institute for Advanced Studies Lucca
Piazza S. Francesco, 19
Lucca, Italy
mirco.tribastone@imtlucca.it

Catia Trubiani
Gran Sasso Science Institute
Viale Francesco Crispi, 7
L'Aquila, Italy
catia.trubiani@gssi.infn.it

ABSTRACT

Complex software systems are required to adapt dynamically to changing workloads and scenarios, while guaranteeing a set of performance objectives. This is not a trivial task, since run-time variability makes the process of devising the needed resources challenging for software designers. In this context, self-adaptation is a promising technique that work towards the specification of the most suitable system configuration, such that the system behavior is preserved while meeting performance requirements.

In this paper we propose a proactive approach based on queuing networks that allows self-adaptation by predicting performance flaws and devising the most suitable system resources allocation. The queueing network model represents the system behavior and embeds the input parameters (e.g., workload) observed at run-time. We rely on fluid approximation to speed up the analysis of transient dynamics for performance indices. To support our approach we developed a tool that automatically generates simulation and fluid analysis code from an high-level description of the queueing network. An illustrative example is provided to demonstrate the effectiveness of our approach.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics—*performance measures*; C.4 [**Performance of Systems**]: Modeling techniques, Performance Attributes

Keywords

Runtime Self-Adaptation, Queueing Networks, Fluid Approximation Analysis

1. INTRODUCTION

In the software development process it is fundamental to understand if performance requirements are fulfilled, since they represent what end users expect from the software system. In fact, if performance targets are not met, a variety of negative consequences (such as damaged customer relations, business failures etc.) can impact the project success.

The performance evaluation of software systems is critical in most application domains, such as distributed web systems and services, enterprise applications or cloud computing, since such domains are increasingly required to adapt dynamically to changing workloads, scenarios and objectives. Guaranteeing performance requirements with a high degree of unpredictability and dynamism in the execution context is a not trivial task, since the presence of runtime variability (e.g., workload peaks) makes the whole process challenging for software designers.

Several approaches have been proposed in literature for performance evaluation [14], however the current status of the proposed approaches is far from an ideal situation. Current methodologies rely on: (i) model-based predictions that may not approximate real systems; (ii) measured runtime values that are very expensive in terms of resource usage. Our work tries to overcome these issues by proposing a proactive approach based on queuing networks (QN) [16] that allows model-based predictions while considering runtime variability by means of fluid approximation analysis. In this way we are able to efficiently predict performance flaws and devise the most suitable system resources allocation.

We propose an approach to guarantee the performance requirements of software systems by explicitly considering variability in the performance analysis process. The performance predictions are leveraged to pro-actively place the software in the optimal configuration with respect to changing conditions and parameter runtime changes. In particular, the QN model represents the system behavior and embeds the input parameters (e.g., workload) observed at runtime. We rely on fluid approximation to speed up the analysis of transient dynamics for performance indices. The goal is to build a framework able to anticipate performance flaws by adapting software on the basis of the monitored runtime variabilities.

In this paper self-adaptation is currently implemented with a repository of design changes that impact the system performance, such as the reallocation of software components to hardware platforms, and mirroring the software components to better distribute incoming requests. To support our approach we developed a tool that automatically generates simulation and fluid analysis code from a high-level description of the queueing network.

QUDOS'15, September 1, 2015, Bergamo, Italy
© 2015 ACM. 978-1-4503-3817-2/15/09...$15.00
http://dx.doi.org/10.1145/2804371.2804375

The remainder of this paper is organized as follows. Section 2 presents related work; Section 3 provides foundations for the QN fluid approximation analysis; Section 4 describes our approach; Section 5 shows the approach at work on an illustrative example to demonstrate its effectiveness; Section 6 discusses the relevant open issues raised by the approach and provides directions for future research.

2. RELATED WORK

The work presented in this paper relates to two main research areas: (i) runtime self-adaptation, and (ii) queueing network fluid analysis.

