Software Performance Self-Adaptation through Efficient Model Predictive Control

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

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

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

***Abstract*—A key challenge in software systems that are exposed to runtime variabilities, such as workload fluctuations and service degradation, is to continuously meet performance requirements. In this paper we present an approach that allows performance self-adaptation using a system model based on queuing networks (QNs), a well-assessed formalism for software performance engineering. Software engineers can select the adaptation knobs of a QN (routing probabilities, service rates, and concurrency level) and we automatically derive a Model Predictive Control (MPC) formulation suitable to continuously configure the selected knobs and track the desired performance requirements. Previous MPC approaches have two main limitations: *i)* high computational cost of the optimization, due to nonlinearity of the models; *ii)* focus on long-run performance metrics only, due to the lack of tractable representations of the QN's time-course evolution. As a consequence, these limitations allow adaptations with coarse time granularities, neglecting the system's transient behavior. Our MPC adaptation strategy is efficient since it is based on mixed integer programming, which uses a compact representation of a QN with ordinary differential equations. An extensive evaluation on an implementation of a load balancer demonstrates the effectiveness of the adaptation and compares it with traditional methods based on probabilistic model checking.**

***Index Terms*—Adaptive software, Control-theory, Model predictive control, Performance requirements**

I. INTRODUCTION

In software-intensive systems, runtime variability [1], [2] is a major impediment to satisfy quantitative non-functional properties such as performance and reliability [3]: a configuration that is initially optimal with respect to some given quality-of-service (QoS) requirements (e.g., throughput or mean time to failure) may suddenly become poor when certain events such as workload fluctuations or breakdowns, steer the system toward a sensibly different operating point. In this context, self-adaptation is a promising paradigm where uncertainty is managed by continuously monitoring the current execution conditions [4] and planning a reconfiguration using a model of the system under analysis [5], [6], [7], [8], [9], [10].

Runtime adaptation introduces two 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 adaptation knobs. Indeed, the typically huge size of the AS represents one of the major limitations for applying state-of-the-art approaches to real world scenarios [11].

Some recent work has looked at ways of efficiently exploring the parametric AS [12], [13], [14], also using SAT-based methods [15], [16]; however available solutions only concern the long-run (i.e., steady-state) performance indices. Any adaptation strategy based on such indices (as also done in [17], [18], [19], [20]) ignores the system's whole time-course dynamics. 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 paper we deal with self-adaptation for performance-related QoS requirements, and focus on software systems that can be described by queuing networks (QNs), a well-estabilished model in software performance engineering (e.g., [21], [22], [23]). The analysis of the QN's time-dependent evolution suffers the well-known problem of state space explosion, arising from the huge state space of the underlying Markov chain. To tackle this issue we consider a compact, *approximate* representation of QNs based on ordinary differential equations (ODEs) [24], [25]. Viewing a QN as a dynamical system governed by ODEs unleashes a range of techniques that would not be otherwise applicable. For instance, Filieri et al. use ODEs in a closed-loop control strategy [26], [27], but the control acts on a single parameter. The novelty of our approach is the formulation of the performance-driven self-adaptation problem using model predictive control (MPC), a well-known technique based on on-line numerical optimization [28], 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 [29]. When MPC is started at the next iteration the newly measured state will readily feed back 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 [29]. 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 paper, we achieve this by formally translating the original nonlinear MPC problem into a Mixed Integer Programming (MIP) 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 [30].

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 MIP approach shows that it can yield control loops as fast as 0.3 s on commodity hardware even for large prediction horizons.

Our 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 [31], [32], [33], [34]. In addition, it compares very favourably 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 [35], similarly to [36]. 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.

The remainder of this paper is organized as follows. Section II provides an overview of our approach. Section III sets up the MPC problem, and the mathematical formulation is reported in Appendix. Section IV numerically demonstrates the effectiveness and the scalability of our approach on a real system. Section V discusses the threats to validity. Section VI presents related works, and Section VII concludes the paper and outlines future work.

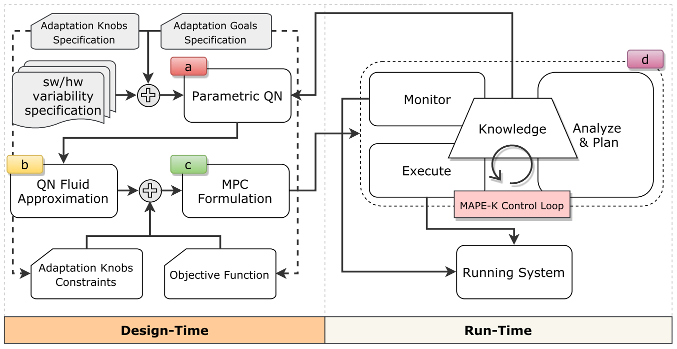


Fig. 1: Approach overview

II. Overview and Running Example

Figure 1 depicts the overview of the approach we propose.

