Preprints of the 21st IFAC World Congress (Virtual)
Berlin, Germany, July 12-17, 2020

Control strategies for adaptive resource allocation in cloud computing

Tiago Salviano Calmon [*,**] **Amit Bhaya** [**] **Oumar Diene** [**]
Jonathan Ferreira Passoni [**] **Vinicius Michel Gottin** [*]
Eduardo Vera Sousa [*]

[*] *Dell EMC R&D Center Rio de Janeiro, Rio de Janeiro, RJ 21941-907 Brazil*
[**] *Federal University of Rio de Janeiro, Rio de Janeiro, RJ 21941-901 Brazil*

Abstract: Using a compute infrastructure efficiently to execute jobs while respecting Service Level Agreements (SLAs) and thereby guaranteeing Quality of Service (QoS) poses a number of challenges. One such challenge lies in the fact that SLAs are set prior to the execution of a job, but the execution environment is subject to a number of possible disturbances, such as poor knowledge about actual resource necessity, demand peaks and hardware malfunctions, amongst others. Thus by using a fixed resource allocation, the manager of a shared computing environment risks violating user SLAs. Furthermore, the complexity of managing several workload executions increases with the number of workloads, implying the need for an automatic method to manage and control the execution of workloads. The execution time SLA is specially important in streaming scenarios such as web applications and continuous video processing, and is the focus of this paper. A method based on adaptive model predictive control (aMPC) is proposed here to adapt the amount of allocated resources to iterative workloads. The methodology is tested applied to Deep Learning Workloads, in standalone and multi-workload versions. The results show that using adaptive optimal control with a linearized model improves performance with respect to simpler control laws as well as reinforcement learning approaches.

Keywords: Cloud computing, model predictive control, adaptive control.

1. INTRODUCTION

Over the last few years, Cloud Computing has gained the attention of businesses because of its benefits, which include pay-per-use computation for customers and resource sharing for providers. Through virtualization, the main technology behind clouds, it is possible to abstract a pool of computation devices and offer computational resources better tailored to customer needs, who might contract more computation as their necessities grow.

In such an environment, multiple resource abstractions have emerged, the most prominent example being containers. Through the usage of containers, an infrastructure provider can operate in a different namespace of the operating system. It is also possible to offer computation without the customer needing to know exactly which underlying infrastructure is running his code. This can be achieved in the Platform as a Service (PaaS) paradigm and also the Function as a Service (FaaS) paradigm, which is also known as serverless computing.

In each of these paradigms, the usual agreements upon quality of service (QoS) expected by the customer are expressed through several Service Level Agreements (SLAs). These include bounds on response time, execution time, uptime percentage, among others. The levels of SLAs are usually agreed upon in the contract prior to the service delivery through reference values called SLA metrics, which the provider attempts to respect regardless of conditions that might affect the execution environment. Violating these agreements results in fines for the service provider and also diminishes the trust that a customer attributes to it.

One way to ensure SLAs is to dedicate a fixed amount of resources to them. There are two issues with this approach. Firstly, in general, an application cannot be assumed to be bounded by one particular resource. Some applications, for example, might have an IO-intensive phase and, afterwards, a compute-intensive phase. Dedicating resources to an application in such a scenario is inefficient, resulting in idle resources at the different phases of the application. Secondly, the initial guess the quantity of resources are needed to run an application might be an under- or overestimate. Obtaining an accurate estimate is certainly not a trivial task, especially if the workload is unknown.

In contrast with the hypothesis underlying SLAs, which are contracted prior to the execution of a job, the execution environment is quite dynamic. New workloads might arrive and compete for resources; unplanned demand peaks might occur, disrupting the original workload planning due to tasks with higher priorities, greater need to share the

[⋆] ABs work was partially supported by BPP grant 309625/2011-4 from the Brazilian agency CNPq and grant APQ1-210.509/2016 from the agency FAPERJ of the state of Rio de Janeiro.

environment and overheads might arise because of context switching.

