Ph.D. Thesis

Formal Design of Performance-driven Self-adaptive Systems under Uncertainty

PhD Program in Computer Science: XXX cycle

Author:

Emilio Incerto
emilio.incerto@gssi.it

Supervisor:

Prof. Mirco Tribastone
mirco.tribastone@imtlucca.it

Internal advisor:

Dr. Catia Trubiani
catia.trubiani@gssi.it

April 2019

GSSI Gran Sasso Science Institute
Viale Francesco Crispi, 7 - 67100 L'Aquila - Italy

Abstract

Performance evaluation is fundamental for modern software systems design since it heavily determines users satisfaction. However, although standard quantitative methods are effective for studying the performance behavior of software in the early stage, they may fail at runtime when systems are exposed to strong external and internal variability (e.g., workload fluctuations or hardware degradation).

In this setting, performance-driven self-adaptation is a promising technique. It manages runtime variability by continuously monitoring the current execution conditions and, when required, by triggering reconfiguration actions using a model of the system under study. This introduces three main difficulties: *i*) the system model must be efficiently analyzed, *ii*) an effective strategy for exploring the adaptation space (i.e., the set of all feasible system configurations) must be devised, *iii*) frequent up-to-date estimates of model parameters need to be provided in a minimally intrusive manner.

Motivated by these challenges, this thesis studies the definition of performance-driven self-adaptation techniques enabling effective and efficient runtime reconfigurations. We focus on systems whose performance dynamics can be described by a compact, *approximate* representation of queuing networks based on ordinary differential equations. We exploit this representation of the system as the enabling technology for defining different performance-driven self-adaptation techniques following two orthogonal approaches.

The *symbolic* approach is based on closed-form formulas expressing the *steady-state* relation between the system parameters and its throughput. The main contribution is an encoding of the performance-driven adaptation problem as a satisfiability modulo theories one. This allows querying for a feasible assignment of the system parameters that satisfy given performance requirements or getting a formal proof of its unavailability (i.e., no such a configuration exists). The *numeric* approach, instead, is founded on the numerical analysis of the dynamical model describing the *transient* performance behavior of the system. In this case, we formulate the performance-driven self-adaptation problem using model predictive control, a well-known technique based on online optimization. The main novel contribution in this area is the exact formulation of the original nonlinear runtime adaptation problem as a linear one, enabling faster adaptation space explorations.

Finally, we propose a moving horizon estimator for the resource demands of queuing networks. Its main novelty lays in the formulation of the estimation problem as an efficient quadratic optimization one built on the observation of transient queue-length dynamics. This yields an accurate and nonintrusive estimation methodology employable as the input phase of any of the performance-driven adaptation techniques presented in this thesis.

Acknowledgements

This thesis would be not possible without the strong support of my family. Their unconditional love has given me the strength to go ahead and overcome even the biggest obstacles I have faced along my path. An even bigger thank you goes to my girlfriend Francesca who with her daily presence, helped me to realize my objectives by struggling to discipline my inattentive soul into the "here and now" mindset.

I have no words to express my gratitude to my advisors, Mirco Tribastone and Catia Trubiani. Under their passionate guidance, I learned how to conduct a rigorous research and to present the results convincingly. I owe a great debt with them for making my love of science bloom definitively. Thank you for your work, patience, and advice. I felt very lucky and proud to work with you for all these years. A special thank goes to Professor Murray Woodside that hosted me at Carleton University for three months. Thank you for all your wise advice and for the stimulating discussions.

Last but not least, I would like to thank the Gran Sasso Science Institute and IMT for giving me this opportunity and for all the services made available all along the Ph.D. program.

Declaration

I declare that this thesis was created by my self, that the works composing it are my own under the guidance of my supervisors Mirco Tribastone and Catia Trubiani. In particular, Chapter 3 is based on [1]. Chapter 4 is based on [2]. Chapter 5 is based on [3]. Chapter 6 is based on [4] and Section 2.4 is mainly taken from [5].

(Emilio Incerto)

Contents

Acknowledgements ii

Declaration iii

List of Figures vi

List of Tables vii

Abbreviations viii

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Contributions . . . 2
- 1.3 Outline . . . 4

2 Background 5
- 2.1 Queuing Networks . . . 5
- 2.2 Fluid Approximation of Queuing Networks . . . 9
- 2.3 Software Engineering for Self-adaptive Systems . . . 11
  - 2.3.1 A Research Landscape . . . 12
  - 2.3.2 Uncertainties in Self-adaptive Systems . . . 14
- 2.4 A Self-adaptation Framework Based on Queuing Networks Fluid Analysis . . . 15
- 2.5 Related Work . . . 17

3 Symbolic Performance Adaptation 19
- 3.1 Overview . . . 19
- 3.2 Related Work . . . 21
- 3.3 Preliminaries . . . 23
- 3.4 The Symbolic Performance Adaptation Approach . . . 24
  - 3.4.1 The Parametric QN . . . 25
  - 3.4.2 SMT Problem Formulation . . . 27
- 3.5 The Case Study: A Three-Tier System . . . 30
- 3.6 Discussion . . . 34

4 Efficient Performance-driven Self-adaptation 37
- 4.1 Overview . . . 37
- 4.2 Related Work . . . 41

4.3 Efficient Performance-driven Self-adaptation . . . 42
4.3.1 Preliminaries . . . 42
4.3.2 Nonlinear MPC Formulation . . . 43
4.3.3 MILP Formulation . . . 44
4.3.4 Adaptation Logic . . . 45
4.4 Numerical Evaluation . . . 47
4.4.1 System Description and Implementation . . . 47
4.4.2 Model Parametrization and Validation . . . 48
4.4.3 Adaptation Evaluation . . . 50
4.4.4 Scalability Evaluation . . . 53
4.4.5 Comparison with Markov Decision Processes . . . 54
4.5 Discussion . . . 54
4.6 An Exact MPC MILP Formulation . . . 55

5 Performance Adaptation of Co-located Distributed Applications 59
5.1 Overview . . . 59
5.2 Combined Vertical and Horizontal Autoscaling . . . 61
5.2.1 A Running Example . . . 61
5.2.2 Multi Class Parametric QN . . . 62
5.2.3 ODE Model . . . 62
5.2.4 LP Performance Adaptation Formulation . . . 63
5.3 Numerical Evaluation . . . 66
5.3.1 System Description and Implementation . . . 66
5.3.2 Model Parametrization and Validation . . . 67
5.3.3 Adaptation Evaluation . . . 68
5.4 Discussion . . . 70

6 Moving Horizon Estimation of Service Demands for Queuing Networks 72
6.1 Overview . . . 72
6.2 Related Work . . . 74
6.3 Moving Horizon Estimation of Service Demands for Queuing Networks . . . 76
6.3.1 Nonlinear service demand estimator . . . 76
6.3.2 Quadratic programming formulation . . . 76
6.3.3 Moving horizon estimation . . . 77
6.3.4 Application to the load balancer case study . . . 78
6.4 Numerical Evaluation . . . 79
6.4.1 Methodology . . . 79
6.4.2 Comparison with QMLE . . . 79
6.4.3 Scalability analysis . . . 82
6.5 Discussion . . . 82

7 Conclusions 84
7.1 Achievements . . . 84
7.2 Future Work . . . 86

List of Figures

1.1 Contributions overview 2

2.1 Load balancer running example: (A) depicts its closed QN model. (B) reports the state space diagram of the underlying stochastic process. In each circle is reported the number of users waiting or in service at each station respectively. . 6
2.2 Comparison between a sample path of the CTMC and the fluid model 10
2.3 MAPE-K control loop 12
2.4 Abstract fluid QN based self-adaptation framework 16

3.1 Overview of the symbolic performance adaptation. 24
3.2 Queuing network model for the load balancing system. 25
3.3 Load balancing $\mathcal{Q}$ fulfilment. 28
3.4 Self-adaption computation time 29
3.5 QN model for a three-tier system based on [6], where each tier has three replicated servers. 30
3.6 System response time. 33
3.7 Minimum and maximum utilizations for each runtime configuration. 33
3.8 Computation time. 34
3.9 Number of faults for each adaptation step. 34
3.10 Scalability of SPA on the case study. 35

4.1 MPC adaptation approach overview 39
4.2 HAT architecture 47
4.3 QN model validation 49
4.4 Hardware degradation experiment 50
4.5 Workload fluctuation experiment 52

5.1 Sample QN model. 61
5.2 Experiment system architecture 66
5.3 Class-1 instantaneous average response time distribution with (5.3c, 5.3d) and without (5.3a, 5.3b) degradation. 69
5.4 Class-2 instantaneous average throughput with (5.4c, 5.4d) and without degradation (5.4a, 5.4b). 70

6.1 Simulation and ODE solution for the queue lengths of the running example. . . . 78
6.2 Queuing network topology used in the comparison experiments against QMLE. . 80

List of Tables

3.1 Parameter choices for the running example; '—' indicates that the variable is considered symbolically. 27

4.1 System parameters for the validation experiments 48
4.2 Optimization runtimes (TO: timeout after 120 s) 53
4.3 MDP comparison (TO: timeout after 120 s) 54

5.1 Model validation results. The errors $\mathcal{E}_{r1}$ and $\mathcal{E}_{r2}$ between the measured and predicted throughputs for class 1 and class 2, respectively, are measured as absolute relative percentage errors. 68
5.2 Optimal control signals. 70

6.1 Model parameters for the load balancer of Fig. 2.1a 78
6.2 Accuracy comparison between QMLE and MHE. 81
6.3 Scalability analysis. 82

Abbreviations

BS-AMVA Bard-Schweitzer Approximate Mean Value Analysis

AS Adaptation Space

CTMC Continuous Time Markov Chain

DSPL Dynamic Software Product Line

DTMC Discrete Time Markov Chain

HAT Heavy Task Processing Application

KAMI Keep Alive Models with Implementations

LP Linear Programming

MAPE-K Monitoring Analysing Planning Execution-Knowledge

MDP Markov Decision Process

MHE Moving Horizon Estimation

MILP Mixed Integer Linear Programming

MINLP Mixed Integer Non Linear Programming

MPC Model Predictive Control

MVA Mean Value Analysis

$\mathcal{NRA}$ Nonlinear Real Arithmetic Theory

ODE Ordinary Differential Equation

QMLE Queue Length Maximum Likelihood

QN Queuing Network

QoS Quality of Service

QP Quadratic Programming

SAT Boolean Satisfiability

SMT Satisfiability Modulo Theories

SOA Service Oriented Architecture

SPA Symbolic Performance Adaptation

SPL Software Product Line

VM Virtual Machine

VPS Virtual Private Server

Chapter 1

Introduction

1.1 Motivation

In the software development process, quantifying quality-of-service (QoS) properties (e.g., response time or throughput) is a fundamental activity since they represent what the end users expect from the system. Moreover, it has been widely recognized by researchers and practitioners how performance needs to be seen as a new facet of software correctness [7].

Queuing networks (QNs) are a class of well established analytic models for software performance engineering [8, 9]. Their key idea is to model customers being routed between different service stations, where they compete for a pool of processing resources. Among the benefits of QNs is the fairly immediate association between their constituent elements and the components of software systems (e.g., a service center may be used either for abstracting a software device or a hardware resource like CPU or disk). Typically, designers employ such model for studying the performance of systems at design time in a "what-if" analysis fashion (i.e., combining different workload and queuing parameters) so as to identify the most suitable hardware/software configuration satisfying requirements under worst-case or average execution conditions [10].

However, although this approach is effective for characterizing the performance behavior of software in the early stage of life cycle [11, 12], it fails when systems are exposed to strong external and internal runtime variabilities such as workload fluctuation (i.e., a variable number of concurrent users) or hardware degradation (i.e., changes in the hardware characteristics). Indeed, due to the uncertainty in the execution condition, an initially optimal configuration with respect to some given QoS requirements (e.g., throughput or mean time to failure) may suddenly become poor, steering the system toward a significantly different operating point. In this context, self-adaptation is a promising paradigm where uncertainty is managed by continuously monitoring the current execution conditions and, if needed, by planning a reconfiguration using a model of the system under study [13, 14].

Conferring runtime self-adaptation capabilities to software systems introduces three main difficulties. First, the system model must be analyzed under strict time constraints to ensure fast reactions. Second, an effective strategy must be devised to explore the adaptation space (AS), i.e., the set of all feasible system configurations that can be obtained by tuning the system parameters. Indeed, it has been recognized how the typically huge size of the AS represents one of the major limitations for applying state-of-the-art approaches to real-world scenarios [15]. Third, effective and efficient parameters estimation techniques need to be provided for obtaining a continuously updated model (i.e., aligned with the current system execution conditions) in a minimally intrusive manner (i.e., without interfering with the behavior of the running system) [16].

Motivated by these challenges, in this thesis we study the definition of novel performance-driven self-adaptation techniques enabling effective and efficient runtime reconfigurations. We focus on systems whose performance dynamics can be described by a compact, *approximate* representation of QNs based on ordinary differential equations. The main benefit of this ODE interpretation of QN is the availability of a range of techniques that would not be otherwise applicable due to the computational complexity of an exact QN transient analysis [17]. We exploit this dynamic model of the system as the starting point for defining different performance-driven self-adaptation methodologies according to two different approaches.

The first makes use of *symbolic* expressions, describing the *steady-state* relation between the system performance dynamics and its parameters, while the second uses *numerical transient* analysis for predicting performance. By doing so we are able to study the quantitative behavior of the system taking into account the effect of its transient dynamics, which is typically neglected by the traditional steady-state solutions of QNs (i.e., when the system is assumed in the equilibrium condition).

1.2 Contributions

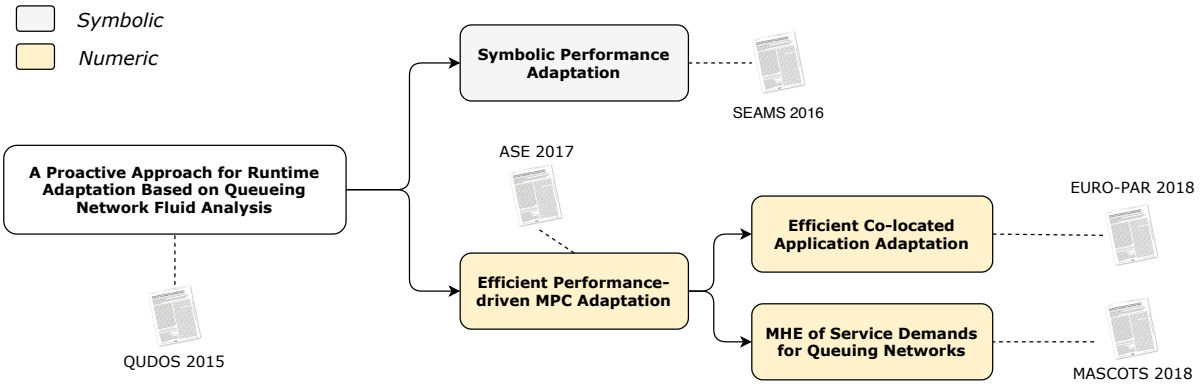


FIG. 1.1: Contributions overview

Figure 1.1 depicts an overview of our contributions highlighting their relations, i.e., each solid arrow in the diagram denotes an use-extend relation between the connecting works. Moreover each paper is decorated with the conference where it has been presented. In the following, each of them is briefly discussed.

QUDOS 2015 [5] This paper introduces an abstract MAPE-K loop [14] based framework that, relying on QN models, allows a predictive performance-driven self-adaptation at runtime. In particular, we exploit the QN model for representing the quantitative behavior of the system taking into account the actual runtime parameters observed during the system executions. Then, relying on QN fluid analysis we are able to anticipate unwanted performance dynamics by triggering the proper system reconfiguration actions. By instantiating each component of this abstract MAPE-K loop, different performance-driven self-adaptation approaches can be derived (i.e., each of the proposed approaches in this thesis is a particular implementation of this framework). We provide an illustrative example to demonstrate the effectiveness of the proposed framework.

SEAMS 2016 [1] In this paper the symbolic performance adaptation is presented. It is based on previous work [18], which introduces a closed-form symbolic expression describing the steady-state relation between the system parameters and its throughput. Then the actual performance of a specific QN can be obtained by simply plugging its concrete parameters into the symbolic expression. Technically, the novel contribution is the encoding of the performance-driven adaptation problem as a satisfiability modulo theories (SMT) one [19]. This allows querying the SMT solver for a feasible assignment of the system parameters needed for satisfying performance requirements or getting a formal proof of its non-existence. One of the main limitations of this work is that the symbolic closed-form expressions are available for steady-state measures only. This implies that the symbolic performance adaptation can be effectively applied exclusively under the assumption of negligible transient dynamics.

ASE 2017 [2] Here, differently from [1] the self-adaptation approach is founded on the *transient* analysis of the QN fluid approximation describing the performance behavior of the system. We formulate the performance-driven self-adaptation problem using model predictive control (MPC), a well-known technique based on on-line numerical optimization [20]. Its basic idea is to perform an optimization at each time step during the system evolution. Specifically, by feeding the system model with the actual runtime execution conditions, the optimization automatically computes the most suitable parameters steering the system toward a given reference trajectory (e.g., a QoS requirement) over a given time horizon. The novel contribution in this area is the exact formulation of the runtime adaptation problem as a mixed integer linear program instead of a nonlinear mixed integer one, thus enabling faster adaptation space explorations with respect to state of the art approaches [21].

EURO-PAR 2018 [3] In this paper we extend the MPC formulation of [2] for controlling the performance dynamics of distributed co-colocated applications. Technically, we present two major improvements with respect to our previous approaches: *i)* A multi-class fluid QN model that enables an accurate representation of performance dynamics of multiple applications sharing the same virtualized environment. *ii)* The specification of latency-based response-time requirements, which in [2] it was limited to queue-length and throughput only. A positively surprising side effect of this multi-class MPC formulation is the reduced computational cost, since the whole control problem is now encoded as a linear programming problem.

MASCOTS 2018 [4] Here we propose a novel moving horizon estimator (MHE) for resource demands of QNs. Its main novelty resides in the formulation of the estimation problem as an efficient quadratic optimization one built on queue lengths transient observations only. To the best of our knowledge, this paper presents the first approach that does not make the assumption that the measurements are taken from a system in the steady-state regime. This yields to an efficient and nonintrusive estimation methodology that can be employed as the input phase of any of the performance-driven adaptation approaches presented in this thesis (i.e., [1–3]).

1.3 Outline

The remainder of this work is organized as follows:

In Chapter 2 we discuss the theoretical concepts underlying this thesis. Moreover, the main state-of-the-art works proposed in the performance-driven self-adaptation area are analyzed. In Chapter 3 we discuss the symbolic performance adaptation [1]. In Chapter 4 the MPC based self-adaptation [2] approach is reported. In Chapter 5 the combined vertical and horizontal auto scaling technique [3] is exhaustively discussed. In Chapter 6 the moving horizon estimator for resource demands of QNs proposed in [4] is examined. Finally, Chapter 7 concludes the thesis by summarizing the main results and suggesting possible future research directions.

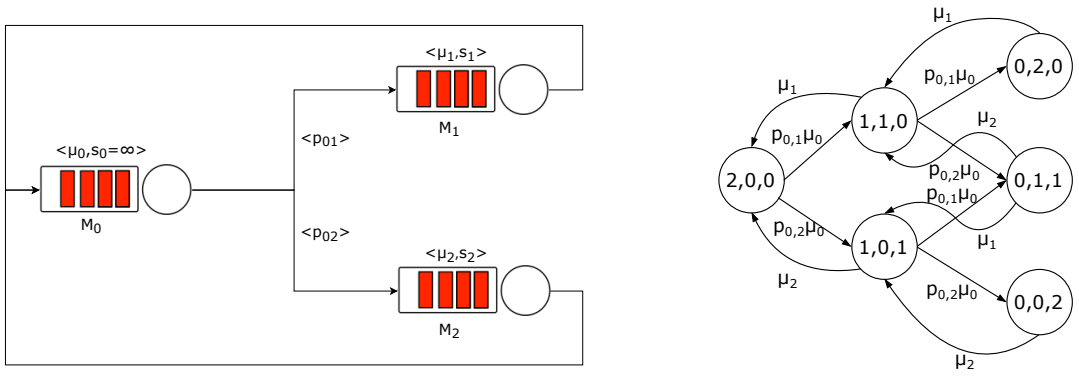
Chapter 2

Background

The scope of this chapter is to present the main concepts and techniques underlying the research work of this thesis. We start by briefly discussing (see Section 2.1) the queuing networks modeling paradigm, showing how it is possible to employ such mathematical abstraction for capturing the steady state performance dynamics of software systems. Section 2.2 introduces the fluid version of queuing networks that enable an approximate but efficient transient analysis based on ordinary differential equations.

2.1 Queuing Networks

Queuing networks are a class of models that has proven very popular in the software performance engineering community [8, 9, 11]. They model customers/jobs being routed between different queuing stations where they compete for a pool of processing resources. After being served, a job leaves the station and goes to possibly another one according to a probability distribution (routing probabilities). Among the benefits of QN is the rather immediate association between their constituent elements and the components of a software system. A service center may either represent a software device (e.g., a web server) or a hardware resource (e.g., CPU or disk). Routing probabilities, instead, can represent the operational profile of the workload. In order to model parallelism, each station can be composed of multiple, independent and identical servers (e.g., to represent thread concurrency levels for software resources or multiple cores for hardware resources). Service times (i.e., the time required by the system for completely serving a job) are described by a probability distribution, where a commonly used one is the exponential. A QN may be closed or open depending on whether or not a fixed population of customers remains within the system. Moreover, if users in the network belong to different classes the network represents a multi-class QN used for modeling systems that execute a set of parallel non-interacting jobs characterized by different service demands or operational profiles.



(A) Closed queuing network model (B) CTMC state space diagram

FIG. 2.1: Load balancer running example: (A) depicts its closed QN model. (B) reports the state space diagram of the underlying stochastic process. In each circle is reported the number of users waiting or in service at each station respectively.

In this thesis we study closed QNs with single (see Chapter 4) and multiple (see Chapter 5) classes of users characterized by exponentially distributed service times. However, we remark that using [18] our frameworks can be extended to the nonexponential case via an approximation based on Coxian distributions [22], which is particular 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 [23, 24] while keeping the underlying stochastic process tractable. For clarifying the different components of the QN model in the following we introduce *load balancing*, which is a fundamental design principle for performance engineering. We will use it extensively as a running example throughout the technical chapters of this thesis.

Load balancing is a very well-known and established design technique [25] employed at different levels of abstraction, from the operating system [26] to the application level [27]. Its goal is to distribute workloads across multiple computing resources, such as CPUs, clusters, network links or disk drives so as to optimize resource usage, maximize throughput, minimize response time, and avoid overload of any single resource. Basically, it operates as follows: once a request enters the system, at runtime a dispatcher (the load balancer) selects a particular station to which the actual processing is delegated according to a particular load distribution policy. Doing so, the resources of a system can be simultaneously utilized giving it linearly scalable performance (i.e., the maximal throughput of a load balancer linearly increase with the number of its replicas). For instance in Amazon EC2 [28], load balancing is handled by uniformly distributing requests between homogeneous instances of a specific Web service. Scalability is achieved by starting new instances when the time taken to process a client request is beyond pre-defined thresholds [29]. Moreover, it has been recognized by both researchers and practitioners how load balancing is the key ingredient for implementing redundancy patterns, which are instrumental to improve systems reliability and availability [30].

Figure 2.1a depicts the reference QN model describing the performance dynamics of a simple

load balancing system with two replicas. Stations M_1, M_2 represent the computational nodes devoted to the processing of incoming requests, while M_0 is the scheduler node responsible for forwarding each request to a particular queuing station according to the probabilities $p_{0,1}$ and $p_{0,2}$. Finally, each station $i \in \{0, 1, 2\}$ in the model is characterized by a service rate μ_i and by the number of servers s_i, respectively denoting the average number of requests processed in the unit of time and the maximum number of requests processable in parallel.

In principle, this kind of system can be configured by tuning different parameters, related to both the software and the hardware, such as the dispatchers strategy (i.e., $p_{i,j}$) and the concurrency level (i.e., s_i) at each station. Events that may threaten the performance of the system include: degradation of the quality of a server (e.g., to account for a hardware issue); or unexpected increases in the system workload (e.g., as a consequence of peak/off-peak patterns). As we will see later in this thesis, our goal is to endow such systems with self-adaptation capabilities that allow fast and automated reconfiguration aimed at proactively taming such unexpected and disruptive events. For instance, it could be necessary to recompute the routing probabilities due to stations or links faults, or to scale up service node resources (e.g., CPU speed) due to workload peaks.

More formally, a general open/closed single-class[1] QN is identified by a set of stations S and the following parameters:

- λ is the rate of the exponential distribution describing the arrival process of open queuing networks only (i.e., in average a new customer arrive at the system within $\frac{1}{\lambda}$ unit of time);
- z is the rate of the exponential distribution describing the thinking process of each clients circulating in a closed queuing networks only (i.e., in average an already processed customer issues a new request within $\frac{1}{z}$ unit of time);
- C is the population level denoting the fixed number of users circulating in closed queuing networks;
- s_i is the concurrency level of station i, with $i \in S$;
- μ_i is the service rate of station i, with $i \in S$;
- $p_{i,j}$ is the *routing probability*, i.e., the probability that a request from station i goes to j, with $i, j \in S$;

The model just defined can be employed for computing the following performance measures:

- *Utilization* $\mathcal{U}$: the fraction of the time in which the server pool of the station is busy, i.e., it is processing a request.

[1]Since the multi-class version is dependent by the adopted scheduling policies, we postpone its presentation until, in Chapter 5, where an approach for controlling distributed co-located applications is presented.

- *Throughput* $\mathcal{T}$: the mean number of jobs processed in a single unit of time.
- *Response Time* $\mathcal{R}$: the total time that a job spends in the queueing system.
- *Queue Length* $\mathcal{Q}$: the number of jobs in the queue.

The exact analysis of QN (e.g, computing throughput or response time) is not an easy task due to the stochastic nature of both its arrivals λ and processing rate μ_i, i.e., the system's performance evolution is influenced by the fact that the probability with which the inter-arrival time between two consecutive clients $a_{i,i+1}$ is shorter than the i-th client processing time d_i is a non zero quantity, causing the possibility that the $i+1$-th job will be enqueued.

Continuous time Markov chains (CTMCs) are employed as the mathematical framework for computing exact QN solutions. Indeed, in principle it is always possible to translate each QN model in a low-level Markovian process that once solved, with standard techniques [22], gives back the value for the performance metrics of interest.

As an example Figure 2.1b depicts the state space diagram for our running example with parameters $C = 2, s_1 = 3, s_2 = 3$. The idea of this graphical representation is to show how the user currently interacting with the system stochastically circulates in the network. In particular, each state of the system (i.e., a node in the diagram) is annotated with the vector $x(t)$ whose individual components $x_i(t)$ indicate the number of users waiting or in service at station i at time t. Each directed edge from station i to station j in the diagram represents the movement (i.e., the jump) of a client between the two stations (i.e., a client that has received the service from station i goes to station j). Each jump is fired according to an exponentially distributed random variable whose rate is reported on the edge. Moreover, since the probability of an arrival at time t and a departure ad the same instant of time is 0, only one user can move per unit of time. Equation (2.1) denotes the mathematical representation of the stochastic behavior of the system, in terms of infinitesimal matrix generators. Then, relying on standard numerical techniques [22] it can be directly employed for performance analysis.

$$Q = \begin{pmatrix} -(p_{0,1}\mu_0 + p_{0,2}\mu_0) & p_{0,1}\mu_0 & p_{0,2}\mu_0 \\ \mu_1 & -\mu_1 & 0 \\ \mu_2 & 0 & -\mu_2 \end{pmatrix} \tag{2.1}$$

Unfortunately, the exact analysis of CTMT suffers from the well known *state explosion problem* [17]. Indeed it is easy to see from Figure 2.1b and (2.1) how the number of the states of the CTMC (hence the number of equations to be solved) exponentially increases with the number of clients interacting with the system, making an exact transient quickly analysis prohibitive.

Steady-state analysis of CTMC has been exploited for efficiently computing performance metrics of interest when dealing with complex systems [31]. The main benefit is the ability to study the system's quantitative behavior without explicitly generating the state space of the underlying

stochastic process, but relying on closed-form expressions only. Many efficient algorithms for computing the steady state performance metrics of queueing networks have been proposed in the literature [22]. Among these, one of the most important one is the mean value analysis (MVA) [32]. This algorithm is grounded on two foundational equations of queuing theory: *i*) the Little's theorem [33] and *ii)* the Arrival theorem [34]. Hereafter only the former is reported since it is used later in this thesis. We refer the reader to [22] for a detailed discussion of MVA.

The following equation states the Little's theorem:

$$\overline{X} = \Lambda \overline{R} \tag{2.2}$$

Basically, it expresses the steady state number $\overline{X}$ of customers in a stationary system as the product between the steady state arrival rate Λ and the steady state time $\overline{R}$ spent by a customer in the system (independently by the arrival and departure process distribution). In Chapter 3 we use it for computing the steady state response time of closed queuing network knowing the average number of customer in the system and the average steady-state throughput.

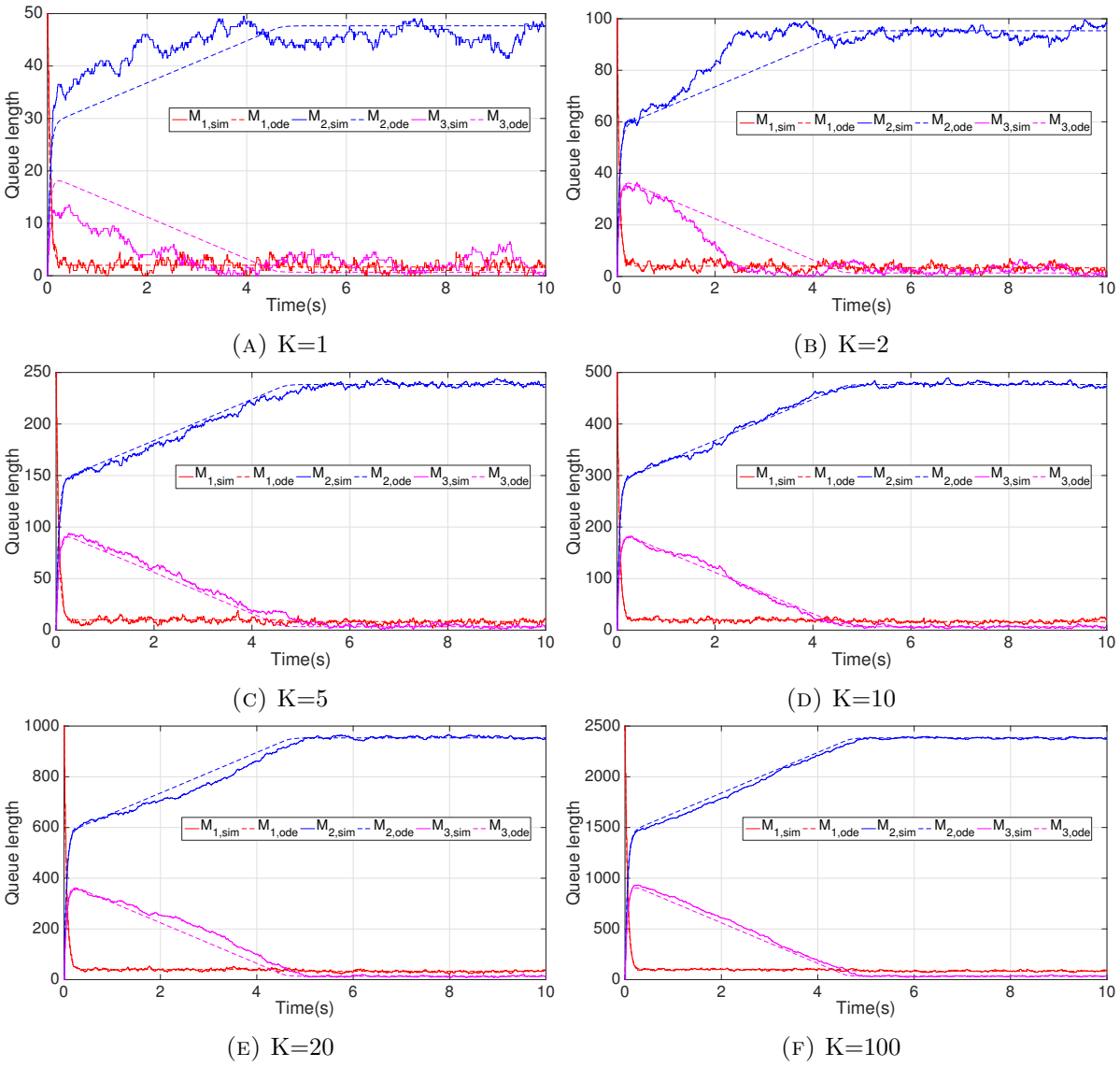
Although the QN steady state solutions have been thoroughly studied in the literature, the conditions needed for ensuring their applicability (i.e., the system equilibrium condition) are often too restrictive to be met by real-world systems. In particular, it is well known how steady state cannot be easily guaranteed on time-inhomogeneous queuing networks [35], which represent a natural modeling abstraction for those systems characterized by time-variant parameters such as in the case of self-adaptive systems.

