Combined Vertical and Horizontal Autoscaling through Model Predictive Control

Emilio Incerto[1], Mirco Tribastone[1], Catia Trubiani[2]

[1] IMT School for Advanced Studies, Piazza San Francesco 19, Lucca, Italy
emilio.incerto@imtlucca.it, mirco.tribastone@imtlucca.it
[2] Gran Sasso Science Institute, Viale Francesco Crispi 7, L'Aquila, Italy
catia.trubiani@gssi.it

Abstract. Meeting performance targets of co-located distributed applications in virtualized environments is a challenging goal. In this context, vertical and horizontal scaling are promising techniques; the former varies the resource sharing on each individual machine, whereas the latter deals with choosing the number of virtual machines employed. State-of-the-art approaches mainly apply vertical and horizontal scaling in an isolated fashion, in particular assuming a fixed and symmetric load balancing across replicas. Unfortunately this may result unsatisfactory when replicas execute in environments with different computational resources.
To overcome this limitation, we propose a novel combined runtime technique to determine the resource sharing quota and the horizontal load balancing policy in order to fulfill performance goals such as response time and throughput of co-located applications. Starting from a performance model as a multi-class queuing network, 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 load balancing adaptation can efficiently achieve desired performance targets in the presence of heterogeneous computational resources.

Keywords: Performance, Queuing Networks, Control, Resource Sharing, Load Balancing

1 Introduction

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 [12]. 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 [20,19]. 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., [11]), must be guaranteed by the computing platform providers [18,17,1].

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 [23]. Unfortunately, state-of-the-art approaches mainly apply vertical and horizontal scaling in an isolated fashion. According to a recent survey on this topic [23], among the 87 surveyed approaches only two have explored optimization techniques to search for combined vertical and horizontal scaling [8,9]. 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 ageing or hardware degradation), since a symmetric load distribution may worsen performance.

To overcome this limitation, we propose a novel technique combining horizontal and vertical scaling that can efficiently determine the load distribution policy to continuously fulfill performance goals of distributed co-located applications (Section 2). We consider a model-based approach using queuing networks (QNs) [3]. In particular, we employ multi-class QNs, where each class represents an application with its own demand on the CPU and specific performance targets. Our analysis is based on a compact, approximate representation of QNs based on ordinary differential equations (ODEs) [16,5]. This avoids the state space explosion problem arising from the exact transient analysis of the Markov chain underlying the QN, thus enabling an effective runtime adaptation.

We formulate the question of finding a combined horizontal and vertical scaling strategy as a Model Predictive Control (MPC) problem [10]. MPC performs runtime optimization which uses the ODE model to predict the future evolution of the system given the currently measured state; 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 [13], but for queuing models with a single class of users only. In this paper we present two significant extensions:

1. A multi-class model that enables an accurate representation of the *capped allocation* paradigm [4]. This is a CPU-sharing mechanism available in most modern off-the-shelf hypervisors (e.g., [2,22]), which defines the maximum share that a VM can receive from the CPU.
2. The specification of latency-based response-time requirements, enriching [13], 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., [6]) as opposed to the mixed-integer program of the single class formulation of [13]. This is due to the fact that in [13] 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 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.

2 Combined Vertical and Horizontal Autoscaling

A Running Example: Figure 1 shows a QN model of a prototypal system on which to perform combined autoscaling. There are two processing nodes represented by the queuing stations N_1 and N_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 N_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. 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.

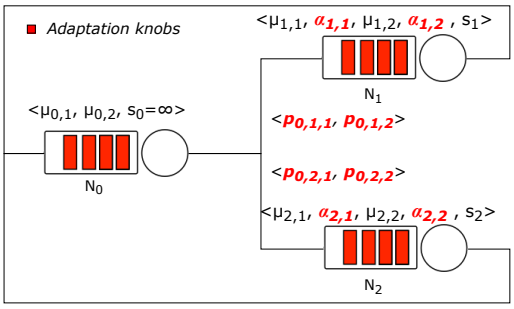


Fig. 1: Sample QN model.