Runtime Self-Adaptation. The need of runtime self-adaptation has been widely recognized in literature. In [11] the problem to dynamically self-adapt software systems (due to resource variability, changing user needs, system faults, etc.) is addressed with a framework using software architectural models to monitor and adapt a running system. Our approach differs from [11], since we are interested in performance properties of running systems and our proactive approach makes use of QN models to avoid performance flaws. In [28] authors claim that one of the major challenges in self-adaptive systems is to assure the required Quality-of-Service (QoS) properties (e.g., performance, reliability), and a relevant part of the studies use formal methods at runtime. However the usage of such methods is limited to modeling and analysis, whereas our approach includes a set of adaptation strategies aimed at executing changes in the running system. In [1] approaches for performance engineering of self-adaptive systems are surveyed and classified (design vs runtime [19], and reactive vs proactive [18]), outlining that model-driven tools including modeling of self-adaptive systems with explicit adaptation strategies are missing in the current state-of-the-art. Several approaches have been defined in the context of QoS driven runtime adaptation, in particular for service-oriented systems [4, 17], for the evolution of runtime parameters [9], for the quantitative verification [3]. The novelty of our approach is that it exploits the QN fluid approximation to speed up the analysis of transient dynamics for performance indices.

Queueing Network Fluid Analysis. We consider fluid analysis in the sense of Kurtz [15], providing a system of ordinary differential equations that associates one equation to each service center in the network. The solution to this equation gives an estimate to the average queue length as a function of time. We refer to [7, 26, 2, 23] for a more in-depth discussion of the derivation of the fluid QN models considered here. The refactoring approach presented in this paper in principle requires the evaluation of each possible QN design alternative from scratch, i.e., without the possibility of reusing the analysis across different alternatives. This is a problem that has been recently tackled in previous literature, for instance in [24] and [13]. However, the approach in [24] allows to reduce the complexity of the exploration of the design space when specific conditions on the constraints for adaptation are met. The symbolic technique of [13] computes the solution of every possible design alternative at once, but it works for Jackson-type queuing networks (see e.g., [22]). Furthermore, it supports only steady-state performance measures, which prevents to apply adaptation when transient violations are detected (as shown in our running example).

Lastly, Table 1 reports some of the QN analysis tools that can be found in literature. We can notice that such tools make use of simulation (providing transient behavior analysis) and/or analytic (providing exact or approximate steady state solutions) evaluation techniques. The novelty of our approach is that it relies on QN fluid approximation analysis that can be seen as an analytical technique for studying the transient behavior of the QN model.

Table 1: Other QN Tools

Tool	Evaluation technique
JMT [6]	Analytic and Discrete Event Simulation
SHARPE [25]	Analytic
JINQS [10]	Discrete Event Simulation
PEPSY-QNS [12]	Discrete Event Simulation and Analytic
TANGRAM-II [5]	Simulation and Analytic
QSIM [21]	Discrete Event Simulation

3. BACKGROUND

The purpose of this section is to briefly describe the main concepts underlying the fluid approximation analysis of queuing networks. Roughly speaking, the goal of this analysis technique is to translate a QN model in a system of ordinary differential equations (ODEs), then solving them to derive the performance indexes of interest. The role of the defined equations is to analytically describe the evolution of the queue length at each service center composing the QN model. The main idea is to view a QN model M as a *Markov Population Process* (MPP) where:

i) $x_i(t)$ is the number of jobs in the queue of service center i at time t, and $N(t) = \sum_{i \in M} x_i(t)$ is the total population of jobs circulating in the network of M centers;

ii) each service center i modifies the number of jobs waiting in the queue, according to a *jump vector* $I_i \in \mathbb{R}^{|M|}$;

iii) the rate of the changes caused by each server center i is proportional to its actual jobs population $x_i(t)$, and is expressed by the definition of the infinitesimal generator $q_{x,x+I}$.

Let us now consider, as an example, a tandem QN model with N circulating jobs. Assume that this network consists of a pure delay station (station 1) and a single server station (station 2) with services rates μ_1 and μ_2 respectively. From the topology of the QN model we can derive the jump vectors $I_1 = (-1, +1)$ and $I_2 = (+1, -1)$ where each component of the jump vector I_i, with $i \in \{1, 2\}$, describes the change of number of jobs at each service center. The elements of the infinitesimal generator for this model are $q_{x_1,x_1+I_1} = \mu_1 x_1$ and $q_{x_2,x_2+I_2} = \mu_2 \min(1, x_2)$. Putting it all together we can write the ODE system as:

$$\frac{dx_1(t)}{dt} = -\mu_1 x_1(t) + \mu_2 \min(1, x_2(t))$$
$$\frac{dx_2(t)}{dt} = +\mu_1 x_1(t) - \mu_2 \min(1, x_2(t))$$