In the following the fluid queuing network model is presented as an effective mathematical formulation for addressing the state space explosion problem of CTMCs while allowing a tractable transient analysis.

2.2 Fluid Approximation of Queuing Networks

The purpose of this section is to briefly pinpoint the essential concepts underlying the fluid approximation analysis of queuing networks. We refer to [36] for a more formal discussion about this topic.

The key intuition behind fluid approximation is to consider a dynamical model, i.e., a set of coupled ordinary differential equations (ODE), of the QN that provides an estimate of its transient evolution approximately. By doing so we are able to avoid the well-known problem of state space explosion that would be incurred by an exact dynamical description of the underlying CTMC [37]. Formally the transient evolution of the fluid approximation of the QN is described



(A) K=1 (B) K=2

(C) K=5 (D) K=10

(E) K=20 (F) K=100

FIG. 2.2: Comparison between a sample path of the CTMC and the fluid model

by the following system of ODEs:

$$\dot{x}_i(t) = -\mu_i(t)\min\{x_i(t), s_i(t)\} + \sum_{j \in S} p_{j,i}(t)\mu_j(t)\min\{x_j(t), s_j(t)\} \tag{2.3}$$

where we use the dot notation in the left-hand sides to denote derivative with respect to time (i.e., $\frac{dx_i(t)}{dt}$) with initial condition $x_i(0)$, for all $i \in S$.

The quantity $\mu_i(t)\min\{x_i(t), s_i(t)\}$ represents the overall *nonlinear* instantaneous throughput of each station [38]: when the queue length $x_i(t)$ in station i is less than the available servers $s_i(t)$, then the $x_i(t)$ jobs are served in parallel; otherwise some of the jobs are enqueued and only $s_i(t)$ of them are processed at once. The throughputs may be weighted by the routing probabilities $p_{j,i}(t)$ because a station may receive only a fraction of the jobs elsewhere.

The following ODE system gives the fluid approximation for the load balancer used as running

example (see Figure 2.1a):

$$\begin{aligned}
\dot{x}_1(t) &= -\mu_1 \min\{x_1(t), s_1\} + \mu_2 \min\{x_2(t), s_2\} + \mu_3 \min\{x_3(t), s_3\} \\
\dot{x}_2(t) &= -\mu_2 \min\{x_2(t), s_2\} + p_{1,2}\mu_1 \min\{x_1(t), s_1\} \\
\dot{x}_3(t) &= -\mu_3 \min\{x_3(t), s_3\} + p_{1,3}\mu_1 \min\{x_1(t), s_1\}
\end{aligned} \tag{2.4}$$

In general, the QN fluid approximation affords two related interpretations. Roughly speaking, the first is a first-order moment-closure approximation of a given fixed CTMC representing a population process [36]; The second interpretation is that the ODE solutions coincide with a sample path of the underlying CTMC under appropriate *limiting and scaling conditions* [39]. Due to this fact, the accuracy of the approximation tends to improve as the number of jobs and server multiplicities increase [40].

As a numerical example for showing the trend of the approximation error, Figure 2.2 depicts the queue-length time-course evolution for the load balancing system (see Figure 2.1a). In each subplot, both the stochastic simulation and ODEs solutions are reported considering a QN where the initial condition $x(0)$ and the server multiplicity s_i are scaled up by different values of the factor $K \in \{1, 2, 5, 10, 20, 100\}$. It is clear that with an increasing scaling factor K the stochastic simulation trace and corresponding ODE solution become indistinguishable. In the following chapters we exploit this key feature of fluid QN as the enabling technology for defining different performance-driven self-adaptation approaches.

2.3 Software Engineering for Self-adaptive Systems

Modern software systems need to become more resilient, efficient and highly configurable by adapting to changes possibly occurring in their operational contexts, environments and requirements [15]. This ever increasing complexity requires software engineers and architects to design systems in innovative ways. It has been widely recognized in the literature that the main challenges arise when conferring to these systems a parsimonious resources usage with respect to the actual users' demand, and the ability to continuously fulfill functional and nonfunctional requirements while executing under an uncertain environment [41–47].

For these reasons, self-adaptation has become an important research topic in many application areas [48]. It works towards the specification at runtime of the most suitable system configurations such that the system behavior is preserved while meeting functional and non-functional requirements. The idea is to make systems aware of themselves by monitoring their runtime dynamics for evaluating the quality attributes of interest. If the obtained metrics are outside the acceptable ranges then the system adapts itself relying on some refactoring policies. The self-adaptation capability of these systems is usually based on the MAPE-K control loop [14].

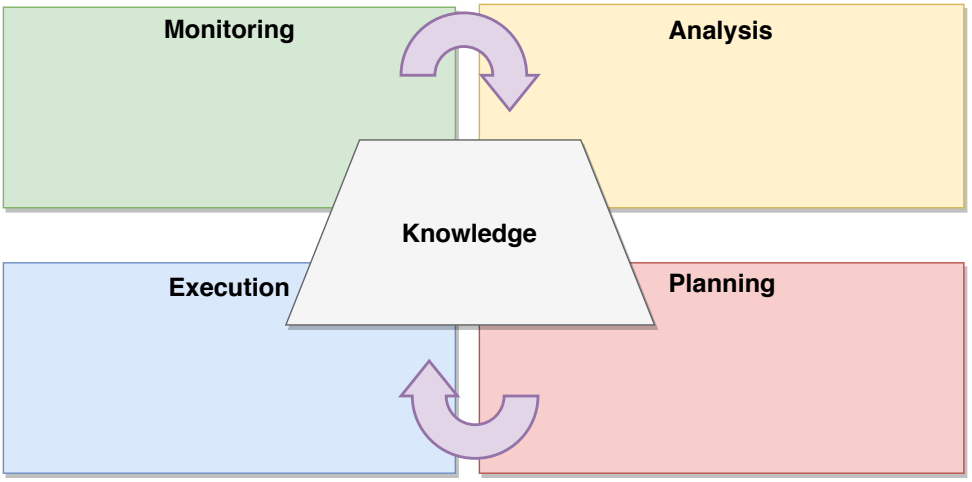


FIG. 2.3: MAPE-K control loop

Figure 2.3 depicts the reference architecture of this runtime control methodology: a self-adaptive system by monitoring its behavior extracts the actual runtime parameters (Monitoring phase), then uses them to feed a model at runtime that allows an accurate analysis of the current execution conditions (Analysis phase). If problems are detected, it is able to plan a set of refactoring strategies (Planning phase) that, once reflected on the running system, steer it in a problem-free state (Execution phase). The knowledge block is used as the shared repository (among all the different phases) to store all the information representing the awareness of the system about itself.

A huge number of self-adaptation approaches based on the MAPE-K control loop and other software engineering techniques have been already proposed in the literature. In the following those that we consider the most relevant with respect to the research directions of this thesis are surveyed.

2.3.1 A Research Landscape

In [48] Macías-Escrivá et al. analyze various techniques applied in the design of self-adaptive systems. The authors classify the different approaches according to the underlying principles used to endow systems with self-adaptation capabilities, such as *component based software engineering* [41, 42], *service based systems* [43, 44], *multiagent systems* [45–47], *model-driven* [49, 50] and *nature-inspired engineering* [51–55].

Beyond the aforementioned classical software engineering paradigms, there is literature on works providing a compact representation of all system configurations for runtime self-adaptation purposes. A step forward along this research direction can be found in [56], where Hallsteinsen et al. introduce the concept of dynamic software product line engineering (DSPL) as an extension of the classic software product line (SPL) development paradigm [57] (i.e., a well assessed methodology according which the software is created and evolved as a family of similar and

interconnected products). The novelty of DSPL is identified in the shift of the variability binding (products derivation) from design-time to runtime allowing the management of the environment uncertainties. Moreover, they identify self-adaptation as one of the possible future application of this new vision in SPL.

As a first extension of these ideas, in [58], the authors propose a feature-oriented method to support runtime variability reconfiguration in DSPLs. The method introduces the concept of a role model, an intermediate level between feature variations and implementations to improve their traceability. They also underline that an important engineering principle that DSPL can bring to self-adaptive systems is the feature dependency and constraint mechanism. By relying on the feature constraints specified in the feature model, it is possible to represent require or exclude dependencies among feature variations, and combine them into runtime adaptation controls to help to ensure consistent and complete feature bindings and re-configurations. From our perspective, the main drawback of this technique is the formalization of the adaptation capabilities (i.e., the adaptation space) according to the principle *on event if condition do action*, hence explicitly enumerating all the possible events and actions relations. This approach quickly becomes computationally intractable when dealing with large adaptation spaces. Moreover, they focus only on functional aspects without considering runtime violation of extra-functional requirements.

Differently, in [59] the authors propose an efficient technique for performance analysis of SPL systems by using a class of UML activity diagrams annotated with quantitative information [60–62]. The main benefit of this approach is the enabling of a family-based evaluation (i.e., the performance evaluation of the system is conducted on the whole family and not product by product) by means of symbolic computation, which has been shown to be orders of magnitude faster than product-based analysis. As we will see later, we exploit similar ideas for defining the different self-adaptation methods proposed in this thesis (see Chapters 3-5).

Other works [63, 64] can be found in the literature that exploit DSPL in the service oriented architecture (SOA) domain for runtime service selection or reconfiguration. Unfortunately, very few of them are addressing reconfiguration problems triggered by quantitative QoS violations but focus only on qualitative non-functional aspects. Moreover, none of the analyzed approaches rely on the solution of analytic models for performance evaluation but just on aggregation functions, i.e., the relations between the performance dynamics of the systems and their parameters are specified by simple mathematical functions that do not reflect the actual quantitative behavior. This is due to the generally used assumptions in the SOA domain of the availability of services with an infinite number of servers (i.e., no queuing phenomena). Moreover, none of the proposed approaches are able to deal with the intrinsic uncertainty arising from the modern application execution conditions such as shared virtualized environment or cloud-based deployment.

The techniques proposed in this thesis extend the state of the art approaches thanks to an

innovative usage of QNs for optimal runtime configuration selection enabling an implicit configurations representation and an efficient adaptation space exploration.

2.3.2 Uncertainties in Self-adaptive Systems

Esfahani and Malek in [65] define the problem of uncertainties in self-adaptive systems. They state that while many promising approaches for the construction of self-adaptive software systems have been developed, the majority of them ignore the uncertainty underlying the adaptation due to its vague understanding. They characterize the sources of uncertainties in two main categories: *internal* and *external*. While the latter arises from the environment or domain in which the software is deployed, the former is rooted in the difficulty of determining the impact of adaptation on the systems quality objectives (i.e., due to incomplete or too abstract model). Following this categorization, the authors identify the sources of uncertainties in system parameters, analytical models, user preferences and environment attributing their causes to the intrinsic variability or lack of knowledge about the system.

From our perspective, the main contribution of that work is the proposal of an alternative definition of optimality in making adaptation decisions that incorporates uncertainty. A system configuration is optimal if it concurrently satisfies the following three objectives: *i)* maximizes the most likely utility for the system under uncertainty; *ii)* maximizes the positive consequence of uncertainty; *iii)* minimizes the negative consequence of uncertainty. Basically, it means that given the variability in the system and in environmental parameters, an optimal solution is not the one that achieves the highest utility at the point in time in which the decision is made, but the one that anticipates the future behavior of the selected configuration over time. Another benefit of the new optimality criteria advocated in the paper is that since under the reconceptualized notion of optimality the system is expected to maintain a higher utility in its future operation, this approach decreases the number of adaptations compared to traditional approaches. This in essence results in more stable self-adaptive software systems.

Perez-Palacin et al. in [66] focus on the usage of both design-time and runtime software models and give a taxonomy for classifying different types of uncertainties. They identify three main dimensions for classifying uncertainties: location, level, and nature. The *location* of uncertainty refers to the place where the uncertainty manifests itself within the model. The *level* of uncertainty is where the uncertainty reveals itself between deterministic knowledge and total ignorance. Finally, the *nature* of uncertainty refers to whether the uncertainty is due to the imperfection of the acquired knowledge or is due to the inherent variability of the phenomena being described. Afterward, they used these dimensions to classify the different sources of uncertainties that can be found in the literature.

In [67, 68] the authors propose *GuideArch*, a quantitative framework for the exploration of architectural solution space under uncertainty. The goal is not to eliminate uncertainty, but to

provide quantitative techniques and tools for making informed decisions in such circumstances. It is worth to notice that the scope of uncertainty in those papers has to do with not knowing the exact impact of architectural alternatives on properties of interest. For this reason, instead of modeling the anticipated impact of an architectural alternative on the systems properties as a point estimate, GuideArch represents it as a range of values. The key contribution of this work is the ability to provide quantitative analysis of the trade-offs between different utility functions relying on *fuzzy math* theory and on *triangular fuzzy value*.

We argue that the works presented in thesis advance the state of the art by showing how the QN fluid approximation can be exploited as a formal basis for an efficient adaptation space exploration while considering the system (e.g., departure process) and environment (e.g., the arrival process) uncertainty as first-class citizens.

2.4 A Self-adaptation Framework Based on Queuing Networks Fluid Analysis [2]

In thesis we focus on performance-driven self-adaptation, i.e., a specialized class of adaptive design employed for those systems whose key aspect is to maintain the prescribed QoS level regardless changes in their execution condition, e.g., varying the number CPUs or virtual machines assigned to a cloud-based application for meeting response time or throughput requirements under a fluctuating workload. Usually, in this adaptation scenario the system model is a quantitative analytic representation of the performance dynamic of the system (e.g., CTMC or QN). Then the model is periodically analyzed at runtime for computing the proper control signals (i.e., parameters value) that steer the system toward an optimal operating point.

Hereafter, an abstract MAPE-K loop for performance-driven self-adaptation based on QN fluid analysis is defined. Its main goal is to give systems the predictive power to anticipate performance flaws by automatically adapting software on the basis of the monitored runtime variabilities. It can be considered as the general framework on which all the approaches presented in this thesis are grounded. As we already said in Section 2.1, here we focus on QN with a single class of users while postponing the discussion of the multi-class case in Chapter 5.

Specifically, our framework can be seen as an implementation of the MAPE-K loop (see Figure 2.3), augmented with a parameter estimation phase. The overview of the proposed approach is depicted in Figure 2.4, in which we can identify the following components:

- **Monitoring:** the idea of this phase is to monitor the running system using some kind of runtime system profiler tool, similarly to [69]. Ideally, beyond the choice of a particular monitoring tool, we expect to periodically extract a set of parameters from the running

[2] The content of this section is mainly taken from our previous work [5]

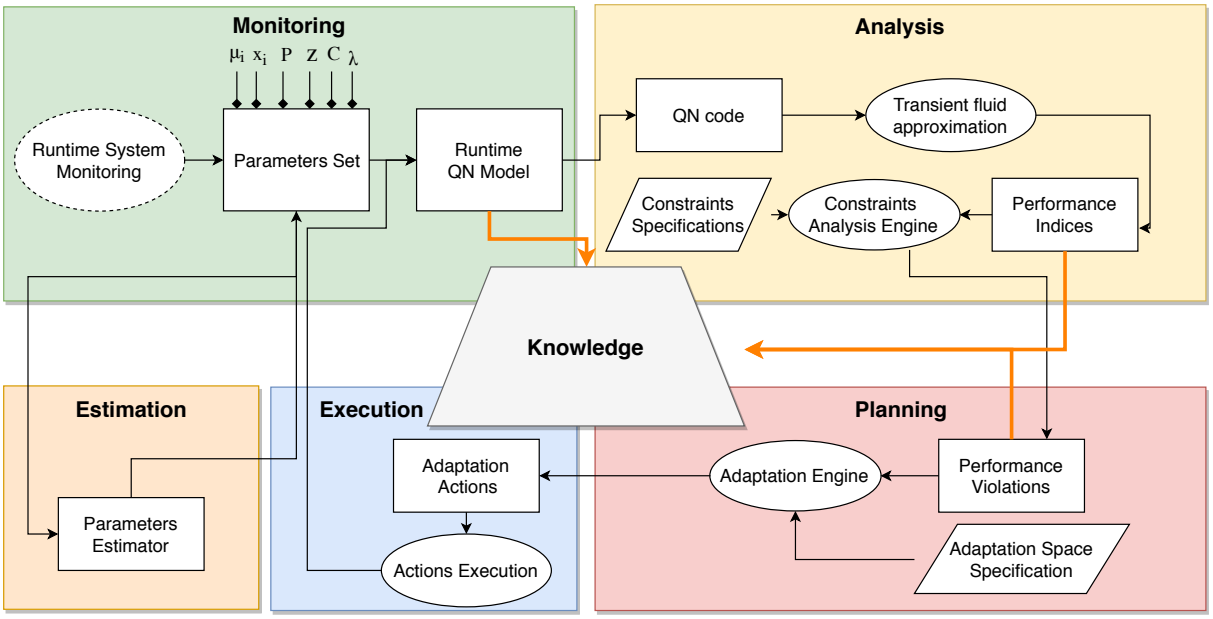


FIG. 2.4: Abstract fluid QN based self-adaptation framework

system. More precisely, we want to collect μ_i as the service rate for each service center, x_i as the estimated queue length at service center i and P as the routing matrix of the QN model. In the case of closed networks, we estimate the number of customers in the system N and the thinking time Z. On the contrary, in the 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 performance evolution of a running system.

- **Analysis:** in this step using an ad-hoc defined encoding (e.g., SMT in Chapter 3 and mathematical programming in Chapters 4,5) 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. The 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 performance violations. We envisage this last component as a modeling and analysis tool applicable for checking the QoS requirements against the performance evaluation of each specific system configuration (e.g., constraint logic programming [70] or an SMT solver [71]).

- **Planning:** starting from the *performance violations* previously computed, an *adaptation engine* component is able to compute a set of refactoring actions that represent the plan for the current adaptation iteration. We can think of these actions as the adding or removal of a service center (e.g., virtual machines), the modification of the service rates

(e.g., assigning different CPU sharing quota) or the change of the routing probability for some branch in the model (e.g., changing load distribution policy).

- **Execution:** executing the previously defined *adaptation actions* on the running system the self-adaptation mechanism is actually implemented.
- **Estimation:** goal of this phase is to use the structural information about the system and the monitored quantities for estimating the relevant QN parameters that cannot be directly measured (e.g., service demands). Then the estimated information is used for feeding the QN model used for predicting the performance dynamics of the systems.

We identify as *knowledge* of the MAPE-K loop the information that is required to describe the system in each self-adaptation iteration. Such information is identified by the runtime QN model, the estimated system parameters, the computed performance indices, and the detected performance violations. The control loop is repeated at runtime for each system evolution step so as to reduce the influence of uncertainty on each adaptation iteration. By instantiating each component of this framework, the different performance-driven self-adaptation approaches presented in this thesis can be derived.

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 generate the ODE representation. Our tool can be downloaded here http://sourceforge.net/projects/qnml.

2.5 Related Work

To give an overview of the novelties of our proposals, in the following we report a summary of the state-of-the-art works in common with all the self-adaptation methods presented in this thesis. A detailed discussion about the related work specific to each presented technique is reported in the corresponding Chapter (i.e., Chapters 3, 4, 5, 6).

All of the approaches presented in this dissertation relate to two main research areas: performance-driven self-adaptation and automated adaptation space exploration.

Performance-driven self-adaptation The need of runtime performance-driven self-adaptation has been widely recognized in the literature. Many approaches have been defined in the context of QoS driven runtime adaptation, in particular for SOA systems [43, 72], for the evolution of runtime parameters [73] and for quantitative verification [74]. However, in [75] the authors claim that assuring the required quantitative properties (e.g., performance, reliability) in self-adaptive systems is still an open challenge, and a relevant part of the studies use formal methods at runtime. Unfortunately, the usage of such methods is limited to modeling and analysis, whereas our approach includes a set of adaptation strategies aimed at automatically executing changes

in the running system. In [76] approaches for performance engineering of self-adaptive systems are surveyed and classified (design vs runtime [77], and reactive vs proactive [78]), outlining that model-driven tools including modeling of self-adaptive systems with an efficient automatic adaptation strategies selection are missing in the current state-of-the-art.

Automated adaptation space exploration A naive implementation of the planning phase of Figure 2.3, in principle, requires the evaluation of each possible QN design alternative from scratch, i.e., without the possibility of reusing the analysis across the different design choices. This induces a substantial overhead in the adaptation process hampering its applications to real world systems. For mitigating this challenge many research works have been proposed in the literature.

In [79] the authors present a declarative language (with a formal semantics defined in terms of a discrete-time Markov chain) that allows to explicitly represent uncertainty in the adaptation process by expressing alternative outcomes in the execution of the same adaptation action according to some probability distribution. In [79] a case study based on a QN model is used. However to compute the utility of a possible adaptation step it is necessary to analyze the model using traditional means (i.e., each design alternative is analyzed one by one). In [80], instead, the author proposes an approach that allows reducing the complexity of the exploration of the adaptation space only when specific conditions on the constraints for adaptation are met. In [81] the Active FORmal Models for Self-adaptation is proposed. It uses an integrated formal model of the adaptation components and knowledge models ensuring that the adaptation goals that are verified offline are then guaranteed at runtime by supporting dynamic adaptation. However, they use timed automata and timed computation tree logic [82] that does not allow to derive adaptations reusing previous computation, thus leading to an expensive exploration.

In order to overcome this limitation, the approaches presented in this thesis are inspired by the symbolic technique of [18, 59] that computes the solution of every possible design alternative at once. By doing so our methods are able perform an efficient automated adaptation space exploration choosing nondeterministically (see Chapter 3) or optimally (see Chapters 4, 5) among all possible feasible system configurations fulling the QoS requirements. In the following each of them is formally presented and discussed.

Chapter 3

Symbolic Performance Adaptation

In this chapter we propose a performance-driven self-adaptation technique based on queuing networks (see Chapter 2.1) as the quantitative model to represent system configurations. To find a model that fits with continuous changes in runtime conditions we rely on an innovative combination of symbolic analysis and satisfiability modulo theory. Through symbolic analysis we represent all possible system configurations as a set of nonlinear real constraints. By formulating an SMT problem we are able to devise feasible system configurations at a small computational cost. We study the effectiveness and scalability of our approach on a three-tier web system featuring different levels of redundancy. The content of this chapter is mainly taken from [1].

3.1 Overview

Performance evaluation of software systems at runtime is not a straightforward task since there are many sources of variability and uncertainty that make the process of guaranteeing QoS requirements very challenging. In this chapter we consider performance requirements such as response time, throughput, and utilization of resources, for software systems exposed to runtime variabilities such as server failures or changes in workload levels. Our key idea is to explicitly consider such uncertainties as first-class citizens in the analysis process. In particular, we propose a novel use of QNs models [83] that represent not only the system topology (e.g., software components, hardware devices) and parameters (e.g., workload, service rates), but also variability through a parametric representation of different system configurations [18]. This provides a symbolic evaluation of QoS predictions that can be leveraged at runtime in order to pro-actively adapt the system under study to changing conditions.

Based on previous works [18, 59] allowing parametric specifications for the number of servers, their service times, and the routing probabilities, it is possible to obtain a closed-form symbolic expressions for typical QN steady-state performance indices such as utilization and response times. The actual performance of a specific QN can then be obtained by simply plugging its

concrete parameters into the symbolic expression. Notably, topological variations to a model (such as the removal/addition of a service centre) can be accommodated by allowing certain parameters to be zero. In [18, 59] the authors are mainly concerned with analyzing many variants of a performance model effectively, but did not consider the problem of exploring these variants in order to find a configuration that satisfies certain QoS requirements. We remark that a naive strategy for doing this, i.e., by testing the feasibility of each possible configurations one by one, may be unfeasible when the parameter space is very large, or even impossible if it is infinite — for instance when allowing rates and probabilities to vary within a range.

In the approach presented in this Chapter, which we call *symbolic performance adaptation* (SPA), we tackle this issue by leveraging the symbolic QN solution as the underlying engine for adaptation. In particular, we consider the solution of [18] as it covers more expressive models than [59] because it allows for multiple-server queuing stations and generally distributed service times.

Technically, the main novel contribution here is to encode the QoS satisfaction problem as a satisfiability modulo theories one [19]. This allows us to formulate statements such as the existence of an assignment for the symbolic variables that satisfy a QoS constraint, specified in first-order logic. The peculiarity of this formulation is that, if the SMT problem is satisfiable, i.e., an assignment exists, then an SMT solver is able to produce a *witness*, i.e., one possible assignment. Instead, in the case the problem is not satisfiable then the modeler has the definite proof that no assignment can ensure the given QoS. This mandates a revision of the system and/or a relaxation of the QoS requirements. In other words, in doing so we exploit the symbolic variables as the runtime "knobs" for guiding the QoS-based adaptation process.

In this way, SPA allows to perform QoS analysis at two stages: *i)* at design-time we can provide assurances that the system can meet desired QoS; *ii*) at runtime, after detecting changing conditions, these can be encoded as further constraints on the model. These, in turn, trigger a new evaluation of the satisfiability problem, leading to a possible new configuration that meets the newly imposed constraints. Thus, it becomes crucial to evaluate the effectiveness and scalability of SPA in re-configuring the system at runtime. Here we present a numerical evaluation on a realistic QN model of a three-tiered distributed web system with service replications. We consider the challenging situation where the system presents various degrees of freedom in the adaptation capability. SPA can find a re-configuration that satisfies the desired QoS in face of random changes of workloads and injection of faults within few hundred milliseconds.

As a final remark we stress how, in relation to the abstract self-adaptation framework presented in Section 2.4, here we exploited the SMT problem formulation for conducting the analysis and planning phase at once. Indeed all the possible adaptation strategies (i.e., the adaptation space), the cost function (i.e., symbolic QN solution), and the target values (i.e., QoS requirements) are encoded in the same model and evaluated simultaneously. Moreover we assumed that all of

the system parameters could be directly measurable in the monitoring phase (i.e., no estimation phase is needed).

The remainder of this chapter is organized as follows. Section 3.2 presents the related work; Section 3.3 provides preliminaries concepts underlying the presented approach; Section 3.4 describes SPA by means of an illustrative example; Section 3.5 shows SPA at work on the three-tier web system; Section 3.6 discusses the main advantages and limitations of our approach and provides directions for future research.

3.2 Related Work

SPA deals with systems changing dynamically at runtime, in fact we use QN models including these uncertainties and providing a symbolic adaptation. In the literature several approaches have been proposed to deal with the problem of capturing runtime variabilities. There are two main streams of research in this direction: *i)* measurements-based approaches that monitor the actual system behavior variations and update parameters of analysis models accordingly; *ii)* prediction-based approaches that use model-based predictions to derive missing parameters of system specifications.

Measurement-based approaches. In [84] extended Kalman filters are used to track changes in parameters of QN models. Such filters model the dynamics of parameter changes that are triggered by measuring system performance and updating the performance model when prediction estimates are far from the actual measurements. KAMI (keep alive models with implementations) models system evolution by runtime adaptation [73] with continuous parameter estimation. Bayesian estimators are exploited to produce updated model parameters enabling continuous automatic verification of requirements at runtime. However, KAMI modifies models through the estimation of numerical parameters but it does not take into account structural changes. Instead, our SPA relies on QN models that anticipate structural modifications, besides others, since the topology of our performance models is symbolically represented. An adaptive filter to accurately learn time-varying transition probabilities of discrete time Markov chains (DTMCs) is proposed in [85]. It provides robustness to noise and fast adaptation to changes with a low overhead. However, this work focuses on learning and updating input parameters of analysis models similarly to [84]. In [13] the authors present an approach for runtime quantitative verification and sensitivity analysis, where the self-adaptation is achieved by bringing software models and probabilistic model checking to runtime, thus to support perpetual automatic reasoning about changes. In principle, all these approaches [13, 73, 84, 85] can be used to keep the parameters of our symbolic model updated. That is, they may represent the input of our adaptation process which can be triggered by a detection of parameter changes. A detailed discussion on this research topic can be found in Chapter 6 where an efficient runtime estimator for service demands of QN is presented.

Prediction-based approaches. In [86] an approach is presented for computing unknown service-demand parameters in multi-class open QN models. In particular, a closed-form solution is given for the case of a single missing parameter; instead, a constrained nonlinear minimization problem can be numerically solved for the case of multiple missing service demands. This approach is extended in [87] by considering input (i.e., arrival rates for each job class) and output metrics (i.e., the average response time for each job class) to estimate the per-class service demands and number of servers for a system QN model. Our approach can be seen as a generalization with respect to the input parameters that can be derived from QN models; indeed as output of the adaptation process SPA may obtain not only service demands and number of servers but other parameters such as routing probabilities. On the other hand, we only consider single class closed QN models.

The control-theoretic approach of [88] to adapt DTMCs for reliability requirements that need to be preserved over time considers specific changes that may lead to requirements violations (i.e., updates in certain probabilities associated with transitions), and a specific way to attempt adaptation (i.e., generating new values for other probabilities, which represent control variables). Our approach works at a different modeling level since it is concerned with performance instead of reliability, hence the adaptation process derives parameter values of different nature (e.g., number of servers, service demands).

In [89] a technique is presented for exploring the design space of complex systems by combining results from domain-specific languages, symbolic execution, and automated theorem proving in order to quickly move from a specification of a design space to a set of distinctive solutions. Similarly to [89], we use an SMT solver to solve a set of global design constraints and search for valid system instances, however our approach allows to encode both uncertainties and constraints in the SMT problem, thus to derive the adaptations directly from its formulation.

In [90, 91] uncertainty in performance model parameters was investigated by considering not only the workload, but also the operational profile, software/hardware demand, etc. However, the performance analysis relied on lower/upper bounds provided by software designers. In this work we broaden the scope of our research since no bounds are required, the system configurations fulfilling QoS requirements are dynamically generated on the basis of runtime variabilities.

A platform for the deployment and autonomic management of web-based applications in cloud is presented in [92]. Performance control is performed with two strategies, i.e., when the system becomes overloaded/underloaded: (i) add/remove one worker to/from the web cluster; (ii) use a performance model to determine the number of worker instances that should be added/removed. In the conducted experimentation the first strategy showed poorer results than the second. This leads to assess the usefulness of our approach that makes use of performance models not constrained with pre-defined changes, in fact the adaptations take place on the basis of runtime conditions.

3.3 Preliminaries

In the following we introduce the main concepts underlying SPA, i.e., the symbolic analysis and and the satisfiability modulo theory. For a discussion about queuing networks and their fluid approximation we refer the reader to Section 2.1.

Symbolic Analysis When *some* of the parameters (e.g., routing probabilities, service rates, etc.) of a QN are left unspecified, the model cannot be solved using conventional numerical algorithms. We make use of a recently presented result [18] which exploits symbolic computation to provide an approximation to steady-state performance indices. This approximation arises from a *fluid limit* of Section 2.2. Finding a stationary solution to these ODEs can provide an estimate to the steady-state indices of the QN. In particular, rewriting the stationary point in terms of the throughputs yields a linear system of equations for which a symbolic solution can be obtained. The output is constituted by expressions that depend on symbols representing the unspecified parameters. Plugging in concrete values in these symbolic expressions provides the performance indices of the corresponding QN where all parameters are instead specified.

The main advantage of this analysis is that the computation of the symbolic expression can be reused among many models that share commonalities while exhibiting differences in topology and/or parameters. Indeed, the results of [18] have been motivated by software product lines applications where the symbolic solution represents the solution of the so-called *150%* model, i.e., an object that subsumes every concrete configuration of a product line model [93].

Satisfiability Modulo Theories As discussed above, we take the results of [18] as the starting point of this chapter. We interpret the symbolic expressions as a set of constraints that the parametric variables have to satisfy together with further constraints that formally express QoS requirements. We encode this into a satisfiability problem, and we are able to automatically obtain an assignment of parameters that satisfy these constraints, or a formal certificate that no assignment can satisfy the requirements.