Service providers aim to maximize profit by minimizing resource usage while respecting SLAs. Static resource allocation, which dedicates a fixed amount of resources to a job from start to completion, is clearly sub-optimal.

This paper proposes a methodology to dynamically allocate resources based on feedback of the job execution and prior knowledge of its stages: specifically, the task of controlling iterative workloads, with finite or infinite lengths. Examples of such workloads are the training of most machine learning algorithms, such as training of Deep Neural Networks (DNNs); processing images in a video in real-time, for example, in autonomous vehicles; processing requests in a RESTful API. (Pautasso et al. (2013))

The remainder of this paper is organized as follows. Section 2 presents a brief literature review. Section 3 formulates the optimization problem for both the single workload/single metric problem as well as for the multi-workload/multi-metric problem. An analytical solution to the Single Input Single Output (SISO) problem is provided. Section 4 presents results and compares them to those obtained using an adaptive deadbeat controller, as well as a Reinforcement Learning (RL) solution based on Deep Q Networks. Finally, section 5 presents conclusions and future work.

2. LITERATURE REVIEW

The area of adaptive software has drawn a lot of attention with the advent of Cloud Computing. Efficiently allocating resources is paramount for infrastructure providers, and, in such a market, might make the difference between company success or failure. Indeed, many solutions for this problem have been proposed in different fields. Solutions based on Control Theory, Queuing Theory and Machine Learning based solutions are the methods of choice, for purposes of comparison with the proposed method.

Queuing theory solutions are somewhat similar to those obtained using control theory in terms of using a transfer function between requests and the SLA metrics. However, these transfer functions are derived directly from queuing models, which gives the system less flexibility in terms of demand rates, controls and fine tuning. The actuation possibilities in these kinds of works are usually the number of servers performing request processing and obeying the same distribution of throughput power. Some examples of publications using this approach are Goudarzi and Pedram (2011), Desnoyers et al. (2012).

Machine Learning based approaches such as those in Gambi et al. (2016), Chen et al. (2016) assume that the correct allocation can be found using a trained algorithm. They usually do not take into account the dynamics of the workload but focus instead on characterizing workloads that are better suited for co-allocation and determining the quantity of resources to be assigned to each workload using a static predictor.

Solutions based on control theory model the relationship between resource allocations and performance metrics as a differential or difference equation system. The controller is then synthesized based on the nominal characteristics of the plant and offers some benefits, stability and set-point tracking being the most important and usually found in the solutions.

Liu et al. (2005) proposed a solution using system identification tools to obtain a plant, and then designed a linear feedback controller for the plant, evaluating performance on real data. In their setting, the plant is linear and assumed to be fixed over time, which is a strong assumption for the particular problem of controlling infrastructure since the plant itself is nonlinear, as shown in figure 1. Unmodeled dynamics also affect the performance of a fixed controller adversely.

Angelopoulos et al. (2016) describe the use of model predictive control (MPC) in software adaptation, for an example problem of setting up meetings using software, and the multiple goals associated with it. The authors provide a formal description of MPC and apply it to their example. The proposed solution requires a learning phase to be carried out before the running stage. Also, even with the adaptation provided by the MPC framework, the model parameters are fixed after the training phase.

Farokhi et al. (2016) uses an adaptive controller and focuses on memory control for VMs, whereas our work is more general in terms of resources. Furthermore, Farokhi et al. (2016) also require a warm-up time using default plant settings before starting to adapt to the environment.

In Nathuji et al. (2010), the authors model the system as a Multi-Input Multi-Output (MIMO) system, with inputs being generically defined as the levers available to operators such as the virtual processor capping mechanism. Outputs are also generally defined as SLA metrics of interest. In a sense, it is more of a definition of a framework than a solution. The main contribution is to consider a sum of step functions, defined as Q-States, which, in turn, define utility.