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., [37], [38], [39], [40], [41]). We call a *parametric QN* the model including the following two main characteristics of the system specification:

I1. the elements that can act as adaptation knobs, and the range of values that each adaptation knob can take;

I2. the adaptation goals defined as desired set-points for certain performance metrics; these represent 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 IV.

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 [42] for a discussion on this.

b) For a given parametric QN, the controller can be automatically synthesized. In particular, we first obtain an ODE system, using an approach similar to [14], [43]. Using well-known arguments that relate the ODE dynamics to a stochastic Markov-chain based interpretation of the QN behavior (e.g., [24], [25]), the ODE solution gives an estimate of the average queue length at each station. Importantly the model size is independent of the number of jobs in the system, unlike in Markov chains for QNs where the size grows exponentially.

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 MIP 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 loop controller. In the *Monitor* phase the current system state is measured and used as input parameter for updating the system model embedded in the MIP formulation. The solution of the optimization problem implements the *Analysis Planning* phase. 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 optimization problem.

Running example: We use a load balancing system but we remark that the proposed approach is applicable to QN models of any topology. Load balancing is an established design technique in performance engineering [44], whose goal is to distribute incoming requests/jobs across several computational units in order to prevent overload conditions and the consequent performance degradation. Once a request enters the system, at runtime a dispatcher (the *load balancer*) selects a particular station to which the processing is delegated. This system can be configured by tuning a number of parameters, related to both the software and the hardware, such as the dispatcher's strategy and the concurrency level (i.e., how many servers can work in parallel) at each station. Events that may jeopardize 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).

The goal of self-adaptation is to efficiently explore the system's configuration space guided by QoS requirements such as throughput maximization and balancing of the station's load, under given resource constraints such as the maximum concurrency level at the stations.

III. Efficient QoS Adaptation

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

A. Parametric QN

Let us consider a set of stations S. A *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 $i \in S$;
- $\mu_i \in P$ is the service rate of station i, with $i \in S$;
- $p_{i,j} \in P$ is the *routing probability*, i.e., the probability that a request from station i goes to j, with $i, j \in S$;

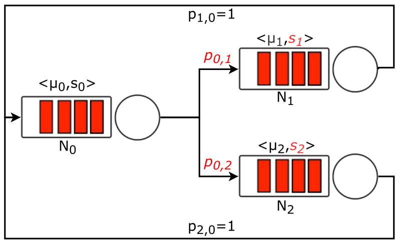


Fig. 2: Parametric QN for a load balancing system

To formally define 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 [13], [14], [16] 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 VII. To formally justify the ODE approximation, the service rates μ_i are assumed to be exponentially distributed. However, using [14] our framework can be extended to the nonexponential case via an approximation based on phase type distributions [45].

Figure 2 depicts the parametric QN for the load balancing system. Station N_0 is the dispatcher while stations N_1 and N_2 are the two processing ones. The adaptation knobs are indicated in red. In Section IV we consider the adaptation goal to have balanced queue lengths at stations N_1 and N_2 with 15 jobs. In our framework this can be done by having $R = 2$ requirements with $\mathsf{kind}_1 = \mathsf{Q}_1$, $\mathsf{value}_1 = 15$, $\mathsf{kind}_2 = \mathsf{Q}_2$, and $\mathsf{value}_2 = 15$. As intuition suggests, if stations N_1 and N_2 are identical, then the optimal strategy chooses equal routing probabilities $p_{0,1} = p_{0,2} = 1/2$. However, this does not hold any longer if, at runtime, there is an event that breaks this symmetry, such as the degradation of the service rate of either server. Intuitively, such a degradation can be compensated by increasing the probability that a job is sent to the faster server and/or increasing the concurrency level of the degraded one.

B. 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). The evolution of the QN is described by the following ODE system:

$$\frac{dx_i(t)}{dt} = -\mu_i(t) \min\{x_i(t), s_i(t)\} + \sum_{j \in S} p_{j,i}(t)\mu_j(t) \min\{x_j(t), s_j(t)\}$$

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: 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 validation of the ODE model has been performed by comparing prediction results against real measurements in order to assess its accuracy, see Section IV for further details.

C. Nonlinear MPC Formulation

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 yield a formulation that reads, in matrix notation:

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

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 $k = 0, 1, \dots, H-1$, we consider the vector of performance metrics $m(k) = \big(m_1(k), \dots, m_R(k)\big)$ where each $m_i(k)$, $1 \le i \le R$ 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 $r(k) = (r_1(k), \dots, r_R(k))$, 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 $k = 0, 1, \dots, H-1$. Thus, overall the objective function is defined as follows:

$$\text{minimize} \sum_{k=0}^{H-1} e(k)^T e(k) \tag{2}$$

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

The discretization in (1) allows us to embed the dynamics of the model as a discrete set of constraints in the optimization problem by unfolding (1) over H time steps:

$$\begin{aligned} x(1) &= A(x(0)) + x(0) \\ x(2) &= A(x(1)) + x(1) \\ &\dots \\ x(H) &= A(x(H-1)) + x(H-1) \end{aligned} \tag{3}$$