The encoding is done by means of satisfiability modulo theories which check the satisfiability of logical formulas over one or more theories [94]. This is needed because we combine boolean satisfiability (SAT) with appropriate domains, such as those studied in convex optimization and term-manipulating symbolic systems [95]. More precisely, the propositional atoms that form the SAT formulae can be represented by uninterpreted functions, linear inequalities over the rational numbers or bit vector, or other elements belonging to a particular theory. In our case, the constraints involve inequalities of symbols interpreted as real values (e.g., routing probabilities and service rates). For instance, given the formula $3x+2y-z \geq 4 \wedge x \geq 0 \wedge y \geq 0$ an SMT solver [71, 96] looks for an assignment of real variables x, y, z that satisfy the inequalities. The solution is based on the tight integration of propositional SAT solvers with dedicated

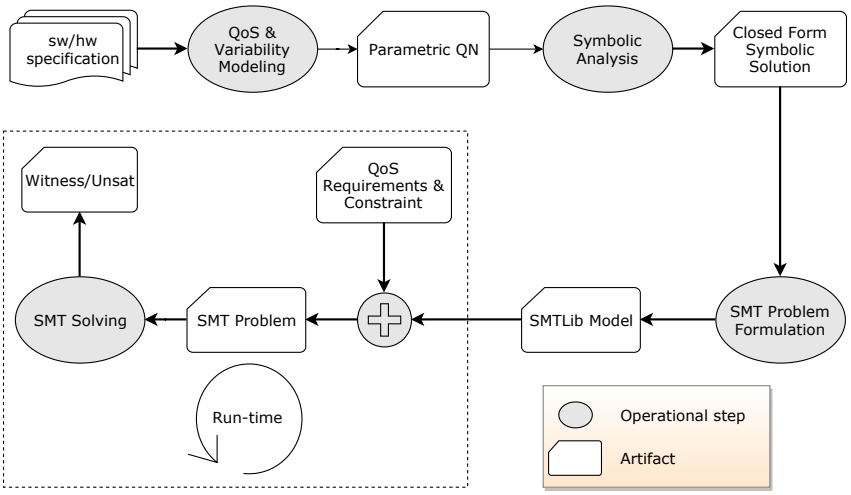


FIG. 3.1: Overview of the symbolic performance adaptation.

procedures to reason about the theory component. It combines the power of the well-known constraint programming techniques such as linear or nonlinear programming with first-order-logic [97]. We exploit this combination to model the quantitative properties of the systems.

3.4 The Symbolic Performance Adaptation Approach

Our idea is to see a self-adaptive system as a highly configurable one, which is able to change its configuration at runtime in response to internal or external events. In this view, the role of self-adaptation becomes finding a suitable system configuration such that QoS requirements are continuously fulfilled. To this end, we define a suitable SMT encoding of the symbolic solution of the QN model augmented with QoS requirements. Then, we use an SMT solver, specifically the well known Z3 [96] in our case, for devising a feasible system configuration. This allows us to enable the runtime adaptation capabilities of the system through a white-box approach.

Figure 3.1 depicts an overview of SPA, where the grey ellipse boxes represent operational steps while the rounded rectangles identify different kind of artifacts (e.g., models). The first step is to model the QoS properties of the system using a parametric version of QNs (i.e., the 150% model) that allows designers to single out the adaption knobs that are accessible in the system. Instantiating these knobs with concrete variables leads to all possible system configurations, compactly and symbolically expressed in a single model. Then, symbolic evaluation yields a set of real nonlinear equations (i.e., the symbolic solution). Finally, augmented with QoS requirements and constraints, we formulate an SMT problem that, once solved, is able to efficiently devise feasible system configurations, or to assess their absence. At runtime, the SMT problem is continuously updated with monitored information (e.g., workload fluctuations) in order to devise feasible system configurations taking into account the actual runtime variabilities. We

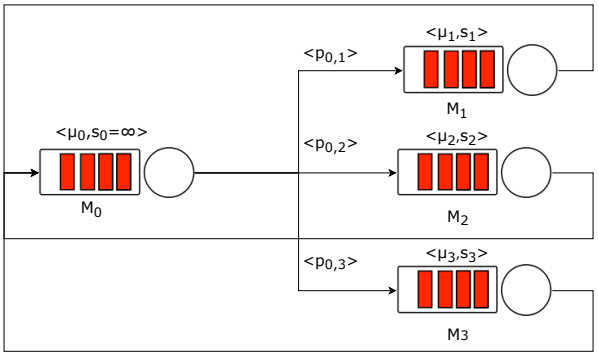


FIG. 3.2: Queuing network model for the load balancing system.

remark that an adaptation knob represents a naturally continuous variable such a routing probability, and the whole configuration space becomes uncountably infinite. Yet the power of the SMT encoding will allow us to make formal and precise statements about the existence (or the absence) of a feasible configuration.

In the remainder of this section we explain in detail all the operational steps belonging to our approach. For clarifying the discussion we use a load balancing system reported in Figure 3.2 as a running example (see Section 2.1 for a discussion on load balancing system). Here our goal is to endow such systems with self-adaptation capabilities, thus to efficiently explore, at runtime, the system's configurations space guided by QoS requirements (e.g., response time upper bound) and resource design constraints (e.g., maximum number of replication) in response to uncertain events.

3.4.1 The Parametric QN

In this Chapter we focus on closed single class QN with multiple and independent servers. Its formal description is essentialy identical to the one reported in Section 2.1 but now ready to explicitly include variability. In particular, the modeler at design time identifies parameters to be left unspecified in the form of *symbolic variables*. These will represent the "knobs" of the model that can be changed at runtime. Following [18], we set up a linear system of equations based on the routing probability matrix $P = (p_{ij})_{i,j \in S}$ as follows:

$$P'\mathcal{T} = \mathcal{T}, \qquad \mathcal{T} = (\mathcal{T}_i)_{i \in S} \tag{3.1}$$

where $'$ denotes matrix transposition and $\mathcal{T}$ is the vector of unknown steady-state *throughputs*, for each queuing station.

It is worth to remark that due to the unspecified parameters, these equations can only be solved symbolically using a computer algebra system. We obtain a symbolic expression for all throughputs $\mathcal{T}_i$, computed exploiting the following theorem [18]:

Theorem 1. Let QN_s be a parametric QN model of the system S, let $e_i = \frac{1}{\mu_i}$ be the service time of station i and P be the routing probability matrix of QN_s with (3.1) its traffic equations. Denote by ζ the unique solution of (3.1) when $\zeta_1 = 1$. Then for all $i \in S$,

$$\mathcal{T}_i = \begin{cases} \zeta_i \left(\sum_{j \in S} e_j \, \zeta_j\right)^{-1} C & \text{if (3.2)} \\ \zeta_i \, \min\{s_j \, (e_j \, \zeta_j)^{-1} | j \in S \wedge s_j < \infty\} & \text{else.} \end{cases}$$

$$s_i > \frac{e_i \, \zeta_i \, C}{\sum_{j \in S} e_j \, \zeta_j}, \qquad \forall i \in S \tag{3.2}$$

For illustration purposes we consider the running example with the parameter choice summarized in Table 3.1, where the symbol '—' refers to the variable parameters. For simplicity, we assume exponentially distributed service times. With this, the system of equations (3.1) reads

$$\begin{pmatrix} 0 & 1 & 1 & 1 \\ p_{0,1} & 0 & 0 & 0 \\ p_{0,2} & 0 & 0 & 0 \\ p_{0,3} & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} \mathcal{T}_0 \\ \mathcal{T}_1 \\ \mathcal{T}_2 \\ \mathcal{T}_3 \end{pmatrix} = \begin{pmatrix} \mathcal{T}_0 \\ \mathcal{T}_1 \\ \mathcal{T}_2 \\ \mathcal{T}_3 \end{pmatrix}$$

where $p_{0,1}$, $p_{0,2}$, and $p_{0,3}$ are the symbolic parameters. Applying the theorem 1 to the running example yields:

$$\mathcal{T}_0 = \begin{cases} \dfrac{C}{(e_0 + \frac{p_{0,1}}{10} + 2p_{0,3} + e_2 \, p_{0,1})} & \text{if } (*) \\ \min\left\{\dfrac{4}{e_0}, \dfrac{10 s_1}{p_{0,1}}, \dfrac{30}{e_2 \, p_{0,2}}, \dfrac{s_3}{(2 p_{0,3})}\right\} & \text{else,} \end{cases} \tag{3.3}$$

$$\mathcal{T}_1 = \begin{cases} \dfrac{C \, p_{0,1}}{(e_0 + \frac{p_{0,1}}{10} + 2p_{0,3} + e_2 \, p_{0,1})} & \text{if } (*) \\ p_{0,1} \min\left\{\dfrac{4}{e_0}, \dfrac{10 s_1}{p_{0,1}}, \dfrac{30}{e_2 \, p_{0,2}}, \dfrac{s_3}{(2 p_{0,3})}\right\} & \text{else,} \end{cases}$$

$$\mathcal{T}_2 = \begin{cases} \dfrac{C \, p_{0,2}}{(e_0 + \frac{p_{0,1}}{10} + 2p_{0,3} + e_2 \, p_{0,1})} & \text{if } (*) \\ p_{0,2} \min\left\{\dfrac{4}{e_0}, \dfrac{10 s_1}{p_{0,1}}, \dfrac{30}{e_2 \, p_{0,2}}, \dfrac{s_3}{(2 p_{0,3})}\right\} & \text{else,} \end{cases}$$

$$\mathcal{T}_3 = \begin{cases} \dfrac{C \, p_{0,3}}{(e_0 + \frac{p_{0,1}}{10} + 2p_{0,3} + e_2 \, p_{0,1})} & \text{if } (*) \\ p_{0,3} \min\left\{\frac{4}{e_0}, \frac{10 s_1}{p_{0,1}}, \frac{30}{e_2 \, p_{0,2}}, \frac{s_3}{(2 p_{0,3})}\right\} & \text{else,} \end{cases}$$

$$(*) = \begin{cases} 4 > (C \, e_0)/(e_0 + \dfrac{p_{0,1}}{10} + 2 \, p_{0,3} + e_2 \, p_{0,2}) \\ s_1 > (C \, p_{0,1})/(10 \, (e_0 + \dfrac{p_{0,1}}{10} + 2 p_{0,3} + e_2 \, p_{0,2}) \\ 30 > (C \, e_2 \, p_{0,2})/(e_0 + \dfrac{p_{0,1}}{10} + 2 p_{0,3} + e_2 \, p_{0,2}) \\ s_3 > (2 \, C \, p_{0,3})/(e_0 + \dfrac{p_{0,1}}{10} + 2 p_{0,3} + e_2 \, p_{0,2}) \end{cases}$$

TABLE 3.1: Parameter choices for the running example; '—' indicates that the variable is considered symbolically.

Parameter	**Value**	**Parameter**	**Value**
e_0	—	s_0	4
e_1	0.1	s_1	—
e_2	—	s_2	30
e_3	2.0	s_3	—
$p_{0,1}$	—	$p_{0,2}$	—
$p_{0,3}$	—	C	—

We remark that the parametric QN representation allows to encode arbitrarily complex and structural changes to the system. For instance, by setting $p_{0,1} = 0$ and $s_1 = 0$ it is possible to "power down" one server, which can model either a power failure or a downscaling of resources due to a lower workload. Moreover, we can express additional performance indices of the system over the specified symbolic variables such as $\mathcal{U}_i = \frac{\mathcal{T}_i e_i}{s_i}$ modeling in this way the utilization ratio at the i-th queuing station.

The important feature of this construction is that the performance of a concrete configuration, e.g., setting $C = 20, s_1 = 3, s_3 = 4, e_0 = 5, e_2 = 6, p_{0,1} = p_{0,2} = p_{0,3} = \frac{1}{3}$, can be obtained by simply plugging in these values in (3.3). For this case, it would yield

$$\mathcal{T}_0 = 0.80 \qquad \mathcal{T}_1 = 0.26 \qquad \mathcal{T}_2 = 0.26 \qquad \mathcal{T}_3 = 0.26.$$

3.4.2 SMT Problem Formulation

We observe that all symbolic expressions can be expressed in nonlinear real arithmetic theory ($\mathcal{NRA}$). Thus, our idea is to encode the system's symbolic solution, combined with the QoS requirements and resources constraints, in an SMT problem that efficiently devises a feasible system configuration. Our formulation reflects Zave and Jackson's seminal work on bridging the gap between requirements and specifications [98]. More precisely, let $\mathcal{M}$ be the symbolic solution of the QN, let $\mathcal{Q}$ be a set of QoS requirements (e.g., an upper bound on the response time), $\mathcal{D}$ be a set of domain assumptions (e.g., workload levels), and $\mathcal{R}$ be a set of resource constraints (e.g., at most 10 servers are available). We encode $\mathcal{M}, \mathcal{Q}, \mathcal{D}$ and $\mathcal{R}$ as first-order-logic formulae over the $\mathcal{NRA}$ theory such that the QoS-based adaptation is turned into the satisfiability problem which we call the *adaptation problem*, $\mathcal{AP}$:

> *Find an assignment of the variables for model $\mathcal{M}$ that ensures $\mathcal{Q}$ subjected to constraints $\mathcal{D} \wedge \mathcal{R}$.*

The SMT model encoding the symbolic solution $\mathcal{M}$ for the load balancing system (see Figure 3.2) is publicly available[1].

[1] https://rise4fun.com/Z3/1jA8

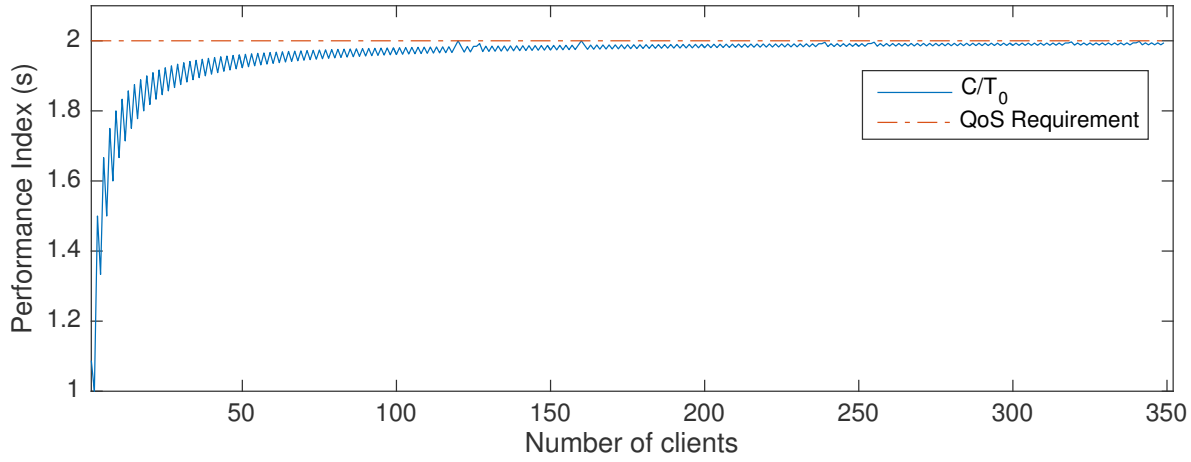


FIG. 3.3: Load balancing $\mathcal{Q}$ fulfilment.

To completely specify the problem we consider in $\mathcal{M}$ the QoS requirements:

$$\mathcal{Q} := \left\{ \mathcal{T}_0 \geq \frac{C}{2} \right\},$$

the domain assumptions

$$\mathcal{D} := \{1 \leq C \leq 350\},$$

and the resource constraints

$$\mathcal{R} := \left\{ \forall i, j \in S. \sum_{k \in S} p_{i,k} = 1.0 \wedge \left(0 \leq p_{i,j} \leq 1.0 \wedge 1 \leq s_i \leq 40 \wedge 0.02 \leq e_i \leq 10\right) \right\}$$

This formulation requires that the ratio between the number of clients and the throughput of the reference station must be lower or equal than 2 time units (e.g., sec, ms) while the system is serving form 1 up to 350 users with available system resources spanning from 1 to 40 servers for each service station and a service time between 0.02 and 10 time units. Moreover, the formulation enforces that the symbols modeling the routing probabilities are associated to values that make the QN a valid closed one.

By embedding in $\mathcal{AP}$ the actual runtime values for $\mathcal{D}$ we are able to devise feasible system configurations that can be used for avoiding QoS violations consuming the available resources constrained by $\mathcal{R}$. Hereafter we report some numerical results about the solution of adaptation problem $\mathcal{AP}$ formulated in Section 3.4.2. We evaluated the self-adaptation capabilities of the load balancing system for a series of 350 self-adaption steps. For each of them we generated a linear increasing workload C satisfying the domain assumptions $\mathcal{D}$ and collecting the information about the throughput of the scheduler node $\mathcal{T}_0$ and the self-adaptation overhead (i.e., needed computation time).

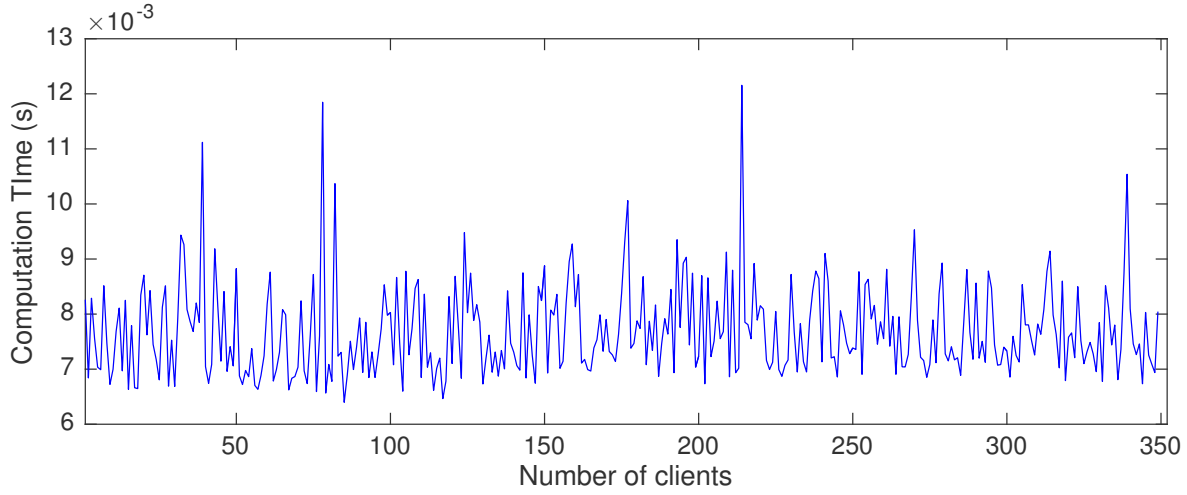


FIG. 3.4: Self-adaption computation time

Figure 3.3 shows the effectiveness of our approach for allowing a load balancing system to continuously meet its QoS requirements $\mathcal{Q}$ (red dotted line) despite a highly variable workload. In Figure 3.3 we can see that the blue line (i.e, $\frac{C}{T_0}$) is always below its threshold (i.e., 2), thus assessing the fulfillment of $\mathcal{Q}$ for any value of the workload C. Instead, Figure 3.4 shows the computation time needed by the load balancing system to recompute a new configuration. For this model the solver was very efficient, yielding a new configuration in a few milliseconds on average.

We remark that the effectiveness of the SMT solver may depend on the choice of the domain used to encode inherently integer parameters such as server multiplicities and client populations. Declaring them as integers lead to nonlinear mixed integer constraints that may turn out to be more difficult to handle than using reals. In fact, in this case Z3's *nlsat* is a decision procedure for such formulae (that belong to the quantifier-free fragment of first-order logic), so the solver will return an answer (given enough resources). In our tests we chose an encoding into reals, appropriately rounding after the solution was obtained.

This is also somewhat justified by the inherent assumptions made for deriving closed form symbolic expressions, whereby one considers the fluid limit of large enough copies of servers and large populations of jobs. It is not difficult to show that the same systems of equations are valid for variables that represent *population densities*, i.e., proportions of server multiplicities for each client in the system [99]. Then, one would equivalently consider an SMT encoding where all variables are naturally reals, and then interpret the solution through rounding to an integer afterward.

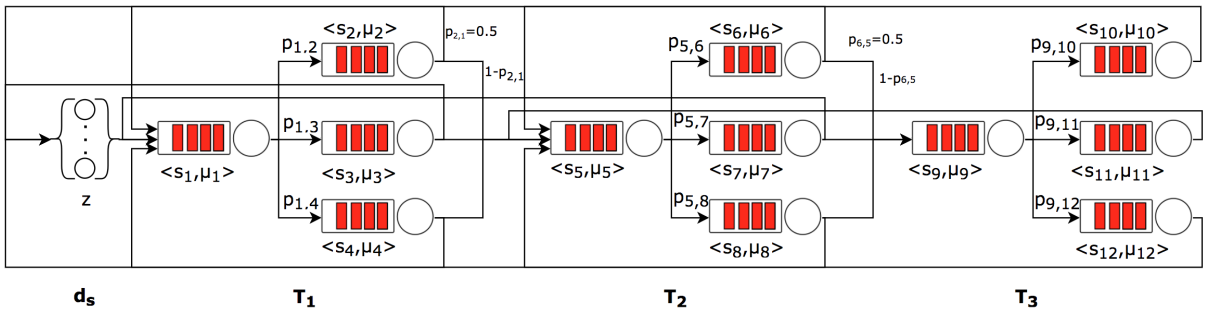


FIG. 3.5: QN model for a three-tier system based on [6], where each tier has three replicated servers.

3.5 The Case Study: A Three-Tier System

In this section we evaluate the effectiveness of SPA on a more substantial case study. We consider a model of a three-tier system consisting of a data-management tier (encompassing one or several database servers), an application tier (business logic), and a client tier (interface functionality). We study the possibility of load balancing within tiers, especially for the application tier, by distributing requests across the different application server instances.

All the numerical results shown in this section are available for replicability at https://goo.gl/wmmnCK.

Model description Our case study is based on a QN model presented and successfully validated against a real system in [6]. The QN is shown in Figure 3.5, where we can see that the model is composed of tiers T_1, T_2, T_3 plus a delay station d_s used for emulating the session-based workload typical of web applications. We modeled d_s as a queuing center with a high number of servers such that at least one is always available for a client. With this consideration its thinking time Z becomes the server service time. Each of the three-tiers has a load balancing system with possibly different degrees of replication. In Figure 3.5 we consider the case where each tier has three replicas.

When a request arrives at tier T_i it triggers one or more requests at its subsequent tier T_{i+1} (e.g., a database search that triggers multiple queries at different product catalogs). This phenomenon is captured by allowing a request to make multiple visits to each of the queues during its overall execution. This is achieved by introducing a transition from each tier to its predecessor. A request, after some processing at tier T_i, either returns to T_{i-1} with a certain probability $p_{i,j}$ or proceeds to T_{i+1} with probability $(1 - p_{i,j})$. At the tier T_3 all requests return to T_2, while at the first tier T_1 a transition to the delay station denotes request completion. For a complete list of the system's parameters together with their semantics we refer the reader to the description of the load balancing system used as the running example of Section 3.4.

Throughout the remainder, without loss of generality, we fixed to 0.5 the probabilities that tier i makes multiple requests to tier $i+1$ and $Z = 0.2$, i.e., we assume that the operational profile of the user is known.

Experimental set-up As said in Section 3.4, once QoS properties of the system and its variabilities are symbolically modeled, then a set of nonlinear real equations is derived from them. This represents the symbolic solution $\mathcal{M}$ for the throughput of each station in the system. By encoding $\mathcal{M}$, augmented with QoS requirements $\mathcal{Q}$, resource constraints $\mathcal{R}$, and domain assumptions $\mathcal{D}$, in a set of first-order-logic formulas we are able to formulate an SMT problem used as the core engine for the self-adaptation process.

In all our tests, we arbitrarily set $\mathcal{Q}$ to represent the requirement that the average response time for a user is no larger than 1.0 time units, together with the condition that all working servers operate at a utilization between 0.6 and 1.0. This encodes a trade-off between competing requirements forcing the solver to avoid trivial solutions: without a condition on the utilizations, the solver could have found an assignment that oversizes the system, at the cost of having servers that are inefficiently working at very low utilization levels. Furthermore, it is worth to point out that our utilization constraint can be interpreted as a QoS requirement related to energy efficiency by relating servers utilization to energy consumption, as done in [100].

In the following we formally specify these constraints. By taking station 1 as the reference station and applying Little's law (see Section 2.1), we obtain that the response time is given by $C/\mathcal{T}_1$. Hence, $\mathcal{T}_1 \geq C$ encodes our desired response time constraint (i.e., ≤ 1.0). Recalling that $\mathcal{U}_i$ denotes the utilization at queuing station i (again, a subformula from $\mathcal{M}$), we get

$$\mathcal{Q} := (\mathcal{T}_1 \geq C) \wedge \forall i \in S.\big((\mathcal{U}_i \leq 1.0 \wedge \mathcal{U}_i \geq 0.6) \vee \mathcal{U}_i = 0.0\big),$$

where the condition $\mathcal{U}_i = 0.0$ allows the solver to explore the possibility not to activate one or more server replicas in a tier. Finally we complete the adaptation problem formulation by specifying $\mathcal{D} \wedge \mathcal{R}$ as

$$\mathcal{D} \wedge \mathcal{R} := (C \geq 1) \wedge \mathcal{R}_1 \wedge \mathcal{R}_2$$

where

$$\mathcal{R}_1 := \forall i, j \in S.\big(0.0 \leq p_{i,j} \leq 1.0\big) \wedge \forall i \in S. \left(\sum_{j \in S} p_{i,j} = 1.0\right),$$

$$\mathcal{R}_2 := \forall i \in S.\big((1.0 \leq s_i \leq 40.0) \wedge (0.06 \leq e_i \leq 10.0)\big),$$

Here $\mathcal{R}_1$ imposes that the symbolic variables $p_{i,j}$ are actual probabilities, such that the sum across all outgoing routing probabilities from a node is equal to 1. Instead, $\mathcal{R}_2$ gives further constraints on the service multiplicities and on the service times. To better describe our formulation, we report the initial specification of the case study, in Z3-like pseudo-code, in Listing 3.1.

```
(assert
        (and
                ;Constraints set
                ∀i ∈ S (≥ s_i 1)
                ∀i ∈ S (≤ s_i 40)
                ∀i ∈ S (≥ 1/μ_i 0.06)
                ∀i ∈ S (≤ 1/μ_i 10)
                ;Routing probabilities constraints
                ∀i, j ∈ S (≤ p_{i,j} 1.0)
                ∀i, j ∈ S (≥ p_{i,j} 0.0)
                ∀i ∈ S (= Σ_{j∈S} p_{i,j} 1.0)
                ;Domain assumptions
                (≥ C 1)
                ;System throughput
                p_{i,j}, 1/μ_i, s_i, k ∈ S T_k = f(p_{i,j}, s_i, 1/μ_i)
                ;QoS requirements
                (≥ T_1 C)
        )
)
```

Listing 3.1: SMT problem for the three-tier system.

This will be expanded with further constraints to represent the runtime changes, as discussed hereafter.

Numerical results We evaluate the effectiveness and efficiency of our approach by simulating adaptation steps on the QN model. Each step triggers the solution of the satisfiability problem by varying the workload C and simulating link/server faults by enforcing to 0.0 one or more of the routing probabilities $p_{1,4}$, $p_{5,8}$, and $p_{9,12}$. More specifically, we considered 300 adaptation steps where at each step we picked a linear increasing workload C in the range $[5, \ldots, 304]$, and we imposed uniformly at random to apply one of the following constraints with the idea of emulating from 0 up to 3 station or link faults: $\mathcal{A}_1$) *true*, i.e., no change to the routing probabilities; $\mathcal{A}_2$) $p_{1,4} = 0.0$, $\mathcal{A}_3$) $p_{1,4} = 0.0 \wedge p_{5,8} = 0.0$; $\mathcal{A}_4$) $p_{1,4} = 0.0 \wedge p_{5,8} = 0.0 \wedge p_{9,12} = 0.0$. We remark that at each iteration the internal status of the used SMT solver is cleared in order to ensure that it did not reuse any assignment computed at the previous step.

Figure 3.9 shows the traces of the adaptation steps considered in our experiment, where each j-th step added the constraint $(C = j) \wedge \mathcal{A}_i$ for $j \in 5, \ldots, 304$ and $i = 1, 2, 3, 4$, to the model reported in Listing 3.1.

To fully understand the efficiency of SPA, we calculated the computational time required by the preprocessing phases of *symbolic analysis* and *SMT problem formulation* (see Figure 3.1) for our three-tier system. We experienced that these two steps were executed with an average of 4.85 s, however we remark that this initial overhead is required once since the runtime adaptations apply to the same SMT problem formulation.

Figures 3.6, 3.7, and 3.8 show the numerical results of our experiments, executed on an ordinary laptop with a 2,6 GHz Intel Core i5 and 8 GB of RAM. On the x axis is reported the number of

clients C at each adaptation step. The response time for each found configuration (Figure 3.6) and the utilization levels of the system (Figure 3.7) always satisfy the requirements (indicated by the red dotted line) even if in the system some faults occur (Figure 3.9). For presentation issue, Figure 3.7 shows only the maximum and minimum utilization levels measured at each adaptation step since they are enough for assessing the fulfillment of the QoS requirements. Moreover, the required computation time for the adaptation (Figure 3.8) shows that our approach can efficiently solve this problem within few hundred milliseconds in the worst case.

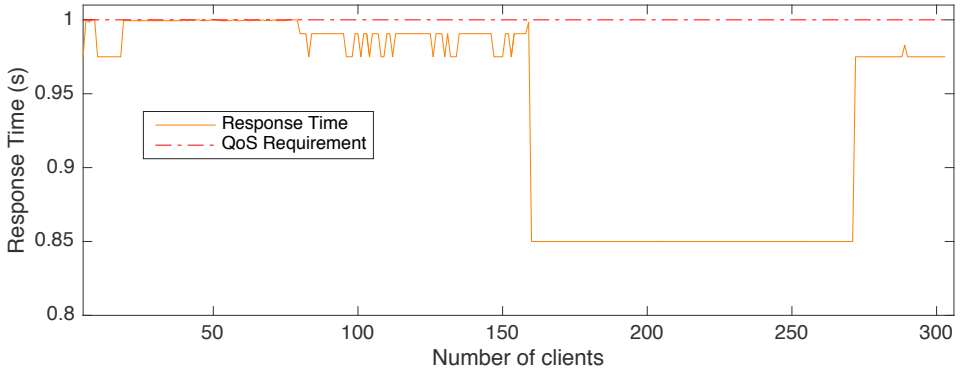


FIG. 3.6: System response time.

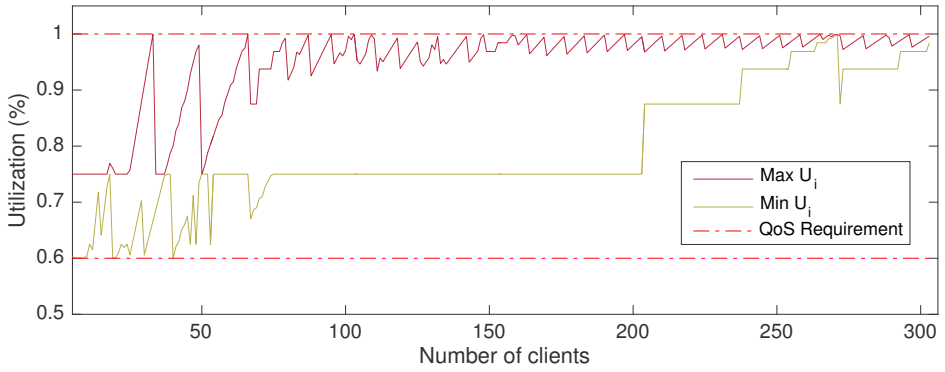


FIG. 3.7: Minimum and maximum utilizations for each runtime configuration.

To evaluate the scalability of SPA, we considered the same experiment (i.e., managing symbolically all the system parameters $s_i, e_i, p_{i,j}$ constrained by the same $\mathcal{Q} \wedge \mathcal{R}$) on larger variants of the system characterized by increasing replication levels up to six replicas per tier. Figure 3.10 reports the average computation time (in seconds) across all adaptation steps for each replication level (where the case of 3 replicas summarizes the results previously reported). The complexity of the adaptation process grows up while increasing numbers of replicas per tier (adding a replica for each tier means to add in the model a number of new parameters equal to 3 times the number of tiers), but still remains reasonable for meaningful system configurations. However, we remark that all the results presented in this section are obtained for the extreme cases in which *all* the system parameters are treated symbolically. Reducing this variability can significantly lower the complexity of the adaptation process, making our approach scalable to

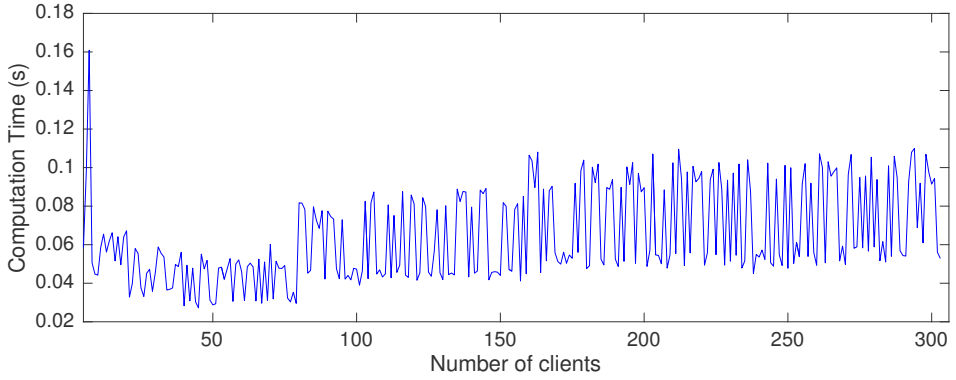


FIG. 3.8: Computation time.