By solving this system, we can compute the queue length of the two stations over time. The transformation of the QN

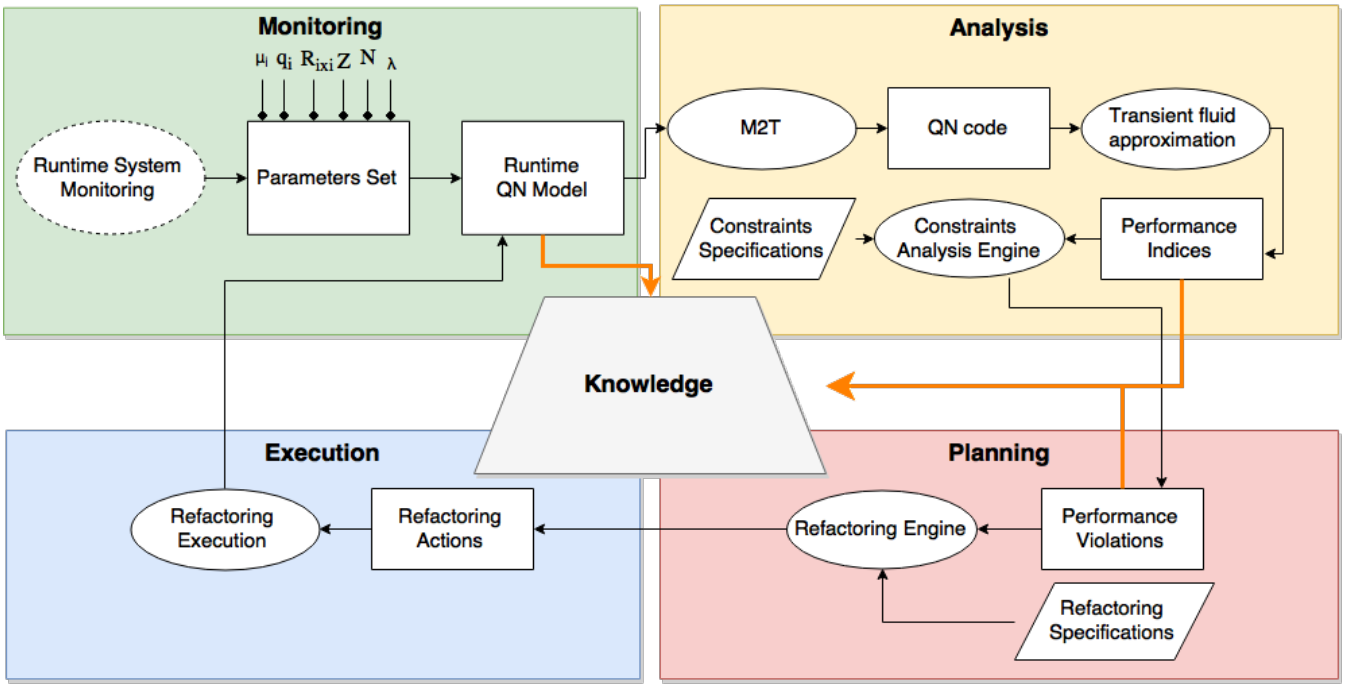

Figure 1: Overview of the proposed technique

model in ODE implies that the solution process shifts from a discrete-state characterization and mean value analysis to a continuous-state representation (based on a system of ordinary differential equations) and fluid approximation analysis. Its computational complexity depends directly on the algorithm used for solving the ODE system that, by referring to the original QN model, is related to the product between the number of stations in the network and the transitions among them.

4. OUR APPROACH

The purpose of this section is to describe a framework that provides the self-adaptation capability to software systems. The goal of this adaptation process is to keep the performance indices of the system within the stated requirements.

The key idea of our approach is to exploit the predictive power of queuing network models by analyzing them at runtime. In particular, feeding a model of the running system with runtime parameters allows us to see if its future evolution will results or not in the degradation of the performance indices of interest.