In Kusic et al. (2009), the proposition is to employ a two-level control using Model Predictive Control (MPC). Level zero control assigns a portion of the Virtual Machine (VM) resource to each cluster. The Level 1 controller is tasked with the control of the size of each cluster and the amount of hosts turned on. These controllers work in tandem with different timespans. Level 0 control is used at each iteration, whereas the level 1 controller algorithm is more time consuming and needs to be triggered at each k timesteps, with k being a ratio between control evaluation time and the actual execution time of the workload.

In Shevtsov and Weyns (2016) the authors couple adaptive controllers for multiple different goals with an optimization procedure to decide allocations based on the aforementioned goals with possible trade-offs between them. This work also relies on a learning phase and a subsequent operation phase. The authors do not provide details of the optimization procedure when trade-offs need to be considered.

In Cerf et al. (2016) the authors rely on a MPC control strategy to control Hadoop Clusters. Their goal is to minimize the usage of control actions, and, thus, the MPC formulation is only triggered when the cost of

changing controls is below a predetermined percentage of the current cost. This feature is becoming less important in a container-based execution environment, where costs of reconfiguration in terms of time and orchestration is small. Furthermore, it only considers a single workload with varying load through time.

In Wang et al. (2015), a hierarchical approach consisting of optimization of resources at the cloud level and dynamic fuzzy model predictive control at the host level is proposed. Wang et al. (2015) relies on fuzzy logic for model estimation and genetic algorithm for solving the MPC problem. These approaches, combined, will result in the amount of resources allocated per workload. On top of that, a cross-host resource manager is responsible for moving VMs through hosts if their capacity is exceeded. This approach is interesting, although there are limitations considering the possibly time-consuming fuzzy estimation plus resource allocation using a genetic algorithm. Furthermore, the FMPC formulation does not address resource limits or multiple workloads sharing the same host, leaving these decisions for the cross host resource manager.

3. PROBLEM FORMULATION AND PROPOSED SOLUTION

In a data center, cloud environment, or any other shared execution environment, users must compete for resources in order to run their workloads within acceptable service levels. The infrastructure provider, on the other hand, wants to ensure that a maximum number of users are running their workloads concurrently in order to enhance infrastructure utilization and maximize profits.

In a simple setting with less than a dozen workloads, an operator can simply schedule these jobs to run without concurrency and this will be an acceptable solution. In the ever-growing real data centers and clouds nowadays, the reality is different: hundreds or even thousands of users want to run their workloads with different service levels. The infrastructure manager must deal with this complexity and assign an appropriate amount of resources to each of the users in accordance with their needs. This situation has the following characteristics:

- The management task becomes too complex for a single operator. With multiple human operators, communication issues arise, and inter-connectivity of workloads becomes close to impossible to be properly dealt with, making infrastructure usage inefficient;
- Since the shared execution environment is highly dynamic, it does not suffice to make a good allocation at the beginning of the workload; instead, executions must be tracked and allocations corrected over time;
- The relationship between allocations and service level for every single resource and every single service level metric cannot be assumed to be known *a priori*. Instead, it is an important task to learn the dynamics ruling these relationships as well.

We now formalize the problem in mathematical terms. Let x denote the state or service level, and let $u[k]$ be the amount of a particular resource allocated to the workload at discrete time instant k. As an example, inspection of response times of a neural network w.r.t. allocated CPU amounts leads to the behavior depicted in figure 1. In this picture, the points represent the mean of observed values in 100 executions, in seconds, and the error bars are the standard deviations for each of the limits.

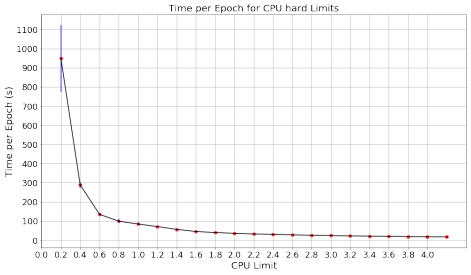

Fig. 1. The relationship of time per epoch and amount of CPU core allocation for a Deep Neural Network Training Workload for 100 executions of epochs using a fixed allocation of resources, which are represented in the x-axis.