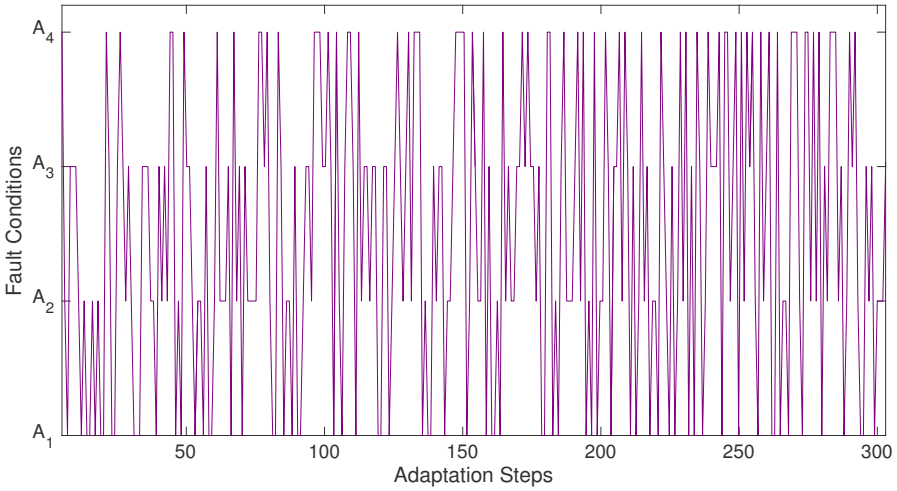


FIG. 3.9: Number of faults for each adaptation step.

larger models. Finally we evaluated when the adaptation problem is unsatisfiable since it may take longer than providing a witness. To experiment this point, in our case study we modified the original problem by requiring its satisfiability with a number of clients equal to: (i) 3 for the three replica system, and (ii) 2 for the four replica one. The SMT solver provided as output *unsat* for both cases, and it took 69.42 s and 224.26 s, respectively. This shows that the difficulty in not finding a witness is model-dependent; in practical uses it is possible to consider setting a time-out that is treated as *unsat*, at the cost of obtaining false negatives.

3.6 Discussion

In this Chapter we presented a symbolic approach for the performance adaptation of a software system, i.e., SPA, using a QN as the underlying model employed to predict QoS-related properties such as response times, throughput, and utilization. The main novelty lies in the use of a symbolic procedure as the core adaptation engine. Our methodology starts with the description of a system as well as all its possible configurations as a single "super-model" that

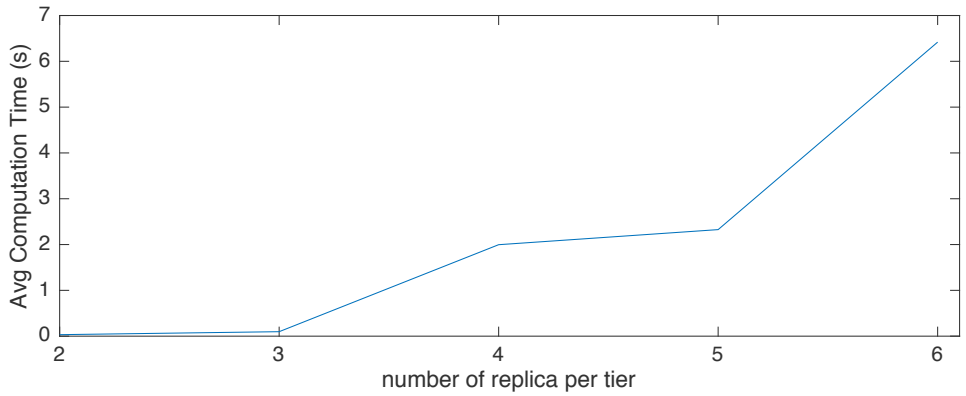


FIG. 3.10: Scalability of SPA on the case study.

encodes uncertainty and variability as symbolic variables. Building on previous results [18, 59], such a model enjoys a closed-form solution yielding an expression that gives the performance prediction of a concrete configuration by simply plugging in the numerical values related to that configuration. The key idea is to encode the model, the QoS constraints, and the necessary assumptions on the parameter spaces as first-order logical formulae over those variables, which thus represent adaptation "knobs" such as the number of servers at a queuing center or routing probabilities. In this way, one can leverage on efficient SMT solvers, in our case the well-known Z3 [96], in order to find an assignment of such variables that satisfies all the constraints. In the following the main advantages and limitations of this work are discussed.

Advantages One major strength of SPA is that a great amount of the computation effort is pushed to the design phase, where the symbolic expressions for the QN performance indices must be produced using a computer algebra system. The numerical experiments reported in this chapter showed that the time taken by the satisfiability problem allows to perform adaptation steps in a few milliseconds even in a realistic three-tier model under the extreme-case condition where the system is able to self-adapt by varying any of its parameters. This scalability is mainly due to the fact that the complexity of the QN solution depends on the network topology and not on the size of the underlying continuous-time Markov chain. In more general approaches where such approximations are not feasible, carrying out runtime adaptation using formal verification techniques becomes more difficult and strategies to improve its effectiveness and responsiveness must be adopted [101].

Another advantage comes from the fact that the adaptation is symbolically performed: the solver may return an unsatisfiability result, meaning that there exists no configuration that can satisfy the current QoS requirements. This peculiar characteristic of our SMT approach can also be leveraged at design time (although this is not the focus of this thesis). Indeed, it is not difficult to envisage a development process where the modeller uses this approach in order to find a rough sizing of the system, for instance using worst-case scenarios about the workload levels and service rates, which is then taken as the starting configuration that will be refined at

runtime. Another possibility might be to include forecast knowledge describing load intensity variations over time, similarly to [102, 103], thus to pro-actively set system parameters [104] and anticipate QoS flaws by means of such additional source of information.

Limitations The main limitations of symbolic performance adaptation are in the assumptions behind the QN model. Despite being employed in many software performance models, QNs lack the necessary expressiveness to cover useful mechanisms such as simultaneous resource possession and layered services. The former is of increasing relevance especially in virtualized environments in order to model the case where more than one application/virtual machine shares the same hardware. The latter is typical in multi-tiered applications when one tier needs processing from tiers below in order to complete a request. Incidentally, our case study from [6] features a similar scenario, but it is successfully rendered as a "flat" QN by means of a careful interpretation of the routing probabilities.

A crucial feature for the derivation of a symbolic QN solution is the assumption of a closed network. However we argue that, in practice, this does not appear particularly limiting since it is possible to approximate exogenous arrivals by choosing a (single-server) station with a large enough initial population of jobs (to which jobs that exit the network are redirected). A further challenging restriction In terms of expressiveness is due to the single-class assumption. We address this problem in Chapter 5 where a multi-class performance runtime self-adaptation method is presented.

Symbolic closed-form expressions are available for steady-state measures. Therefore, our performance adaptation can be effectively applied under the assumption that the duration of the transient dynamics is negligible with respect to the representative time scales of the system under consideration. The fluid solution of the QN (see Section 2.4), instantiated with the current parameters, may assist the modeler in deciding to which extent the assumption is acceptable in practice. Since it gives an approximation of steady state as the equilibrium point of a time-dependent trajectory, the fluid solution directly provides an estimate of the duration of the transient behavior. If, at runtime, the QoS requirements are not violated during that period of time, the modeler gains confidence that the new configuration converges to the desired steady state.

Another critical issue of the symbolic approach presented in this chapter lays in its inability to finding the *optimal* configurations with respect to some optimality measures but just a feasible one. This is due to the unavailability of optimization modulo theory techniques applicable to efficiently deal with general nonconvex nonlinear problem as the one that would be generated by our adaptation problem. This limitation and the steady state assumption are addressed in Chapter 4 where a transient and efficient optimal runtime self-adaptation technique is proposed.

Chapter 4

Efficient Performance-driven Self-adaptation

In this Chapter we present a novel technique that allows transient performance-driven self-adaptation using a system model based on fluid queuing networks. Software engineers can select the adaptation knobs of a QN (routing probabilities, service rates, and concurrency level) and we automatically derive a model predictive controller able to continuously track the desired performance requirements by properly tuning the system parameters. Previous self-adaptation methods (e.g., see Chapter 3) have two main limitations: *i)* high computational cost of the optimization, due to the nonlinearity of the models; *ii)* focus on long-run performance metrics only. As a consequence they allow adaptations with coarse time granularities neglecting the system's transient behavior. Our MPC adaptation strategy is efficient at it is based on mixed integer programming and transient since it uses a dynamical representation of a QN. An extensive evaluation of an implementation of a load balancer demonstrates the effectiveness of the adaptation and compares it with traditional methods based on probabilistic model checking. The content of this chapter is mainly taken from [2].

4.1 Overview

In software-intensive systems, runtime variability [105, 106] is a major impediment to satisfy quantitative non-functional properties such as performance and reliability [107]. However, the available QoS self-adaptation solutions [1, 108] only concern the long-run (i.e., steady-state) performance indices. Any adaptation strategy based on such indices (as also done in [6, 109–111]) ignores the system's whole time-course dynamics (see Chapter 3). Actually, reaching a steady state may not even be guaranteed based on the degree of variability, hence defying any prediction and adaptation effort.

In this chapter, we deal with self-adaptation for performance-related requirements and focus on software systems that can be described by queuing networks. Since the analysis of the QN's time-dependent evolution suffers the well-known problem of state space explosion (see Section 2.1), we consider a their approximate representation based on ordinary differential equations (see Section 2.2).

The novelty of our approach is the formulation of the performance-driven self-adaptation problem using model predictive control, a well-known technique based on on-line numerical optimization [20], which allows multiple knobs. This provides a more expressive adaptation strategy that may automatically act on system settings of different nature (routing probabilities, service rates, and concurrency levels).

The basic idea behind MPC is to perform an optimization at each time step during the system evolution. The model is initialized with the currently measured state of the system. The optimization automatically finds the values of the *control signals*, i.e., the model parameters related to the adaptation knobs that best steer the system toward a given reference trajectory (e.g., a QoS requirement such as throughput or response time) over a given time horizon. Thus MPC returns an optimal value for each control signal at each time step across that horizon. Adaptation takes place according to the *receding horizon control* paradigm: only the values for the next time step are applied, whereas all subsequent ones are discarded [112]. When MPC is started at the next iteration the newly measured state will readily feedback the effect of the adaptation into the system, and become the starting point for the optimization over the next time horizon.

Unfortunately, applying MPC is not straightforward. Its main limitation is the typically high computational cost. Indeed, state-of-the-art approaches report significant overheads even for small models and short prediction horizons [21, 112]. For QNs, the main sources of complexity are: *i*) the nonlinearity of the ODE model, as it needs to account for threshold-type service rates that depend on the state of the system; *ii*) the exponential complexity of the optimization due to the multi-dimensionality of the control signals space. The practical consequence is that the execution of a single optimization problem can be time-consuming, reducing the maximum frequency at which the controller can operate. This may significantly degrade its effectiveness if the system under control has faster dynamics, because the controller will not keep track of potentially many events across two steps.

Ideally one would like to design the controller's computational loop such that it can operate at a frequency close to the system dynamics. As a main technical result of this work, we achieve this by formally translating the original nonlinear MPC problem into a mixed integer linear programming (MILP) one. This is an equivalent representation where the model becomes linear time-invariant and the control signals become exogenous inputs, leading to a quadratic programming problem that enjoys very efficient solution techniques [113].

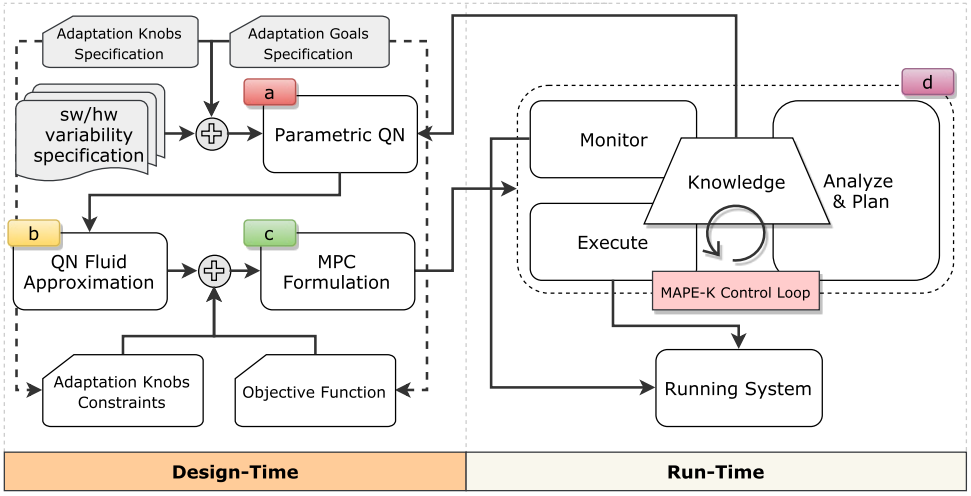


FIG. 4.1: MPC adaptation approach overview

We tested our MPC approach on an implementation of a load-balancing system. Our results show its effectiveness in adapting to events such as performance degradation of a server and sudden changes in the system workload. Our formulation does allow reaction times that are orders of magnitude faster than a naive nonlinear MPC design. Indeed, we show that such nonlinear MPC would not be able to track the dynamics of our testbed effectively. A scalability assessment of the computational requirements of our MILP approach shows that it can yield control loops as fast as 0.3 s on commodity hardware even for large prediction horizons.

This work advances the state-of-the-art in applications of MPC, which so far have been designed for ad-hoc solutions to specific case studies, always considering a single adaptation knob [114–117]. In addition, it compares very favorably against analogous runtime adaptation strategies based on probabilistic model checking. To show this, we consider the Markov chain interpretation of the QN and develop a controller based on Markov decision processes using the PRISM model checker [118], similarly to [119]. We show that the execution time of MPC controller is essentially independent from the system size, whereas the Markov decision process suffers from state explosion to the extent that it does not return an adaptation strategy fast enough to track runtime variability.

MPC Performance Adaptation

Figure 4.1 depicts the overview of the different phases composing the MPC performance adaptation approach proposed in this Chapter:

a) The starting point is a system specification that can be translated into a QN. Although the definition of such a system specification is outside the scope of this work, we argue that it could be based on existing model-driven approaches. Indeed, in the literature there are techniques that translate software designs into performance models (e.g., [11, 120–123]). As we already did in Chapter 3.4.1, we call a *parametric QN* the performance model of the systems including the

elements that can act as adaptation knobs (i.e., the parameters), and the range of values that each adaptation knob can take. Moreover, since in this Chapter our goal is to explore the AS looking for an optimal configuration satisfying QoS requirements, we include in this parametric specification the adaptation goals defined as desired set-points for certain performance metrics (i.e., the given QoS requirements for the system).

Which adaptation knobs to use mainly depends on the possibility that those architectural elements may be changed at runtime in the actual implementation. For instance, in the model of a server, choosing its concurrency levels as an adaptation knob requires the implementation to dynamically kill/spawn new threads/processes. We will study such a case in Section 4.4. The adaptation goal can be described in terms of set-points for either *queue lengths* (how many clients/jobs are waiting for service at a station) or *throughputs* (how many clients are served per unit time). We focus on these for ease of presentation, but we stress that this is general enough to encode other important performance metrics such as utilization and response time, see [40] for a discussion on this.

b) For a given parametric QN, the controller can be automatically synthesized. In particular, we obtain an ODE system using the approach described in Section 2.2.

c) The ODE system will then be discretized in time such that it can give a finite set of constraints for the MPC problem. The constraints further encode the feasible ranges of the adaptation knobs and the values for all other parameters, which are assumed to be fixed (i.e., non-controllable). The adaptation goal is translated into the objective function of the MPC problem. This can be systematically turned into the equivalent MILP problem, which can be efficiently solvable by a wide array of both commercial and open-source solvers.

d) The MPC philosophy leads to a natural implementation as a MAPE-K (see Figure 2.4) loop controller. In the *Monitor* phase the current system state is measured and used as the input parameter for updating the system model embedded in the MILP formulation. Similarly to Chapter 3, the solution of the optimization problem implements the *Analysis* and *Planning* phase at once but here with the aim to find the optimal configuration satisfying requirements and not just a feasible one. Finally, in the *Execution* phase the computed control signals are applied to the running system. The *Knowledge* block is thus represented by the model constraints embedded in the optimization problem. Even in this work we assume that all of the system parameters could be directly measurable in the monitoring phase (i.e., no estimation phase is needed). We address the problem of parameter estimation in Chapter 6 whose foundation is the MPC formulation given in this Chapter.

The remainder of this Chapter is organized as follows. Section 4.2 presents the relevant related works. Section 4.3 sets up the MPC problem which complete mathematical formulation is reported in Section 4.6. Section 4.4 numerically demonstrates the effectiveness and the scalability of our approach on a real system exposed to two different adaptation scenarios (i.e., hardware

degradation and workload fluctuation). Finally Section 4.5 concludes the chapter and outlines future work.

4.2 Related Work

Our approach focuses on two main critical points of self-adaptation, i.e., expressiveness and effectiveness, as discussed hereafter.

Adaptation expressiveness. Some approaches aim to control and adapt the admission of incoming requests. In [115] a load control mechanism for a web-server system with control-theoretic methods is designed. The system is modeled as a $G/G/1$ queue and the server is controlled by an admission system that collects the steady-state server utilization and adapts the probability of incoming requests. In [109] a QN model predictor is used to strengthen the adaptive feedback. Similarly to [115], the controller checks steady-state performance results vs requirements and adjusts the admitting probability to meet such requirements. Predictions of the QN model are demonstrated to provide a better accuracy for the adaptation. Our work differs from [109, 115] since it considers transient dynamics and is not restricting to admission control.

In [117, 124] a MPC algorithm is developed to forecast incoming workload and derive adaptation strategies on cloud resources. Differently from our approach, the optimization is limited to the allocation of resources, and performance results are computed with the MVA [32] algorithm, without considering transient dynamics. In [125] a control-theoretic solution to the dynamic capacity provisioning for cloud is presented. Specifically, the number of active servers is adapted with MPC to reduce the total operational cost in terms of energy consumption. Our approach is different since our adaptation is not limited to switching on and off servers.

Adaptation effectiveness. In [116] an auto-scaling methodology for cloud resources is presented for web application providers. It predicts the workload of web requests and returns the optimal number of virtual machines by utilizing queueing theory. However, the optimization results to be a complex nonlinear function that is hard to simplify by mathematical methods. On the contrary, our approach enables the formulation of the QoS adaptation problem as an efficient MILP one.

Further approaches that make use of control theory formal guarantees are [126, 127] where a closed-loop strategy is proposed to enable an adaptive software system's dynamic behavior. In [126] the designed controller continuously determines the value of a single control variable (e.g., the processor's clock speed) which represents a setting for the corresponding tunable parameter. In [127] QN performance models are investigated, and each queueing center is equipped with a controller that considers predefined values of service rates and provides as output the rate of that specific QN center while considering external disturbances. Differently from [126, 127], our approach simultaneously considers multiple adaptive parameters of different nature, and the

optimal control provides as output values to scale service rates, number of servers, and routing probabilities at the same time.

Recently, Baresi et al. presented a technique to auto-scale the CPU cores of containerized applications by means of a planner which consists of a discrete-time feedback controller [128]. Instead, our approach considers the system transient dynamics relying on a model predictive technique with receding control horizon to forecast the system evolution.

4.3 Efficient Performance-driven Self-adaptation

In this section we precisely define all the steps informally overviewed in Section 4.1.

4.3.1 Preliminaries

Parametric QN

The single-class *parametric QN* has been already introduced in Section 3.4.1. In addition here, in order to formally specify the adaptation space, we denote by $V \subseteq S$ the subset of adaptation knobs, consisting of the user-defined parameters that can be controlled at runtime. For each adaptation knob $v \in V$ we indicate by $\underline{v}_i$ and $\bar{v}_i$ the minimum and maximum values allowed for that parameter, respectively. For each parameter that is fixed, i.e., $p \in S - V$, $\hat{p}$ is its given value. Finally we denote by $x_i(0), i \in S$ the initial condition, i.e., the initial number of jobs assigned to station i.

We consider a set of R QoS requirements. For the r-th requirement, we let kind_r be either Th_i or Q_i, to denote the throughput or the queue length at some station i, respectively. Analogously, the corresponding setpoint is denoted by value_r.

Similarly to [1, 18, 59] we focus on a model supporting the specification of a single class of jobs. We comment on the limitations of this assumption and on possible ways to mitigate it in Section 4.5. Finally, as we already said in Section 2.1, the service rates μ_i are assumed to be exponentially distributed but using [18] our framework can be easily extended to the nonexponential case.

ODE Model

The ODE model is automatically derived from the parametric QN and it gives estimate of the average queue lengths $x_i(t)$ as a function of the QN parameters (i.e., service rate, routing probabilities, number of servers). We refer the reader to Section 2.2 for a detailed discussion about the queuing network fluid approximation.

In order to assess its accuracy, we validate the used ODE model by comparing prediction results against real measurements. See Section 4.4 for further details.

4.3.2 Nonlinear MPC Formulation

Discrete-time model

In order to employ MPC, we rely on a time discretization of the fluid model with a finite step size Δt. MPC finds the optimal values of the adaptation knobs over a time horizon of H steps. Simple algebraic manipulations yield a formulation that reads, in matrix notation:

$$x(k+1) = A(x(k)) + x(k) \tag{4.1}$$

where $x(k) = (x_i(k))_{i \in S}$ are the queue lengths of the QN at step k (i.e., at time $k\Delta t$) and matrix $A(x(k))$ has components $a_{i,j}(x(k))$, with $i, j \in S$, given by:

$$a_{i,j}(x(k)) = \begin{cases} -\mu_i(k)\Delta t \min\{x_i(k), s_i(k)\} & , i = j \\ p_{j,i}(k)\mu_j(k)\Delta t \min\{x_j(k), s_j(k)\} & , i \neq j. \end{cases}$$

Objective function

We now define the objective function of the MPC optimization problem, starting from the QoS requirements in the parametric QN. Specifically, for each time step $k = 0, 1, \ldots, H-1$, we consider the vector of performance metrics $m(k) = \big(m_1(k), \ldots, m_R(k)\big)$ where each $m_i(k)$, $1 \leq i \leq R$ can be either of the following:

$$m_i(k) = \begin{cases} \mu_j(k) \min\{x_j(k), s_j(k)\} & \text{if } \mathsf{kind}(R_i) = \mathsf{Th}_j \\ x_j(k) & \text{if } \mathsf{kind}(R_i) = \mathsf{Q}_j \end{cases}$$

Similarly, the required set points are collected in the vectors $r(k) = (r_1(k), \ldots, r_R(k))$, with $r_i(k) = \mathsf{value}(R_i)$.

Our goal is to minimize the error between the performance indices and their reference values, i.e., $e(k) = m(k) - r(k)$, across all time steps in the horizon $k = 0, 1, \ldots, H-1$.

Thus, overall the objective function is defined as follows:

$$\text{minimize} \sum_{k=0}^{H-1} e(k)^T e(k) \tag{4.2}$$

where $(\cdot)^T$ indicates matrix transposition.

The discretization in (4.1) allows us to embed the dynamics of the model as a discrete set of

constraints in the optimization problem by unfolding (4.1) over H time steps:

$$\begin{aligned} x(1) &= A(x(0)) + x(0) \\ x(2) &= A(x(1)) + x(1) \\ &\dots \\ x(H) &= A(x(H-1)) + x(H-1) \end{aligned} \tag{4.3}$$

To these, we add:

$$s_i(k) \in \{\underline{s}_i, \underline{s}_i + 1, \dots, \overline{s}_i\}, \qquad s_i(k) \in V \tag{4.4}$$

$$\underline{\mu}_i \leq \mu_i(k) \leq \overline{\mu}_i, \qquad \mu_i(k) \in V \tag{4.5}$$

$$0 \leq p_{i,j}(k) \leq 1, \ \sum_{j \in S} p_{i,j}(k) = 1, \qquad p_{i,j}(k) \in V \tag{4.6}$$

$$s_i(k) = \hat{s}_i, \qquad s_i(k) \in S - V \tag{4.7}$$

$$\mu_i(k) = \hat{\mu}_i, \qquad \mu_i(k) \in S - V \tag{4.8}$$

$$p_{i,j}(k) = \hat{p}_{i,j}, \qquad p_{i,j}(k) \in S - V \tag{4.9}$$

$$\text{for } k = 0, \dots, H-1$$

where (4.4)–(4.6) define the ranges for the adaptation knobs, and (4.7)–(4.9) set the values for the fixed parameters. We remark that the specification of (4.4) includes integer variables because they represent server multiplicities which may vary discretely.

MPC is the problem of minimizing (4.2) subject to the constraints (4.3)–(4.9). In addition to all QN parameters, the decision variables of this problem are also the vectors $x(1)$, $x(2)$, …, $x(H)$; these represent the predictions on future states of the system once the optimal values of the adaptation knobs are applied. Instead, $x(0)$ is the vector of the current queue lengths, which are measured and plugged into the problem at runtime. The efficiency issues of this MPC setup (as will be demonstrated numerically in Section 4.4) are due to the nonlinearities arising from the bilinear terms, i.e., $p_{i,j}(k)\mu_i(k)$ and the threshold-type service dynamics arising from expressions such as $\min\{x_i(k), s_i(k)\}$.

In the next section we will use this nonlinear problem as the basis for developing an equivalent MILP formulation.

4.3.3 MILP Formulation

The MILP formulation relies on a linear time-varying system with auxiliary, "virtual" adaptation knobs which will be then related to the original ones. We define the new ODE system:

$$\frac{dx_i(t)}{dt} = \gamma_i(t) + \sum_{j \in S}(-\gamma_j(t) + \zeta_{j,i}(t)), \quad i \in S, \tag{4.10}$$

where $\gamma_i(t)$ represents the virtual service rate for station i and $\zeta_{i,j}(t)$ is a virtual routing probability. Setting $\gamma(k) = (\gamma_i(k))_{i \in S}$ and $\zeta(k) = (\zeta_{i,j}(k))_{i,j \in S}$, (4.10) can be written

$$x(k+1) = x(k) + \gamma(k) + \mathbf{1}(-\gamma(k)) + \mathbf{1}\zeta(k) \tag{4.11}$$

where $\mathbf{1}$ denotes a matrix of all ones of appropriate size.

In Section 4.6 we demonstrate an exact correspondence between the original nonlinear MPC problem (4.2) and the MILP formulation over the variables appearing in (4.11). This leads us to state the following theorem:

Theorem 2. Denoting by $\mathcal{S} = \{\mu_i^*(k), p_{i,j}^*(k),\ s_i^*(k)\}$ an optimal solution to (4.2), there exists an MPC problem based on an MILP formulation with dynamics (4.11) such that its optimal solution $\mathcal{S}' = \{\gamma_i'(k),\ x_i'(k), \zeta_{i,j}'(k), s_i'(k)\}$ satisfies:

$$\mu_i^*(k) = \begin{cases} -\frac{\gamma_i'(k)}{x_i'(k)\Delta t} & \text{if } x_i'(k) \leq s_i'(k) \\ -\frac{\gamma_i'(k)}{s_i'(k)\Delta t} & \text{if } x_i'(k) > s_i'(k) \end{cases}$$
$$p_{i,j}^*(k) = \frac{\gamma_i'(k) - \zeta_{i,j}'(k)}{\gamma_i'(k)}, \quad s_i^*(k) = s_i'(k).$$

for all $k = 0, \ldots, H-1$.

This is of key relevance in our formulation since all nonlinearities have been removed and solutions are efficiently derived.

4.3.4 Adaptation Logic

The MILP problem can be used for performance self-adaptation. The algorithm is re-iterated at each time step:

a) Feed the MILP program with the state of the QN by assigning:

   - the current queue length at station i to variable $x_i(0)$;
   - the fixed service rates of station i to variables $\hat{\mu}_i$;
   - the fixed routing probability between station i and station j to variables $\hat{p}_{i,j}$;
   - the fixed number of servers of station i to the variable $\hat{s}_i$;

b) Solve the MILP problem, obtaining the optimal sequence $\gamma_i'(k)$, $x_i'(k)$, $\zeta_{i,j}'(k)$, $s_i'(k)$, for $k = 0, \ldots, H-1$.

c) With Theorem 2, obtain the values for the concrete adaptation knobs, $\mu_i^*(k)$, $s_i^*(k)$, and $p_{i,j}^*(k)$ for $k = 0$ only, discarding the others.

d) Apply $\mu_i^*(0), s_i^*(0)$, and $p_{i,j}^*(0)$ to the system.

By discarding the future predictions and running the optimization at every time step instead (i.e., the receding horizon control), the system can react to changes as early as possible. We postpone to Section 4.5 the discussion on how to track the state of the system and parameterize the QN model.

MPC parameter setting. In our approach, the modeler is required to choose two parameters: the ODE discretization step Δt and the optimization horizon H. Here we discuss how these parameters may be chosen in practice.

The discretization step Δt affects the quality of the numerical solution of the ODE model for the QN network—the smaller Δt the more accurate the solution. On the other hand, H represents the optimization horizon in terms of the number of such Δt steps. So, for a given time interval of interest, smaller values Δt require larger values of H. This has an effect of the computational cost of the optimization because constraints and decision variables grow with H, see (4.3)–(4.9).

Since the optimization is performed at runtime, the time to complete an entire adaptation loop (consisting of measuring the current state of the system, running MPC, and applying the control) should be comparable to the dynamics of the system under control. Indeed, an adaptation loop that is slow relatively to the system will not be able to keep track of the possibly many events occurring during the adaptation itself.

Taking into account these constraints, we propose the following strategy for choosing H and Δt:

i) Choose a duration A of the adaptation loop comparable to the system dynamics. This can be done by examining the service rates of the QN model, which provide an estimation of the time required to process the different activities in the system.

ii) Fix a value of Δt that ensures an accurate enough ODE solution. This can be done by solving the ODE with standard numerical solvers, imposing a desired level of accuracy. Their output will provide the sequence of discrete time steps to maintain that accuracy.

iii) Choose a value of H such that the adaptation loop can execute within A time units. This can be done off-line by running tests on the hardware on which the controller will be deployed.

With this procedure, it may be possible that H becomes small enough that it does not cover the time of the adaptation loop D, i.e., $H \cdot \Delta t \leq A$. This invalidates the controller. To tackle this issue, steps i)–iii) may be repeated by, for instance, considering a faster hardware for the controller (to decrease A) or accepting a larger tolerance by increasing Δt.

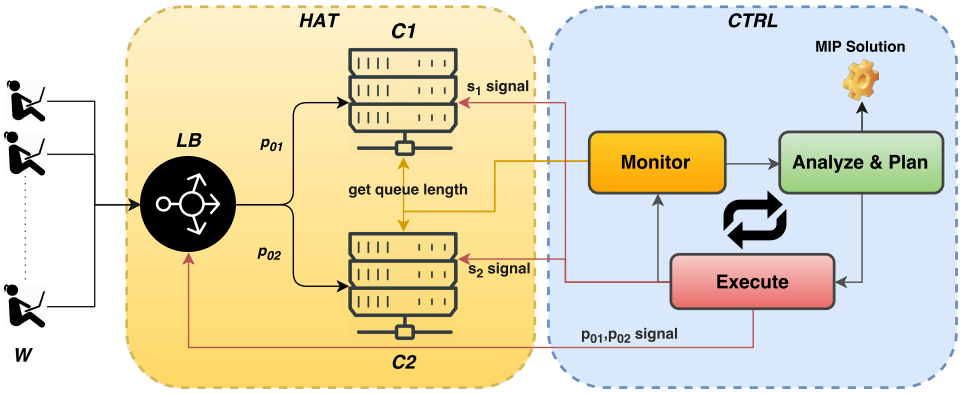


FIG. 4.2: HAT architecture

4.4 Numerical Evaluation

We evaluate the effectiveness and the scalability of the proposed MPC approach on a real system. The replication package containing the experimental infrastructure is publicly available at https://goo.gl/rpWCHv.

4.4.1 System Description and Implementation

For conducting our study we relied on an in-house developed web application, called HAT (heavy task processing application), meant to serve user requests characterized by a CPU-intensive load. HAT was deployed as a NodeJs[1] based load-balancing system running on the Ubuntu server 14.04 Linux distribution.

Figure 4.2 depicts the architecture: ***C1*** and ***C2*** represent the processing nodes, ***LB*** identifies the dispatcher, while ***CTRL*** is the controller component. Component ***W*** represents the workload generator, issuing requests to ***LB***. In particular, following the load balancing paradigm (see Section 2.1), user requests are randomly distributed across the processing components according to the routing probabilities p_{01}, p_{02}. Requests are processed on the server using PHP and reported to the user as an HTML page.