Our work, can be considered as an implementation of the MAPE-K [8] loop that is a very well assessed methodology in the context of *self-adaptive systems*. The overview of the proposed approach is depicted in Figure 1 in which we can identify the following phases:

- **Monitoring:** the idea of this phase is to monitor the running system using some kind of runtime system profiler tool, similarly to [27]. Ideally, beyond the choice of a particular tool, we expect to periodically extract a set of parameters from the running system. More precisely we want to collect μ_i as the service rate for each service center, q_i as the estimated queue length at service center i and R_{ixi} as the routing matrix of the QN model. In case of closed networks, we estimate the number of customers in the system N and the thinking time Z. On the contrary, in case of open networks, we identify the jobs arrival rate λ. Then we use these parameters to feed the *runtime QN model* built in order to describe the running system.

- **Analysis:** in this step using an ad-hoc defined model-to-text transformation (M2T) we translate the high-level queuing network model of the previous phase, in a low-level representation (*QN code*) suitable for automatic analysis. Executing this code we can analyze the transient behavior of the QN exploiting the fluid approximation technique. Result of this step is the set of *performance indices* of interest that are influenced by the previously estimated parameters. Now giving the computed indices and the performance constraints model as input to the *constraint analysis engine* we detect the eventual performance violations.

- **Planning:** starting from the *performance violations* previously computed and from a refactoring specification model, the *refactoring engine* component is able to compute a set of refactoring actions that represent the plan for the current adaptation iteration. We can think at this actions as the adding or removal of a service center, the modification of the service rates or for example the change of the routing probability for some branch in the model.

- **Execution:** executing the previously defined *refactoring actions* we are able to devise the new QN model that is aimed at avoiding violations of performance constraints. Several refactoring actions may be available and each action gives rise to a new runtime QN

model that undergoes the same process of the initial one. Our approach aims to evaluate several QN models and then reflect in the running system the change actually beneficial only.

We identify as **knowledge** of the MAPE-K loop the information that is required to describe the system in each self-adaptation iteration t. Such information is identified by the runtime QN model, the computed performance indices and the detected performance violations.

Note that the presented approach is tool supported. We developed an Eclipse-based tool used for the definition of the Queuing network models and to automatically execute the M2T transformation on them. Our tool can be downloaded here `http://sourceforge.net/projects/qnml`.

5. ILLUSTRATIVE EXAMPLE

In this section we show a simple but meaningful illustrative example suitable to demonstrate our approach. In particular we consider a constraint model requiring that the percentage of jobs in a queue of every service center do not exceed 0.5% of the total jobs population. In the following the example is described step by step.

5.1 Monitoring

Figure 2 depicts the QN model describing the running system used in our illustrative example. It represents a closed network composed by: one delay station ($D1$), four service centers (*Server1, Server2, Server3, Server4*) and four routing stations ($R1, R2, R3, R4$). In Figure 2 the solid lines are used to represent the connections between the routing station and service center; on the contrary, the dashed line is used to represent the opposite direction.

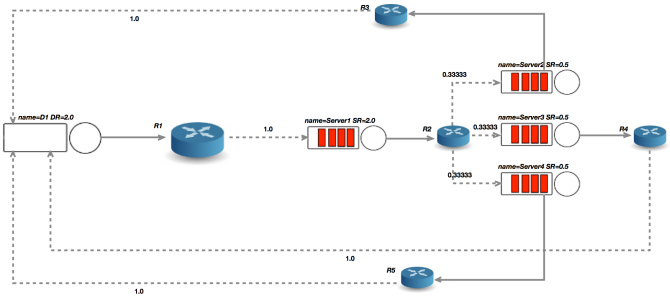

Figure 2: Initial Model

Although, in our vision, the parameters embedded by the model should to be automatically and periodically extracted from the running system, for sake of simplicity, we fed the model for the example with specific parameters suitable to trigger interesting behavior into the system. More precisely the used parameters are listed in Table 2 while the routing matrix is reported hereafter.

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1/3 & 1/3 & 1/3 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Table 2: Initial Model Parameters

Station	Type	Init. Pop.	μ_i	Z
D1	Delay Center	10	n.d.	0.5
Server1	Service Center	0	2.0	n.d.
Server2	Service Center	0	0.5	n.d.
Server3	Service Center	0	0.5	n.d.
Server4	Service Center	0	0.5	n.d.

5.2 Analysis

Goal of this step is to produce the performance indices and to detect the eventual constraints violation of the system. Figure 3 shows the structure of the Eclipse plug-in used in order to implement the Model to Text Transformation (M2T) through the ACCELEO framework [20].

