Model-free control for resource harvesting in computing grids

Quentin Guilloteau, Bogdan Robu, Cédric Join, Michel Fliess, Éric Rutten, Olivier Richard

Model-Free Control for Resource Harvesting in Computing Grids

Quentin Guilloteau[1], Bogdan Robu[2], Cédric Join[3,5], Michel Fliess[4,5], Eric Rutten[1] and Olivier Richard[1]

***Abstract*— Cloud and High-Performance Computing (HPC) systems are increasingly facing issues of dynamic variability, in particular w.r.t. performance and power consumption. They are becoming less predictable, and therefore demand more runtime management by feedback loops. In this work, we describe results addressing autonomic administration in HPC systems through a control theoretical approach. We more specifically consider the need for controllers that can adapt to variations a long timee in the behavior of controlled systems, but also to being reused on different systems and processors. We therefore explore the application of Model-Free Control (MFC) in the context of resource harvesting in a Computing Grid, by regulating the injection of flexible jobs while limiting perturbation of the prioritary applications.**

Keywords: Model-Free Control, Control for Computing, Resource Harvesting

I. Introduction

A. Need of Control Theory for High Performance Computing

Using Control Theory for computing systems is a relatively recent approach, for which the motivations detailed elsewhere [1] are summarized here to introduce their need. Scientists from many fields have to run experiments or analysis requiring some kind of computations. Such computations, also called job in High-Performance Computing (HPC), due to the quantity of read/write operations, amount of required memory, processor speed, etc. need to be submitted to super computers with a high level of parallelism. These Computing Grids or data-centers, used for Cloud and HPC, are increasingly facing issues of dynamic variability, in particular w.r.t. performance and power consumption. For example, variability can come from computing architectures, where processors, even of the very same model, show variability in speed or thermal behavior. Clusters of such processors can be constructed following different architectures and organizations, involving memory benches, and cache systems. Variability also comes at run-time, due to phases involving more memory accesses, read/write or input/outputs (*I/O*), slowing the computation, whereas phases with less of them have a higher speed ; it can also be due to temperature of processors influencing their speed.

Hence, HPC systems are becoming less predictable, and therefore demand more run-time management by feedback loops. In Computer Science and Software Engineering, Autonomic Computing [2] is addressing this concern, proposing software architectures with feedback loops involving a variety of decision mechanisms (*e.g.,* programmatic, based on AI and Machine Learning, scheduling, constraint programming). A particularly interesting approach involves Control Theory [3]–[5] with applications to HPC *e.g.,* [6]–[8].

In our work, we more specifically consider the need for controllers that can adapt to variations a long time in the behavior of controlled systems, but also to being reused on different systems and processors.

In consideration of our application domain of Computing Systems, we also face a need for solutions with a particular simplicity of implementation and operation, because HPC system administrators are experts in Computer Science, but typically not of Control Theory: indeed, for historical and academic organization reasons, the curricula and scientific cultures are very separate between these two domains.

B. Applying Model-Free Control

These two needs of adaptivity and simplicity of operation motivate us to consider applying Model-Free Control (MFC) [9], [10]:

- This data-driven setting is easy to implement.
- It has been successfully applied in many concrete applications, like energy management (see, *e.g.,* [11]), car driving (see, *e.g.,* [12]) or electro-active actuators (see, e.g., [13]).
- Several illustrations in computer science have already been published: improving resource elasticity in cloud computing [14], service in the Internet of Things [15], and cybersecurity [16].

C. Our Control Problem: Resource Harvesting in HPC

Supercomputers are very expensive, therefore they are shared between many users from research groups and laboratories. A reservation process is used to access the machines and execute the jobs. Users submit their computations, along the estimate duration and the required number of machines, to the scheduler. The latter is responsible to map the jobs to the physical machines. However, the reservation system also leads to the idleness of some machines, for different reasons, *e.g.,* a lack of demand from the users, representing a non-negligible loss of computing power.