We implemented ***W*** as a multi-threaded Python based workload generator. Each thread runs an independent concurrent user that iteratively accesses the system, interposing an exponentially distributed delay (i.e., the *think time*) between successive requests. Nodes ***LB***, ***C1***, and ***C2*** are NodeJs servers equipped with runtime reconfiguration capabilities. In particular, relying on the HTTP-PROXY and Cluster libraries we enabled the reconfigurations of the routing probabilities used by ***LB***, and the number of active processes and CPU cores assigned to ***C1*** and ***C2***. These reconfigurations are triggered at runtime by the ***CTRL*** component, in a MAPE-K [129] loop, through operating system signals (see Figure 4.2). ***CTRL*** ran the MILP optimizations using the well-known CPLEX tool. To this end, we defined a model to text transformation employed

[1] https://nodejs.org/

TABLE 4.1: System parameters for the validation experiments

Parameter	Value	Description
μ_0	0.1	User's think rate
s_0	∞	Users think activities are independent
s_1	1	Number of cores for node ***C1***
s_2	1	Number of cores for node ***C2***
p_{01}, p_{02}	0.5	Balanced routing
p_{10}, p_{20}	1	User issues a new request after service

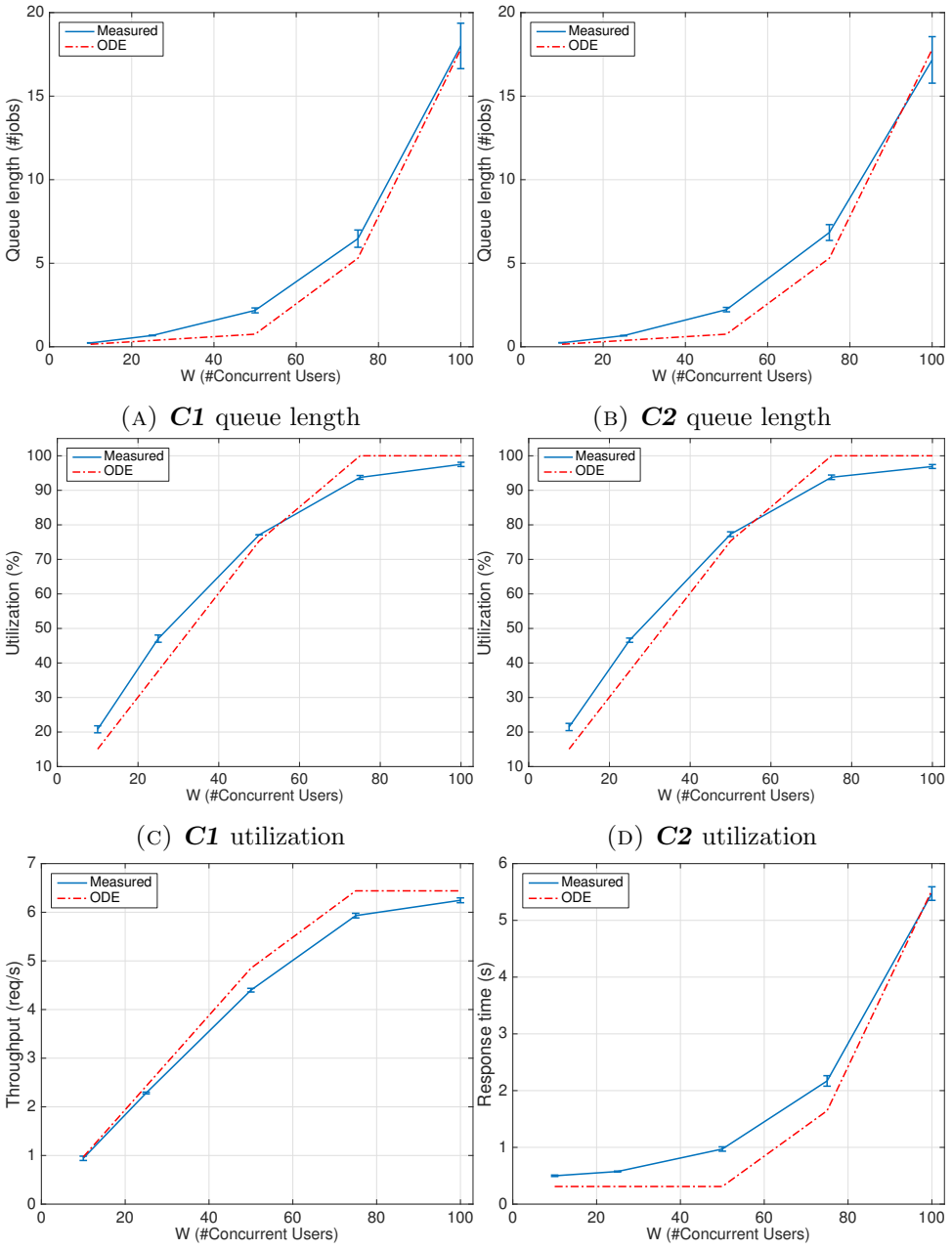
for translating the system of ODEs in a Julia [130] specification that, combined with the JuMp library, provides a common programming interface for different optimization solvers. Finally, we monitored the queue lengths of ***C1*** and ***C2*** using the *netstat* utility. To facilitate the replicability, we deployed all nodes of HAT on a single Linux machine with 24 GB of RAM, 400 GB SSD and 12 Xeon cores at 2.0 GHz. Each HAT node was assigned dedicated CPU cores. We remark that the implementation choices do not reduce the generality of the conducted evaluation since similar reconfigurations can be implemented using other existent technologies, as further discussed in Section 4.5.

4.4.2 Model Parametrization and Validation

The model for this system is the QN in Figure 2.1a. Station M_0 represents the workload generator. The service rate accounts for both the think time of the users and the time spent by the controller to dispatch the requests. Indeed, since the latter was negligible, ***CTRL*** is not explicitly represented in the QN. Instead, stations M_1 and M_2 model the processing nodes of HAT. The number of servers s_1 and s_2 represents the number of CPU cores assigned to the processing nodes.

With our hardware configuration we measured the average time to serve a request as 0.3125 s. Hence we fixed rate parameters $\mu_1 = \mu_2 = 1/0.3125 = 3.20$ (i.e., the approach presented in Chapter 6 can be employed for measuring these quantities at runtime relying on queue length transient observations only). To validate the model, we executed the system with a *balanced* configuration consisting of equal routing probabilities and one core for both ***C1*** and ***C2***, while increasing the population of circulating users, denoted by W, with $W \in \{10, 25, 50, 75, 100\}$. Each user has an average think time of 10 s (drawn from an exponential distribution). The corresponding parameters of the QN model are summarized in Table 4.1.

For each value of W we measured the following average indices in steady state: throughput, response time, and utilization of the processing nodes. We verified that a session duration of 10 minutes was sufficient to attain the steady state in all cases. Figure 4.3 depicts the validation results by comparing the steady-state prediction values of the QN model (dashed



(A) ***C1*** queue length

(B) ***C2*** queue length

(C) ***C1*** utilization

(D) ***C2*** utilization

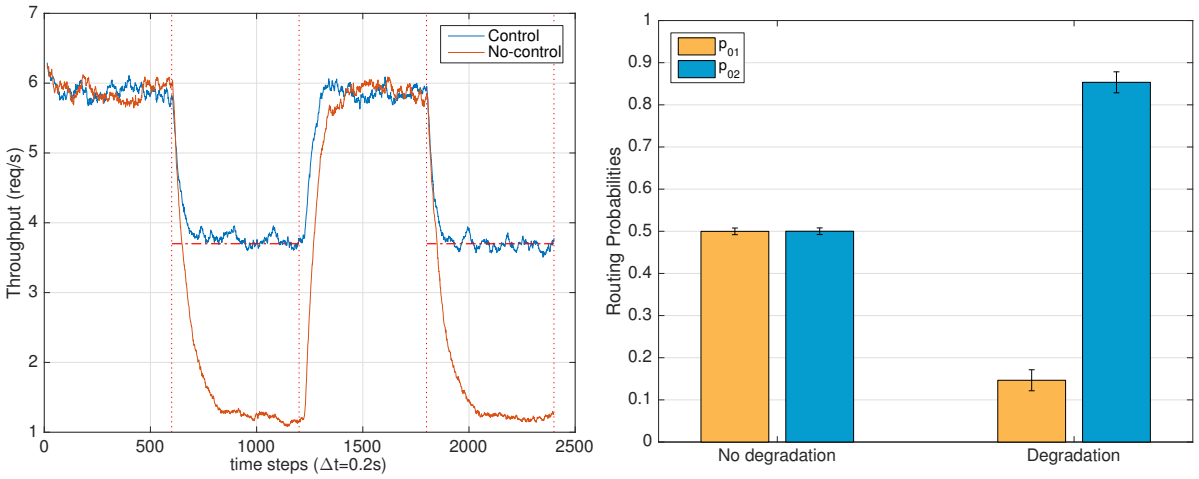
(E) System throughput

(F) System response time

FIG. 4.3: QN model validation

lines) with the measured ones of the real system averaged on 30 repetitions. The error bars on the measurements curves show the 95% confidence intervals.

The results demonstrate that the model can predict the trends of the real system satisfactorily. We consider the errors acceptable, especially since a fairly simple three-equation deterministic model abstracts away from many low-level interactions (e.g., virtual machine/host operating system, message passing, networking stack) which introduce sources of disturbance, in addition to the stochasticity of the workload.



(A) System throughput

(B) Optimal control signals

FIG. 4.4: Hardware degradation experiment

4.4.3 Adaptation Evaluation

We evaluated the effectiveness of our approach by studying two non-trivial adaptation scenarios:

S1) *Hardware Degradation.* Starting from a balanced set-up where the processing nodes ***C1*** and ***C2*** are identical, we inject degradation events where the service rate of ***C1*** is drastically reduced. The objective of the adaptation is to properly redirect users to ***C2*** with higher probability.

S2) *Workload Fluctuation.* We inject an increasing number of users into the system. The objective of the adaptation is to keep the queue lengths at ***C1*** and ***C2*** balanced and around a target value, by acting on both the routing probabilities and the number of cores at the processing nodes.

Hardware Degradation: We synthesized hardware degradation events by allowing node ***C1*** to perform six times the amount of original processing of a request whenever instructed to do so by our experimental infrastructure (by means of a signal triggered by the workload generator; in Chapter 6 an approach for automatically tracking these runtime changes is given). Thus, upon degradation the service rate of ***C1*** was assumed to be $\mu_1 = 3.20/6 \approx 0.53$. We ran the system for 8-minute long sessions under a workload of $W = 70$ concurrent users, where we alternated periods of normal operation (i.e., $\mu_1 = 3.20$) and degradation every 2 minutes. During normal operation we considered settings as in Table 4.1 (the number of server cores was fixed). The control objective has been set to find the value for the routing probabilities p_{01}, p_{02} that maximize the system throughput. Following the strategy described in Section 4.3.4, we fixed an ODE sampling interval $\Delta t = 0.1\,\text{s}$ and an activation loop $A = 0.5\,\text{s}$ after a profiling and simulation phase. This allowed a control horizon $H = 10$ which ensured a look-ahead of

two adaptation loops. In that phase we numerically verified that the chosen parameters did not reduce the accuracy of the ODEs steady state solution.

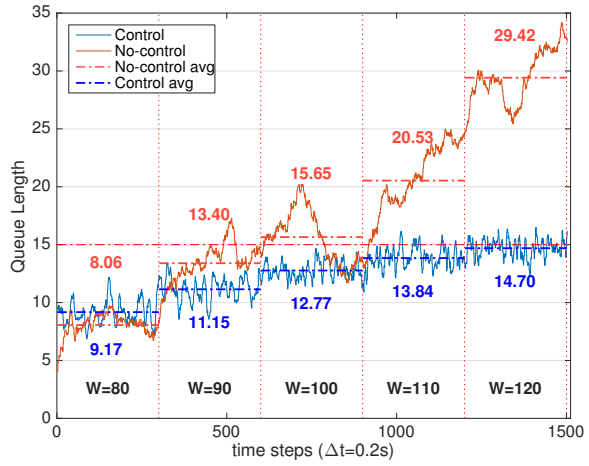
We ran the system both with and without the controller, for comparison. Figure 4.4a reports the average throughput dynamics in both cases averaged over 30 repetitions (the x-axis reports time steps with a granularity of 0.2 s). The occurrence of hardware degradation events is indicated by the dotted vertical lines. The straight horizontal lines indicate the maximum theoretical system throughputs under the degradation condition, simply estimated as the sum of the processing rates of nodes ***C1*** and ***C2*** (equal to 3.73).

During normal operations, the controlled system and the uncontrolled achieve the same throughput. This is because the balanced deployment ($p_{01} = p_{02} = 0.5$) represents the optimal solution. When degradation kicks in, the effect of the control is evident: the throughput of the uncontrolled system decreases to as little as 1.2 requests per second, while the controlled system attains values around the theoretical maximum.

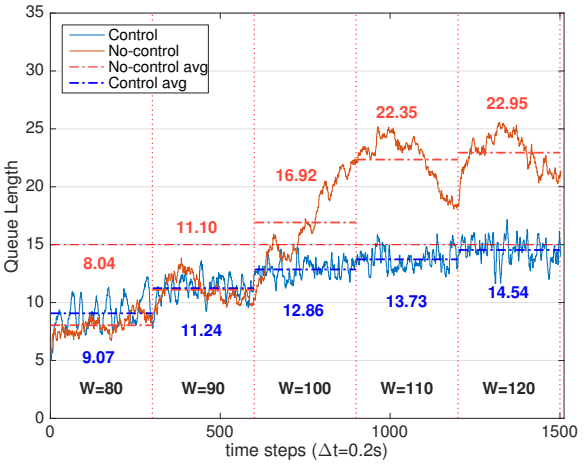
Maintaining equal probabilities when ***C1*** is degraded causes the majority of user requests to be waiting at ***C1***. Instead, the controller tends to increase p_{02} during degradation. These considerations are supported by Figure 4.4b, showing the statistics (averages and standard deviations) of the optimal values of the control signals. During degradation, the controller sends 84% of requests to ***C2*** on average. Instead without degradation, the MILP returns the balanced configuration as the optimal one.

Workload Fluctuation: We designed this experiment by starting with a workload of $W = 80$ users and introducing 10 new users into the system every minute, until reaching $W = 120$. Here the control objective is to maintain the queue lengths of ***C1*** and ***C2*** at the fixed threshold, i.e., $Q_l = 15$. This represents a critical condition for the balanced single-server deployment, from Figures 4.3a and 4.3b it is possible to observe that the system could not maintain the requirement on its own for larger workloads than 90 concurrent users. Here, the decision variables are the routing probabilities of the load balancer and s_1 and s_2, i.e., the number of cores for ***C1*** and ***C2***, respectively, with $s_1, s_2 \in \{1, 2\}$. For the uncontrolled case used for comparison we chose $s_1 = s_2 = 1$ and $p_{01} = p_{02} = 0.5$. We also chose $H = 10, \Delta t = 0.1, A = 0.5$ following similar reasoning made for the previous scenario.

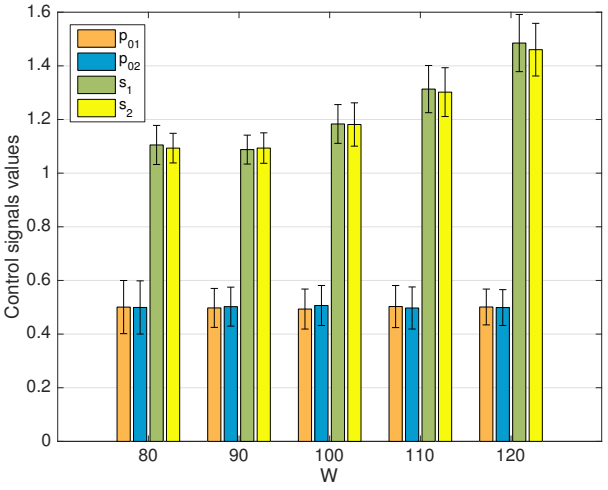
Figures 4.5a and 4.5b report the queue lengths dynamics (averaged over 30 repetitions) for ***C1*** and ***C2***, respectively. The dotted vertical lines denote the points when the workload changes. The straight horizontal lines indicate the average queue length computed at each workload level. At low workload levels ($W = 80$ and $W = 90$) the behavior of both cases is similar since the balanced single-server version is still able to fulfil the requirement $Q_l = 15$, as expected from Figures 4.3a and 4.3b. However, for larger workloads the queue lengths of the uncontrolled system start to increase. Larger values of W lead to less balanced queues in the uncontrolled case. This is not in contradiction with the intuitive argument that they should be equal on



(A) ***C1*** queue length



(B) ***C2*** queue length



(C) Statistics of optimal control signals

FIG. 4.5: Workload fluctuation experiment

average, because this holds true for a large enough number of replicas. Instead, the 30 replicas analyzed do show a strong stochastic noise. This is robustly controlled by our MPC approach which considerably reduces the queue length difference between ***C1*** and ***C2*** at all workload levels. More importantly it keeps the queue lengths always under the prescribed threshold on average.

Figure 4.5c reports the statistics (averages and standard deviations) of the optimal values of the control signals during each level of workload intensity. The average number of servers used increases with W. To reach the target for $W = 120$ the optimal strategy used about 1.5 servers on average. The statistics on the routing probabilities confirm that the average optimal strategy is to direct users to either server equally likely.

TABLE 4.2: Optimization runtimes (TO: timeout after 120 s)

	Hardware degradation		*Workload fluctuation*	
H	MINLP(s)	MILP(s)	MINLP(s)	MILP(s)
5	1.1735	0.0070	3.2633	0.0103
8	2.6391	0.0093	38.8508	0.0229
12	16.9091	0.0117	TO	0.0530
15	TO	0.0150	TO	0.0950
20	TO	0.0215	TO	0.2260

4.4.4 Scalability Evaluation

In this section we provide the numerical evidence that the main technical result of the paper, namely the equivalent MILP formulation and correspondence through Theorem 2, appears to be essential for the applicability of MPC to the QN models previously examined. We argue this by reporting considerably larger runtimes for the solution of the original nonlinear program, which preclude its use in our experiments. To this end, we used the well-known mixed integer nonlinear programming (MINLP) solver Couenne [131, 132] for dealing with the nonlinear instances. We remark that since our problem formulation is convex, its solutions are globally optimal; thus for a fair comparison we compared against a nonlinear solver that searches a global solution as well.

We parameterized the optimization problems for both the adaptation scenarios of Section 4.4.3 as a function of the control horizon H, since it represents the largest source of complexity for both the linear and the nonlinear formulations. Each entry of Table 4.2 gives the optimization runtimes (expressed in seconds), averaged over 100 adaptation steps. In the experiments we set a timeout of 120 s.

Three main observations can be made based on these results. *i)* The solution of our formulation is obtained orders of magnitude faster than the globally optimal solution of the equivalent nonlinear problem for the most challenging examples; this promotes our approach as an effective one when fast adaptation times are required, being able to optimize over 300 variables in few tenths of a second. *ii)* As expected, thanks to the linearity of the MILP problem, the required solution time does not explode when the control horizon H is increased. *iii)* The workload fluctuation scenario is harder than the hardware degradation one for equal values of H. This is due to the larger number of integer variables involved, required to represent the multiplicities of CPU cores at ***C1*** and ***C2***. We stress how, in that case, even for smaller values of H the nonlinear solution becomes practically inapplicable while our formulation is still feasible for $H = 20$. For instance, the MINLP controller would not be able to track the workload fluctuation experiment since its average solution time with $H \geq 10$ is larger than the time between changes of the workload (i.e., 60 s).

TABLE 4.3: MDP comparison (TO: timeout after 120 s)

	MILP	*Markov Decision Processes*		
W	Runtime(s)	Runtime(s)	# States	# Transitions
80	0.0037	71	3 018 789	334 732 743
90	0.0036	87	3 805 074	421 958 628
100	0.0040	TO	4 682 259	519 272 613
110	0.0038	TO	5 650 344	626 674 698
120	0.0041	TO	6 709 329	744 164 883

4.4.5 Comparison with Markov Decision Processes

In this section we compare our MILP formulation against an analogous one using probabilistic model checking, when both are employed for solving the same adaptation step of the *Workload fluctuation* scenario. We used the Markov chain interpretation of a QN (which is approximated by our ODEs), and developed a Markov decision process (MDP) controller in the PRISM model checker [118], similarly to what presented in [119]. In order to allow the adaptation of routing probabilities in the discrete state space, we discretized the interval $[0, 1]$ in 100 steps, a granularity used by real-world load balancers [133, 134]. Table 4.3 compares the runtimes for the first adaptation step as a function of the workload level W (varying from 80 to 120), and with a control horizon fixed to $H = 15$.

We observe that MILP is orders of magnitude faster than MDP, which appears to be inapplicable for that adaptation scenario since its solution time is larger than the time between two workload variations (i.e., the control signals are computed when the workload level is already changed). Furthermore, due to the state space explosion the MDP runtime depends on the workload size, while the MILP formulation is almost unaffected. We remark, however, that MDP is a general approach to self-adaptation, whereas our MILP formulation is specific to QNs (for any topology).

4.5 Discussion

In this chapter we have presented an efficient self-adaptive approach to continuously meet performance requirements using model predictive control. Selected adaptation knobs of a QN (such as routing probabilities, service rates, and concurrency levels) are automatically and continuously configured. In the following the main threat to validity and future research directions are discussed.

The building of a QN model brings about two main concerns: the definition of the structure/-topology and its parameterization. These are, in general, specific to the system under study. However we remark that there are already robust solutions for particular classes of systems.

For well-known software architectures such as multi-tiered and load-balancing systems there is substantial literature with validated models (see Section 4.2). Moreover, as discussed in Section 3.4, many approaches have been proposed to derive QN models from higher-level software designs (e.g., [11, 120, 121]).

In the experiments of this chapter, we considered a straightforward parameterization of the parameters that were not adaptation knobs (e.g., the service rates) performing it off-line. For the hardware degradation scenario, we assumed to detect the decrease in the service rate when it happened. These choices were made for simplicity, the parameterization being outside the scope of this work. However, we remark that our MPC formulation naturally allows for a more careful on-line parameterization: the current estimates may be encoded in the optimization problem as new constraints. We explore this research direction in Chapter 6 where an efficient moving horizon estimator for service demands of queuing networks is proposed.

As already discussed in Section 4.4.4, our simple load balancing system would not be controllable using a naive nonlinear MPC approach. It is easy to see that the number of variables in the MPC problem grows quadratically with the system size. A thorough study of how this impacts on the practical usability of our approach will be part of our future work as will further experimentation in other hardware/software settings, such as different servers (e.g., Apache with graceful restart) or deployment platforms (e.g., Microsoft Azure and Amazon EC2).

Furthermore, the linearization technique proposed in this chapter can be straightforwardly applied to other more expressive models such as stochastic Petri nets (which feature minimum-like expressions for the firing rates due to the amount of tokens at each incoming place, e.g., [135]). Instead, in future work we aim to develop similar ideas for more expressive models such as fluid layered queuing networks [40].

4.6 An Exact MPC MILP Formulation

In this section we describe how to get our linear representation from the original one of (4.1), which contains nonlinear terms such as $p_{i,j}(k)\mu_i(k)\Delta t \min\{x_i(k), s_i(k)\}$. To do so, we initially consider the case where $x_i(k) \leq s_i(k)$; then we remove the nonlinearity by using appropriate constraints to account for the fact that the actual instantaneous throughput cannot exceed $\mu_i(k)s_i(k)$.

Under the assumption that $x_i(k) \leq s_i(k)$ the consistency between (4.1) and (4.11) is ensured by the relations:

$$\begin{aligned} \gamma_i(k) &= -\mu_i(k)x_i(k)\Delta t \\ -\gamma_i(k) + \zeta_{i,j}(k) &= p_{i,j}(k)\mu_i(k)x_i(k)\Delta t \end{aligned} \tag{4.12}$$

Solving (4.12) as a function of the variables $\mu_i(k)$ and $p_{j,i}(k)$ gives:

$$\mu_i(k) = -\frac{\gamma_i(k)}{x_i(k)\Delta t}, \quad p_{i,j}(k) = \frac{\gamma_i(k) - \zeta_{i,j}(k)}{\gamma_i(k)} \tag{4.13}$$

Substituting these expressions in (4.5) and (4.6) we get the following constraints for the new variables:

$$\gamma_i(k) \geq -x_i(k)\Delta t\overline{\mu}_i \tag{4.14}$$

$$\gamma_i(k) \leq -x_i(k)\Delta t\underline{\mu_i} \tag{4.15}$$

$$\zeta_{i,j}(k) \geq \gamma_i(k) \tag{4.16}$$

$$\sum_{j \in S} \zeta_{i,j}(k) = \gamma_i(k)(|S| - 1) \tag{4.17}$$

To account for the nonlinearity of the service rates, we interpret the expressions for throughput as follows:

$$\mu_i(k) \min\{x_i(k), s_i(k)\}$$

This means that at most $s_i(k)$ jobs can be served at once (i.e., a concurrency-level limitation). Using the upper bound for the service rate

$$\mu_i(k) \leq \overline{\mu}_i \tag{4.18}$$

we can write:

$$\mu_i(k)x_i(k) \leq \overline{\mu}_i s_i(k) \Longleftrightarrow \mu_i(k) \leq \frac{\overline{\mu}_i s_i(k)}{x_i(k)} \tag{4.19}$$

The basic idea under these equations is that we are translating the upper bound on the throughput into an upper bound of the service rate $\mu_i(k)$. In particular, if $x_i(k) > s_i(k)$ then $\frac{\overline{\mu}_i s_i(k)}{x_i(k)} < \overline{\mu}_i$; thus the maximum service rate for station i is limited by constraint (4.19), instead, if $x_i(k) \leq s_i(k)$ then $\frac{\overline{\mu}_i s_i(k)}{x_i(k)} \geq \overline{\mu}_i$ hence the service rate of station i is limited by the resource constraints (4.18) as if the system was linear. Using (4.13), we are able to translate the constraints (4.18)-(4.19) over the original adaptation knobs into linear constraints over the virtual ones, yielding:

$$\gamma_i(k) \geq -\overline{\mu}_i x_i(k)\Delta t, \quad \gamma_i(k) \geq -\overline{\mu}_i s_i(k)\Delta t \tag{4.20}$$

A similar reasoning is made for the lower bound of each service rate. The constraint

$$\mu_i(k) \geq \min\left\{\underline{\mu_i}, \frac{\underline{\mu_i} s_i(k)}{x_i(k)}\right\} \tag{4.21}$$

captures the fact that the chosen rate cannot be less than its given lower bound $\underline{\mu_i}$. The second argument of the above minimum expression, instead, handles the case when $x_i(k) > s_i(k)$. The

minimum allowed service rate will be scaled down by the factor $s_i(k)/x_i(k)$ with respect to the user-defined minimum value $\underline{\mu}_i$. This ensures that the minimum throughput allowed in the linear model it is always equal to the minimum throughput specified in the original formulation. In this case, as expected, using (4.12) Equation (4.21) can be written as

$$\gamma_i(k) \leq -\underline{\mu}_i \Delta t \min\{x_i(k), s_i(k)\}$$

We remove the nonlinearity of the above min expression by introducing the binary variables $d_i(k)$, for each $i \in S$. With the further variables $dmin_i(k)$, $i \in S$, we encode the value $\min\{x_i(k), s_i(k)\}$ using the constraints:

$$\begin{aligned} dmin_i(k) &\geq x_i(k) - \alpha d_i(k) \\ dmin_i(k) &\geq s_i(k) - \alpha(1 - d_i(k)) \\ dmin_i(k) &\leq x_i(k) \\ dmin_i(k) &\leq s_i(k) \end{aligned} \tag{4.22}$$

where α is an arbitrary scaling factor chosen such that $\alpha \geq |x_i(k) - s_i(k)|$ for $\forall i \in S$. This leads to the desired upper bound for $\gamma_i(k)$:

$$\gamma_i(k) \leq -\underline{\mu}_i dmin_i(k) \Delta t \tag{4.23}$$

We develop an objective function consistent with (4.2), specifically we denote by $m'(k) = \big(m'_1(k), \ldots, m'_R(k)\big)$ the vector of performance metrics to be tracked where each $m_i(k)$, $1 \leq i \leq R$ is defined as follows:

$$m'_i(k) = \begin{cases} -\gamma_j(k) & \text{if } \mathsf{kind}(R_i) = \mathsf{Th}_j \\ x_j(k) & \text{if } \mathsf{kind}(R_i) = \mathsf{Q}_j \end{cases}$$

Similarly to (4.2), $r(k)$ is the vector of reference values for each time step k with $r_i(k) = \mathsf{value}(R_i)$, and $e'(k) = m'(k) - r(k)$ is the error to be minimized. The objective function for the MILP formulation is given by:

$$f := \sum_{k=0}^{H-1} e'(k)^T e'(k) \tag{4.24}$$

Putting all these results together, the MILP formulation results to be specified as follows:

$$\begin{aligned} &\text{minimize}_{u'(k)} \quad f \\ &u'(k) = \{\gamma_i(k), \zeta_{i,j}(k), s_i(k), d_i(k)\} \end{aligned} \tag{4.25}$$

subject to:

$$\text{Eqs. (4.11), (4.16), (4.17), (4.20), (4.22), (4.23),}$$

$$s_i(k) = \hat{s}_i, \qquad s_i(k) \in S - V \tag{4.26}$$

$$\gamma_i(k) = -\hat{\mu}_i x_i(k) \Delta t, \qquad \mu_i \in S - V \tag{4.27}$$

$$\zeta_{i,j}(k) = \gamma_i(k)(1 - \hat{p}_{i,j}), \qquad p_{i,j} \in S - V \tag{4.28}$$

$$\text{for } k = 0, \ldots, H-1,$$

where with (4.26)–(4.28) we set the values for the QN parameters.

The integer variables $d_i(k)$ ensure that the lower bounds used during the optimization are consistent with the user-defined values $\underline{\mu_i}$. Indeed, the removal of (4.22) from our formulation leads the minimum throughput $\underline{\mu_i} x_i(k)$ to increase by a factor $\frac{x_i(k)}{s_i(k)}$ when $x_i(k) > s_i(k)$.

It is worth noticing that the objective function is composed only of quadratic terms over the decision variables, making the above MILP program a quadratic one and still efficiently solvable with standard convex optimization techniques. This theoretical formulation results to be of key relevance for the efficiency of our MPC formulation.

Chapter 5

Efficient Performance Adaptation of Co-located Distributed Applications

In this Chapter we study the definition of an efficient performance-driven self-adaptation technique for controlling the performance behaviour of multiple applications (e.g., multiple virtual machines) sharing the same execution environment (i.e., the same CPUs or disks). Basically, it can be seen as an extension of Chapter 4 for controlling multi-class QN, where each application is modeled by a different class of service with its own service demands and routing schema. Under this new perspective, the goal of the runtime adaptation become to find the optimal resource sharing quota (i.e., vertical scaling) and load distribution policy (i.e., horizontal scaling) needed for fulling QoS requirements.

To this end, starting from a the multi-class QN model, we formulate a model-predictive control problem which can be efficiently solved by linear programming. A validation performed on a shared virtualized environment hosting two real applications shows that only a combined vertical and horizontal adaptation can efficiently achieve desired performance targets in the presence of heterogeneous computational resources. The content of this chapter is mainly taken from [3].

5.1 Overview

Performance adaptation of co-located distributed applications consists in satisfying quality-of-service agreements expressed as response-time or throughput requirements for multiple applications that share common resources. It is considered a challenging activity [136]. Indeed, current resource schedulers blindly operate in a *performance unaware* fashion, both at the level of the hypervisor of virtual machines (VMs) or of the operating system [137, 138]. As a consequence, the expected *performance isolation*, i.e., the behavior of one VM should not negatively impact performance of other running VMs (e.g., [139]), must be guaranteed by the computing platform providers [140–142].

Here we focus on CPU-intensive applications running on a virtualized environment. To effectively allocate resources at runtime and in an adaptive manner, *vertical* and *horizontal* scaling are promising techniques; the former varies the resource shares on each individual machine, whereas the latter deals with choosing the number of virtual machines employed [143]. Unfortunately, state-of-the-art approaches mainly apply vertical and horizontal scaling in an isolated fashion. According to a recent survey on this topic [143], among the 87 surveyed approaches only two have explored optimization techniques to search for combined vertical and horizontal scaling [144, 145]. However, in both cases horizontal scaling assumes a fixed symmetric load balancing toward all the horizontal replicas. This may not be appropriate when machines have different hardware characteristics (i.e., due to uncertain runtime disruptive events such as software aging or hardware degradation), since a symmetric load distribution may worsen performance.

To overcome this limitation, we propose a novel performance driven self-adaptation technique that combining horizontal and vertical scaling, can efficiently determine the load distribution policy to continuously fulfill performance goals of distributed co-located applications. We consider a model-based approach using QNs (see Section 2.1). In particular, we employ fluid multi-class QNs, where each class represents an application with its own demand on the CPU and specific performance targets.

We formulate the question of finding a combined horizontal and vertical scaling strategy as an efficient model predictive control problem (see Chapter 4). At each time step, the output is an allocation of the resource-sharing quotas on each machine and the routing probabilities across replicas that steer the model toward the reference set points for each application.

The use of MPC with an ODE model to control performance-related indices of a distributed application has been studied in Chapter 4, but for queuing models with a single class of users only. In this Chapter we present two significant extensions:

1. A multi-class model that enables an accurate representation of the *capped allocation* paradigm [146]. This is a CPU-sharing mechanism available in most modern off-the-shelf hypervisors (e.g., [147, 148]), which defines the maximum share that a VM can receive from the CPU.

2. The specification of latency-based response-time requirements, enriching [2], which was limited to queue-length and throughput requirements only.