In particular, there is a class of workloads that is iterative, i.e., consists of repeated iterations of equal sizes over time. Such workloads are present in stream use cases such as request processing by a RESTful API or real-time surveillance, and also on batch workloads such as training of the majority of machine learning algorithms, optimization procedures, and many others.

This leads us to propose the following non-linear model:

$$u[k] = f(x[k]) \tag{1}$$
$$x[k+1] = g(u[k] + \Delta u[k]) \tag{2}$$

where f and g are assumed to be continuous functions, that are inverses of each other: f mapping service level to resource usage and g vice versa. Using a Taylor series expansion, we write:

$$x[k+1] = g(f(x[k]) + \Delta u[k]) \tag{3}$$
$$x[k+1] = g(f(x[k])) + g'(u[k])\Delta u[k] + O(g'') \tag{4}$$

By neglecting the higher order terms, and noting that $g(f(x(k)) = x(k)$ (since g and f are inverses of each other), we are led to a model in the increment $\Delta u[k]$ of the form:

$$x[k+1] = x[k] + b(x)\Delta u[k] \tag{5}$$

where $b(x)$ denotes $g'(f(x))$. Observe that this approximate model contains the product of a function of the state x and the control Δu and is therefore nonlinear. To use this approximate model in an environment where the dynamics change, the usage of Online Learning algorithms is employed. One of the simplest forms of Online Learning is Recursive Least-Squares (RLS) Kailath et al. (2000). By using a online adaptation algorithm such as RLS, one is able to learn instantaneous values of estimates $\hat{b}(x[k])$ of $b(x)$. The dynamical model to be controlled is then written as:

$$x[k+1] = x[k] + \hat{b}(x[k])\Delta u[k] \tag{6}$$

3.1 Analytical Solution for optimal control of one workload

In this section, we provide a formalization of the optimal control problem to be solved iteratively. This problem can be solved efficiently with several available optimization tools or, for the specific case of a single workload, solved analytically to generate an optimal update rule for the

controller. A convenient formulation is as the following discrete-time optimal control problem with a quadratic objective function which is a weighted sum of the deviations from the SLA summed with the square of the incremental control:

$$\begin{aligned} \min_u J(x,u) &:= w_x(x[k]-r)^2 + w_u(u[k])^2 \qquad (7)\\ s.t.\ x[k+1] &= x[k] + \hat{b}[k]\Delta u[k] \\ u[k] &< U_{\max} \end{aligned}$$

where w_x and w_u are the weights assigned to set-point deviation and control action, respectively.

Defining $w = \frac{w_x}{w_u}$, the compact form of the objective function is:

$$J(x,u) = w(y-r)^2 + u[k]^2 \qquad (8)$$

and appropriate choice of w allows a trade-off between meeting the SLA and resource allocation.

For this simple constrained optimization problem, the solution can be found by calculus plus constraint checking:

$$\frac{\partial J}{\partial u} = -2wb(x)(x - b(x)u - r) + 2u \qquad (9)$$

Setting the right hand side to zero yields u^*:

$$u^* = \frac{wb(x)}{(1+wb(x))^2}(x-r) \qquad (10)$$

Finally, we must check if the local extremum u^* is a minimum, maximum or saddle point, by checking the second derivative of J w.r.t. u

$$\frac{\partial^2 J}{\partial u^2} = 2(wb^2(x)+1) \qquad (11)$$

which is always a positive number for $w > 0$.