[1] Univ. Grenoble Alpes, Inria, CNRS, LIG, F-38000 Grenoble France
Firstname.Lastname@inria.fr
[2] Univ. Grenoble Alpes, CNRS, Grenoble INP, GIPSA-lab, 38000 Grenoble France
Bogdan.Robu@univ-grenoble-alpes.fr
[3] CRAN (CNRS, UMR 7039), Université de Lorraine, 54506 Vandoeuvre-lès-Nancy, France
Cedric.Join@univ-lorraine.fr
[4] LIX (CNRS, UMR 7161), École polytechnique, 91128 Palaiseau, France
Michel.Fliess@polytechnique.edu
[5] AL.I.E.N., 7 rue Maurice Barrès, 54330 Vézelise, France
{cedric.join, michel.fliess}@alien-sas.com

Therefore researchers have been looking at the problem of harvesting those idle resources [17]–[19]. Resources are harvested by submitting jobs that are more flexible (smaller, interruptible) and viewed as second class citizens. However, they still impact the shared resources of the cluster (e.g. file-system, communication network), thus inevitably perturbing the jobs of the premium users. There is a trade-off to exploit between the amount of harvesting and the maximum perturbation that these users can accept.

Our control problem is therefore to explore the application of MFC in the context of resource harvesting in a Computing Grid, by regulating the injection of flexible jobs while limiting perturbation of the priority applications.

D. Contributions of this paper

Within the above context, the work presented in this paper presents contributions in:

- a formulation of the problem in terms of MFC
- an analysis of the parameterization and method
- an experimental evaluation.

II. Background

A. MFC : the ultra-local model and intelligent controllers

We summarize from [9] the main ideas and building blocks of Model-Free control.

For simplicity's sake, let us restrict ourselves to single-input single-output (SISO) systems. The unknown global description of the plant is replaced by the following first-order *ultra-local model*:

$$\dot{y}(t) = F(t) + \alpha u(t) \tag{1}$$

where:

1) the control and output variables are respectively $u(t)$ and $y(t)$, which are of course time-dependent.
2) the constant $\alpha \in \mathbb{R}$ is chosen by the practitioner such that the three terms in Equation (1) are of the same magnitude.

The following comments are useful:

- $F(t)$ is *data driven*: it is given by the past values of $u(\tau)$ and $y(\tau)$, $\tau \leq t$.
- $F(t)$ includes not only the unknown structure of the system but also any disturbance.

Close the loop with the *intelligent proportional controller*, or *iP*,

$$u(t) = -\frac{\hat{F}(t) - \dot{y}^\star(t) + K_P e(t)}{\alpha} \tag{2}$$

where

- $y^\star(t)$ is the reference trajectory,
- $e(t) = y(t) - y^\star(t)$ is the tracking error,
- $\hat{F}(t)$ is an estimated value of $F(t)$,
- the constant $K_P \in \mathbb{R}$ is the proportional gain.

Equations (1) and (2) yield

$$\dot{e}(t) + K_P e(t) = F(t) - \hat{F}(t) \tag{3}$$

If the estimation $\hat{F}$ is "good" *i.e.,* $F - \hat{F} \simeq 0$, then $\lim_{t \to +\infty} e(t) \simeq 0$ if $K_P > 0$. It implies that the tuning of K_P is quite straightforward. This is a major benefit when compared to the tuning of classic PID control (see, *e.g.*, [20]).

The computation of $\hat{F}$, can be done with different techniques as presented in [9]. However, as a first approach, we can also compute $\hat{F}$ using Equation (1) as

$$\hat{F}(t_k) = \dot{s}(t_k) - \alpha u(t_{k-1}) \tag{4}$$

where t_k is the current instant and $\dot{s}(t_k)$ the filtered derivative of $\dot{y}$.[1] For sake of simplicity, from now on we will write y_k or s_k instead of $y(t_k)$ or $s(t_k)$.

B. Background on the HPC system

1) The CiGri Middleware: The computing resource harvesting we consider (see Section I-C) takes places in the framework of the *CiGri* architecture depicted in Figure 1. The injected jobs come from *Bag-of-Tasks* (BoT) applications, composed of many small independent parametric tasks that can thus be executed in parallel. Some examples of BoT applications are Monte-Carlo applications where the user has to execute thousands of small independents tasks to produce statistical results from the results of all the tasks. Other examples can be jobs from the Big Data field, or parameter sweep applications.

An important issue is linked to the potential overloading of the file-server, due to too many *I/O* operations (*i.e.,* reading/writing files), which risk to disturb the high priority users of the clusters.

In the general architecture of the system the computing grid is composed of clusters, grouping compute nodes resources, connected by a network to a storage server, also called file-server. *OAR* [21] servers are managing computations on the clusters, scheduling tasks coming mainly from priority users, and also from *CiGri*. In this paper we only focus on the case of a single cluster.

CiGri jobs are viewed by *OAR* as *Best-effort* jobs *i.e.,* with the lowest priority on the cluster. If a priority user of the cluster needs the resource where a *Best-effort* job is running, then the latter will be stopped and the priority user will get the resource for her job. *Best-effort* jobs will only get scheduled on idle cluster nodes, as opposed to higher priority jobs that can interrupt lower priority jobs in order to run on a specific resource.