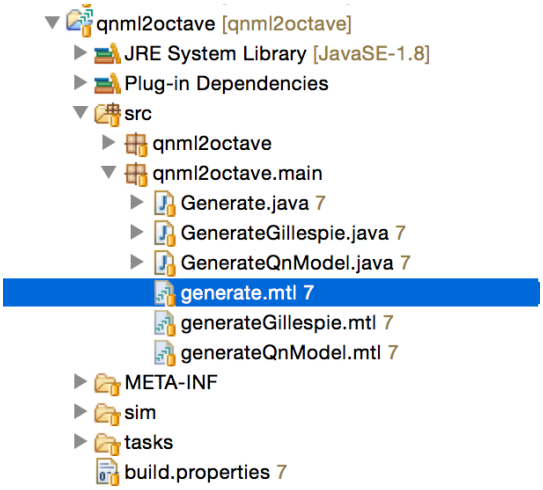

Figure 3: M2T plug-in structure

Using this transformation we can translate the previously defined QN model into Octave[1] executable code, shown in Listing 1. It is worth to notice that this low-level representation of the model allows us to analyze the transient behavior of the system through the fluid approximation technique.

Listing 1: Octave Code For The Initial Model

```
function dx = qn_ode(x,t)
dx = zeros(5,1);
dx(1) = -mu1*x(1)+mu3*min(x(3),1)
   +mu4*min(x(4),1)+mu5*min(x(5),1);
dx(2) = +mu1*x(1)-mu2_1*min(x(2),1)
    -mu2_2*min(x(2),1)-mu2_3*min(x(2),1);
dx(3) = +mu2_1*min(x(2),1)-mu3*min(x(3),1);
dx(4) = +mu2_2*min(x(2),1)-mu4*min(x(4),1);
dx(5) = +mu2_3*min(x(2),1)-mu5*min(x(5),1);
endfunction
```

Figure 4 shows how the queue length at each service center changes during the observation time. We can see that the population of the jobs in the network, at the beginning of the analysis, is almost in the queue of *server*1. This happens due to the fact that the thinking time Z of $D1$ is lower than the time needed by *Server*1 in order to complete a job.

[1] `http://www.gnu.org/software/octave/`

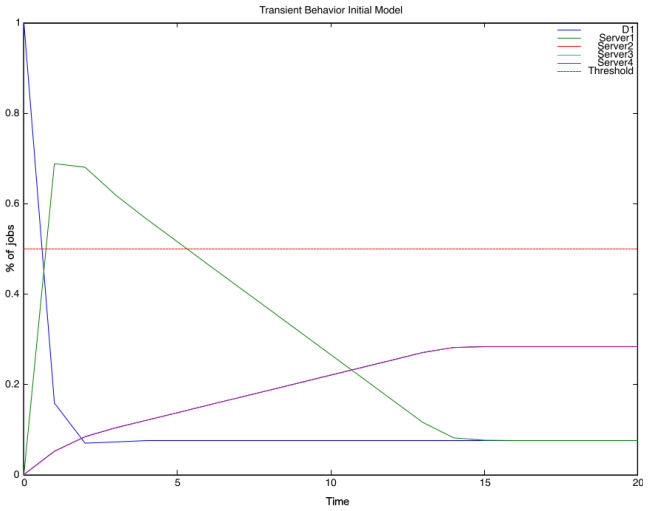

Figure 4: Transient behavior for the initial model

Considering our constraints model, the behavior shown in Figure 4 is not acceptable and represents a performance violation. The set of perfomance violations are used as input for the planning phase in order to produce the appropriate set of refactoring actions.

5.3 Planning

The refactoring engine relying on the refactoring specification is able to generate a set of refactoring actions suitable to react to the identified performance violation. In our example the refactoring specification suggests to duplicate the queuing center suffering of these kind of performance violations. Figure 5 depicts the refactored QN model for our example.

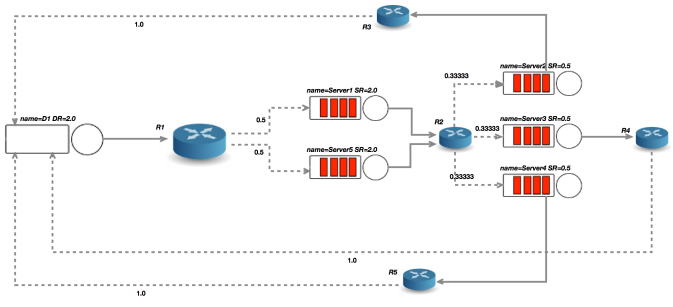

Figure 5: Refactored Model

In particular, we added a new service station in parallel to $Server1$ with an equal routing probability. In this way the jobs coming from $D1$ are equally distributed among $Server1$ and $Server2$.