3.2 Optimal Control Problem Proposition for Multiple Workloads

After posing the problem of allocating a single workload in section 3.1, we now present an optimal control strategy that solves the problem of allocating multiple workloads in a single machine with a finite amount of resources. In this case, $U_{\max}$ is once again defined as the maximum amount of resources available in the machine. Finally, a new version of the optimal adaptive control problem is posed, for multiple workloads

The value of $\mathbf{u}^*[k]$ to be used at step k is determined by solving the following constrained nonlinear programming problem:

$$\begin{aligned} \min_u \sum_{i=k}^{k+d} \|\mathbf{x}[i] - \mathbf{r}\|^2 + \|\mathbf{u}[\mathbf{i}]\|^2 \qquad (12)\\ \text{subject to } \mathbf{x}[i+1] = \mathbf{x}[i] + \hat{\mathbf{B}}_k \Delta \mathbf{u}[i] \\ \mathbf{1}\cdot\mathbf{u}[i] < U_{\max};\ i \in \{k, \cdots, k+d\} \end{aligned}$$

Using this formulation, we propose to use a control scheme based on Model Predictive Control (MPC) to iterativelly solve this problem for a finite number of steps at each iteration using the current estimated matrix $\hat{\mathbf{B}}_k$.

where $\mathbf{1}$ is a vector of ones of appropriate size and d is the size of the optimization window. The resulting control loop is depicted as a block diagram in figure 2.

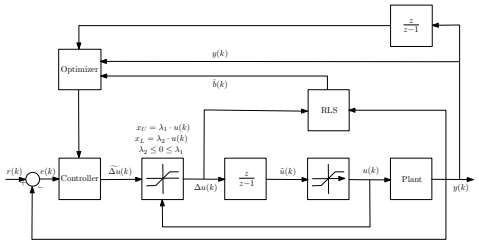

Fig. 2. The block-diagram for the solution for multi-workload control. There are three main components, the control module, the adaptation module, represented by the RLS block and the optimization procedure. The two blocks representing saturation prevent actuations signals from becoming too large and affecting the validity of the approximation in equation (5).

4. RESULTS AND ANALYSIS

In this section we present the architecture used for experimental validation architecture, the results obtained using our methods and baselines usually used in such scenarios.

We begin by describing our testing environment to compare control schemes. We used a Virtual Machine inside a Dell EMC vBlock, and used container-based solution to share resources between applications. The virtual machine used contains 32 virtual CPU cores, each operating at 2.8 GhZ with 32 GB of RAM memory. The virtual machine also had the Ubuntu Operating System version 16.04 installed. Although we used a Virtual Machine on an enterprise-level array, all experiments reported in this paper can be executed in a common desktop computer or laptop. The container-based solution used was Docker, which allows the manager to switch allocations by just sending command-line instructions, which can be embedded in the code. Another interesting feature of Docker is that it allows for fractional number of CPU cores to be assigned to each task, since it does time-sharing of the pool of available CPU cores. Finally, it is noteworthy that, while Docker needs a time window to adapt to the new allocation, this time window is usually much smaller than 1 epoch of training of a Neural Network. The overall architecture is detailed in figure 3.

First, we test the controllers against a single workload with unknown resource usage. Justification for the usage of control is also provided by an execution with disturbance. The second test is to control multiple workloads at once, using again the different control methodologies. We use the adaptive Model Predictive Control methodology proposed in this paper against a fixed feedback controller.

4.1 Standalone Test

In this test, a Deep Neural Network, as described in table 1 to predict handwritten digits from the MNIST dataset LeCun et al. (2010) was trained alone. There are two sets of tests in this section, one in which we just want to

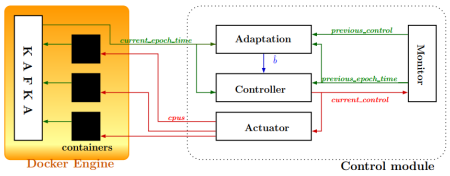

Fig. 3. The testing environment used to test all three methodologies. The workloads are executed as docker containers, and, at each iteration, they report their metrics to kafka, a high-performance framework for real time streaming messages. These metrics are captured by the monitor module, which shares it to other components of the control module, namely the adaptation module, and the controller. After a control decision is formulated, the actuator applies it in the docker daemon.