In its original version, *CiGri* uses a very simple algorithm every 30 seconds, to decide of the quantity of jobs to submit to *OAR*, and then will wait until all these jobs have finished executing to submit a new subset of jobs. This current solution has the drawback that it leads to an under-utilization of the cluster, by waiting for the injected jobs to terminate before the next injection, and also by not taking into account the current state of the system (*e.g.,* number of idle resources, load of the file-system).

A significant aspect of this load is related to the fact that every *HPC* job either writes a file (*e.g.,* saving results), reads a file (*e.g.,* importing configuration) or both. Usually, in a cluster, there is one, or several, dedicated machines by cluster

[1] See Section III-B for details on the implementation.

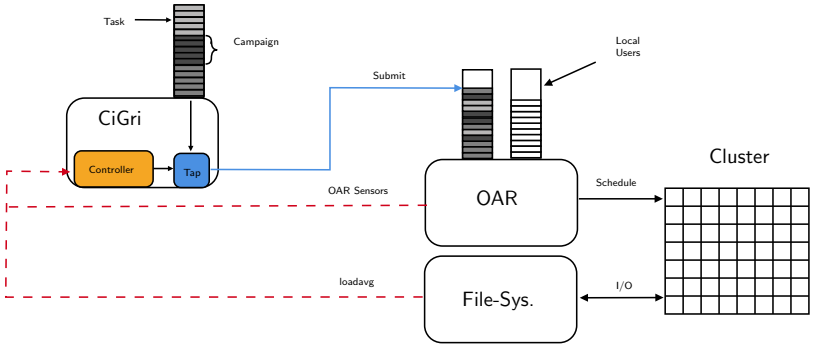

Fig. 1. Architecture of the *CiGri* System

for the storage of the users data. The users then access the data on this server using mechanisms such as NFS (Network FileSystem). However, the performances of this server are limited, and too many write/read requests can lead to an overload and thus to a drop in performance, affecting the priority users of the cluster by slowing them down. (*i.e.,* increase in read/write times).

As we can see, the current behavior of the *CiGri Middleware* does not take into account the dynamic behavior of the grid (sudden presence or absence of machines or higher priority users) as well as the file-server load. The harvesting of the idle resources by *CiGri* must be done dynamically by taking into account the current state of the system: load of the file-server and number of idle resources. Moreover, it should be easily deployable in the production environment and independent of the number of resources and job size. For these reasons we decide to use tools from Control Theory to tackle this issue.

2) CiGri in a Control Theory framework: The goal of the controller is to **avoid any file-server overload** while **minimizing the under-utilization of the cluster** by submitting tasks from a *Bag-of-Tasks* application. Thus, by controlling the load of the file-server to a constant value, we could make sure that the latter is not overloaded by *CiGri* and thus is not the bottleneck of some high priority user applications.

For sensing the load of the filesystem, we chose the UNIX `/proc/loadavg` sensor. This sensor is well known in system administration. A rule of thumb is that a system is overloaded if the value returned by `loadavg` is greater than the number of threads in the system. This metric has the particularity of having some unknown inertia depending on the number of threads.

The way of controlling the system (the `actuator`) is by adjusting the number of jobs that *CiGri* can submit to *OAR* at each cycle.

The number of jobs submitted by the premium users of the cluster plays the role of an external signal on which we do not have any influence (*i.e.,* perturbation).

III. Model-Free Controller Design

A. Applying MFC to our problem

Administrators of HPC systems are not control theory experts. This is why we want to benefit from the simplicity of operation of the model-free approach, as well as from its performance.

In [9], the authors present multiple *intelligent* controllers (intelligent PIDs and their variations). In this work, as a first approach, we decided to use an intelligent Proportional controller (*iP*) due to its simplicity of implementation and the fact that it leads to a zero steady state error.

a) Inputs: In our system, we have a single input signal which is the number of jobs submitted by *CiGri* to *OAR* at each submission cycle. We note $u = \#jobs_{CiGri}$.

b) Outputs: In this work we are interested in the regulation of the fileserver load. We measure the UNIX metric y = `loadavg` to sense this load. The machine hosting the filesystem does also non *I/O* related work, (*e.g.,* network requests). As the `loadavg` metric measured the CPU activity, it also captures the non *I/O* behavior, thus introducing perturbation in the sensing of the *I/O* load. This is why we need to filter the `loadavg` sensor output to get the *I/O* load of the filesystem.

c) Control Objective: Our objective is to harvest the idle resources of a cluster of machines without perturbing the premium users of the cluster and guarantee a *Quality-of-Service*. The *Quality-of-Service* can be expressed as the the inverse of load of the filesystem (y): the higher the load the more perturbed the *I/O* of the users as it will take longer to execute the *I/O* request.