To these, we add:

$$\begin{aligned} & s_i(k) \in \{\underline{s}_i, \underline{s}_i + 1, \dots, \overline{s}_i\}, && s_i(k) \in V && (4) \\ & \underline{\mu}_i \le \mu_i(k) \le \overline{\mu}_i, && \mu_i(k) \in V && (5) \\ & 0 \le p_{i,j}(k) \le 1, \ \sum_{j\in S} p_{i,j}(k) = 1, && p_{i,j}(k) \in V && (6) \\ & s_i(k) = \hat{s}_i, && s_i(k) \in S - V && (7) \\ & \mu_i(k) = \hat{\mu}_i, && \mu_i(k) \in S - V && (8) \\ & p_{i,j}(k) = \hat{p}_{i,j}, && p_{i,j}(k) \in S - V && (9) \\ & \text{for } k = 0, \dots, H-1 \end{aligned}$$

where (4)–(6) define the ranges for the adaptation knobs, and (7)–(9) set the values for the fixed parameters. We remark that the specification of (4) includes integer variables because they represent server multiplicities which may vary discretely.

MPC is the problem of minimizing (2) subject to the constraints (3)–(9). In addition to all QN parameters, the decision variables of this problem are also the vectors $x(1)$, $x(2), \dots, x(H)$; 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 IV) are due to the nonlinearities from the multiplication of decision variables, 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 MIP formulation.

D. MIP Formulation

The MIP 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:

$$\frac{dx_i(t)}{dt} = cn_i(t) + \sum_{j\in S}(-cn_j(t) + cp_{j,i}(t)), \quad i \in S, \tag{10}$$

where $cn_i(t)$ represents the virtual service rate for station i and $cp_{i,j}(t)$ is a virtual routing probability. Setting $cn(k) = (cn_i(k))_{i\in S}$ and $cp(k) = (cp_{i,j}(k))_{i,j\in S}$, (10) can be written

$$x(k+1) = x(k) + cn(k) + \mathbf{1}(-cn(k)) + \mathbf{1}cp(k) \tag{11}$$

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

In Appendix A we demonstrate an exact correspondence between the original nonlinear MPC problem (2) and the MIP formulation over the variables appearing in (11). This leads us to state the following theorem:

Theorem 1. *Denoting by $\mathcal{S} = \{\mu_i^*(k), p_{i,j}^*(k),\ s_i^*(k)\}$ an optimal solution to (2), there exists an MPC problem based on an MIP formulation with dynamics (11) such that its optimal solution $\mathcal{S}' = \{cn_i'(k),\ x_i'(k), cp_{i,j}'(k), s_i'(k)\}$ satisfies:*

$$\mu_i^*(k) = \begin{cases} -\frac{cn_i'(k)}{x_i'(k)\Delta t} & \textit{if } x_i'(k) \le s_i'(k) \\ -\frac{cn_i'(k)}{s_i'(k)\Delta t} & \textit{if } x_i'(k) > s_i'(k) \end{cases}$$

$$p_{i,j}^*(k) = \frac{cn_i'(k) - cp_{i,j}'(k)}{cn_i'(k)}, \qquad s_i^*(k) = s_i'(k).$$

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

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

E. Adaptation Logic

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

a) Feed the MIP 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 MIP problem, obtaining the optimal sequence $cn_i'(k)$, $x_i'(k)$, $cp_{i,j}'(k)$, $s_i'(k)$, , for $k = 0, \dots, H-1$.

c) With Theorem 1, 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 VII 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 (3)–(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.

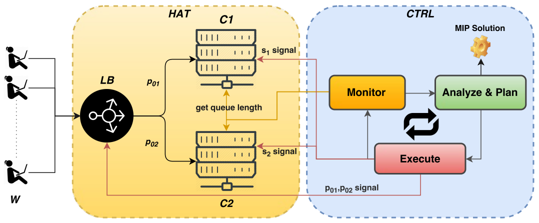


Fig. 3: HAT architecture

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., $H \cdot \Delta t \le A$. 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.

Finally, we remark that it is possible to take into account the interplay between A and H in the MPC setup: one may additionally impose that the control signals do not vary for a number of Δt steps equal to A, so as to exclude optimal control sequences that vary faster than A, which is the technological constraint of the problem. Formally, this is done by extending the constraint set (3)–(9) with

$$v(0) = v(1) = \ldots = v(c),$$
$$v(c+1) = v(c+2) = \ldots = v(2c),$$

and so on, for all decision variables $v \in V$, where c is the least integer such that $c\Delta t \ge A$. We used this approach in the experiments reported in the next section.

IV. Evaluation

We evaluate the effectiveness and the scalability of the proposed MPC approach on a real system. A virtual machine hosting the experimental infrastructure is publicly available at https://goo.gl/rpWCHv.

A. 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 based load-balancing system running on the Ubuntu server 14.04 Linux distribution. Figure 3 depicts the architecture: ***C1*** and ***C2*** represent the processing nodes, ***LB*** identifies the dispatcher, while ***CTRL*** is the controller component. Component

TABLE I: 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

W represents the workload generator, issuing requests to ***LB***. In particular, following the load balancing paradigm (see Section II), 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 a HTML page.