Layer	# of Neurons	# of Inputs
Dense 1 + ReLU	512	401920
Dense 2	10	5130
Softmax	10	10

Table 1. Description of Tensorflow's sample Deep Neural Network to predict handwritten digits from the MNIST dataset LeCun et al. (2010).

regulate to their set-points without any disturbances and another one with the same goal but with another unknown workload starting at the middle of the execution of the controlled workload.

In both executions, we are able to see in figure 4 that the controller is able to track the set-points, even though for higher values of the weighting parameter w (defined in 3.1) oscillatory behavior is observable. This happens due to the fact that w is being used to scale the importance of set-point deviation in comparison to control usage. It is natural that, for higher values of w, more aggressive actions are chosen. This leads to oscillatory behavior in some cases, as depicted in the choice of $w = 3.33$.

Another experiment done was to validate the necessity of a controller, instead of using a nonlinear regression technique only at the beginning. First, the non-linear regression assumes some knowledge of the workload, whereas the controller does not. Second, and most important, having a reasonable estimate at the beginning will not help in the case delineated in figure 5, in which a concurrent workload (in this experiment, just a smaller execution of the same training procedure of a Neural Network to recognize handwritten digits) is added in the middle of the execution of the controlled workload.

The proposed analytical optimal control methodology is compared with a discrete action Deep Neural Network trained using Q-Learning theory, whose allowable actions are described in equation (13), using $\epsilon = 0.1$ and $K_{MAX} = 5$ and an adaptive deadbeat controller, whose control law is described in equation (14). We also couple the deadbeat controller to a saturation in order to not allow it to deviate from the linear zone corresponding to the instantaneous linearization found with the RLS block.

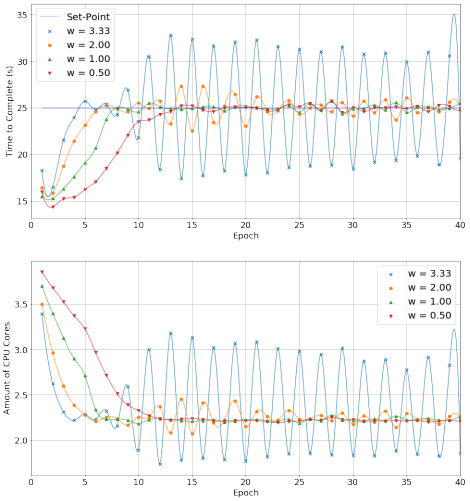

Fig. 4. Using the analytical optimal feedback law to control a single workload with varying values of w. Oscillatory behavior appears with lower values, and settling time increases in correspondence to the magnitude of the weighting parameter w (defined in 3.1).

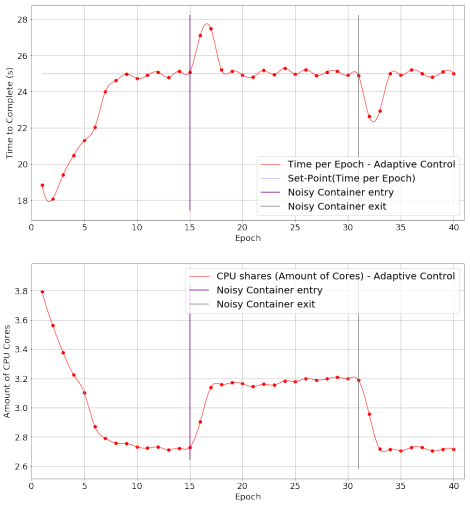

Fig. 5. Using the analytical optimal feedback law to control a single workload and $w = 1$, adding a noisy container in the middle of the execution. The controller is able to increase the resources used by the container in order to not suffer performance issues.

$$a_i = \epsilon \cdot 2^{\pm k}; k \in \{0, 1, \cdots \ K_{MAX}\} \tag{13}$$

$$u[k] = \frac{1}{\hat{b}_k} \cdot e[k] \tag{14}$$

The comparison between the methodologies is performed using the Sum of Time Absolute Error (STAE), which is the discrete version of the widely used Integral of Time Absolute Error (ITAE). This criterion can be calculated according to equation (15).