B. The need to filter the sensor values

As we see from Equation (4) the approximation of the system F needs the values of y and it's temporal derivative $\dot{y}$. As computing the derivative of a noisy signal can result in an even noisier signal, we decided to use an exponential moving average filter to smooth the output signal. The filter has the following form:

$$s_k = \beta s_{k-1} + (1-\beta) y_k \tag{5}$$

where s_k is the filtered (smoothed) output at iteration k, y_k the real (noisy) output and $\beta \in [0, 1]$ is the smoothing factor.

Figure 2 depicts the variations of the output signal derivative when smoothed (top) as well as the reaction delay induced by the introduction of the smoothing (shaded area on bottom graph). We can see that the greater β, the smoother

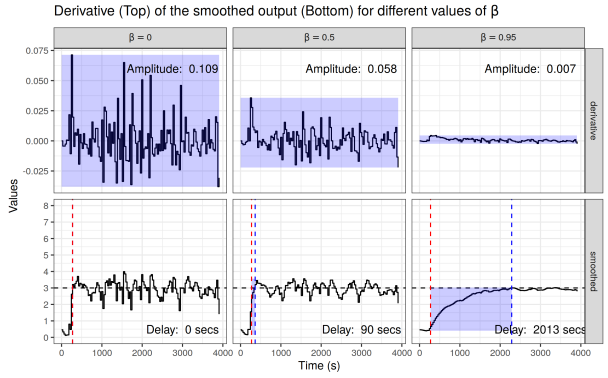

Fig. 2. Smoothed output, its derivative and the induced delay for different smoothing factor (β).

is the output signal (bottom) as well as the derivative (top). The smoother the derivative, the better the estimation of F. However, we can also see that increasing β leads to some delay in response, which will slow down the reaction time of the closed loop system. There is thus a tradeoff between smoothness and response time. A discussion about the choice of β is given below in Subsection III-D.2.

C. The Control Law

CiGri works in a discrete fashion by submitting jobs periodically, every $\Delta t = 30sec$. We thus compute the control law in a discrete manner. As described in Section II-A, the control law at each iteration k is defined as follows:

$$\begin{cases} \hat{F}_k & = \dfrac{s_k - s_{k-1}}{\Delta t} - \alpha u_k \\ u_{k+1} & = -\dfrac{\hat{F}_k - \dot{y}^\star_k + K_p \times (s_k - y^\star_k)}{\alpha} \end{cases} \tag{6}$$

where s_k is the smoothed value of the output y_k, u_k is the control value, $\hat{F}_k$ is the estimation of the plant model, $y^\star_k$ is the reference value and $\dot{y}^\star_k$ the derivative of the reference value. If we want for example a constant *Quality-of-Service* for the file-server, we then take $y^\star_k$ constant and its derivative ($\dot{y}^\star_k$) is null.

D. Choice of the Parameters

The model-free approach requires us to set up a few parameters:

1) α: In [9], the authors recommend taking α such that $\dot{y}$ and $\alpha \times u$ have the same order of magnitude.

From Figure 2, we see that an estimate for the order of magnitude of the derivative $\dot{y}$ is approximately $0.1 \times (1-\beta)$, where 0.1 is the amplitude for the non filtered output (*i.e.,* $\beta = 0$). Let us thus set:

$$\begin{cases} A_0 & \simeq 0.1 \\ A_\beta & \simeq A_0 \times (1-\beta) \end{cases} \tag{7}$$

Hence, as $u \in [0, r_{\max}]$, where $r_{\max}$ is the total number of resources available in the cluster, we find from Equation (6) that for $\dot{y}$ and $\alpha \times u$ to have the same order of magnitude we need:

$$\alpha \simeq \frac{A_\beta}{r_{\max}} \tag{8}$$

2) Smoothing factor: The smoothing factor β is used in filtering the derivative, as we see in Equation (5).

Furthermore, we try to find a relation between the value of β and the reaction delay it introduces.