Multi Class Parametric QN: Formally, let us consider a set of stations S and a set of service classes 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 $i \in S, c \in C$;
- $p_{i,j,c} \in P$ 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 $c \in C$;

– $\alpha_{i,c}$ is the processing share assigned to jobs of class c at station i such that $\sum_{c \in C} \alpha_{i,c} \leq 1$ and $\alpha_{i,c} \geq 0$, with $i \in S, c \in C$.

Finally, to formally justify the ODE approximation, the service rates $\mu_{i,c}$ are assumed to be exponentially distributed. However, using [15] our framework can be extended to the nonexponential case with phase-type distributions [3].

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.

ODE Model: The ODE model is systematically derived from the parametric QN and it gives estimate of the average queue lengths $x_{i,c}(t)$ as a function of time. The evolution of the multi-class QN under a cap share resource allocation policy is described by the following ODE system:

$$\frac{dx_{i,c}(t)}{dt} = -\mu_{i,c}(t) \min\{x_{i,c}(t), \alpha_{i,c}(t)s_i(t)\} + \sum_{j \in S} p_{j,i,c}(t)\mu_{j,c}(t) \min\{x_{j,c}(t), \alpha_{j,c}s_j(t)\} \quad (1)$$

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

Here 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 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.

LP Performance Adaptation Formulation: In [13] 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 (MIP). 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:

$$\frac{dx_{i,c}(t)}{dt} = \gamma_{i,c}(t) + \sum_{j \in S}(-\gamma_{j,c}(t) + \zeta_{j,i,c}(t)), \quad i \in S, c \in C \tag{2}$$

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 $p_{j,i,c}$).

We show how (2), augmented with appropriate constraints, can be used for building an LP problem suitable for controlling systems whose performance dynamics can be described by the discrete time version of (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 (2) yield a formulation that reads:

$$x_{i,c}(k+1) = x_{i,c}(k) + \gamma_{i,c}(k) + \sum_{j \in S}(-\gamma_{j,c}(k) + \zeta_{j,i,c}(k)), \quad i \in S, c \in C. \tag{3}$$

Unfolding (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 [13].

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:

$$-\gamma_{i,c}(k) \geq 0, \qquad -\gamma_{i,c}(k) \leq \mu_{i,c} s_i \Delta t \tag{4}$$

$$-\gamma_{i,c}(k) \leq \mu_{i,c} x_{i,c}(k) \Delta t, \qquad -\sum_{c \in C} \frac{\gamma_{i,c}(k)}{\mu_{i,c}} \leq s_i \Delta t \tag{5}$$

with $i \in S, c \in C$. In (4),(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 [13] 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 $k = 0, 1, \ldots, H-1$, in the vector $m(k) = \big(m_1(k), \ldots, m_R(k)\big)$ 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 $\mathcal{R}_{i,c}(k) = \frac{x_{i,c}(k)}{\mathcal{T}_{i,c}(k)} = \frac{x_{i,c}(k)\Delta t}{-\gamma_{i,c}(k)}$. 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

$$\tilde{x}_{i,c}(k) \geq 0 \tag{6}$$

$$\tilde{x}_{i,c}(k) \geq x_{i,c}(k)\Delta t + \beta_{i,c}\gamma_{i,c}(k) \tag{7}$$

$$-\tilde{x}_{i,c}(k) \leq x_{i,c}(k)\Delta t + \beta_{i,c}\gamma_{i,c}(k) \tag{8}$$

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:

$$\text{minimize}_{\{\gamma_{i,c}(k), \zeta_{i,j,c}(k), \tilde{x}_{i,c}(k)\}} \quad \sum_{k=0}^{H-1} e(k)^T e(k) \tag{9}$$