A positively surprising side effect of our new multi-class MPC formulation is the reduced computational cost, since the whole control problem is now encoded as a linear programming problem (LP, see e.g., [149]) as opposed to the mixed-integer program of the single class formulation of Chapter 4. This is due to the fact that in that work the control was acting on an integer variable representing the number of CPUs in each machine, whereas here we control a continuous variable that represents the CPU share allocated to each application.

We conducted the evaluation of the proposed approach on a real shared virtualized environment hosting two load-balanced applications (Section 5.3) by showing that only a combined vertical and horizontal load balancing adaptation can efficiently achieve desired performance targets when heterogeneous computational resources are considered.

As a final remark, we would stress as even in this work we assumed that all of the system parameters could be directly measurable in the monitoring phase. We postpone the parameter estimation problem in Chapter 6.

5.2 Combined Vertical and Horizontal Autoscaling

5.2.1 A Running Example

Figure 5.1 shows a QN model of a prototypal system (i.e., a multi-class load balancer) on which to perform combined autoscaling. There are two processing nodes represented by the queuing stations M_1 and M_2. Each node serves two application classes; each class may have different service demands and performance requirements. The demands are expressed as exponential distributions; for instance, $1/\mu_{1,2}$ is the average service time of class-2 application on node 1. Node M_0 represents a dispatcher that submits user's requests (the *jobs* circulating in the QN) to either computational node. Horizontal scaling is achieved by choosing the routing probability with which the dispatcher selects the actual processing node. For example, setting $p_{0,1,1} = p_{0,2,1} = p_{0,1,2} = p_{0,2,2} = 0.5$ induces a symmetric load balancing policy according to which requests are evenly distributed across all the processing nodes. Vertical scaling is achieved by choosing the CPU quotas assigned to the applications in each machine. For example, fixing $\alpha_{1,2} = 0.3$ assigns a share of 30% of computational resources to class-2 jobs. The parameter s_1 indicates the total number of CPU cores available in node 1. We note that the shares need not to sum up to one at a node in which case the computation resources are not fully utilized.

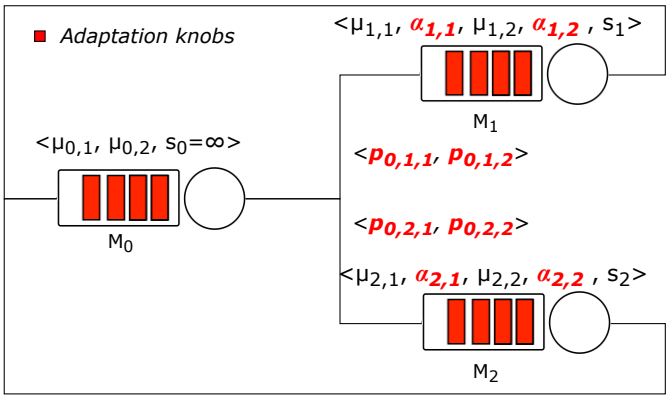


FIG. 5.1: Sample QN model.

The *adaptation knobs*, i.e., the values that can be changed at runtime are indicated in red. In the following we formally define all the different components of the proposed approach.

5.2.2 Multi Class Parametric QN

We now introduce the concept of multi-class parametric queuing network as an extension of the single-class one presented in Section 3.4.1. Formally, let us consider a set of stations S and a set of service classes $\mathcal{C}$. A *Multi-class parametric QN* is described by a set of parameters, denoted by P, as follows:

- $s_i \in P$ is the concurrency level of station i, with $i \in S$;
- $\mu_{i,c} \in P$ is the service rate of station i for the jobs of service class c, with $i \in S, c \in \mathcal{C}$;
- $p_{i,j,c} \in P$ is the *routing probability*, i.e., the probability that a request of class c from station i goes to j, with $i, j \in S$ and $c \in \mathcal{C}$;
- $\alpha_{i,c}$ is the processing share assigned to jobs of class c at station i such that $\sum_{c \in \mathcal{C}} \alpha_{i,c} \leq 1$ and $\alpha_{i,c} \geq 0$, with $i \in S, c \in \mathcal{C}$.

Finally, to formally justify the ODE approximation, the service rates $\mu_{i,c}$ are assumed to be exponentially distributed. However, as we already said in previous Section 2.1, using [18] our framework can be extended to the nonexponential case with phase-type distributions [22].

Moreover in order to formally define the adaptation, we denote by $V \subseteq P$ the subset of adaptation knobs. For each parameter that is fixed, i.e., $p \in P - V$, $\hat{p}$ is its given value. Finally we denote by $x_{i,c}(0), i \in S$ the initial condition, i.e., the initial number of jobs of class c assigned to station i.

5.2.3 ODE Model

The evolution of the multi-class QN under a cap share resource allocation policy is described by the following ODE system:

$$\frac{dx_{i,c}(t)}{dt} = -\mu_{i,c}(t) \min\{x_{i,c}(t), \alpha_{i,c}(t) s_i(t)\} + \sum_{j \in S} p_{j,i,c}(t) \mu_{j,c}(t) \min\{x_{j,c}(t), \alpha_{j,c} s_j(t)\} \quad (5.1)$$

with initial condition $x_{i,c}(0)$, for all $i \in S, c \in C$.

The dynamical system in (5.1) can be seen as an extension of the single-class one (see Section 2.2) in which we introduced the sharing factor $\alpha_{i,c}$. Indeed, the term $\mathcal{T}_{i,c}(t) = \mu_{i,c}(t) \min\{x_{i,c}(t), \alpha_{i,c}(t) s_i(t)\}$ represents the overall *nonlinear* instantaneous average throughput of station i for jobs of class c: when the queue length $x_{i,c}(t)$ in station i is less than the reserved fraction of servers $\alpha_{i,c}(t) s_i(t)$, then the $x_{i,c}(t)$ jobs are served in parallel; otherwise some of the jobs are enqueued and only

$\alpha_{i,c}(t)s_i(t)$ of them are processed at once. The throughputs may be weighted by the class-dependent routing probabilities $p_{j,i,c}(t)$, because a station may receive only a fraction of the jobs elsewhere. Using the instantaneous average queue length $x_{i,c}(t)$ and the throughput $\mathcal{T}_{i,c}(t)$, we define $\mathcal{R}_{i,c}(t) = x_{i,c}(t)/\mathcal{T}_{i,c}(t)$ as the instantaneous average response time for jobs of class c at station i; this is the time spent by the last job of class c while competing for service at station i.

Basically, assuming a cap share resource allocation policy is equivalent to assuming that $s_{i,c} = \alpha_{i,c}(t)s_i(t)$ is the fraction of the original physical station capacity $s_i(t)$ assigned to the service class c, scaled by the sharing factor $\alpha_{i,c}$, such that $\sum_{c \in C} \alpha_{i,c}(t)s_i(t) \leq s_i(t)$. In Section 5.3 we validate this ODE model by comparing prediction results against real measurements taken from a multi-class system hosted in a shared virtualized environment.

5.2.4 LP Performance Adaptation Formulation

In Chapter 4 we showed how employing MPC for performance runtime adaptation of single-class queuing network could be reduced to the solution of a mixed-integer program. This formulation relies on a linear time-varying system with auxiliary, "virtual" adaptation knobs which will be then related to the original ones. Here we extend this formulation for controlling the multi-class QN under the cap allocation sharing. The linear time-varying system that we consider is defined as:

$$\frac{dx_{i,c}(t)}{dt} = \gamma_{i,c}(t) + \sum_{j \in S}(-\gamma_{j,c}(t) + \zeta_{j,i,c}(t)), \quad i \in S, c \in C \tag{5.2}$$

where $\gamma_{i,c}(t)$ represents the virtual throughput of station i of class c and $\zeta_{j,i,c}(t)$ is a virtual routing probability (which will be related to $p_{j,i,c}$).

In the following we show how (5.2), augmented with appropriate constraints, can be used for building an LP problem applicable for controlling systems whose performance dynamics can be described by the discrete time version of (5.1).

Discrete Time Model: In order to employ MPC, we rely on a time discretization of the ODE model with a finite step size Δt. MPC finds the optimal values of the adaptation knobs over a time horizon of H steps. Simple algebraic manipulations of (5.2) yield a formulation that reads:

$$x_{i,c}(k+1) = x_{i,c}(k) + \gamma_{i,c}(k) + \sum_{j \in S}(-\gamma_{j,c}(k) + \zeta_{j,i,c}(k)), \quad i \in S, c \in C. \tag{5.3}$$

Unfolding (5.3) over H time steps allows us to embed the dynamics of the model as a discrete set of constraints in the optimization problem:

$$
\begin{aligned}
x_{i,c}(1) &= x_{i,c}(0) + \gamma_{i,c}(0) + \sum_{j \in S}(-\gamma_{j,c}(0) + \zeta_{j,i,c}(0)) \\
x_{i,c}(2) &= x_{i,c}(1) + \gamma_{i,c}(1) + \sum_{j \in S}(-\gamma_{j,c}(1) + \zeta_{j,i,c}(1)) \\
&\dots \\
x_{i,c}(H) &= x_{i,c}(H-1) + \gamma_{i,c}(H-1) + \sum_{j \in S}(-\gamma_{j,c}(H-1) + \zeta_{j,i,c}(H-1))
\end{aligned}
$$

for all $i \in S, c \in C$.

Virtual Knobs Constraints: In order to relate the virtual adaptation knobs, i.e., $\gamma_{i,c}(k), \zeta_{i,j,c}(k)$, to the original ones, i.e., $\alpha_{i,c}(k), p_{j,i,c}(k)$, respectively, we add specific constraints to the optimization problem. Hereafter we focus only on establishing a formal correspondence between the $\gamma_{i,c}(k)$ and the actual shares since the equivalence between the virtual routing probabilities and the actual ones is analogous to what discussed in Chapter 4.

The term $-\gamma_{i,c}(k)$ represents the throughput of station i for service of class c at discrete time step k, i.e., $\mu_{i,c}(k)\min\{x_{i,c}(k), \alpha_{i,c}(k)s_i(k)\}$. Since the shares can be chosen as close to 0 as desired, this consistency is given by adding the following constraints to the optimization problem:

$$
-\gamma_{i,c}(k) \geq 0, \qquad -\gamma_{i,c}(k) \leq \mu_{i,c} s_i \Delta t \tag{5.4}
$$

$$
-\gamma_{i,c}(k) \leq \mu_{i,c} x_{i,c}(k) \Delta t, \qquad -\sum_{c \in C} \frac{\gamma_{i,c}(k)}{\mu_{i,c}} \leq s_i \Delta t \tag{5.5}
$$

with $i \in S, c \in C$. In (5.4),(5.5) we consider a time invariant service rate $\mu_{i,c}(k) = \mu_{i,c}$ for each station i of service class c and a time invariant parallelism level $s_i(k) = s_i$. Indeed differently from Chapter 4 in the new LP control formulation the number of cores assigned to each machine is statically determined and only the share parameters are used to control the runtime performance of the system. However we remark that this formulation can be easily extended to those cases in which the number of virtual machines need to be computed at runtime (i.e., by considering a time variant s_i).

Objective Function: We define the objective function of the optimization problem. We consider R performance metrics to be optimized. For each time step $k = 0, 1, \dots, H-1$, in the vector $m(k) = \big(m_1(k), \dots, m_R(k)\big)$ each component $m_r(k)$ represents the variable associated with the r-th performance metric, with $1 \leq r \leq R$. We specify the values that this can take according to the type of instantaneous average index to optimize: throughput, queue length, or

response time. For all k we set:

$$m_r(k) = \begin{cases} -\dfrac{\gamma_{i,c}(k)}{\Delta t} & \text{if the } r\text{-th metric is the class-}c \text{ throughput at station } i, \\ x_{i,c}(k) & \text{if the } r\text{-th metric is the class-}c \text{ queue length at station } i. \end{cases}$$

For the encoding of response time, the treatment is different because we need to handle the nonlinear expression $\mathcal{R}_{i,c}(k) = \frac{x_{i,c}(k)}{T_{i,c}(k)} = \frac{x_{i,c}(k)\Delta t}{-\gamma_{i,c}(k)}$. We linearize this problem as follows. We let $\beta_{i,c}$ denote the desired response time requirement for class c in station i at time step k. Then, the idea is to minimize the quantity $|x_{i,c}(k)\Delta t + \beta_{i,c}(k)\gamma_{i,c}(k)|$. In order to do so, we consider auxiliary variables $\tilde{x}_{i,c}(k)$ and let $m_r(k) = \tilde{x}_{i,c}(k)$ if the r-th metric is the class-c response time at station i. Then, we can observe that by adding the following constraints to the LP problem

$$\tilde{x}_{i,c}(k) \geq 0 \tag{5.6}$$

$$\tilde{x}_{i,c}(k) \geq x_{i,c}(k)\Delta t + \beta_{i,c}\gamma_{i,c}(k) \tag{5.7}$$

$$-\tilde{x}_{i,c}(k) \leq x_{i,c}(k)\Delta t + \beta_{i,c}\gamma_{i,c}(k) \tag{5.8}$$

minimizing $\tilde{x}_{i,c}(k)$ results in finding the value for the adaptation knobs such that the response time at time k for station i and class c is as close as possible to the desired value $\beta_{i,c}(k)$.

Thus, overall we can collect the set points in vectors $o(k) = (o_1(k), \ldots, o_R(k))$. Each component of this vector, $o_r(k)$, is equal to the desired set point if the r-th requirement is throughput or a queue length, or 0 if the r-th requirement is a response time.

Our goal is to minimize the error between the performance indices and their reference values, i.e., $e(k) = m(k) - o(k)$, across all time steps in the horizon $k = 0, 1, \ldots, H-1$. Thus, overall the LP formulation can be specified as follows:

$$\text{minimize}_{\{\gamma_{i,c}(k), \zeta_{i,j,c}(k), \tilde{x}_{i,c}(k)\}} \quad \sum_{k=0}^{H-1} e(k)^T e(k) \tag{5.9}$$

subject to constraints Eqs. (5.3), (5.4), (5.6), (5.7), (5.8)

$$\begin{aligned} \zeta_{i,j,c}(k) &\geq \gamma_{i,c}(k) && \text{if } p_{i,j,c} \in V \\ \sum_{j \in S} \zeta_{i,j,c}(k) &= \gamma_{i,c}(k)(|S| - 1), \\ \zeta_{i,j,c}(k) &= \gamma_{i,c}(k)(1 - \hat{p}_{i,j,c}) && \text{if } p_{i,j,c} \in P - V \end{aligned} \tag{5.10}$$

for all $k = 0, \ldots, H-1,\ i \in S,\ c \in C$, where with (5.10) we set the values for all the parameters of the QN that are fixed.

Following [2], it is possible to define a *nonlinear MPC* formulation built on a discrete-time representation of the model (5.1). With the following result, we can recover the shares and

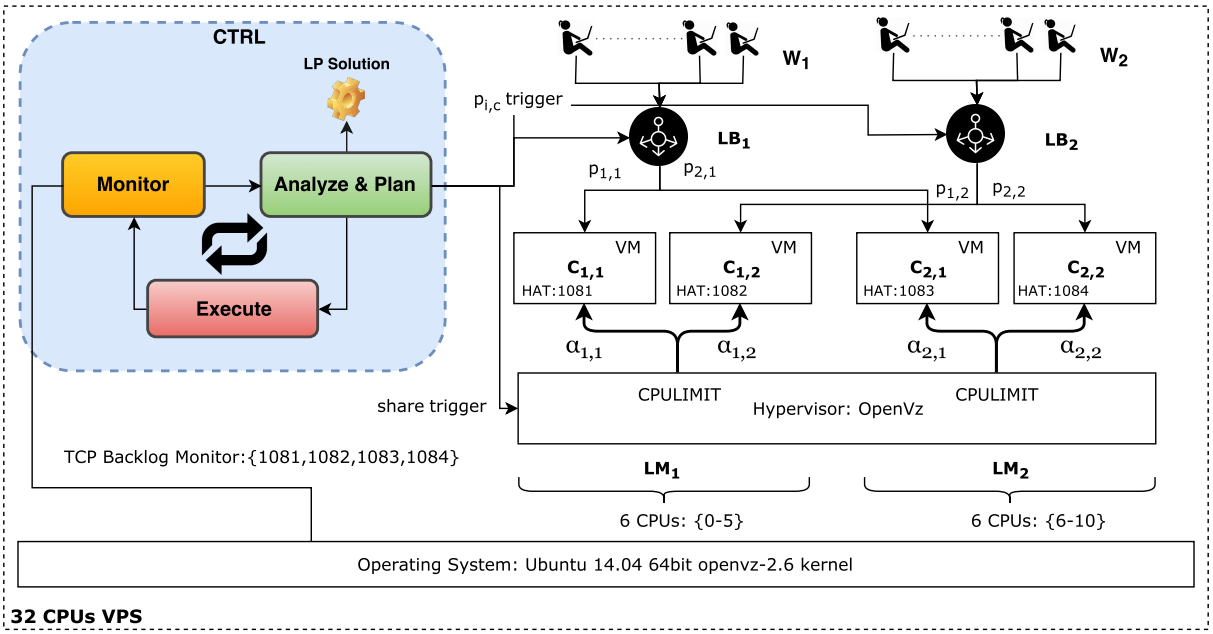


FIG. 5.2: Experiment system architecture

routing probabilities for the original nonlinear model from the LP formulation above.

Theorem 3. Denoting by $\mathcal{S} = \{\alpha^*_{i,c}(k), p^*_{i,j,c}(k)\}$ an optimal solution to the nonlinear MPC formulation built on (5.1) based on theorem 2, there exists an MPC problem encoded as an LP formulation with dynamics (5.3) such that its optimal solution $\mathcal{S}' = \{\gamma'_{i,c}(k),\ x'_{i,c}(k), \zeta'_{i,j,c}(k)\}$ satisfies:

$$\alpha^*_{i,c}(k) = \frac{\gamma'_{i,c}(k)}{s_i \Delta t}, \quad p^*_{i,j,c}(k) = \frac{\gamma'_{i,c}(k) - \zeta'_{i,j,c}(k)}{\gamma'_{i,c}(k)}$$

for all $k = 0, \ldots, H-1$.

5.3 Numerical Evaluation

In this section we evaluate the effectiveness of the proposed adaptation approach on a real multi-class load-balanced system. The code needed for setting up the experimental infrastructure is publicly available at https://goo.gl/6bNR23.

5.3.1 System Description and Implementation

Similarly to Chpater 4, for running our evaluation we relied on the in-house developed web application, namely HAT, specifically designed for resembling the behavior of CPU-intensive systems [150, 151]. We conducted our study on a single virtual private server (VPS) equipped with 32 cores and 6 GB of memory. As a vertically scalable virtualized environment we used the *OpenVz* hypervisor [148], while the horizontal scaling has been enabled through a load balancer

implemented in NodeJS. In order to validate our control approach in a resource contention scenario, we ran two instances of the same load balanced HAT deployment, each consisting of two OpenVz virtual machines.

Figure 5.2 depicts the architecture of the considered system. There are two classes of applications and two processing nodes. We emulated a distributed scenario by partitioning the available CPU cores into two Logical Machines **LM** with 6 cores each. The remaining cores are used for running the monitoring infrastructure and the controller. Component $\mathbf{C}_{i,j}$ is the instance of the computational service for class j running on logical machine $\mathbf{LM}_i$; $\mathbf{LB}_j$ is the dispatcher for class j; **CTRL** is the runtime controller. Component $\mathbf{W}_j$ represents the workload generator which injects requests of class-j service into the system. With these settings, $\mathbf{LB}_j$ dispatches user requests for class j to processing node i with routing probability $p_{i,j}$, while the resource share of class j executing at node i is $\alpha_{i,j}$. These values are adapted at runtime by the **CTRL** component, in a MAPE-K [129] fashion, through operating system signals (see Figure 4.2) and the OpenVz interface.

CTRL ran the LP optimizations using the academic version of CPLEX tool. We implemented each $\mathbf{W}_j$ as a multi-process Python based workload generator running independent concurrent users that iteratively issue requests, waiting an exponentially distributed delay (i.e., the *think time* given by $1/\mu_{0,i} \geq 0$) between successive requests. Components $\mathbf{C}_{i,j}$ and $\mathbf{LB}_j$ have been implemented as multi-threaded NodeJs servers using the NAPA library.

5.3.2 Model Parametrization and Validation

We modeled the system under study with a multi-class QN as depicted in Figure 5.1. The QN processing node N_0 represents the $\mathbf{W}_1$ and $\mathbf{W}_2$ workload generators, while nodes N_1 and N_2 model the logical machines $\mathbf{LM_1}$ and $\mathbf{LM_2}$.

For model validation, we set think times $1/\mu_{0,1} = 1/\mu_{0,2} = 1\,\mathrm{s}$, and population levels $X_1 = X_2 = 200$ users (i.e., we assumed a closed workload). We assigned $s_1 = s_2 = 2$ cores to each processing node and service rates $\mu_{i,j} = 20.5\,\mathrm{s}^{-1}$ for $i, j = 1, 2$. These rates were estimated by measuring the maximum throughput on the actual hardware platform. Finally we deployed the system in its symmetric configuration, i.e., $p_{i,j} = 0.5$ for $i, j = 1, 2$. To exercise the system under different conditions we considered 10 different resource share allocations.

Table 5.1 reports the validation results in terms of the measured and predicted throughputs for class-j application, denoted by $\mathcal{T}_{mj}$ and $\mathcal{T}_{pj}$, respectively, as well as their relative percentage errors $\mathcal{E}_{rj}$. For each resource share assignment, the throughputs were measured by averaging 10 independent executions, each lasting 2 minutes. The results show that the model can predict the trends of the real system adequately. We consider the errors acceptable, since a simple deterministic model omits many low-level interactions between the operating system and the virtualization environment.

TABLE 5.1: Model validation results. The errors $\mathcal{E}_{r1}$ and $\mathcal{E}_{r2}$ between the measured and predicted throughputs for class 1 and class 2, respectively, are measured as absolute relative percentage errors.

$\alpha_{1,1}, \alpha_{2,1}$	0.15	0.30	0.40	0.50	0.60	0.65	0.70	0.75	0.80	0.85
$\alpha_{1,2}, \alpha_{2,2}$	0.85	0.70	0.60	0.50	0.40	0.35	0.30	0.25	0.20	0.15
$\mathcal{T}_{m1}$	10.85	22.47	29.85	40.85	45.90	50.10	54.57	57.73	62.50	65.06
$\mathcal{T}_{p1}$	12.00	24.00	32.00	39.99	48.00	52.00	56.00	60.00	63.99	67.97
$\mathcal{E}_{r1}$	10.53	6.78	7.21	2.12	4.56	3.80	2.61	3.93	2.39	4.48
$\mathcal{T}_{m2}$	65.94	53.40	44.54	40.80	30.67	26.83	22.90	18.72	14.79	10.84
$\mathcal{T}_{p2}$	67.97	56.00	48.00	39.99	32.00	28.00	24.00	20.00	16.00	12.00
$\mathcal{E}_{r2}$	3.08	4.87	7.75	2.00	4.34	4.34	4.79	6.82	8.18	10.62

5.3.3 Adaptation Evaluation

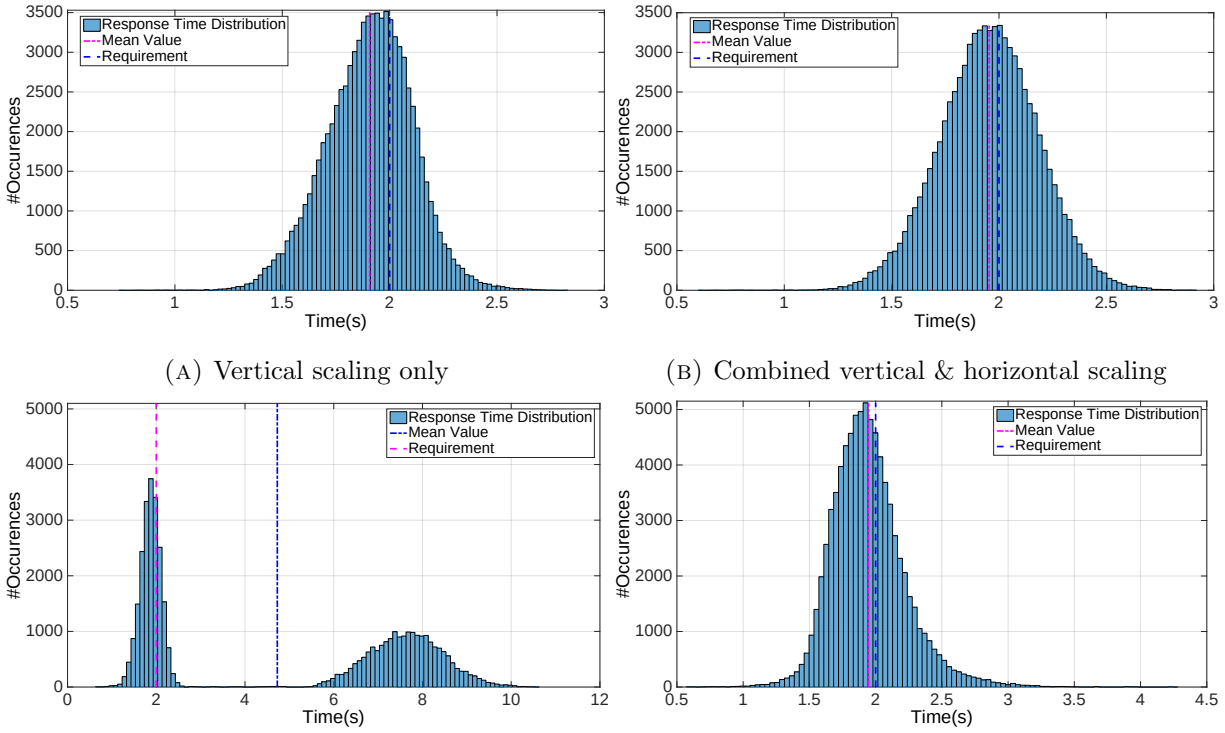
We evaluate the effectiveness of our approach by showing that the combined vertical and horizontal load balancing adaptation can efficiently meet performance targets when either of the two techniques alone cannot. We focus on a scenario of *hardware degradation*. Starting from a symmetric set-up where the service rates of nodes $\mathbf{C}_{i,j}$ are identical and equal to 20.5 (as in the validation set-up), we inject a degradation event where the service rate at node $\mathbf{LM}_1$ becomes 3 times smaller.

For both the symmetric and the degraded case, the objective of the adaptation was to maintain the following set points: instantaneous average response time of the class-1 application equal to 2 s; and class-2 instantaneous average throughput equal to 50 requests per second.

We controlled the performance of the system under a workload of $X_1 = X_2 = 200$ concurrent users with a think rate $\mu_{0,1} = \mu_{0,2} = 1\,\mathrm{s}^{-1}$. According to the system description (see Figure 4.2), we assigned $s_1 = s_2 = 6$ to each logical machine. For the vertical scaling control we fixed the symmetric distribution $p_{i,j} = 0.5$, with $i, j = 1, 2$, while the controller could change all resource shares $\alpha_{i,j}$ in an isolated fashion. When the combined control approach is applied also the routing probabilities are changed at runtime.

We evaluated both the control approaches, i.e., vertical and combined, in two separated sessions, i.e., symmetric and degraded, each of which was 20-minutes long. We fixed an ODE sampling interval $\Delta t = 0.1\,\mathrm{s}$, an activation loop rate= 0.1 s, and control horizon $H = 10$.

Figures 5.3a and 5.3b report the class-1 instantaneous average response time distributions in the symmetric set-up (i.e., no degradation) when vertical scaling only and the combined approaches are applied, respectively. In this case both the control approaches are able to fulfill the requirements. This is due to the fact that the performance target is locally achievable on each logical machine by varying the allocation shares only. Under degradation, the advantage of the combined control (i.e., vertical plus dynamic load distribution policy) becomes evident. Indeed, Figure 5.3c depicts the class-1 response time distribution when the vertical scaling is



(A) Vertical scaling only

(B) Combined vertical & horizontal scaling

(C) Vertical scaling only

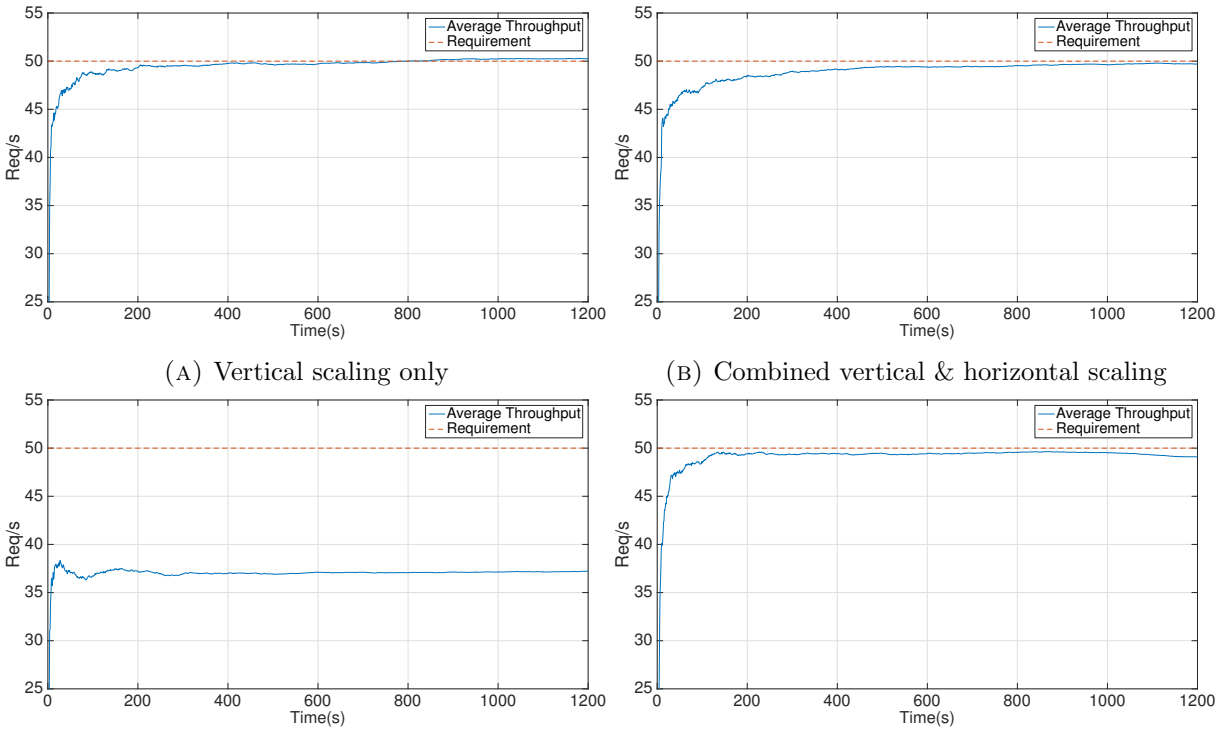
(D) Combined vertical & horizontal scaling

FIG. 5.3: Class-1 instantaneous average response time distribution with (5.3c, 5.3d) and without (5.3a, 5.3b) degradation.

applied under the degradation scenario. Since the joint requirements for class-1 and class-2 are no longer satisfiable locally, all the users sent to $\mathbf{LM}_2$ will still experience the intended response time, while the ones sent to $\mathbf{LM}_1$ will be served by a saturated system characterized by poor performance. Figure 5.3d, instead, depicts the class-1 response time distribution when the combined vertical and horizontal load balancing autoscaling is applied. In this case the routing probabilities of both classes are properly adjusted, steering the system toward the requirements fulfillment regardless of the differences in the service rates. We remark how, under the same scenario, applying a state-of-the-art horizontal scaling technique (i.e., [144, 145]) would lead to a system with a larger number of provisioned virtual machines, thus incurring higher costs and adaptation delays.

Regarding class-2 throughput adaptation, when the system works in the symmetric case both the control approaches are able to fulfill the requirements (see Figures 5.4a, 5.4b). With degradation, only the combined approach is able to steer the system toward the desired set points (see Figures 5.4c,5.4d).

Table 5.2 reports the average values for the control signals used during each adaptation trace. We can observe that during the degradation case $\mathbf{LM}_1$ is saturated since $\alpha_{1,1} + \alpha_{1,2} \simeq 1$. In this case the only way to satisfy the requirements of both classes is to operate at the load distribution level redirecting the majority of the user requests on the faster machine $\mathbf{LM}_2$ while properly



(A) Vertical scaling only

(B) Combined vertical & horizontal scaling

(C) Vertical scaling only

(D) Combined vertical & horizontal scaling

FIG. 5.4: Class-2 instantaneous average throughput with (5.4c, 5.4d) and without degradation (5.4a, 5.4b).

TABLE 5.2: Optimal control signals.