If we consider that y is constant and suppose the initial condition of the smoothing filter $s_0 = 0$, we can write:

$$s_k = \beta s_{k-1} + (1-\beta)y = y\left(1-\beta^k\right) \tag{9}$$

We want to know the number of iterations needed to reach a p percentage of the input step, with $p \in [0,1]$. Hence, by setting $s_k = p \times y$, we deduce the following relations:

$$\begin{cases} k & = \ln_\beta (1-p) \\ \beta & = (1-p)^{\frac{1}{k}} \end{cases} \tag{10}$$

As the system output changes very rapidly, we can not accept a delay greater than two cycles but we also need the filtered value to be as close as possible to the real one (*i.e.,* $p \simeq 1$). From Equation (10) if we set $k = 2$ and $p = 0.95$ we obtain $\beta = 0.22$ which manages to smooth enough the signal by only delaying the output very slightly.

3) Controller gain K_p: The value of the gain of the controller is also responsible for the closed loop behavior of the system. Small values of K_p yield conservative and slow controllers whereas greater values yield more aggressive controllers prone to overshooting and oscillations.

From [9], we want the contribution of the controller ($K_p \times error$) and the estimation of the system ($\hat{F}$) to have the same order of magnitude. Indeed, the role played by $\hat{F}$ in rejecting errors and/or perturbations of the model must be important with respect to the controller.

From Equations (6), (7) and (8), we get that for $K_p \times error$ and $\hat{F}$ to have the same order of magnitude we need:

$$K_p \simeq A_\beta = A_0 \times (1-\beta) \tag{11}$$

4) Summary: From the experiments done in Figure 2, as well as Equations (8), (10) and (11), we infer the value of the parameters of our Model-Free controller. The total number of resources available in the cluster is $r_{\max} = 100$. We first pick the value of β as explained in Section III-D.2 and then derive the values for α and K_p based on the smoothing factor. This gives:

$$\alpha = 0.008, \;\; K_p = 0.078, \;\; \beta = 0.22 \tag{12}$$

IV. Experimental Validation

In this section we detail the implementation of the Model-Free controller defined in Section III on the experimental setup containing the *CiGri* middleware. The behavior of the controller and its capacity to reject disturbances will be tested.

A. Experimental Setup

To test the controller described in Section III, we used the following setup:

- One *CiGri* Server
- One *OAR* Server (Version 3)
- One Fileserver (implemented with NFS)
- One Cluster of 100 *OAR* resources ($r_{\max} = 100$)

The experiments were done by using nodes from the Grisou Cluster in Grid'5000 [22] which is a shared French testbed for experimental research in distributed and parallel computing. Each node of this cluster has two Intel Xeon E5-2630 v3 CPU with eight cores per CPU and 128 GB of memory. Each server of our system is being deployed onto a single Grid'5000 node.

The experiment consisted in submitting campaigns to *CiGri* with different *I/O* loads in order to test the robustness of the method. The *I/O* load is given by writing a different file of 50, 100 or 200MBytes to the fileserver.

We want to regulate the load of the fileserver around a reference value ($y^\star$) given by the cluster administrators. A discussion about this value is given below in Section but for now we consider it constant (*i.e.,* $y^\star = 3$ in our case). In practice, these load values translate an overhead on the reading/writing time of a file: the higher the load, the longer the overhead. Thus, the system administrators would choose the reference value that yields an acceptable overhead for their system and users, based on their experience.

B. Controller validation

In the following experiments we submitted campaigns with a different quantity of *I/O* per job. We have three different campaigns, each one has jobs that first emulate a CPU bound computation (sleep for 30 seconds) and then write a file to the file-server (emulating the saving of results). Different file sizes of 50, 100 and 200MBytes are considered which have a different impact on the file-server.

Figure 3 depicts the evolution of the load of the fileserver (1st row), the number of jobs submitted by *CiGri* cycle (2nd row), the contribution of $\hat{F}$ (3rd row) and the contribution of the error (4th row) for the control law.

We can see that, as desired, most of the contribution to u (2nd row), the number of jobs submitted by *CiGri*, is done by the estimation of F (3rd row), and that the error term (4th row) only corrects slightly the input. We also see that when the system reaches a steady state, the error term is close to zero. This comforts us in the choice of the parameters.

Another remark concerns the stability of the load when reaching a steady state. We can see that for 100MBytes files, the load is quite stable, whereas for 200MBytes files it does slightly oscillate. This might indicate that the choice of K_p might not be completely adequate for this quantity of *I/O*. Similarly for smaller quantity of *I/O*, the response is slightly slower, indicating that a larger K_p might be more appropriate. However, the choice of this parameters (Equation (12)) still covers a large range of *I/O*, making this controller relevant for such a range.