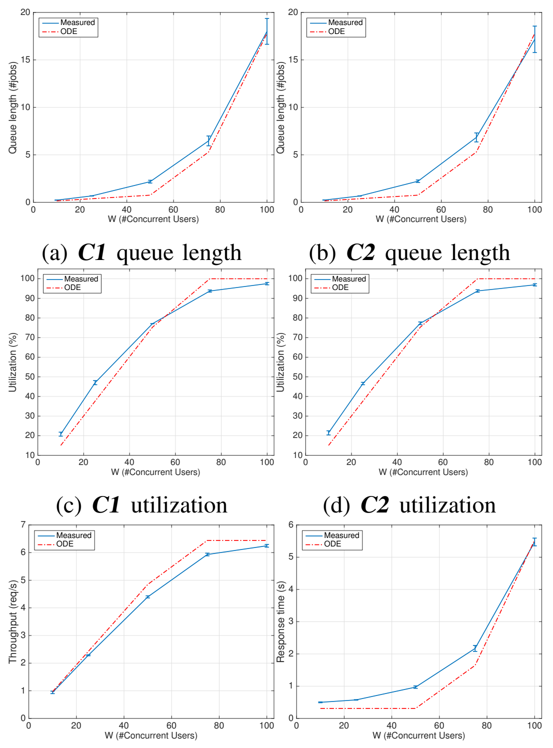
We implemented ***W*** as a multi-threaded Python based workload generator. Each thread runs an independent concurrent user that iteratively accesses the system, interposing an exponentially distributed delay (i.e., the *think time*) between successive requests. Nodes ***LB***, ***C1***, and ***C2*** 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 [46] loop, through operating system signals (see Figure 3). ***CTRL*** ran the MIP optimizations using the well-known CPLEX tool. To this end, we defined a model to text transformation (M2T) suitable to translate the system of a ODEs in a Julia [47] specification that, combined with the JuMp library, provides a common programming interface (API) for different optimization solvers. Finally, we monitored the queue lengths of ***C1*** and ***C2*** using the *netstat* utility.

To facilitate the replicability of our experiments, we deployed all nodes of HAT on a single Linux machine equipped with 24 GB of RAM, 400 GB SSD and 12 Xeon cores at 2.0 GHz. Each HAT node was assigned dedicated CPU cores.

B. Model Parametrization and Validation

The model for this system is the QN in Figure 2. Station N_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 N_1 and N_2 model the processing nodes of HAT. The number of servers s_1 and s_2 represent 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 $\mu_1 = \mu_2 = 1/0.3125 = 3.20$. 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 I.



(a) ***C1*** queue length (b) ***C2*** queue length

(c) ***C1*** utilization (d) ***C2*** utilization

(e) System throughput (f) System response time

Fig. 4: QN model validation

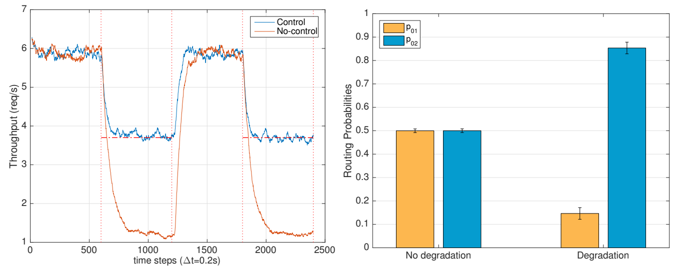
For each value of W we measured the following average indices in steady state: throughput, response time, and utilization of the processing nodes. We verified that a session duration of 10 minutes was sufficient to attain steady state in all cases. Figure 4 depicts the validation results by comparing the steady-state prediction values of the QN model (dashed 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.

C. 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.



(a) System throughput (b) Optimal control signals

Fig. 5: Hardware degradation experiment

1) 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). 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 I (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 III-E, 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 5a 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 5b, 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 MIP returns the balanced configuration as the optimal one.

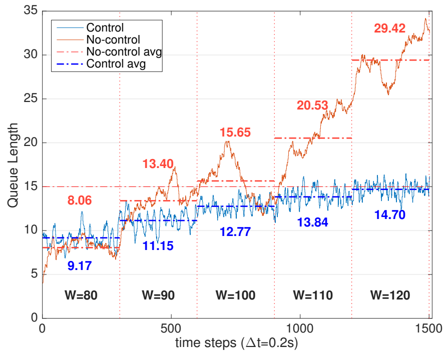
2) Workload fluctuation: We designed this experiment by starting with a workload of $W = 80$ users and introducing 10 new users into the system every minute, until reaching $W = 120$. Here the control objective is to maintain the queue lengths of ***C1*** and ***C2*** at the fixed threshold, i.e., $Q_l = 15$. This represents a critical condition for the balanced single-server deployment, from Figures 4a and 4b 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 6a and 6b 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 suitable to fulfil the requirement $Q_l = 15$, as expected from Figures 4a and 4b. 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 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.