Scenario	**Ctrl type**	$\alpha_{1,1}$	$\alpha_{2,1}$	$\alpha_{1,2}$	$\alpha_{2,2}$	$\mathbf{p}_{1,1}$	$\mathbf{p}_{2,1}$	$\mathbf{p}_{1,2}$	$\mathbf{p}_{2,2}$
No Degradation	Vertical	0.26	0.26	0.25	0.27	0.50	0.50	0.50	0.50
	Combined	0.25	0.28	0.29	0.25	0.50	0.50	0.62	0.38
Degradation	Vertical	0.46	0.13	0.53	0.28	0.50	0.50	0.50	0.50
	Combined	0.46	0.37	0.32	0.38	0.27	0.73	0.24	0.76

varying the CPU shares, i.e., in a combined vertical and horizontal autoscaling fashion.

5.4 Discussion

In this chapter we have presented an efficient approach for the performance adaptation of distributed co-located applications using fluid multi-class queuing network and model predictive control. The main novelties lay in the combined usage of vertical and horizontal auto scaling techniques and in the extension of the fluid approximation presented in chapter 4 for modeling multi-class distributed applications under a capped resources allocation scheduler. At each time step during system evolution a linear programming problem is solved for computing the adaptation knobs (i.e., routing probability and allocation shares) necessary for steering the system to throughput or response time set points. As future work, we aim to extend our adaptation problem formulation to explicitly model the response time distribution instead of

its instantaneous average only. Moreover, we also plan to: *i)* study the scalability of the approach while varying the system size, e.g., increasing the number of VMs; *ii)* extend our method to include resource contention policies for network, memory, and I/O; *iii)* consider more expressive resource schedulers and system performance interactions such as the completely fair scheduler [137] and layered queuing networks [40].

Chapter 6

Moving Horizon Estimation of Service Demands for Queuing Networks

Accurate estimation of resource demands is one of the key challenges to be able to use QNs for performance prediction. This problem is worsened when one needs to obtain a continuously updated model (e.g., for control and adaptation purposes as the ones presented in Chapters 3-5) because it becomes crucial to use fast estimation methods that do not interfere with the behavior of the running system. In this respect, a key limitation in the state of the art is the assumption that the measurements are taken from a system in the steady state regime. To the best of our knowledge, this chapter presents the first approach—here developed for single-class QNs—that does not make such an assumption. In particular, we set up a *moving-horizon estimation* problem whereby the governing equations of the model (i.e., the QN fluid approximation) represent the constraints of a quadratic program that seeks to find the optimal choice of service demands that minimize the error between the measured queue lengths and the predicted ones. An extensive numerical evaluation demonstrates the efficiency and effectiveness of our approach against the state-of-the-art techniques for service demands estimation. The content of this chapter is mainly taken from [4].

6.1 Overview

Accurately predicting the performance of a computing system hinges on the availability of a model with well calibrated parameters. When the system under consideration can be satisfactorily modeled as a queuing network, key quantities to estimate are the service demands, i.e., the parameters that uniquely characterize the distributions of the service times that govern each node of the network. Moreover, the runtime usage of such quantitative models to perform

performance-driven adaptation (see Chapters 3-5) calls for efficient techniques that can provide frequent up-to-date estimates of service demands in a minimally intrusive manner so as not to alter the running system significantly.

The new perspective of online estimation brings about two main challenges.

i) It may not be appropriate to assume that the system is in the steady-state regime, in particular when it is subjected to control actions that update its parameters frequently enough. The steady-state assumption is important because it allows the use of many well-known relationships and/or analytical results for QNs (see e.g., [22]). For example, the Utilization Law permits the derivation of the service demands from the knowledge of the steady-state throughput and utilization at a station [152]. Importantly, none of the techniques reported in the recent survey on service demand estimation [153] is applicable to systems in a transient regime.

ii) It is not possible to employ techniques that require major intrusions into the running system such as, for instance, the injection of measurement traffic to exercise the system under different utilization levels (e.g., [16]). Regression techniques that involve different quantities (e.g., throughput and utilization) may introduce instrumentation overheads and may cause inaccuracies and bias in the estimation due to multicollinearity [154]. This has motivated the development of techniques that make use of queue-length information only [155, 156], which in some cases may be accessible externally (i.e., from the operating system) without the need for direct instrumentation of the application.

In this chapter we present the first estimation technique that, to our knowledge, does not make the steady-state assumption, using low-overhead measurements of queue lengths only for systems that can be modeled as single-class QNs with exponentially distributed service times and load-dependent (i.e., multiple-server) service rates.

As we already did throughout this thesis, here we consider a discrete-time model of the QN providing an estimate of its transient evolution. Then, starting from some known initial condition that gives the queue length at each station in the QN, the discrete dynamics are unfolded over H steps, obtaining explicit equations for the queue lengths at such subsequent steps. This unfolding is interpreted as a set of constraints for an optimization problem where the decision variables are the service demands to be estimated, with the objective of minimizing the error between the predicted queue lengths and the measured ones across the whole observation horizon H. Naturally, this set-up is agnostic to the knowledge of the measured system being in steady state since it involves dynamical equations that can be used over any given fixed time horizon. In addition, it allows for an on-line estimation of the service demands through a *moving horizon* strategy, by shifting forward the time window at each step in order to continuously obtain updated estimates. This represents a novel approach to service demands estimation which

is fundamentally due to the availability of a simple (approximating) ODE system, unleashing the adaptation of ideas on system identification based on moving-horizon estimation that are originally rooted in control theory (e.g., [157–159]).

As discussed in Section 2.2, the ODEs have piecewise-linear derivatives due to the presence of minimum functions that encode the load-dependent rates in a multi-server queue [18, 40, 160]. However, we avoid to explicitly model this nonlinearity in the optimization problem since it is naturally encoded in the measured queue-length dynamics. Indeed, in this chapter we show how to formulate the moving horizon estimation into a quadratic program (QP) which can be efficiently used for the estimation. Then, only after the solution of each optimization problem the explicit throughput nonlinear relation is considered for computing the actual system service demands. We remark how this approach is somewhat dual to that in Chapter 4, where a mixed integer linear optimization (based on the same class of QNs) was obtained for the purposes of model predictive control. Indeed, in that work, similar constraints as in this Chapter are used, but the objective function represents a quality-of-service requirement (i.e., throughput, response time, or queue length) that can be achieved using decision variables related to the routing probabilities and number of servers at each station; here, as discussed, the objective function is the prediction error, while routing probabilities and server multiplicities are assumed to be known.

The accuracy of the service-demand estimates ultimately depends on the accuracy of the approximation of the fluid model with respect to the true system behavior. We refer the reader to Section 2.2 for a detailed discussion about the accuracy error of QN fluid approximation.

With a substantial numerical assessment using both controlled experiments and randomized ones, we show that our service demand technique is efficient and yield accurate estimates, with an average error of 3.82% and a maximum average runtime of 0.52s on QNs with over 20 stations, thus promoting our approach as effective technique for on-line estimation of service demands.

The remainder of this chapter is organized as follows. Section 6.2 summarizes the related state of the art approaches. Section 6.3 presents our QP-based estimation method. Section 6.4 discusses the results of our numerical evaluation, while Section 6.5 provides perspectives on future work.

6.2 Related Work

When applied to a system in the steady state, our technique can be compared with the literature on service demand estimation. As previously mentioned, many approaches rely on a corollary of Little's Law (see Section 2.1), the Utilization Law [153], using different statistical inference approaches such as linear regression [161], non-linear optimization [86], clustering regression [162], independent component analysis [163], pattern matching [164] and Gibbs sampling [155, 165] based on measured values of utilization and/or throughput. A drawback of these approaches

is that they require observations of quantities that are difficult to obtain: indeed, utilization is not available when there is no complete control of the underlying physical or virtual layer (e.g., when using a Platform-as-a-Service environment).

Most approaches require *active probing*, i.e., observing the system in different configurations (e.g., at different utilization levels). For example, the technique presented in [166], which is related to ours since it is based on a QP optimization, measures the utilization of every station under different system configurations (i.e., different load combinations, load intensity, and so on), and it estimates demands only relying on a steady-state closed equation for the QN. However, active probing at runtime induces extra traffic that interferes with the one generated by the actual users of the system under identification. This extra-traffic is necessarily intrusive and the observed metrics will differ from those that would have been generated by the regular traffic only (see [167]). This difference makes the identification problem more difficult since reconstructing the original metric (i.e. the *ground-truth*) from the measured one is not straightforward.

The approach presented in [16] is non-intrusive since it measures only the end-to-end response time and throughput of the transactions submitted to the system, modeled as a QN. The service demands of all network stations are estimated by means of a nonlinear optimization problem that fits the computed performance metrics with the measured ones. The technique requires active probing for the collection of several steady-state observations at different utilization levels of the network. In addition, because it relies only on end-to-end measurements taken where the workload is generated, the method can only yield a feasible assignment for the service demands; for example it cannot distinguish service demands that differ up to a permutation in a series of queuing stations.

Most closely related to the work presented in this chapter are two recent techniques that share the parsimony of monitoring by considering queue-length measurements only [155, 156], which are often easily accessible, e.g., from logs or operating-system application programming interfaces. In [155], the authors develop an estimation method based on Gibbs sampling. However, the computational cost of the algorithm is high already for networks of small/moderate size, making it not applicable online. Instead, in [156] a closed-form expression to evaluate all the network's stations service demands for a multi-class application is presented for the load-independent case. The load-dependent case is treated approximated in closed form by an appropriate scaling of the computed service demands. We postpone a more detailed comparison of that work against our technique in Section 6.4.

6.3 Moving Horizon Estimation of Service Demands for Queuing Networks

As mentioned in Section 6.1, the moving horizon estimation approach can be seen as the dual interpretation of the model predictive control one proposed in [2]. In that work, the MPC problem has been formulated for the dynamical system (2.3), with the goal of finding the model parameters that steer the system toward a desired set point (e.g., a throughput or queue-length target), encoded as a suitable objective function. Here, instead, we rely on the same dynamical model but with the objective to find the most suitable system parameters (i.e., service demands) reproducing the observed queue length dynamics. This translates in the minimization of a cost function expressing the error between the measured queue lengths and the predicted ones. In the following the moving horizon estimation (MHE) approach is formally discussed.

6.3.1 Nonlinear service demand estimator

In a straightforward MHE formulation equations (4.1) can be directly used as constraints in an optimization problem which seeks to minimize the error between the predicted queue lengths $x_i(k)$ and the measured queue lengths over a given time horizon H. Denoting by $\tilde{x}_i(k)$ the measured queue length at time step k, we can write the optimization problem as follows:

$$\begin{aligned} &\text{minimize} \sum_{k=1}^{H} \sum_{i=1}^{M} (x_i(k) - \tilde{x}_i(k))^2 \\ &\text{subject to:} \\ &\qquad \text{Eq. (4.1)}, \quad \text{for } 0 \leq k \leq H-1,\ 1 \leq i \leq M, \\ &\qquad x_i(0) = \tilde{x}_i(0), \quad 1 \leq i \leq M. \end{aligned} \tag{6.1}$$

In other words, we search for the optimal vector of service rates μ that minimizes the difference between the measurements and the model predictions across all stations and all discrete time points over the horizon H, when the model dynamics is initialized with the measured queue lengths.[1]

6.3.2 Quadratic programming formulation

The main drawback of the optimization problem (6.1) is the presence of the nonlinear terms appearing in (4.1). We now consider a formulation in terms of a quadratic programming problem. The key point is to replace each nonlinear term with an auxiliary variable which linearizes

[1] In this general formulation we assume that all service rates are unknown; known service rates can be accommodated by simply treating them as constants, i.e., by adding further equalities to the optimization problem.

the constraints. We do this by setting

$$T_i(k) := \Delta t \mu_i \min\{x_i(k), s_i\}, \quad k \geq 0, 1 \leq i \leq M. \tag{6.2}$$

Essentially, $T_i(k)$ represents the instantaneous discretized throughput at station i. Then, the constraints (4.1) become:

$$x_i(k+1) = x_i(k) - T_i(k) + \sum_{j=1}^{M} p_{j,i} T_j(k) \tag{6.3}$$

Overall, the optimization problem (6.1) is rewritten thus:

$$\begin{aligned} &\text{minimize} \sum_{k=1}^{H} \sum_{i=1}^{M} (x_i(k) - \tilde{x}_i(k))^2, \\ &\text{subject to:} \\ &\qquad \text{Eq. (6.3)}, \quad \text{for } 0 \leq k \leq H-1,\ 1 \leq i \leq M, \\ &\qquad x_i(0) = \tilde{x}_i(0), \quad 1 \leq i \leq M. \end{aligned} \tag{6.4}$$

We denote by $x_i^*(k)$ and $T_i^*(k)$, with $0 \leq k \leq H-1$ and $1 \leq i \leq M$, the solution found by the above optimization problem. Using (6.2), we can define our service demand estimator as follows:

$$\mu_i^* := \frac{\sum_{k=0}^{H-1} T_i^*(k)}{\sum_{k=0}^{H-1} \min\{x_i^*(k), s_i\}}, \qquad 1 \leq i \leq M. \tag{6.5}$$

This essentially estimates the average service demand at station i, i.e., $1/\mu_i^*$, across the entire observation window.

6.3.3 Moving horizon estimation

The optimization problem presented in the previous section can be used as a moving horizon estimator for service demands of a QN by iterating the following steps at runtime:

a) Feed the QP program (6.4) with the measured state of the QN over the past observation window H by assigning:

   - the measured queue length $\tilde{x}_i(k)$ at station i for all stations in the network and for every observation step $k = 0, \ldots, H-1$
   - the (fixed) number of servers at station i to variables s_i;
   - the (fixed) routing probability between station i and station j to variables $p_{i,j}$;

b) Solve (6.4), obtaining the optimal sequence for $x_i^*(k)$ and $T_i^*(k)$.

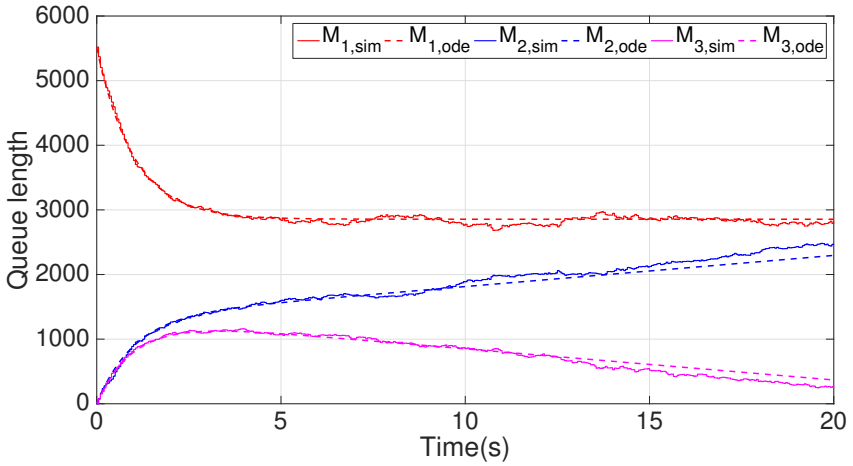


FIG. 6.1: Simulation and ODE solution for the queue lengths of the running example.

TABLE 6.1: Model parameters for the load balancer of Fig. 2.1a

N	μ_1	μ_2	μ_3	s_0	s_1, s_2	$p_{1,2}$, $p_{1,3}$	$p_{2,1}$, $p_{3,1}$
5500	1.0	68.97	73.80	∞	20	0.5	1.0

c) With (6.5), obtain the average estimation values for $1/\mu_i^*$ over the observation window H.

d) Shift the estimation window forward by discarding the oldest queue length samples, i.e., $\tilde{x}_i(0)$, with $1 \leq i \leq M$, while including the newly measured ones.

e) Return to step a).

Continuously applying the moving horizon estimation can detect changes in service demands as early as possible. Furthermore, it makes the approach applicable regardless of the assumption of the system exhibiting a steady-state behavior.

6.3.4 Application to the load balancer case study

Let us now apply the moving-horizon estimation discussed in the previous paragraph to our running example (see Figure 2.1a). We consider a parametrization as in Table 6.1, where we assume that the service demands at stations 2-3 (here chosen at random) must be estimated, while the demand at station 1 is known and fixed to 1.0. In order to model station 1 as a delay, in practice we encode the infinite-server semantics by simply choosing a value $s_0 \geq N$.

Queue-length traces for each station were generated from the simulation of one sample path of the CTMC underlying the QN model for 20 time units, starting from an initial condition where all jobs are located in station 1. Then, we resampled the obtained queue-length traces with a time step $\Delta t = 0.01$, hence for a total of 2000 time steps. Figure 6.1 depicts the simulated trace against the numerical solution of the ODE (2.4). The plot shows that the ODE gives an accurate estimate of the ground-truth dynamics represented by stochastic simulation. Furthermore, it

highlights that with the chosen parameterization the system does not attain steady state within the whole time course.

We then applied the moving horizon estimator iteratively over fixed observation windows with $H = 100$, hence solving 1901 optimization problems of the form (6.4) with the last H samples of the queue length at each station. The percentage errors of the service demand estimates, averaged across all iterations, are 1.4% and 2.6% for station 2 and station 3, respectively, with an average solution time of 0.01 s on an ordinary laptop.

6.4 Numerical Evaluation

In this section we evaluate the effectiveness and the scalability of the moving horizon estimation approach, on simulated queuing networks of different sizes and topologies. The code needed for replicating all the experiments is publicly available at https://goo.gl/zNdr5f.

6.4.1 Methodology

In Section 6.4.2 we assess the effectiveness of our MHE approach against the recent queue length maximum likelihood estimation (QMLE) method proposed in [156]. As discussed in Section 6.1, QMLE estimates service demands from queue-length data only. In Section 6.4.3 we carry out an analysis of the scalability of MHE, by measuring runtime statistics on networks with increasing number of stations. In both cases the queue length traces, i.e., the inputs of the estimation experiments, have been collected by simulating the underlying CTMC of the QN using stochastic simulation based on Gillespie's algorithm [168]. Moreover, we ran the QP optimization problems underlying MHE by using the python interface of the well-known CPLEX optimization tool.[2] We executed all the experiments on a laptop machine equipped with an Intel dual-core i5 processor operating at a 2.6 GHz with 8 GB of memory.

6.4.2 Comparison with QMLE

In [156] QMLE is presented in two variants. The first uses nonlinear optimization on an exact closed-form expression based on the BCMP theorem [169]. The second relies on a formula derived from the Bard-Schweitzer approximate mean value analysis (BS-AMVA) [170, 171], for both load-independent (i.e., single-server) and load-dependent (i.e., multi-server) queuing centers. Here we propose a comparison with the latter approach because the exact one is not suitable for online estimation purposes in real-world systems due to its computational complexity: indeed, in [156] are reported execution times of the order of 10^4 s for tandem networks.

[2] http://www.ibm.com/software/integration/optimization/cplex-optimizer/

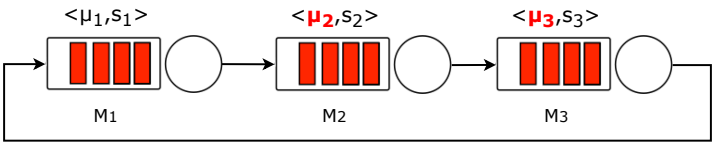


FIG. 6.2: Queuing network topology used in the comparison experiments against QMLE.

In order to conduct as fair a comparison as possible, we considered synthetic networks, with topology given in Fig. 6.2, that stress known sources of approximation errors by both methods. MHE is essentially derived from a deterministic dynamical system, the ODE system (2.3), that approximates the true stochastic dynamics of the QN. Thus, we expect the estimation error to depend on that approximation.

As we previously said in Section 2.2, the precision of the ODE approximation ultimately depends on the *system size*, intended as the number of jobs and the server multiplicities of the QN. More precisely, for a given network topology one can fix a sequence of QNs where both the initial job population $x(0)$ and the vector of server multiplicities K are scaled by the same factor $K \in \mathbb{N}$. The results of Kurtz [39], applied to the QNs of this chapter, states that in the limit when K goes to infinity, a sample path of the Markov chain becomes indistinguishable from the ODE solution. Thus, for larger values of K one may expect an increasingly high accuracy of the ODE solution with respect to a sample queue-length path. Figure 2.2 provides a graphical illustration of this effect.

The precision of QMLE depends on the accuracy of BS-AMVA and the normalization factor used in [156] in order to treat load-dependent service rates. The accuracy of the former tends to increase with the number of users circulating in the network [170]. The latter approximation essentially uses the closed-form expression for a load-independent queue and scales it by a factor equals to the minimum between the number of servers in the queue and the average queue length, similarly to the denominator in (6.5). This may introduce an error at low-utilization regimes (i.e., when the queue length tends to be less than the number of servers with higher probability).

In order to take into account all these potential sources of approximation error, we generated our synthetic benchmarks by controlling the system size as well as the utilization levels of the QN. In order to do so, we considered 6 experiments which spanned the steady-state utilization levels between 0.1 and 0.8. For each experiment we considered varying system sizes $K = 1$, 2, 5, 10, 20, and 50; the case $K = 1$ corresponds to a QN with a delay station at node M_1 and two single-server queues at nodes M_2 and M_3, i.e., $s = (\infty, 1, 1)$; we fixed the service demands arbitrarily with $\mu_1 = 1.0$, $\mu_2 = 27.0$ and $\mu_3 = 48.0$. The experiments differed in the choice of the initial condition $x(0)$, which was fixed in order to achieve varying steady-state utilization of the queues.

In accordance with the experimental settings proposed in [156], we executed QMLE with 10^5

TABLE 6.2: Accuracy comparison between QMLE and MHE.

	$x(0)=(3,0,0)$ $H=2347, U_2\approx 0.10$		$x(0)=(9,0,0)$ $H=688, U_2\approx 0.30$		$x(0)=(12,0,0)$ $H=521, U_2\approx 0.40$		$x(0)=(19,0,0)$ $H=353, U_2\approx 0.60$		$x(0)=(26,0,0)$ $H=262, U_2\approx 0.80$	
K	QMLE	MHE	QMLE	MHE	QMLE	MHE	QMLE	MHE	QMLE	MHE
1	0.52	9.25 ± 1.03	1.37	9.63 ± 1.06	2.07	7.90 ± 1.01	3.40	6.58 ± 0.81	5.15	4.89 ± 0.69
2	448.30	4.13 ± 0.62	126.54	3.93 ± 0.58	67.18	4.20 ± 0.63	5.46	3.90 ± 0.56	2.33	3.59 ± 0.54
5	184.02	2.26 ± 0.33	60.41	3.02 ± 0.43	42.09	2.76 ± 0.38	8.78	2.07 ± 0.33	1.65	2.06 ± 0.34
10	92.29	1.65 ± 0.27	30.53	1.99 ± 0.31	23.18	1.82 ± 0.31	9.50	2.09 ± 0.30	3.89	1.50 ± 0.24
20	45.18	1.37 ± 0.21	15.01	1.13 ± 0.19	11.32	1.36 ± 0.18	6.41	1.36 ± 0.19	5.81	1.17 ± 0.18
50	18.67	0.74 ± 0.10	6.08	0.81 ± 0.14	4.57	0.78 ± 0.11	2.72	0.81 ± 0.12	5.17	0.73 ± 0.10

steady-state queue-length samples. MHE depends on the observation window H and the discretization parameter Δt. Here we used a different value of H for each experiment (i.e., for each utilization level) such that in the time window $H\Delta t$ we observed a roughly constant number of service events (i.e., approximately 1500) while choosing a discretization step $\Delta t = 0.1$ that allowed an accurate enough ODE solution. For MHE we considered 100 non-overlapping intervals of length $H\Delta t$ which were used as input to the optimization problem (6.4), in order to compute statistics about the estimation error.

Table 6.2 shows the results of the comparison. For each experiment we report the value of H that was used, as well as the utilization at the station M_2 (i.e., detailed results reporting statistics about station M_3 can be found in the replication package of this paper), denoted by U_2 (this utilization was roughly constant at every system size K), and the initial condition $x(0)$. For each experiment, each station, and each value of K we measured the accuracy of QMLE and MHE, computed as the mean absolute percentage errors between the estimate and the true service demands; for MHE, we report the 95% confidence intervals computed over the 100 independent samples, and the average execution runtime. Relying on the obtained results we make three main observations:

i) In the load-independent (i.e., $K=1$) case, QMLE tends to outperform MHE at low utilizations, while at higher utilizations ($U_2 \geq 0.60$) the two techniques provide comparable accuracy. Larger MHE errors at $K=1$ can be explained by the fact that this system size is considerably away from a deterministic regime, as previously discussed. Conversely, here QMLE does not feature the multi-server correction to the BS-AMVA solution. Overall, the largest extremum of the confidence interval for MHE is ca. 16%, still indicating an acceptable performance even for small QNs.

ii) In the load-dependent scenarios, i.e., for all $K>1$, MHE consistently outperforms QMLE. We attribute this to the aforementioned multi-server correction to BS-AMVA approach. Indeed, for a fixed K the error tends to decrease for larger utilizations, coherently with the fact that when the queue is highly utilized the difference between the dynamics of a single server (i.e., under BS-AMVA) and that of multiple servers with the same overall maximum service rate (i.e.,

TABLE 6.3: Scalability analysis.

	Errors				*Runtimes* (s)			
M	min	avg	95-th	max	min	avg	95-th	max
5	1.60	2.53	4.12	4.50	0.03	0.03	0.03	0.04
10	1.63	2.46	3.28	3.56	0.08	0.08	0.09	0.09
15	1.59	2.63	3.48	4.56	0.18	0.18	0.19	0.19
20	1.62	2.52	3.19	3.82	0.34	0.38	0.48	0.52

the multi-service correction) become negligible.
iii) As expected for MHE, larger values of the system size K tend to yield more accurate estimates. The results show that this also holds for QMLE.

6.4.3 Scalability analysis

Here we further study the effectiveness and scalability of MHE on QNs with randomly generated topologies and parameters. We fixed total number of stations M equal to 5, 10, 15, 20. For each case we generated 20 QNs with random parameters. In particular, the routing probability matrices were randomly generated stochastic matrices (i.e., ensuring a closed QN workload); the number of servers at each station was picked uniformly at random in $\{20, \ldots, 50\}$ while the service rates were chosen randomly from the interval $[10, 50]$. For each such random QN, the initial number of jobs was chosen such that the bottleneck queue attained a utilization of about 0.8, using the approximate formulas presented in [18]. For each random QN we computed the average service demand estimate across all stations and with 100 non-overlapping observation windows with $H = 200$ taken from a sample path.

Table 6.3 shows the collected results. In particular for each value of M we report the minimum, average, the 95-th quantile and the maximum error across the 20 random QNs. We observe that the number of stations M does not significantly affect the accuracy of the estimation. As expected, the execution time of MHE grows almost linearly with increasing number of stations in the network. Moreover, for large systems (i.e., $M = 20$) the average time needed for obtaining new estimations by solving the QP optimization, does not exceed the fraction of a second in the worst case (i.e., 0.52s).

6.5 Discussion

In this chapter we have presented a technique for service demand estimation of queueing networks using a moving horizon approach. The key insight is the interpretation of the QN as a deterministic dynamical system that approximates the stochastic queue-length processes. While

state-of-the-art techniques are based on steady-state analysis of the QN, ours can naturally be used also in the transient regime.

The method requires the solution of a quadratic programming problem that fits the instantaneous throughputs of each station to minimize the error between the measured queue lengths and the estimated ones. The numerical results demonstrate high accuracy across a wide range of operating regimes and network sizes at a low computational cost, hence promoting our approach as an effective method for online estimation.

Future works will be concerned with the development of extensions for multi-class QNs with non-exponentially distributed service times.

Chapter 7

Conclusions

The novel contribution of this thesis can be summarized as follow: *i*) through the definition of innovative optimization methods based on queuing networks (QNs) fluid analysis we enabled fast adaptation reactions under strict time constraints. Indeed, employing the techniques described in Chapters 4 and 5, it is possible to continuously steer, at runtime, average performance indexes (i.e., queue length, throughput, response time) of single and multi-class applications toward prescribed operating point regardless the presence of disruptive uncertain events (i.e., workload fluctuation and hardware degradation). Through extensive numerical validation we have shown how complex multi-dimensional adaptation actions (i.e., simultaneous vertical and horizontal scaling) can be computed at a small computational cost (i.e., order of hundreds of milliseconds) in real-world scenarios. *ii)* Through the development of the first transient estimation technique for service demands of QNs, we enabled effective and efficient parameter estimation in a minimally intrusive manner. Indeed, employing the technique proposed in Chapter 6 the service demands of a queuing network model of any topology can be estimated without the injection of measurement traffic.

In the following the achievements of each proposed technique are detailed. Finally, we discuss some of possible future research directions.

7.1 Achievements

Symbolic Performance Adaptation In Chapter 3 a symbolic approach for performance adaptation of software systems has been proposed. Its main novelty resides in the encoding of the adaptation problem as a satisfiability modulo theories one.

One major strength of this approach is that a great amount of the computation is shifted to design-time where the symbolic QN solution need to be computed. The numerical experiments

showed that the computational overhead introduced by this technique allows to perform adaptation steps in a few milliseconds even in a realistic three-tier model under nontrivial adaptation scenarios.

Two relevant issues of the presented approach are the availability of closed-form expressions for steady-state systems only and its inability to find the *optimal* configurations but just a feasible one. We addressed both these limitations in Chapter 4 where a transient and efficient optimal runtime self-adaptation technique is proposed.

Efficient Performance-driven Self-adaptation In Chapter 4 we have presented an efficient performance-driven self-adaptive approach based QN fluid approximation and model predictive control. Its main novel contribution is the exact formulation of the original nonlinear runtime adaptation problem as a mixed integer linear one, hence enabling faster adaptations with respect to state-of-the-art approaches.

One of its main limitations resides in the parametrization of the QN model of a system. However we remark that our MPC formulation can be used as a framework for defining an accurate online parameterization, i.e., the current observations (e.g., queue-length) may be encoded in the optimization problem as new constraints. We explored this research direction in Chapter 6 where an efficient moving horizon estimator for service demands of queuing networks has been proposed. Moreover we extended this work for controlling multi-class applications in Chapter 5.

Efficient Performance Adaptation of Co-located Distributed Applications In Chapter 5 we have presented an efficient approach for the performance adaptation of co-located applications using fluid multi-class queuing network and model predictive control. The main novelties are: *i)* the combined usage of vertical and horizontal auto scaling techniques, *ii)* the extension of the fluid approximation for modeling multi-class distributed system under a capped resources allocation scheduler. The numerical experiments have shown how only the combined usage of vertical and horizontal auto scaling can effectively fulfill performance requirements in presence of heterogeneous computational resources.

Moving Horizon Estimation of Service Demands for Queuing Networks In Chapter 6 we have presented a technique for service demand estimation of queueing networks using a moving horizon approach. The key idea has been to see a QN as the deterministic approximation of the measured stochastic queue-length processes. Its main advantage is that the estimation procedure can be used in the transient regime while state-of-the-art techniques are limited to steady-state analysis of the QN. Moreover, our technique does not require the injection of any measurement traffic so as does not interfere with the normal operational behaviour of the system.

The numerical results demonstrate high accuracy across a wide range of utilizations and network sizes at a low computational cost, thus making our approach employable as an effective method for continuous estimation.

7.2 Future Work

One of the main limitations of all the performance-driven self-adaptation techniques proposed in this thesis is the assumption on "flat" nature of the used QN models. This results in the lack of the necessary expressiveness to cover realistic mechanisms such as simultaneous resource possession and layered services (e.g., multi-tiered applications when one tier needs processing from tiers below in order to complete its job).

In order to overcome this limitation, we plan to build a more expressive model predictive control formulation based on the fluid interpretation of layered queuing networks [40]. To this end, smart techniques allowing to encode the nonlinearities introduced by the new contention mechanism in an efficient optimization problem [172] need to be devised.

Another key issue of the presented approaches resides in the fact that the considered fluid model can only be employed for capturing the first order approximation of the underlying stochastic process (i.e., only the mean is estimated). This results in the impossibility to define adaptation strategies driven by distribution requirements, e.g., 95% of clients need to be processed within a target time t.

Tackling this issue, we plan to embed in our QN fluid approximation further dynamics that allow modeling higher order statistics of the estimated quantities [173]. Then relying on optimization problem, similar to the ones already proposed in this thesis, higher order performance-driven adaptation techniques can be defined.

Two main limitations of the proposed moving horizon estimator approach reside in the assumptions of exponentially distributed service demands and single-class queuing networks. These could hamper the applicability of our estimation approach to those systems characterized by generally distributed service demands and multiple classes of users.

Facing such complexity, as future work we plan the development of minimally intrusive estimation techniques for multi-class QNs with non-exponentially distributed service times by fitting service demands against phase-type distributions or Markov arrival process [174].

Bibliography

[1] Emilio Incerto, Mirco Tribastone, and Catia Trubiani. Symbolic performance adaptation. In *International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 140–150, 2016.

[2] Emilio Incerto, Mirco Tribastone, and Catia Trubiani. Software performance self-adaptation through efficient model predictive control. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 485–496, 2017.

[3] Emilio Incerto, Mirco Tribastone, and Catia Trubiani. Combined vertical and horizontal autoscaling through model predictive control. In *European Conference on Parallel Processing*, pages 147–159. Springer, 2018.

[4] Emilio Incerto, Annalisa Napolitano, and Mirco Tribastone. Moving horizon estimation of service demands in queuing networks. In *International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2018.