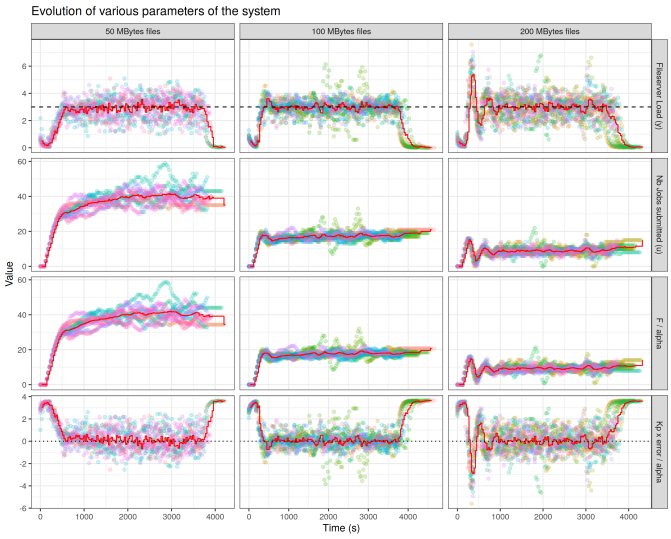

Fig. 3. Temporal evolution of the fileserver load (1st row), the number of jobs submitted by *CiGri* (2nd row), and the contribution of F (3rd row) and the contribution of the error term (4th row). Experiments run 10 times.

C. Controller robustness to external disturbances

In order to come closer to the real life behavior of the *CiGri* environment, we need to observe the behavior of the model-free controller with some perturbations, which in the case of *CiGri* are the jobs from the premium users of the cluster. The arrival of these jobs is considered unpredictable. A degenerative case can be represented as a step input which could represent the writing of a long file, or the frequent writing of medium files for example.

In this experiment we submit a campaign to *CiGri*, as discussed above, and set the reference value for the controller to 3 (black dashed line in Figure 4). After some time, we introduce a step perturbation (black dashed-dotted line) and finally, we end the step perturbation and go back to a disturbances free scenario.

For each campaign, we ran the experiment 10 times. The color of the points in Figure 4 corresponds to a specific experiment. We also show the mean behavior of the 10 experiments in plain lines.

We can see that the controller does increase progressively the number of jobs submitted to *OAR* (bottom row) to get the load of the fileserver (top row) to the reference value. Before the start of the step, the load has converged to the reference value.

At the start of the perturbation, we can see that the load increases and that the controller reacts by decreasing the number of jobs to submit in order to correct it and stabilize it back to the reference value.

At the end of the step, the load drops and we can see the controller increasingly submitting more and more jobs in order to get back to the reference value, where it stabilizes.

We can also note that for campaigns with larger *I/O*, the response of the controller is faster but also more aggressive.

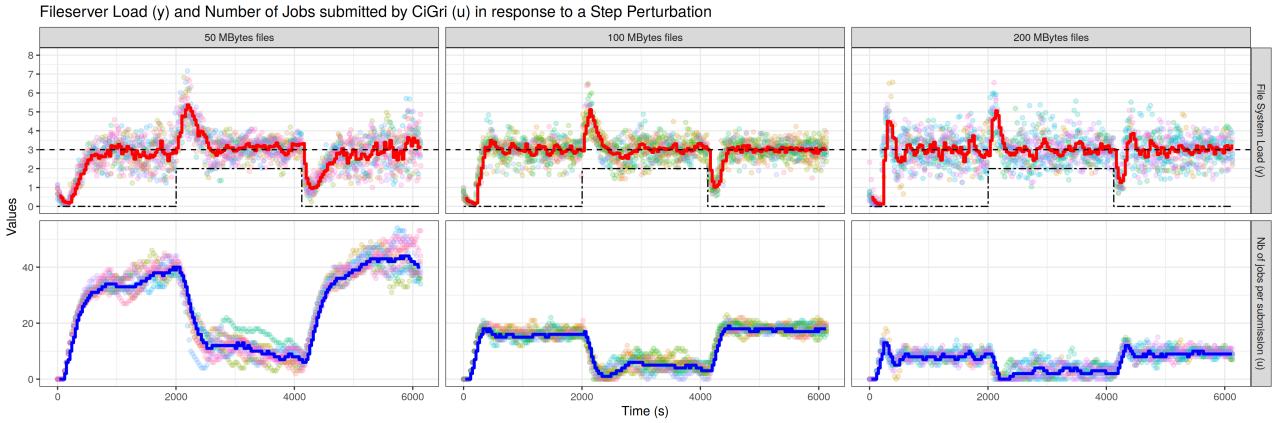

Fig. 4. Evolution of the Fileserver Load and the number of submitted jobs for a step disturbance. Experiments are run 10 times.