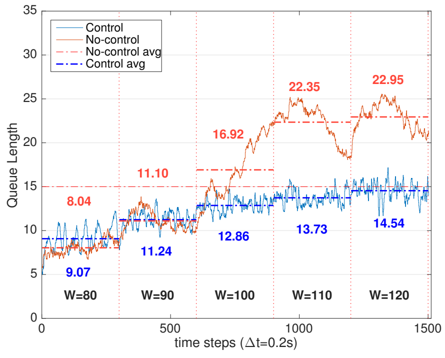
Figure 6c reports the statistics (averages and standard deviations) of the optimal values of the control signals during each level of workload intensity. The average number of servers used increases with W. To reach the target for $W = 120$ the optimal strategy used about 1.5 servers on average. The statistics on the routing probabilities confirm that the average optimal strategy is to direct users to either server equally likely.

D. Scalability Evaluation

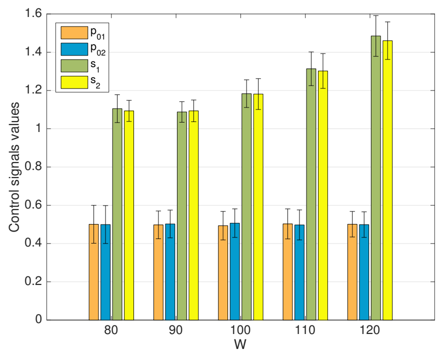
In this section we provide the numerical evidence that the main technical result of the paper, namely the equivalent MIP formulation and correspondence through Theorem 1, appears to be essential for the applicability of MPC to the QN models previously examined. We argue this by reporting considerably 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 Couenne algorithm [48], [49] for solving 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.



(a) ***C1*** queue length



(b) ***C2*** queue length



(c) Statistics of optimal control signals

Fig. 6: Workload fluctuation experiment

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

	Hardware degradation		*Workload fluctuation*	
H	MINLP(s)	MIP(s)	MINLP(s)	MIP(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

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

	MIP	*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

We parameterized the optimization problems for both the adaptation scenarios of Section IV-C 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 II 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 MIP 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, 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).

E. Comparison with Markov Decision Processes

In this section we compare our MIP 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 [35], similarly to what presented in [36]. 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 [50], [51]. Table III 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 MIP 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 MIP formulation is almost unaffected. We remark, however, that MDP is a general approach to self-adaptation, whereas our MIP formulation is specific to QNs (for any topology).

V. Threats to Validity

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 VI). Moreover, as discussed in Section II, many approaches have been proposed to derive QN models from higher-level software designs (e.g., [37], [38], [39]).

In our experiments 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. On-line estimation can be done using: (i) measurement-based approaches that monitor the actual system and update parameters accordingly, e.g., with Kalman filters [52], [53], [54] and Bayesian estimators [55]; (ii) model-based predictions to derive missing parameters [56], [57]; (iii) user defined probability distribution functions [58], [59].

As already discussed in Section IV-D, 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.

VI. Related Work

Research challenges for self-adaptive software systems [60] are provided in [61], [62], [11]. 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 [32] 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 [18] a QN model predictor is used to strengthen the adaptive feedback. Similarly to [32], 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 [32], [18] since it considers transient dynamics and is not restricting to admission control.

In [63], [34] an 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 algorithm, without considering transient dynamics. In [64] 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 [33] 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 VMs 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 MIP one.

Further approaches that make use of control theory formal guarantees are [26], [27] where a closed-loop strategy is proposed to enable an adaptive software system's dynamic behavior. In [26] 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 [27] 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 [26], [27], 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 [65]. Instead, our approach considers the system transient dynamics relying on a model predictive technique with receding control horizon to forecast the system evolution.

VII. Conclusion

In this paper 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.

A technical limitation of our own approach is the single-class assumption in the QN model. We remark however, that for some real-world systems this can be sufficiently expressive, e.g., [17]. Furthermore, the linearization technique proposed in this paper 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., [66]). Instead, in future work we aim to develop similar ideas for other multi-class models such as fluid layered queuing networks [42].

Appendix

In this Appendix we describe how to get our linear representation from the original one of (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 (1) and (11) is ensured by the relations:

$$\begin{aligned} cn_i(k) &= -\mu_i(k)x_i(k)\Delta t \\ -cn_i(k) + cp_{i,j}(k) &= p_{i,j}(k)\mu_i(k)x_i(k)\Delta t \end{aligned} \tag{12}$$

Solving (12) as a function of the variables $\mu_i(k)$ and $p_{j,i}(k)$ gives:

$$\mu_i(k) = -\frac{cn_i(k)}{x_i(k)\Delta t}, \quad p_{i,j}(k) = \frac{cn_i(k) - cp_{i,j}(k)}{cn_i(k)} \tag{13}$$