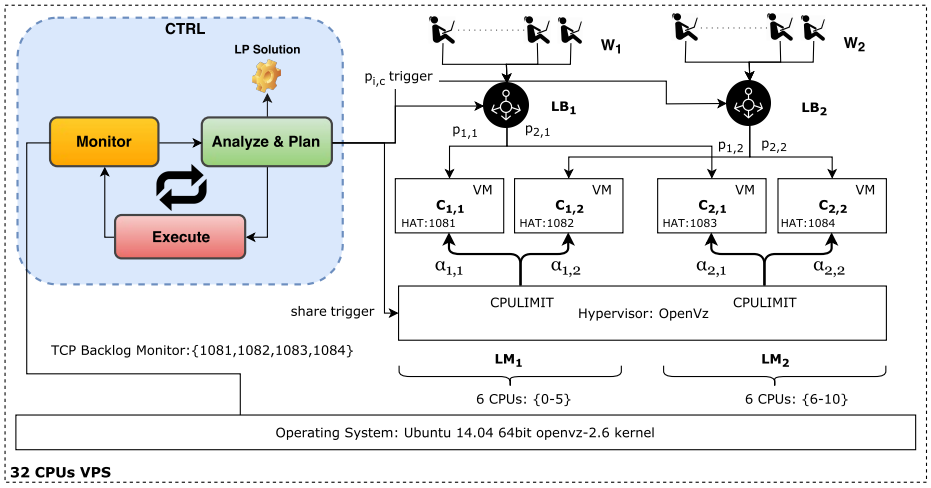


Fig. 2: Experiment system architecture

subject to constraints Eqs. (3), (4), (6), (7), (8)

$$\begin{aligned} \zeta_{i,j,c}(k) &\geq \gamma_{i,c}(k) && \text{if } p_{i,j,c} \in V \\ \sum_{j \in S} \zeta_{i,j,c}(k) &= \gamma_{i,c}(k)(|S|-1), \\ \zeta_{i,j,c}(k) &= \gamma_{i,c}(k)(1-\hat{p}_{i,j,c}) && \text{if } p_{i,j,c} \in P - V \end{aligned} \tag{10}$$

for all $k = 0, \ldots, H-1$, $i \in S$, $c \in C$, where with (10) we set the values for all the parameters of the QN that are fixed.

Following [13], it is possible to define a *nonlinear MPC* formulation built on a discrete-time representation of the model (1). With the following result, we can recover the shares and routing probabilities for the original nonlinear model from the LP formulation above.

Theorem 1 *Denoting by* $\mathcal{S} = \{\alpha^*_{i,c}(k), p^*_{i,j,c}(k)\}$ *an optimal solution to the nonlinear MPC formulation built on (1) based on [13], there exists an MPC problem based on an LP formulation with dynamics (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 $k = 0, \ldots, H-1$.

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`.

System Description and Implementation: For running our evaluation we relied on an in-house developed web application, namely HAT (Heavy Task Processing application), specifically designed for resembling the behavior of CPU-intensive systems [7,21]. 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 [22], 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 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 [14] fashion, through operating system signals (see Figure 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.

Model Parametrization and Validation: We modeled the system under study with a multi-class QN as depicted in Figure 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 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

Table 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

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.

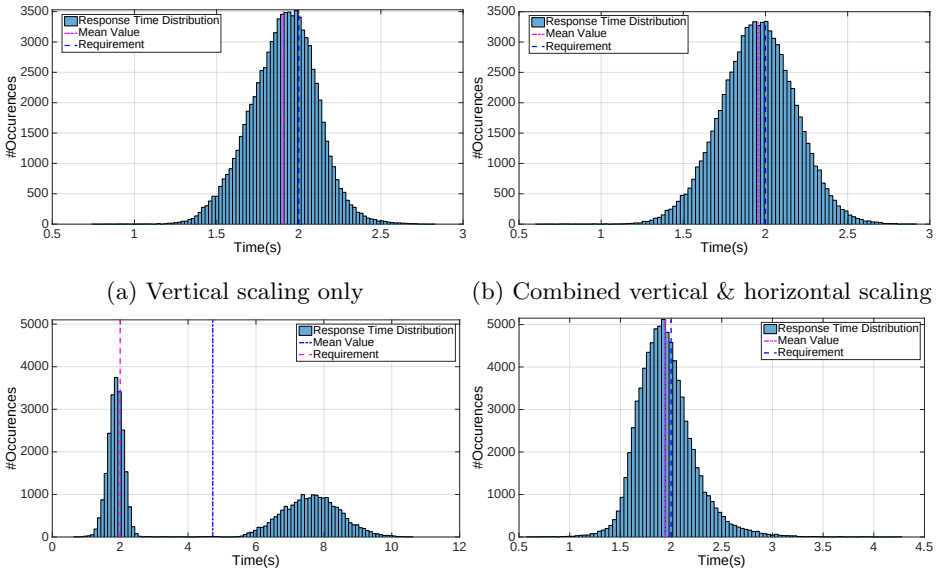
Adaptation Evaluation: We evaluate the effectiveness of our approach by showing that the combined vertical and horizontal load balancing adaptation can efficiently meet performance targets when either of the two techniques alone cannot. We focus on a scenario of *hardware degradation*. Starting from 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 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 3a and 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