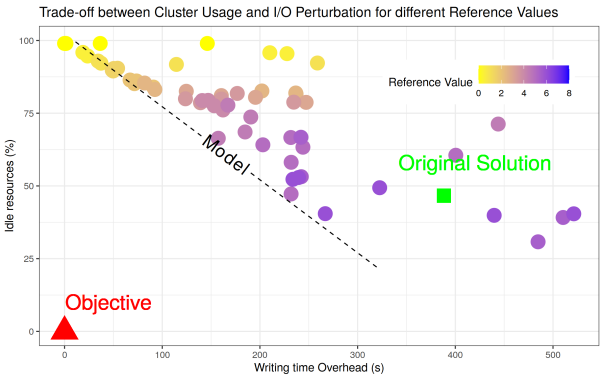

Fig. 5. Trade-off between the Cluster Utilization and the Time Overhead (*I/O* Perturbation to the Premium users jobs) with different reference values

D. Comparison with the Original CiGri Algorithm for several reference values

We want in this Section to evaluate the advantage of using our solution compared to the original *CiGri* submission mechanism [7] presented in Section II-B.1. We will run the *MADbench2* application [23] as a premium user's application to assess of the impact of the idle resources harvesting. *MADbench2* is a tool for testing the integrated performance of the *I/O*, communication and calculation subsystems of massively parallel architectures under the stresses of a real scientific application.

We submit a *CiGri* campaign and run our controller with different reference values between 0 and 8 (range of possible values for this configuration of the fileserver with NFS). At the termination of the *MADbench2* application, we will measure the amount of resources left idle and the time difference between the execution time with and without the harvesting (time overhead). The results are depicted in Figure 5. We can see that the smaller the reference value, the less we harvest (*i.e.,* more resources are left idle) but also the smaller the time overhead. For greater reference values, the quantity of harvested resources is more important (*i.e.,* few resources are left idle) as well as the perturbation.

The black dashed line in Figure 5 represents the ideal (optimal) values linking the reference value to the best amount of idle resources and best *I/O* perturbation. We also plot in Figure 5 the cluster usage and time overhead using the original *CiGri* solution (square point).

Nevertheless, as this solution does not take into account the load of the fileserver, it is not able to adapt its submission to a changing *Quality-of-Service*.

V. Conclusion

In this work we applied the Model-Free approach to the *CiGri* middleware. The objective was to harvest idle resources of a set of computing machines while regulating the load of the filesystem in order to maintain a given level of perturbations, or *Quality-of-Service*.

The advantages of using a Model-Free control are the simplicity of setup, the performances and the adaptativity. The simplicity is important for HPC systems administrators not experts in Control Theory, the adaptativity is important w.r.t. variations in time of the system behavior, but also, more originally, when using the controller for different machines, or different applications running on the system.

In Section III, we defined the values of the different parameters of the MFC. Then, in Section IV-C, we evaluated our solution on a synthetic step perturbation. The controller showed good tracking of the desired reference value, as well as stability. Section IV-D presented the evaluation of the perturbation of the harvesting on a premium user job. It exhibited a trade-off between the harvesting and the perturbing with the reference value being the main knob.

Inspired by [24], another interesting perspective is also the adaptation of the parameters (especially K_p) to the operating domain. Although the overall results obtained are good, this idea could further improve the results obtained in the extreme cases shown Figure 3.

Acknowledgment

Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see `https://www.grid5000.fr`).

REFERENCES

[1] M. Litoiu, M. Shaw, G. Tamura, N. M. Villegas, H. A. Müller, H. Giese, R. Rouvoy, and E. Rutten, "What Can Control Theory Teach Us About Assurances in Self-Adaptive Software Systems?" in *Software Engineering for Self-Adaptive Systems III. Assurances*, R. de Lemos, D. Garlan, C. Ghezzi, and H. Giese, Eds. Cham: Springer International Publishing, 2017, vol. 9640, pp. 90–134. [Online]. Available: http://link.springer.com/10.1007/978-3-319-74183-3_4

[2] J. O. Kephart and D. M. Chess, "The Vision of Autonomic Computing," *IEEE Computer*, vol. 36, pp. 41–50, Jan. 2003. [Online]. Available: http://pages.cs.wisc.edu/swift/classes/cs736-fa06/papers/autonomic-computing.pdf

[3] J. L. Hellerstein *et al.*, *Feedback Control of Computing Systems*. Wiley, 2004.