Substituting these expressions in (5) and (6) we get the following constraints for the new variables:

$$cn_i(k) \geq -x_i(k)\Delta t\overline{\mu}_i \tag{14}$$
$$cn_i(k) \leq -x_i(k)\Delta t\underline{\mu_i} \tag{15}$$
$$cp_{i,j}(k) \geq cn_i(k) \tag{16}$$
$$\sum_{j\in S} cp_{i,j}(k) = cn_i(k)(|S|-1) \tag{17}$$

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

$$\mu_i(k) \leq \overline{\mu}_i \tag{18}$$

we can write:

$$\mu_i(k)x_i(k) \leq \overline{\mu}_i s_i(k) \iff \mu_i(k) \leq \frac{\overline{\mu}_i s_i(k)}{x_i(k)} \tag{19}$$

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 (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 (18) as if the system was linear. Using (13), we are able to translate the constraints (18)-(19) over the original adaptation knobs into linear constraints over the virtual ones, yielding:

$$cn_i(k) \geq -\overline{\mu}_i x_i(k)\Delta t, \quad cn_i(k) \geq -\overline{\mu}_i s_i(k)\Delta t \tag{20}$$

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

$$\mu_i(k) \geq \min\left\{\underline{\mu}_i, \frac{\underline{\mu}_i s_i(k)}{x_i(k)}\right\} \tag{21}$$

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 (12) Equation (21) can be written as

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

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:

$$\begin{aligned} dmin_i(k) &\geq x_i(k) - \alpha d_i(k) \\ dmin_i(k) &\geq s_i(k) - \alpha(1 - d_i(k)) \\ dmin_i(k) &\leq x_i(k) \\ dmin_i(k) &\leq s_i(k) \end{aligned} \tag{22}$$

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 $cn_i(k)$:

$$cn_i(k) \leq -\underline{\mu}_i dmin_i(k)\Delta t \tag{23}$$

We develop an objective function consistent with (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:

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

Similarly to (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 MIP formulation is given by:

$$f := \sum_{k=0}^{H-1} e'(k)^T e'(k) \tag{24}$$

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

$$\begin{aligned} &\text{minimize}_{u'(k)} \quad f && (25)\\ &\qquad u'(k) = \{cn_i(k), cp_{i,j}(k), s_i(k), d_i(k)\} \\ &\text{subject to:} \\ &\qquad \text{Eqs. (11), (16), (17),} \\ &\qquad\qquad \text{(20), (22), (23),} \\ &\qquad s_i(k) = \hat{s}_i, \qquad\qquad s_i(k) \in S - V && (26)\\ &\qquad cn_i(k) = -\hat{\mu}_i x_i(k)\Delta t, \qquad \mu_i \in S - V && (27)\\ &\qquad cp_{i,j}(k) = cn_i(k)(1 - \hat{p}_{i,j}), \qquad p_{i,j} \in S - V && (28)\\ &\qquad\quad \text{for } k = 0, \ldots, H-1, \end{aligned}$$

where with (26)–(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 (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 MIP 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.

References

[1] G. Blair, N. Bencomo, and R. B. France, "Models@runtime," *Computer*, vol. 42, no. 10, pp. 22–27, 2009.

[2] B. Morin, O. Barais, J.-M. Jezequel, F. Fleurey, and A. Solberg, "Models@runtime to support dynamic adaptation," *Computer*, vol. 42, no. 10, pp. 44–51, 2009.

[3] C. Ghezzi, L. S. Pinto, P. Spoletini, and G. Tamburrelli, "Managing non-functional uncertainty via model-driven adaptivity," in *International Conference on Software Engineering (ICSE)*, 2013, pp. 33–42.

[4] N. Esfahani, S. Malek, and K. Razavi, "Guidearch: guiding the exploration of architectural solution space under uncertainty," in *International Conference on Software Engineering (ICSE)*, 2013, pp. 43–52.

[5] R. Calinescu, C. Ghezzi, M. Kwiatkowska, and R. Mirandola, "Self-adaptive software needs quantitative verification at runtime," *Commun. ACM*, vol. 55, no. 9, pp. 69–77, 2012.

[6] D. Sykes, D. Corapi, J. Magee, J. Kramer, A. Russo, and K. Inoue, "Learning revised models for planning in adaptive systems," in *International Conference on Software Engineering (ICSE)*, 2013, pp. 63–71.

[7] B. Chen, X. Peng, Y. Yu, B. Nuseibeh, and W. Zhao, "Self-adaptation through incremental generative model transformations at runtime," in *International Conference on Software Engineering (ICSE)*, 2014, pp. 676–687.

[8] N. D'Ippolito, V. A. Braberman, J. Kramer, J. Magee, D. Sykes, and S. Uchitel, "Hope for the best, prepare for the worst: multi-tier control for adaptive systems," in *International Conference on Software Engineering (ICSE)*, 2014, pp. 688–699.

[9] A. von Rhein, A. Grebhahn, S. Apel, N. Siegmund, D. Beyer, and T. Berger, "Presence-condition simplification in highly configurable systems," in *International Conference on Software Engineering (ICSE)*, 2015, pp. 178–188.