(a) Vertical scaling only

(b) Combined vertical & horizontal scaling

(c) Vertical scaling only

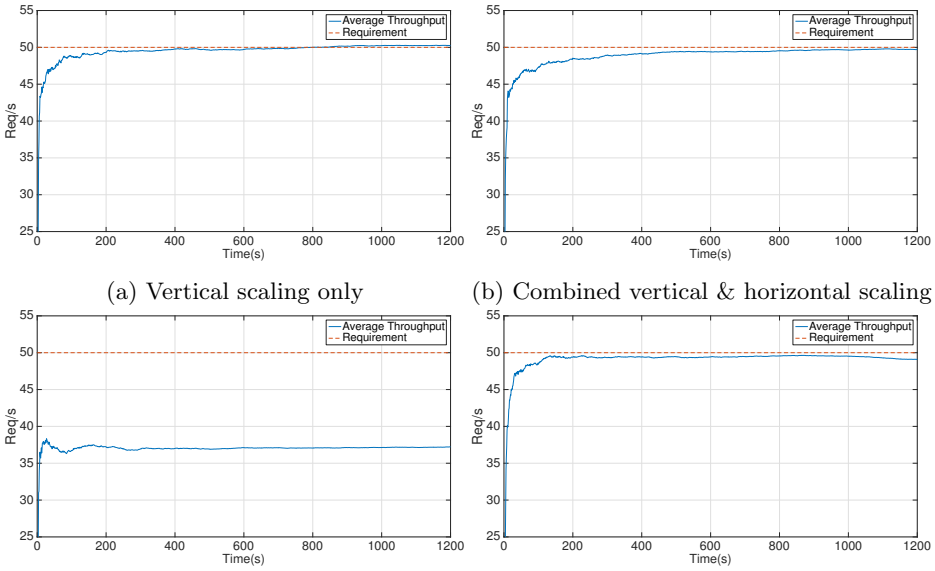
(d) Combined vertical & horizontal scaling

Fig. 3: Class-1 instantaneous average response time distribution without degradation (a-b) and with degradation (c-d).

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 3c depicts the class-1 response time distribution when the vertical scaling is 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 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., [8,9]) 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 4a, 4b). With degradation, only the combined approach is able to steer the system toward the desired set points (see Figures 4c,4d).

Table 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



(a) Vertical scaling only (b) Combined vertical & horizontal scaling

(c) Vertical scaling only (d) Combined vertical & horizontal scaling

Fig. 4: Class-2 instantaneous average throughput without degradation (a-b) and with degradation (c-d).

Table 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

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

4 Conclusion

In this paper we have presented an efficient approach for the performance adaptation of distributed co-located applications using fluid multi class queuing network (QN) and model predictive control (MPC). The main novelties lay in the combined usage of vertical and horizontal load balancing autoscaling techniques and the extension of the fluid model presented in [13] for modeling multiclass distributed systems under a capped resources allocation scheduler. At each time step during system evolution a linear programming problem is solved for comput-

ing the adaptation knobs (i.e., routing probability and allocation shares) suitable to steer 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 [20] and layered queuing networks [24].

Acknowledgement

Mirco Tribastone is partially funded by a DFG Mercator Fellowship (SPP 1593, DAPS2 Project).