5.4 Execution

In this phase we check if the new QN topology is suitable to avoid the performance violations and we actually execute the refactoring actions. This means to update the QN model and, once verified the effectiveness of the refactoring, to report this change in the structure of the running system. Figure 6 shows that the executed refactoring action is suitable to fulfill the constraints specification.

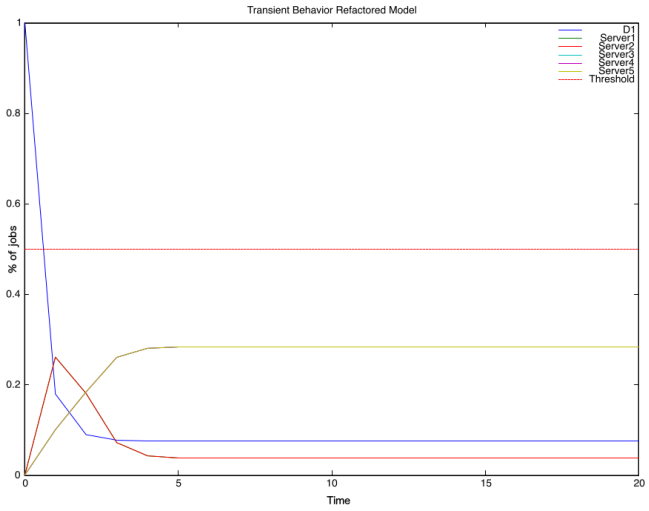

Figure 6: Transient behavior for the refactored model

This change in the QN topology (i.e., the addition of a further server) makes the peak no longer happening. After this phase the adaptation iteration ends and the self-adaptation process continues with new iterations using this model as the starting point for the future analysis and adaptation. We are aware that our illustrative refactoring action is straightforward, but the experimental results seem promising to implement further actions for self-adaptation purpose.

6. CONCLUSION

In this paper we presented a proactive approach that provides self-adaptation capabilities to software systems in order to guarantee the fulfillment of performance requirements.

The novelty with respect to current state-of-art is that our approach exploits the QN fluid approximation to speed up the analysis of transient dynamics for performance indices. To this end, we developed a tool that automates the generation of simulation and fluid analysis code from a high-level description of QNs.

As future work, our short-term research agenda includes the task of providing a more formal definition of the constraints analysis and refactoring engines, together with the introduction of a language for constraints and refactoring specifications. For the last mentioned task we plan to investigate the use of formal specifications such as Signal Temporal Logic. Furthermore, a systematic comparison between the proposed approach and other simulation techniques is needed. In the long-term we intend to apply our approach to other case studies, possibly coming from real-world domains. This wider experimentation will allow us to deeply investigate the scalability and the usefulness of our approach for self-adaptation, thus quantifying its effectiveness.

7. REFERENCES

[1] M. Becker, M. Luckey, and S. Becker. Model-driven performance engineering of self-adaptive systems: a survey. In *Proceedings of the 8th international ACM SIGSOFT conference on Quality of Software Architectures (QoSA)*, pages 117–122, 2012.

[2] L. Bortolussi and M. Tribastone. Fluid limits of queueing networks with batches. In *WOSP/SIPEW International Conference on Performance Engineering (ICPE)*, pages 45–56, 2012.

[3] R. Calinescu, C. Ghezzi, M. Z. Kwiatkowska, and R. Mirandola. Self-adaptive software needs quantitative verification at runtime. *Commun. ACM*, 55(9):69–77, 2012.

[4] V. Cardellini, E. Casalicchio, V. Grassi, S. Iannucci, F. L. Presti, and R. Mirandola. MOSES: A framework for qos driven runtime adaptation of service-oriented systems. *IEEE Trans. Software Eng.*, 38(5):1138–1159, 2012.

[5] R. M. Carmo, L. R. de Carvalho, E. d. S. e Silva, M. C. Diniz, and R. R. Muntz. Tangram-ii: A performability modeling environment tool. In *Computer Performance Evaluation Modelling Techniques and Tools*, pages 6–18. Springer, 1997.