[10] N. Siegmund, A. Grebhahn, S. Apel, and C. Kästner, "Performance-influence models for highly configurable systems," in *Foundations of Software Engineering (ESEC/FSE)*, 2015, pp. 284–294.

[11] R. de Lemos and et al., *Software Engineering for Self-Adaptive Systems: A Second Research Roadmap*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 1–32.

[12] G. Su, D. S. Rosenblum, and G. Tamburrelli, "Reliability of run-time quality-of-service evaluation using parametric model checking," in *International Conference on Software Engineering (ICSE)*, 2016, pp. 73–84.

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

[14] M. Kowal, M. Tschaikowski, M. Tribastone, and I. Schaefer, "Scaling size and parameter spaces in variability-aware software performance models," in *International Conference on Automated Software Engineering (ASE)*, 2015, pp. 407–417.

[15] S. Ben-David, B. Sterin, J. M. Atlee, and S. Beidu, "Symbolic model checking of product-line requirements using sat-based methods," in *International Conference on Software Engineering (ICSE)*, 2015, pp. 189–199.

[16] E. Incerto, M. Tribastone, and C. Trubiani, "Symbolic performance adaptation," in *International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, 2016, pp. 140–150.

[17] B. Urgaonkar, G. Pacifici, P. Shenoy, M. Spreitzer, and A. Tantawi, "An analytical model for multi-tier internet services and its applications," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 33, no. 1, 2005, pp. 291–302.

[18] X. Liu, J. Heo, L. Sha, and X. Zhu, "Adaptive control of multi-tiered web applications using queueing predictor," in *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2006, pp. 106–114.

[19] Y. Diao, J. L. Hellerstein, S. Parekh, H. Shaikh, and M. Surendra, "Controlling quality of service in multi-tier web applications," in *International Conference on Distributed Computing Systems (ICDCS)*, 2006, pp. 25–25.

[20] N. Roy, A. Dubey, A. Gokhale, and L. Dowdy, "A capacity planning process for performance assurance of component-based distributed systems," in *ACM SIGSOFT Software Engineering Notes*, vol. 36, no. 5, 2011, pp. 259–270.

[21] C. M. Woodside, G. Franks, and D. C. Petriu, "The future of software performance engineering," in *Workshop on the Future of Software Engineering (FOSE)*, 2007, pp. 171–187.

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

[23] C. U. Smith, "Software performance engineering then and now: A position paper," in *Workshop on Challenges in Performance Methods for Software Development (WOSP-C)*, 2015, pp. 1–3.

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

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

[26] A. Filieri, H. Hoffmann, and M. Maggio, "Automated design of self-adaptive software with control-theoretical formal guarantees," in *International Conference on Software Engineering (ICSE)*, 2014, pp. 299–310.

[27] D. Arcelli, V. Cortellessa, A. Filieri, and A. Leva, "Control theory for model-based performance-driven software adaptation," in *International Conference on Quality of Software Architectures (QoSA)*, 2015, pp. 11–20.

[28] C. E. García, D. M. Prett, and M. Morari, "Model predictive control: Theory and practice—a survey," *Automatica*, vol. 25, no. 3, pp. 335–348, 1989.

[29] 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*, vol. 6, no. 4, pp. 212–225, 2009.

[30] T. Achterberg and R. Wunderling, "Mixed integer programming: Analyzing 12 years of progress," in *Facets of Combinatorial Optimization*, 2013, pp. 449–481.

[31] L. Sha, X. Liu, Y. Lu, and T. Abdelzaher, "Queueing model based network server performance control," in *Real-Time Systems Symposium (RTSS)*, 2002, pp. 81–90.

[32] A. Robertsson, B. Wittenmark, M. Kihl, and M. Andersson, "Design and evaluation of load control in web server systems," in *American Control Conference*, vol. 3, 2004, pp. 1980–1985.

[33] J. Jiang, J. Lu, G. Zhang, and G. Long, "Optimal cloud resource auto-scaling for web applications," in *International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, 2013, pp. 58–65.

[34] H. Ghanbari, M. Litoiu, P. Pawluk, and C. Barna, "Replica placement in cloud through simple stochastic model predictive control," in *International Conference on Cloud Computing (CLOUD)*, 2014, pp. 80–87.

[35] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of probabilistic real-time systems," in *International Conference on Computer-Aided Verification (CAV)*, 2011, pp. 585–591.

[36] G. A. Moreno, J. Cámara, D. Garlan, and B. Schmerl, "Proactive self-adaptation under uncertainty: A probabilistic model checking approach," in *Joint Meeting on Foundations of Software Engineering (FSE)*, 2015, pp. 1–12.

[37] V. Cortellessa and R. Mirandola, "PRIMA-UML: a performance validation incremental methodology on early uml diagrams," *Science of Computer Programming*, vol. 44, no. 1, pp. 101–129, 2002.

[38] S. Balsamo and M. Marzolla, "Performance evaluation of UML software architectures with multiclass queueing network models," in *International Workshop on Software and Performance (WOSP)*, 2005, pp. 37–42.