References

1. Adam, O., Lee, Y.C., Zomaya, A.Y.: Ctrlcloud: Performance-aware adaptive control for shared resources in clouds. In: International Symposium on Cluster, Cloud and Grid Computing (CCGRID). pp. 110–119 (2017)
2. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. In: ACM SIGOPS operating systems review. vol. 37, pp. 164–177 (2003)
3. Bolch, G., Greiner, S., De Meer, H., Trivedi, K.S.: Queueing networks and Markov chains: modeling and performance evaluation with computer science applications. Wiley (2006)
4. Bolker, E., Ding, Y.: On the performance impact of fair share scheduling. In: International CMG Conference. pp. 71–82 (2000)
5. Bortolussi, L., Hillston, J., Latella, D., Massink, M.: Continuous approximation of collective system behaviour: A tutorial. Performance Evaluation **70**, 317–349 (2013)
6. Boyd, S., Vandenberghe, L.: Convex optimization. Cambridge university press (2004)
7. Corp., I.: Linpack, `https://software.intel.com/en-us/articles/intel-math-kernel-library-linpack-download`
8. Dutta, S., Gera, S., Verma, A., Viswanathan, B.: Smartscale: Automatic application scaling in enterprise clouds. In: International Conference on Cloud Computing (CLOUD). pp. 221–228 (2012)
9. Gandhi, A., Dube, P., Karve, A., Kochut, A., Zhang, L.: Modeling the impact of workload on cloud resource scaling. In: International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD). pp. 310–317 (2014)
10. Garcia, C.E., Prett, D.M., Morari, M.: Model predictive control: Theory and practice—a survey. Automatica **25**, 335–348 (1989)
11. Gupta, D., Cherkasova, L., Gardner, R., Vahdat, A.: Enforcing performance isolation across virtual machines in xen. In: International Conference on Distributed Systems Platforms and Open Distributed Processing (IFIP/USENIX). pp. 342–362 (2006)

12. Huang, D., He, B., Miao, C.: A Survey of Resource Management in Multi-Tier Web Applications. IEEE Communications Surveys & Tutorials **16**, 1574–1590 (2014)
13. Incerto, E., Tribastone, M., Trubiani, C.: Software performance self-adaptation through efficient model predictive control. In: International Conference on Automated Software Engineering (ASE). pp. 485–496 (2017)
14. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. Computer **36**, 41–50 (2003)
15. Kowal, M., Tschaikowski, M., Tribastone, M., Schaefer, I.: Scaling size and parameter spaces in variability-aware software performance models. In: International Conference on Automated Software Engineering (ASE). pp. 407–417 (2015)
16. Kurtz, T.G.: Solutions of ordinary differential equations as limits of pure Markov processes. In: J. Appl. Prob. vol. 7, pp. 49–58 (1970)
17. Lakew, E.B., Klein, C., Hernandez-Rodriguez, F., Elmroth, E.: Performance-based service differentiation in clouds. In: International Symposium on Cluster, Cloud and Grid Computing (CCGRID). pp. 505–514 (2015)
18. Lakew, E.B., Papadopoulos, A.V., Maggio, M., Klein, C., Elmroth, E.: Kpi-agnostic control for fine-grained vertical elasticity. In: International Symposium on Cluster, Cloud and Grid Computing (CCGRID). pp. 589–598 (2017)
19. Li, L., Franks, G.: 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). pp. 1–10 (2009)
20. Molnar, I.: This is the CFS scheduler (1999), `https://www.kernel.org/doc/Documentation/scheduler/sched-design-CFS.txt`
21. NASA: Nas parallel benchmarks, `http://www.nas.nasa.gov/Resources/Software/npb.html`
22. Parallels: OpenVz user guide (2016), `https://docs.openvz.org/openvz_users_guide.webhelp/`
23. Qu, C., Calheiros, R.N., Buyya, R.: Auto-scaling web applications in clouds: A taxonomy and survey. ACM Computing Surveys **9**(4) (2017)
24. Tribastone, M.: A fluid model for layered queueing networks. IEEE Transactions on Software Engineering **39**, 744–756 (2013)