[6] G. Casale and G. Serazzi. Quantitative system evaluation with java modeling tools. In *Proceedings of the 2Nd ACM/SPEC International Conference on Performance Engineering*, ICPE '11, pages 449–454, New York, NY, USA, 2011. ACM.

[7] G. Casale, M. Tribastone, and P. G. Harrison. Blending randomness in closed queueing network models. *Performance Evaluation*, 82(0):15 – 38, 2014.

[8] R. De Lemos, H. Giese, H. A. Müller, M. Shaw, J. Andersson, M. Litoiu, B. Schmerl, G. Tamura, N. M. Villegas, T. Vogel, et al. Software engineering for self-adaptive systems: A second research roadmap. In *Software Engineering for Self-Adaptive Systems II*, pages 1–32. Springer, 2013.

[9] I. Epifani, C. Ghezzi, R. Mirandola, and G. Tamburrelli. Model evolution by run-time parameter adaptation. In *International Conference on Software Engineering ICSE*, pages 111–121, 2009.

[10] T. Field. Jinqs: An extensible library for simulating multiclass queueing networks, v1. 0 user guide, 2006.

[11] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste. Rainbow: Architecture-based self-adaptation with reusable infrastructure. *Computer*, 37(10):46–54, Oct. 2004.

[12] M. Kirschnick. The performance evaluation and prediction system for queueing networks-pepsy-qns. 1994.

[13] M. Kowal, I. Schaefer, and M. Tribastone. Family-based performance analysis of variant-rich software systems. In *Fundamental Approaches to Software Engineering (FASE)*, pages 94–108, April 2014.

[14] H. Koziolek. Performance evaluation of component-based software systems: A survey. *Perform. Eval.*, 67(8):634–658, 2010.

[15] T. G. Kurtz. Solutions of ordinary differential equations as limits of pure jump markov processes. *Journal of Applied Probability*, 7(1):pp. 49–58, 1970.

[16] E. D. Lazowska, J. Zahorjan, G. Scott Graham, and K. C. Sevcik. *Computer System Analysis Using Queueing Network Models*. Prentice-Hall, Inc., Englewood Cliffs, 1984.

[17] D. A. Menascé, H. Gomaa, S. Malek, and J. P. Sousa. SASSY: A framework for self-architecting service-oriented systems. *IEEE Software*, 28(6):78–85, 2011.

[18] A. Metzger, O. Sammodi, K. Pohl, and M. Rzepka. Towards pro-active adaptation with confidence: Augmenting service monitoring with online testing. In *Proceedings of ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS, pages 20–28, New York, NY, USA, 2010. ACM.

[19] R. Mirandola and C. Trubiani. A deep investigation for qos-based feedback at design time and runtime. In *IEEE International Conference on Engineering of Complex Computer Systems ICECCS*, pages 147–156, 2012.

[20] J. Musset, É. Juliot, S. Lacrampe, W. Piers, C. Brun, L. Goubet, Y. Lussaud, and F. Allilaire. Acceleo user guide, 2006.

[21] P. Saxena and L. Sharma. Simulation tool for queuing models: Qsim. *International Journal of Computers and Technology*, 5(2):74–79, 2013.

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

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

[24] M. Tribastone. Efficient optimization of software performance models via parameter-space pruning. In *ACM/SPEC International Conference on Performance Engineering (ICPE)*, pages 63–73, 2014.

[25] K. S. Trivedi and R. Sahner. Sharpe at the age of twenty two. *SIGMETRICS Perform. Eval. Rev.*, 36(4):52–57, Mar. 2009.

[26] M. Tschaikowski and M. Tribastone. Insensitivity to service-time distributions for fluid queueing models. ICST, 1 2014.

[27] A. Van Hoorn, J. Waller, and W. Hasselbring. Kieker: A framework for application performance monitoring and dynamic software analysis. In *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*, pages 247–248. ACM, 2012.

[28] D. Weyns, M. U. Iftikhar, D. G. de la Iglesia, and T. Ahmad. A survey of formal methods in self-adaptive systems. In *Proceedings of the International Conference on Computer Science and Software Engineering*, C3S2E, pages 67–79, New York, NY, USA, 2012. ACM.