[39] C. U. Smith, C. M. Lladó, and R. Puigjaner, "Performance model interchange format (PMIF 2): A comprehensive approach to queueing network model interoperability," *Perform. Eval.*, vol. 67, no. 7, pp. 548–568, 2010.

[40] R. Tawhid and D. Petriu, "Integrating performance analysis in the model driven development of software product lines," in *International Conference on Model Driven Engineering Languages and Systems (MODELS)*, 2008, pp. 490–504.

[41] R. Tawhid and D. C. Petriu, "Automatic derivation of a product performance model from a software product line model," in *International Conference on Software Product Line (SPLC)*, 2011, pp. 80–89.

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

[43] E. Incerto, M. Tribastone, and C. Trubiani, "A proactive approach for runtime self-adaptation based on queueing network fluid analysis," in *International Workshop on Quality-Aware DevOps, (QUDOS)*, 2015, pp. 19–24.

[44] A. N. Tantawi and D. Towsley, "Optimal static load balancing in distributed computer systems," *J. ACM*, vol. 32, no. 2, pp. 445–465, 1985.

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

[46] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[47] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A fresh approach to numerical computing," *SIAM Review*, vol. 59, no. 1, pp. 65–98, 2017.

[48] S. Burer and A. N. Letchford, "Non-convex mixed-integer nonlinear programming: A survey," *Surveys in Operations Research and Management Science*, vol. 17, no. 2, pp. 97–106, 2012.

[49] S. Burer, "On the copositive representation of binary and continuous nonconvex quadratic programs," *Mathematical Programming*, vol. 120, no. 2, pp. 479–495, 2009.

[50] A. Piórkowski, A. Kempny, A. Hajduk, and J. Strzelczyk, "Load balancing for heterogeneous web servers," in *International Conference on Computer Networks (CN)*, 2010, pp. 189–198.

[51] S. Heinzl and C. 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)*, 2013, pp. 342–345.

[52] T. Zheng, C. M. Woodside, and M. Litoiu, "Performance model estimation and tracking using optimal filters," *IEEE Trans. Software Eng.*, vol. 34, no. 3, pp. 391–406, 2008.

[53] T. Zheng, M. Litoiu, and M. Woodside, "Integrated estimation and tracking of performance model parameters with autoregressive trends," in *International Conference on Performance Engineering (ICPE)*, 2011, pp. 157–166.

[54] A. Filieri, L. Grunske, and A. Leva, "Lightweight adaptive filtering for efficient learning and updating of probabilistic models," in *International Conference on Software Engineering (ICSE)*, 2015, pp. 200–211.

[55] I. Epifani, C. Ghezzi, R. Mirandola, and G. Tamburrelli, "Model evolution by run-time parameter adaptation," in *International Conference on Software Engineering (ICSE)*, 2009, pp. 111–121.

[56] D. A. Menascé, "Computing missing service demand parameters for performance models," in *International Computer Measurement Group Conference (CMG)*, 2008, pp. 241–248.

[57] M. Awad and D. A. Menascé, "On the predictive properties of performance models derived through input-output relationships," in *European Workshop on Performance Engineering (EPEW)*, 2014, pp. 89–103.

[58] C. Trubiani, I. Meedeniya, V. Cortellessa, A. Aleti, and L. Grunske, "Model-based performance analysis of software architectures under uncertainty," in *International Conference on Quality of Software Architectures (QoSA)*, 2013, pp. 69–78.

[59] L. Etxeberria, C. Trubiani, V. Cortellessa, and G. Sagardui, "Performance-based selection of software and hardware features under parameter uncertainty," in *International Conference on Quality of Software Architectures (QoSA)*, 2014, pp. 23–32.

[60] J. Kramer and J. Magee, "Self-managed systems: an architectural challenge," in *Workshop on the Future of Software Engineering (FOSE)*, 2007, pp. 259–268.

[61] B. H. C. Cheng and et al., "Software engineering for self-adaptive systems: A research roadmap," in *outcome of a Dagstuhl Seminar*, 2009, pp. 1–26.

[62] J. Andersson, L. Baresi, N. Bencomo, R. de Lemos, A. Gorla, P. Inverardi, and T. Vogel, "Software engineering processes for self-adaptive systems," in *Software Engineering for Self-Adaptive Systems II - Dagstuhl Seminar*, 2010, pp. 51–75.

[63] N. Roy, A. Dubey, and A. Gokhale, "Efficient autoscaling in the cloud using predictive models for workload forecasting," in *International Conference on Cloud Computing (CLOUD)*, 2011, pp. 500–507.

[64] Q. Zhang, M. F. Zhani, S. Zhang, Q. Zhu, R. Boutaba, and J. L. Hellerstein, "Dynamic energy-aware capacity provisioning for cloud computing environments," in *International Conference on Autonomic Computing (ICAC)*, 2012, pp. 145–154.

[65] L. Baresi, S. Guinea, A. Leva, and G. Quattrocchi, "A discrete-time feedback controller for containerized cloud applications," in *International Symposium on Foundations of Software Engineering (FSE)*, 2016, pp. 217–228.

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