[5] Emilio Incerto, Mirco Tribastone, and Catia Trubiani. A proactive approach for runtime self-adaptation based on queueing network fluid analysis. In *International Workshop on Quality-Aware DevOps, (QUDOS)*, pages 19–24, 2015.

[6] Bhuvan Urgaonkar, Giovanni Pacifici, Prashant Shenoy, Mike Spreitzer, and Asser Tantawi. An analytical model for multi-tier internet services and its applications. In *ACM SIGMETRICS Performance Evaluation Review*, volume 33, pages 291–302, 2005.

[7] Mark Harman and Peter O'Hearn. From start-ups to scale-ups: Opportunities and open problems for static and dynamic program analysis. In *IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM)*, 2018.

[8] Vittorio Cortellessa and Raffaela Mirandola. Deriving a queueing network based performance model from UML diagrams. In *International Workshop on Software and Performance*, pages 58–70, 2000.

[9] Antinisca Di Marco and Paola Inverardi. Compositional generation of software architecture performance qn models. In *Working IEEE/IFIP Conference on Software Architectures*, pages 37–46, 2004.

[10] Yu Chi Ho and Xiren Cao. Perturbation analysis and optimization of queueing networks. *Journal of Optimization Theory and Applications*, 40(4):559–582, 1983.

[11] Vittorio Cortellessa and Raffaela Mirandola. PRIMA-UML: a performance validation incremental methodology on early uml diagrams. *Science of Computer Programming*, 44 (1):101–129, 2002.

[12] Hyunsang Youn, Suhyeon Jang, and Eunseok Lee. Deriving queuing network model for UML for software performance prediction. In *Software Engineering Research, Management & Applications, 2007. SERA 2007. 5th ACIS International Conference on*, pages 125–131, 2007.

[13] Antonio Filieri, Giordano Tamburrelli, and Carlo Ghezzi. Supporting self-adaptation via quantitative verification and sensitivity analysis at run time. *IEEE Trans. Software Eng.*, 42(1):75–99, 2016.

[14] Paolo Arcaini, Elvinia Riccobene, and Patrizia Scandurra. Modeling and analyzing mape-k feedback loops for self-adaptation. In *Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 13–23. IEEE Press, 2015.

[15] Rogério De Lemos, Holger Giese, Hausi A. Müller, Mary Shaw, Jesper Andersson, Marin Litoiu, Bradley Schmerl, Gabriel Tamura, Norha M. Villegas, Thomas Vogel, and others. Software engineering for self-adaptive systems: A second research roadmap. In *Software Engineering for Self-Adaptive Systems II*, pages 1–32. 2013.

[16] Mahmoud Awad and Daniel A. Menasce. Deriving parameters for open and closed qn models of operational systems through black box optimization. In *Proceedings of the International Conference on Performance Engineering (ICPE)*, 2017.

[17] Antti Valmari. The state explosion problem. *Lectures on Petri nets I: Basic models*, pages 429–528, 1998.

[18] Matthias Kowal, Max Tschaikowski, Mirco Tribastone, and Ina Schaefer. Scaling size and parameter spaces in variability-aware software performance models. In *International Conference on Automated Software Engineering (ASE)*, pages 407–417, 2015.

[19] Clark W Barrett, Roberto Sebastiani, Sanjit A Seshia, and Cesare Tinelli. Satisfiability modulo theories. *Handbook of satisfiability*, 185:825–885, 2009.

[20] Carlos E. García, David M. Prett, and Manfred Morari. Model predictive control: Theory and practice—a survey. *Automatica*, 25(3):335–348, 1989.

[21] Xiaoqiang Wang, Vladimir Mahalec, and Feng Qian. Globally optimal nonlinear model predictive control based on multi-parametric disaggregation. *Journal of Process Control*, 52:1–13, 2017.

[22] Gunter Bolch, Stefan Greiner, Hermann de Meer, and Kishor Trivedi. *Queueing networks and Markov chains: modeling and performance evaluation with computer science applications*. Wiley, 2005.

[23] Aldo Cumani. On the canonical representation of homogeneous Markov processes modelling failure-time distributions. *Microelectronics Reliability*, 22(3):583–602, 1982.

[24] William J Stewart. *Probability, Markov chains, queues, and simulation: the mathematical basis of performance modeling*. Princeton University Press, 2009.

[25] Asser N Tantawi and Don Towsley. Optimal static load balancing in distributed computer systems. *Journal of the ACM*, 32(2):445–465, 1985.

[26] Abhishek Chandra, Micah Adler, Pawan Goyal, and Prashant Shenoy. Surplus fair scheduling: A proportional-share cpu scheduling algorithm for symmetric multiprocessors. In *Proceedings of the 4th conference on Symposium on Operating System Design & Implementation-Volume 4*, page 4. USENIX Association, 2000.

[27] Valeria Cardellini, Michele Colajanni, and Philip S Yu. Dynamic load balancing on web-server systems. *IEEE Internet computing*, 3(3):28–39, 1999.

[28] EC Amazon. Amazon web services. *Available in: http://aws. amazon. com/es/ec2/(November 2012)*, 2015.

[29] Afkham Azeez. Auto-scaling axis2 web services on amazon ec2. *ApacheCon Europe*, 2009.

[30] Zenon Chaczko, Venkatesh Mahadevan, Shahrzad Aslanzadeh, and Christopher Mcdermid. Availability and load balancing in cloud computing. In *International Conference on Computer and Software Modeling, Singapore*, volume 14, 2011.

[31] William J Anderson. *Continuous-time Markov chains: An applications-oriented approach*. Springer Science & Business Media, 2012.

[32] Martin Reiser and Stephen S Lavenberg. Mean-value analysis of closed multichain queuing networks. *Journal of the ACM (JACM)*, (2):313–322, 1980.

[33] John DC Little. A proof for the queuing formula: L= λW. *Operations research*, 9(3): 383–387, 1961.

[34] Steve S Lavenberg and Martin Reiser. Stationary state probabilities at arrival instants for closed queueing networks with multiple types of customers. *Journal of Applied Probability*, 17(4):1048–1061, 1980.

[35] John Hajnal and MS Bartlett. Weak ergodicity in non-homogeneous Markov chains. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 54, pages 233–246. Cambridge University Press, 1958.

[36] Luca Bortolussi, Jane Hillston, Diego Latella, and Mieke Massink. Continuous approximation of collective system behaviour: A tutorial. *Performance Evaluation*, 70(5):317–349, 2013.

[37] Jane Hillston, Mirco Tribastone, and Stephen Gilmore. Stochastic process algebras: From individuals to populations. *Comput. J.*, 55(7):866–881, 2012.

[38] Mirco Tribastone, Jie Ding, Stephen Gilmore, and Jane Hillston. Fluid rewards for a stochastic process algebra. *IEEE Transactions on Software Engineering*, 38:861–874, 2012.

[39] T. G. Kurtz. Solutions of ordinary differential equations as limits of pure Markov processes. In *J. Appl. Prob.*, number 1, pages 49–58, 1970.

[40] Mirco Tribastone. A fluid model for layered queueing networks. *IEEE Trans. Software Eng.*, 39:744–756, 2013.

[41] Nelly Bencomo, Paul Grace, Carlos Flores, Danny Hughes, and Gordon Blair. Genie: Supporting the model driven development of reflective, component-based adaptive systems. In *Proceedings of the 30th international conference on Software engineering*, pages 811–814, 2008.

[42] Arun Mishra and AK Misra. Component assessment and proactive model for support of dynamic integration in self adaptive system. *SIGSOFT Softw. Eng. Notes*, 34:1–9, 2009.

[43] Daniel A. Menascé, Hassan Gomaa, Sam Malek, and João Pedro Sousa. SASSY: A framework for self-architecting service-oriented systems. *IEEE Software*, 28(6):78–85, 2011.

[44] Javier Cubo and Ernesto Pimentel. Damasco: A framework for the automatic composition of component-based and service-oriented architectures. In *Software Architecture*, pages 388–404. 2011.

[45] Danny Weyns and Michael Georgeff. Self-adaptation using multiagent systems. *Software, IEEE*, 27:86–91, 2010.

[46] Kiwon Yeom and Ji-Hyung Park. Morphological approach for autonomous and adaptive systems based on self-reconfigurable modular agents. *Future Generation Computer Systems*, 28:533–543, 2012.

[47] Toly Chen. A self-adaptive agent-based fuzzy-neural scheduling system for a wafer fabrication factory. *Expert Systems with Applications*, 38:7158–7168, 2011.

[48] Frank D Macías-Escrivá, Rodolfo Haber, Raul Del Toro, and Vicente Hernandez. Self-adaptive systems: A survey of current approaches, research challenges and applications. *Expert Systems with Applications*, 40:7267–7279, 2013.

[49] Kurt Geihs, Paolo Barone, Frank Eliassen, Jacqueline Floch, Rolf Fricke, Eli Gjorven, Svein Hallsteinsen, Geir Horn, Mohammad Ullah Khan, and Alessandro Mamelli. A comprehensive solution for application-level adaptation. *Software: Practice and Experience*, 39:385–422, 2009.

[50] Thomas Vogel, Stefan Neumann, Stephan Hildebrandt, Holger Giese, and Basil Becker. Model-driven architectural monitoring and adaptation for autonomic systems. In *Proceedings of the 6th International Conference on Autonomic Computing*, pages 67–68, 2009.

[51] Yuriy Brun, Giovanna Di Marzo Serugendo, Cristina Gacek, Holger Giese, Holger Kienle, Marin Litoiu, Hausi Müller, Mauro Pezzè, and Mary Shaw. Engineering self-adaptive systems through feedback loops. In *Software engineering for self-adaptive systems*, pages 48–70. 2009.

[52] Hal Abelson, Jacob Beal, and Gerald Jay Sussman. Amorphous computing. *Commun. ACM*, 43:74–82, 2000.

[53] Yuriy Brun and Nenad Medvidovic. An architectural style for solving computationally intensive problems on large networks. In *Proceedings of the 2007 International Workshop on Software Engineering for Adaptive and Self-Managing Systems*, page 2, 2007.

[54] Yuriy Brun and Nenad Medvidovic. Fault and adversary tolerance as an emergent property of distributed systems' software architectures. In *Proceedings of the 2007 workshop on Engineering fault tolerant systems*, page 7, 2007.

[55] Giovanna Di Marzo Serugendo, John S Fitzgerald, Alexander B Romanovsky, and Nicolas Guelfi. A generic framework for the engineering of self-adaptive and self-organising systems. *Organic Computing-Controlled Self-organization*, pages 165–189, 2008.

[56] Svein Hallsteinsen, Mike Hinchey, Sooyong Park, and Klaus Schmid. Dynamic software product lines. *Computer*, 41:93–95, 2008.

[57] David M Weiss and Chi Tau Robert Lai. *Software product-line engineering: a family-based software development process*, volume 12. Addison-Wesley Reading, 1999.

[58] Liwei Shen, Xin Peng, Jindu Liu, and Wenyun Zhao. Towards feature-oriented variability reconfiguration in dynamic software product lines. In *Top Productivity through Software Reuse*, pages 52–68. 2011.

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

[60] Mirco Tribastone and Stephen Gilmore. Automatic extraction of pepa performance models from uml activity diagrams annotated with the marte profile. In *Proceedings of the 7th international workshop on Software and performance*, pages 67–78. ACM, 2008.

[61] Mirco Tribastone and Stephen Gilmore. Automatic translation of uml sequence diagrams into pepa models. In *2008 Fifth International Conference on Quantitative Evaluation of Systems*, pages 205–214. IEEE, 2008.

[62] Stephen Gilmore, László Gönczy, Nora Koch, Philip Mayer, Mirco Tribastone, and Dániel Varró. Non-functional properties in the model-driven development of service-oriented systems. *Software & Systems Modeling*, 10(3):287–311, 2011.

[63] Hassan Gomaa and Koji Hashimoto. Dynamic software adaptation for service-oriented product lines. In *Proceedings of the 15th International Software Product Line Conference, Volume 2*, page 35. ACM, 2011.

[64] Javier Cubo, Nadia Gamez, Lidia Fuentes, and Ernesto Pimentel. Composition and self-adaptation of service-based systems with feature models. In *Safe and Secure Software Reuse*, pages 326–342. 2013.

[65] Naeem Esfahani and Sam Malek. Uncertainty in self-adaptive software systems. In *Software Engineering for Self-Adaptive Systems II*, pages 214–238. Springer, 2013.

[66] Diego Perez-Palacin and Raffaela Mirandola. Uncertainties in the modeling of self-adaptive systems: a taxonomy and an example of availability evaluation. pages 3–14. ACM Press, 2014.

[67] Naeem Esfahani and Sam Malek. Guided exploration of the architectural solution space in the face of uncertainty. Technical report, 2011.

[68] Naeem Esfahani, Sam Malek, and Kaveh Razavi. Guidearch: guiding the exploration of architectural solution space under uncertainty. In *International Conference on Software Engineering (ICSE)*, pages 43–52, 2013.

[69] André Van Hoorn, Jan Waller, and Wilhelm 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.

[70] Joxan Jaffar and Michael J Maher. Constraint logic programming: A survey. *The journal of logic programming*, 19:503–581, 1994.

[71] Alessandro Cimatti, Alberto Griggio, Bastiaan Joost Schaafsma, and Roberto Sebastiani. The mathSAT5 SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 93–107. Springer, 2013.

[72] Valeria Cardellini, Emiliano Casalicchio, Vincenzo Grassi, Stefano Iannucci, Francesco Lo Presti, and Raffaela Mirandola. MOSES: A framework for qos driven runtime adaptation of service-oriented systems. *IEEE Trans. Software Eng.*, 38(5):1138–1159, 2012.

[73] Ilenia Epifani, Carlo Ghezzi, Raffaela Mirandola, and Giordano Tamburrelli. Model evolution by run-time parameter adaptation. In *International Conference on Software Engineering (ICSE)*, pages 111–121, 2009.

[74] Radu Calinescu, Carlo Ghezzi, Marta Kwiatkowska, and Raffaela Mirandola. Self-adaptive software needs quantitative verification at runtime. *Commun. ACM*, 55(9):69–77, 2012.

[75] Danny Weyns, M. Usman Iftikhar, Didac Gil de la Iglesia, and Tanvir Ahmad. A survey of formal methods in self-adaptive systems. In *Proceedings of the International Conference on Computer Science and Software Engineering*, pages 67–79, 2012.

[76] Matthias Becker, Markus Luckey, and Steffen 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.

[77] Raffaela Mirandola and Catia 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.

[78] Andreas Metzger, Osama Sammodi, Klaus Pohl, and Mark Rzepka. Towards pro-active adaptation with confidence: Augmenting service monitoring with online testing. In *Workshop on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS, pages 20–28. ACM, 2010.

[79] Javier Cámara Moreno, Antónia Lopes, David Garlan, and Bradley R. Schmerl. Impact models for architecture-based self-adaptive systems. In *International Symposium on Formal Aspects of Component Software*, pages 89–107, 2014.

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

[81] M. Usman Iftikhar and Danny Weyns. Activforms: Active formal models for self-adaptation. In *SEAMS*, pages 125–134. ACM, 2014.

[82] Farn Wang, Geng-Dian Huang, and Fang Yu. TCTL inevitability analysis of dense-time systems: From theory to engineering. *IEEE Transactions on Software Engineering*, 32(7): 510–526, 2006.

[83] Vittorio Cortellessa, Antinisca Di Marco, and Paola Inverardi. *Model-Based Software Performance Analysis*. Springer, 2011.

[84] C. Murray Woodside, Tao Zheng, and Marin Litoiu. The use of optimal filters to track parameters of performance models. In *International Conference on the Quantitative Evaluation of Systems (QEST)*, pages 74–84, 2005.

[85] Antonio Filieri, Lars Grunske, and Alberto Leva. Lightweight adaptive filtering for efficient learning and updating of probabilistic models. In *International Conference on Software Engineering (ICSE)*, pages 200–211, 2015.

[86] Daniel A Menasce. Computing missing service demand parameters for performance models. In *Int. CMG Conference*, pages 241–248, 2008.

[87] Mahmoud Awad and Daniel A. Menascé. On the predictive properties of performance models derived through input-output relationships. In *European Workshop on Performance Engineering (EPEW)*, pages 89–103, 2014.

[88] Antonio Filieri, Carlo Ghezzi, Alberto Leva, and Martina Maggio. Self-adaptive software meets control theory: A preliminary approach supporting reliability requirements. In *International Conference on Automated Software Engineering (ASE)*, pages 283–292, 2011.

[89] Eunsuk Kang, Ethan Jackson, and Wolfram Schulte. An Approach for Effective Design Space Exploration. In *Foundations of Computer Software. Modeling, Development, and Verification of Adaptive Systems*, number 6662 in Lecture Notes in Computer Science, pages 33–54. Springer Berlin Heidelberg, March 2010.

[90] Catia Trubiani, Indika Meedeniya, Vittorio Cortellessa, Aldeida Aleti, and Lars Grunske. Model-based performance analysis of software architectures under uncertainty. In *International Conference on Quality of Software Architectures (QoSA)*, pages 69–78, 2013.

[91] Leire Etxeberria, Catia Trubiani, Vittorio Cortellessa, and Goiuria Sagardui. Performance-based selection of software and hardware features under parameter uncertainty. In *International Conference on Quality of Software Architectures (QoSA)*, pages 23–32, 2014.

[92] C. Barna, H. Ghanbari, M. Litoiu, and M. Shtern. Hogna: A Platform for Self-Adaptive Applications in Cloud Environments. In *SEAMS*, pages 83–87, 2015.

[93] Thomas Thüm, Sven Apel, Christian Kästner, Ina Schaefer, and Gunter Saake. A classification and survey of analysis strategies for software product lines. *ACM Comput. Surv.*, 47(1):6:1–6:45, June 2014.

[94] Leonardo De Moura and Nikolaj Bjørner. Satisfiability modulo theories: An appetizer. In *Formal Methods: Foundations and Applications*, pages 23–36. 2009.

[95] Marco Bozzano, Roberto Bruttomesso, Alessandro Cimatti, Tommi Junttila, Peter Van Rossum, Stephan Schulz, and Roberto Sebastiani. MathSAT: Tight Integration of SAT and Mathematical Decision Procedures. *Journal of Automated Reasoning*, 35: 265–293, 2005.

[96] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

[97] Alessandro Cimatti. Beyond Boolean Sat: Satisfiability Modulo Theories. In *Discrete Event Systems, 2008. WODES 2008. 9th International Workshop on*, pages 68–73, 2008.

[98] Pamela Zave and Michael Jackson. Four dark corners of requirements engineering. *ACM Trans. Softw. Eng. Methodol.*, 6(1):1–30, 1997.

[99] Thomas G. Kurtz. Solutions of Ordinary Differential Equations as Limits of Pure Jump Markov Processes. *Journal of Applied Probability*, 7:pp. 49–58, 1970.

[100] D Cerotti, M Gribaudo, P Piazzolla, R Pinciroli, and G Serazzi. Multi-class queuing networks models for energy optimization. In *International Conference on Performance Evaluation Methodologies and Tools*, pages 98–105, 2014.

[101] Simos Gerasimou, Radu Calinescu, and Alec Banks. Efficient runtime quantitative verification using caching, lookahead, and nearly-optimal reconfiguration. In *SEAMS*, pages 115–124, 2014.

[102] J. Von Kistowski, N. Herbst, D. Zoller, S. Kounev, and A. Hotho. Modeling and extracting load intensity profiles. In *SEAMS*, pages 109–119, 2015.

[103] N.R. Herbst, S. Kounev, A. Weber, and H. Groenda. BUNGEE: An Elasticity Benchmark for Self-Adaptive IaaS Cloud Environments. In *SEAMS*, pages 46–56, 2015.

[104] L. Sabatucci and M. Cossentino. From Means-End Analysis to Proactive Means-End Reasoning. In *SEAMS*, pages 2–12, 2015.

[105] Gordon Blair, Nelly Bencomo, and Robert B France. Models@runtime. *Computer*, 42 (10):22–27, 2009.

[106] Brice Morin, Olivier Barais, Jean-Marc Jezequel, Franck Fleurey, and Arnor Solberg. Models@runtime to support dynamic adaptation. *Computer*, 42(10):44–51, 2009.

[107] Carlo Ghezzi, Leandro Sales Pinto, Paola Spoletini, and Giordano Tamburrelli. Managing non-functional uncertainty via model-driven adaptivity. In *International Conference on Software Engineering (ICSE)*, pages 33–42, 2013.

[108] Shoham Ben-David, Baruch Sterin, Joanne M. Atlee, and Sandy Beidu. Symbolic model checking of product-line requirements using sat-based methods. In *International Conference on Software Engineering (ICSE)*, pages 189–199, 2015.

[109] Xue Liu, Jin Heo, Lui Sha, and Xiaoyun Zhu. Adaptive control of multi-tiered web applications using queueing predictor. In *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, pages 106–114, 2006.

[110] Yixin Diao, Joseph L Hellerstein, Sujay Parekh, Hidayatullah Shaikh, and Maheswaran Surendra. Controlling quality of service in multi-tier web applications. In *International Conference on Distributed Computing Systems (ICDCS)*, pages 25–25, 2006.

[111] Nilabja Roy, Abhishek Dubey, Aniruddha Gokhale, and Larry Dowdy. A capacity planning process for performance assurance of component-based distributed systems. In *ACM SIGSOFT Software Engineering Notes*, volume 36, pages 259–270, 2011.

[112] S. Abdelwahed, J. Bai, R. Su, and N. Kandasamy. On the application of predictive control techniques for adaptive performance management of computing systems. *IEEE Transactions on Network and Service Management*, 6(4):212–225, 2009.

[113] Tobias Achterberg and Roland Wunderling. Mixed integer programming: Analyzing 12 years of progress. In *Facets of Combinatorial Optimization*, pages 449–481. 2013.

[114] Lui Sha, Xue Liu, Ying Lu, and Tarek Abdelzaher. Queueing model based network server performance control. In *Real-Time Systems Symposium (RTSS)*, pages 81–90, 2002.

[115] Anders Robertsson, Björn Wittenmark, Maria Kihl, and Mikael Andersson. Design and evaluation of load control in web server systems. In *American Control Conference*, volume 3, pages 1980–1985, 2004.

[116] Jing Jiang, Jie Lu, Guangquan Zhang, and Guodong Long. Optimal cloud resource auto-scaling for web applications. In *International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 58–65, 2013.

[117] Hamoun Ghanbari, Marin Litoiu, Przemyslaw Pawluk, and Cornel Barna. Replica placement in cloud through simple stochastic model predictive control. In *International Conference on Cloud Computing (CLOUD)*, pages 80–87, 2014.

[118] Marta Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *International Conference on Computer-Aided Verification (CAV)*, pages 585–591, 2011.

[119] Gabriel A Moreno, Javier Cámara, David Garlan, and Bradley Schmerl. Proactive self-adaptation under uncertainty: A probabilistic model checking approach. In *Joint Meeting on Foundations of Software Engineering (FSE)*, pages 1–12, 2015.

[120] Simonetta Balsamo and Moreno Marzolla. Performance evaluation of UML software architectures with multiclass queueing network models. In *International Workshop on Software and Performance (WOSP)*, pages 37–42, 2005.

[121] Connie U. Smith, Catalina M. Lladó, and Ramón Puigjaner. Performance model interchange format (PMIF 2): A comprehensive approach to queueing network model interoperability. *Perform. Eval.*, 67(7):548–568, 2010.

[122] Rasha Tawhid and Dorina Petriu. Integrating performance analysis in the model driven development of software product lines. In *International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pages 490–504, 2008.

[123] Rasha Tawhid and Dorina C Petriu. Automatic derivation of a product performance model from a software product line model. In *International Conference on Software Product Line (SPLC)*, pages 80–89, 2011.

[124] Nilabja Roy, Abhishek Dubey, and Aniruddha Gokhale. Efficient autoscaling in the cloud using predictive models for workload forecasting. In *International Conference on Cloud Computing (CLOUD)*, pages 500–507, 2011.

[125] Qi Zhang, Mohamed Faten Zhani, Shuo Zhang, Quanyan Zhu, Raouf Boutaba, and Joseph L Hellerstein. Dynamic energy-aware capacity provisioning for cloud computing environments. In *International Conference on Autonomic Computing (ICAC)*, pages 145–154, 2012.

[126] Antonio Filieri, Henry Hoffmann, and Martina Maggio. Automated design of self-adaptive software with control-theoretical formal guarantees. In *International Conference on Software Engineering (ICSE)*, pages 299–310, 2014.

[127] Davide Arcelli, Vittorio Cortellessa, Antonio Filieri, and Alberto Leva. Control theory for model-based performance-driven software adaptation. In *International Conference on Quality of Software Architectures (QoSA)*, pages 11–20, 2015.

[128] Luciano Baresi, Sam Guinea, Alberto Leva, and Giovanni Quattrocchi. A discrete-time feedback controller for containerized cloud applications. In *International Symposium on Foundations of Software Engineering (FSE)*, pages 217–228, 2016.

[129] Jeffrey O Kephart and David M Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[130] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. Julia: A fresh approach to numerical computing. *SIAM Review*, 59(1):65–98, 2017.

[131] Samuel Burer and Adam N Letchford. Non-convex mixed-integer nonlinear programming: A survey. *Surveys in Operations Research and Management Science*, 17(2):97–106, 2012.

[132] Samuel Burer. On the copositive representation of binary and continuous nonconvex quadratic programs. *Mathematical Programming*, 120(2):479–495, 2009.

[133] Adam Piórkowski, Aleksander Kempny, Adrian Hajduk, and Jacek Strzelczyk. Load balancing for heterogeneous web servers. In *International Conference on Computer Networks (CN)*, pages 189–198, 2010.

[134] Steffen Heinzl and Christoph Metz. Toward a cloud-ready dynamic load balancer based on the apache web server. In *International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, pages 342–345, 2013.

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

[136] Dong Huang, Bingsheng He, and Chunyan Miao. A Survey of Resource Management in Multi-Tier Web Applications. *IEEE Communications Surveys & Tutorials*, 16:1574–1590, 2014.

[137] I. Molnar. This is the CFS scheduler, 1999. URL https://www.kernel.org/doc/Documentation/scheduler/sched-design-CFS.txt.

[138] Lianhua Li and Greg Franks. Performance modeling of systems using fair share scheduling with layered queueing networks. In *International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 1–10, 2009.

[139] Diwaker Gupta, Ludmila Cherkasova, Rob Gardner, and Amin Vahdat. Enforcing performance isolation across virtual machines in xen. In *International Conference on Distributed Systems Platforms and Open Distributed Processing (IFIP/USENIX)*, pages 342–362, 2006.

[140] Ewnetu Bayuh Lakew, Alessandro Vittorio Papadopoulos, Martina Maggio, Cristian Klein, and Erik Elmroth. Kpi-agnostic control for fine-grained vertical elasticity. In *International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 589–598, 2017.

[141] Ewnetu Bayuh Lakew, Cristian Klein, Francisco Hernandez-Rodriguez, and Erik Elmroth. Performance-based service differentiation in clouds. In *Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on*, pages 505–514. IEEE, 2015.

[142] Omer Adam, Young Choon Lee, and Albert Y Zomaya. Ctrlcloud: Performance-aware adaptive control for shared resources in clouds. In *International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 110–119, 2017.

[143] Chenhao Qu, Rodrigo N Calheiros, and Rajkumar Buyya. Auto-scaling web applications in clouds: A taxonomy and survey. *ACM Computing Surveys*, 9(4), 2017.

[144] Sourav Dutta, Sankalp Gera, Akshat Verma, and Balaji Viswanathan. Smartscale: Automatic application scaling in enterprise clouds. In *International Conference on Cloud Computing (CLOUD)*, pages 221–228, 2012.

[145] Anshul Gandhi, Parijat Dube, Alexei Karve, Andrzej Kochut, and Li Zhang. Modeling the impact of workload on cloud resource scaling. In *International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 310–317, 2014.

[146] Ethan Bolker and Yiping Ding. On the performance impact of fair share scheduling. In *International CMG Conference*, pages 71–82, 2000.

[147] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *ACM SIGOPS operating systems review*, volume 37, pages 164–177, 2003.

[148] Parallels. OpenVz user guide, 2016. URL https://docs.openvz.org/openvz_users_guide.webhelp/.

[149] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[150] Intel Corp. Linpack. URL https://software.intel.com/en-us/articles/intel-math-kernel-library-linpack-download.

[151] NASA. Nas parallel benchmarks. URL http://www.nas.nasa.gov/Resources/Software/npb.html.

[152] Fabian Brosig, Samuel Kounev, and Klaus Krogmann. Automated extraction of palladio component models from running enterprise java applications. In *Proceedings of the Fourth International ICST Conference on Performance Evaluation Methodologies and Tools*, VALUETOOLS '09, 2009.

[153] Simon Spinner, Giuliano Casale, Fabian Brosig, and Samuel Kounev. Evaluating approaches to resource demand estimation. *Performance Evaluation*, 92:51–71, 2015.

[154] A. Kalbasi, D. Krishnamurthy, J. Rolia, and M. Richter. MODE: mix driven on-line resource demand estimation. In *7th International Conference on Network and Service Management*, 2011.

[155] Weikun Wang and Giuliano Casale. Bayesian service demand estimation using Gibbs sampling. In *21st International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2013.

[156] Weikun Wang, Giuliano Casale, Ajay Kattepur, and Manoj Nambiar. Maximum likelihood estimation of closed queueing network demands from queue length data. In *Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering*, pages 3–14. ACM, 2016.

[157] Alberto Bemporad, Domenico Mignone, and Manfred Morari. Moving horizon estimation for hybrid systems and fault detection. In *Proceedings of the American Control Conference*, volume 4, pages 2471–2475, 1999.

[158] Frank Allgöwer, Thomas A Badgwell, Joe S Qin, James B Rawlings, and Steven J Wright. Nonlinear predictive control and moving horizon estimation—an introductory overview. In *Advances in control*, pages 391–449. Springer, 1999.

[159] James B. Rawlings and Bhavik R. Bakshi. Particle filtering and moving horizon estimation. *Computers & Chemical Engineering*, 30(10):1529–1541, 2006.

[160] Mirco Tribastone, Stephen Gilmore, and Jane Hillston. Scalable differential analysis of process algebra models. *IEEE Transactions on Software Engineering*, 38(1):205–219, 2012.

[161] Giovanni Pacifici, Wolfgang Segmuller, Mike Spreitzer, and Asser Tantawi. Cpu demand for web serving: Measurement analysis and dynamic estimation. *Performance Evaluation*, 65(6-7):531–553, 2008.

[162] Paolo Cremonesi, Kanika Dhyani, and Andrea Sansottera. Service time estimation with a refinement enhanced hybrid clustering algorithm. In *International Conference on Analytical and Stochastic Modeling Techniques and Applications*, pages 291–305. Springer, 2010.

[163] Abhishek B Sharma, Ranjita Bhagwan, Monojit Choudhury, Leana Golubchik, Ramesh Govindan, and Geoffrey M Voelker. Automatic request categorization in internet services. *ACM SIGMETRICS Performance Evaluation Review*, 36(2):16–25, 2008.

[164] Paolo Cremonesi and Andrea Sansottera. Indirect estimation of service demands in the presence of structural changes. *Performance Evaluation*, 73:18–40, 2014.

[165] Charles Sutton and Michael I Jordan. Bayesian inference for queueing networks and modeling of Internet services. *The Annals of Applied Statistics*, pages 254–282, 2011.

[166] Zhen Liu, Laura Wynter, Cathy H Xia, and Fan Zhang. Parameter inference of queueing models for IT systems using end-to-end measurements. *Performance Evaluation*, 63(1): 36–60, 2006.

[167] François Baccelli, Bruno Kauffmann, and Darryl Veitch. Inverse problems in queueing theory and Internet probing. *Queueing Systems*, 63(1-4):59, 2009.

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

[169] Forest Baskett, K Mani Chandy, Richard R Muntz, and Fernando G Palacios. Open, closed, and mixed networks of queues with different classes of customers. *Journal of the ACM*, 22(2):248–260, 1975.

[170] Yonathan Bard. Some extensions to multiclass queueing network analysis. In *Proceedings of the Third International Symposium on Modelling and Performance Evaluation of Computer Systems: Performance of Computer Systems*, pages 51–62. North-Holland Publishing Co., 1979.

[171] Paul Schweitzer. Approxiamate analysis of multiclass closed networks of queues. *J. ACM*, 29(2), 1981.

[172] Pedro M Castro. Normalized multiparametric disaggregation: an efficient relaxation for mixed-integer bilinear problems. *Journal of Global Optimization*, 64(4):765–784, 2016.

[173] Richard A Hayden, Anton Stefanek, and Jeremy T Bradley. Fluid computation of passage-time distributions in large Markov models. *Theoretical Computer Science*, 413(1):106–141, 2012.

[174] Hiroyuki Okamura and Tadashi Dohi. Fitting phase-type distributions and Markovian Arrival Processes: Algorithms and tools. In *Principles of Performance and Reliability Modeling and Evaluation*, pages 49–75. Springer, 2016.