$$STAE = \sum_{k=k_0}^{k_f} = k \cdot \|(y[k] - r[k])\|_2 \tag{15}$$

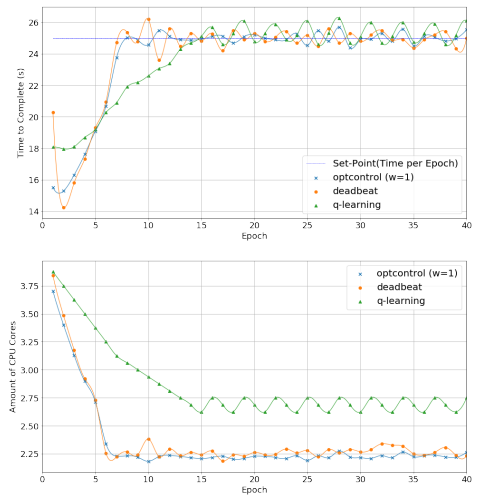

Fig. 6. Comparing analytical optimal solution with q-learning and deadbeat controller approaches. Clearly, the approach using Q-Learning takes a longer time to settle, whereas the deadbeat controller and the analytical optimal have similar performance.

Methodology	STAE
OptControl	$\mathbf{351.43 \pm 25.55}$
Deadbeat	409.30 ± 32.21
DeepRL-Q	653.10 ± 77.44

Table 2. Comparison between each of the methodologies' performance in terms of STAE for five runs.

Table 2 shows a comparison between different methodologies using the sample workload of training a Deep Neural Network to predict handwritten digits from MNIST with 40 epochs and using TensorFlow (Abadi et al. (2015)). At the end of each epoch, which is considered the iteration in this particular test, the time is measured and the controller decides what kind of adaptation it will perform on the number of CPU cores.

4.2 Multi-Workload test

The adaptive MPC controller was also tested for multiple workloads interfering with each other. In the multi workload test, we set two DNN trainings to run in parallel and assign a time per epoch set-point to each of them. Each of these workloads are executed using the set-point as 25s and with starting allocation of 4 full CPU cores. By performing this experiment, the goal is to check if the proposed methodologies are robust to work with multiple workloads. Details of both executions can be found in figure 7, in which both workloads track the set-points determined to them after a few epochs.

5. CONCLUSION AND FUTURE WORK

In modern data centers and other shared computational environments such as clouds and edge computing, it is increasingly important to abstract away the management of workloads due to the increasing complexity of the environment. In such a scenario, it is important to automatize SLA compliance for workloads in order to enhance profitability and avoid penalties, fines and dissatisfaction of the users.

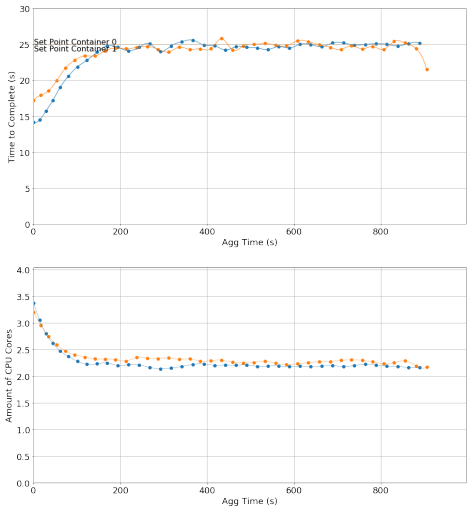

Fig. 7. Multi workload test. Both Neural Networks were trained at the same time and machine. At each iteration, their metrics was sent to a solver in a different machine, that would process the data and optimize for 10 steps. The next allocation was chosen and the controller would wait another epoch before optimizing again.

In this work, a method for automatically controlling iterative workloads was presented. This method relies on system approximation and online identification of a coupling matrix between SLA metrics and the amount of resources dedicated to a workload. We further evaluated this method against two other methodologies, a deadbeat controller based on the same adaptation engine and a Deep-Q network that uses a Deep Neural Network as an approximator for its reward function, which is just the absolute error.

Finally, this method was also evaluated when more than one workload is running in the same environment, with results similar to those of the single workload experiment.

For future work, it would be interesting to inspect the combined behavior of limiting multiple resources for a workload and evaluate the best kinds of models for them. Another interesting direction is to further explore Reinforcement Learning algorithms, in special actor-critic setting and other continuous action variants. Finally, applying the same methodologies here proposed to different problems such as online selecting the correct batch size for training a neural network within a time constraint is also interesting.

REFERENCES

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heteroge-

neous systems. URL `http://tensorflow.org/`. Software available from tensorflow.org.

Angelopoulos, K., Papadopoulos, A.V., Silva Souza, V.E., and Mylopoulos, J. (2016). Model predictive control for software systems with CobRA. In *Proceedings - 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2016*, 35–46.

Cerf, S., Berekmeri, M., Robu, B., Marchand, N., and Bouchenak, S. (2016). Cost function based event triggered model predictive controllers application to big data cloud services. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, 1657–1662. IEEE.

Chen, M., Huang, S., Fu, X., Liu, X., and He, J. (2016). Statistical Model Checking-Based Evaluation and Optimization for Cloud Workflow Resource Allocation. *IEEE Transactions on Cloud Computing*.

Desnoyers, P., Wood, T., Shenoy, P., Singh, R., Patil, S., and Vin, H. (2012). Modellus: Automated modeling of complex internet data center applications. *ACM Transactions on the Web*, 6(2).

Farokhi, S., Jamshidi, P., Bayuh Lakew, E., Brandic, I., and Elmroth, E. (2016). A hybrid cloud controller for vertical memory elasticity: A control-theoretic approach. *Future Generation Computer Systems*, 65, 57–72.

Gambi, A., Pezzè, M., and Toffetti, G. (2016). Kriging-Based Self-Adaptive Cloud Controllers. *IEEE Transactions on Services Computing*, 9(3), 368–381.

Goudarzi, H. and Pedram, M. (2011). Multi-dimensional SLA-based resource allocation for multi-tier cloud computing systems. In *Proceedings - 2011 IEEE 4th International Conference on Cloud Computing, CLOUD 2011*, 324–331.

Kailath, T., Sayed, A.H., and Hassibi, B. (2000). *Linear estimation*. Prentice Hall.

Kusic, D., Kephart, J.O., Hanson, J.E., Kandasamy, N., and Jiang, G. (2009). Power and performance management of virtualized computing environments via lookahead control. *Cluster Computing*, 12(1), 1–15.

LeCun, Y., Cortes, C., and Burges, C. (2010). Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, 2.

Liu, X., Zhu, X., Singhal, S., and Arlitt, M. (2005). Adaptive entitlement control of resource containers on shared servers. *2005 9th IFIP/IEEE International Symposium on Integrated Network Management, IM 2005*, 2005, 163–176.

Nathuji, R., Kansal, A., and Ghaffarkhah, A. (2010). Q-clouds: Managing performance interference effects for QoS-aware clouds. *EuroSys'10 - Proceedings of the EuroSys 2010 Conference*, 237–250.

Pautasso, C., Wilde, E., and Alarcon, R. (2013). *REST: advanced research topics and practical applications*. Springer.

Shevtsov, S. and Weyns, D. (2016). Keep It SIMPLEX: Satisfying multiple goals with guarantees in control-based self-Adaptive systems. *Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, 13-18-Nove, 229–241.

Wang, L., Xu, J., Duran-Limon, H.A., and Zhao, M. (2015). Qos-driven cloud resource management through fuzzy model predictive control. In *2015 IEEE International Conference on Autonomic Computing*, 81–90. IEEE.