[4] A. Filieri *et al.*, "Software engineering meets control theory," in *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, 2015, pp. 71–82.

[5] P. K. Janert, *Feedback control for computer systems: introducing control theory to enterprise programmers*. O'Reilly Media, Inc, 2013.

[6] A. G. Yabo *et al.*, "A control-theory approach for cluster autonomic management: maximizing usage while avoiding overload," in *2019 IEEE Conference on Control Technology and Applications (CCTA)*. Hong Kong, China: IEEE, Aug. 2019, pp. 189–195. [Online]. Available: https://ieeexplore.ieee.org/document/8920473/

[7] Q. Guilloteau *et al.*, "Controlling the Injection of Best-Effort Tasks to Harvest Idle Computing Grid Resources," in *ICSTCC 2021 - 25th International Conference on System Theory, Control and Computing*, Iași, Romania, Oct. 2021, pp. 1–6. [Online]. Available: https://hal.inria.fr/hal-03363709

[8] S. Cerf *et al.*, "Sustaining Performance While Reducing Energy Consumption: A Control Theory Approach," in *EURO-PAR 2021 - 27th International European Conference on Parallel and Distributed Computing*, ser. Euro-Par, vol. 12820. Lisbon, Portugal: Springer, Aug. 2021, pp. 334–349. [Online]. Available: https://hal.inria.fr/hal-03259316

[9] M. Fliess and C. Join, "Model-free control," *International Journal of Control*, vol. 86, no. 12, pp. 2228–2252, 2013.

[10] ——, "An alternative to proportional-integral and proportional-integral-derivative regulators: Intelligent proportional-derivative regulators," *International Journal of Robust and Nonlinear Control*, 2021.

[11] T. Kuruganti, M. M. Olama, J. Dong, Y. Xue, C. Winstead, J. J. Nutaro, S. Djouadi, L. Bai, G. Augenbroe, and J. M. Hill, "Dynamic building load control to facilitate high penetration of solar photovoltaic generation: Final technical report," Oak Ridge National Lab.(ORNL), Oak Ridge, TN (United States), Tech. Rep., 2021.

[12] Z. Wang, A. Cosio, and J. Wang, "Implementation resource allocation for collision-avoidance assistance systems considering driver capabilities," *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[13] C. Sancak, F. Yamac, M. Itik, and G. Alici, "Force control of electro-active polymer actuators using model-free intelligent control," *Journal of Intelligent Material Systems and Structures*, vol. 32, no. 17, pp. 2054–2065, 2021.

[14] M. Bekcheva, M. Fliess, C. Join, A. Moradi, and H. Mounier, "Meilleure élasticité "nuagique" par commande sans modèle," *Automatique*, vol. 2, no. 1, Oct. 2018. [Online]. Available: https://hal-polytechnique.archives-ouvertes.fr/hal-01884806

[15] C. Join, M. Fliess, and F. Chaxel, "Model-free control as a service in the industrial internet of things: Packet loss and latency issues via preliminary experiments," in *2020 28th Mediterranean Conference on Control and Automation (MED)*. IEEE, 2020, pp. 299–306.

[16] M. Fliess, C. Join, and D. Sauter, "Defense against dos and load altering attacks via model-free control: A proposal for a new cybersecurity setting," in *2021 5th International Conference on Control and Fault-Tolerant Systems (SysTol)*. IEEE, 2021, pp. 58–65.

[17] D. Anderson, "BOINC: A System for Public-Resource Computing and Storage," in *Fifth IEEE/ACM International Workshop on Grid Computing*. Pittsburgh, PA, USA: IEEE, 2004, pp. 4–10. [Online]. Available: http://ieeexplore.ieee.org/document/1382809/

[18] M. Mercier, D. Glesser, Y. Georgiou, and O. Richard, "Big data and HPC collocation: Using HPC idle resources for Big Data analytics," in *2017 IEEE International Conference on Big Data (Big Data)*. Boston, MA: IEEE, Dec. 2017, pp. 347–352. [Online]. Available: http://ieeexplore.ieee.org/document/8257944/

[19] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke, "Condor-G: a computation management agent for multi-institutional grids," in *Proceedings 10th IEEE International Symposium on High Performance Distributed Computing*. San Francisco, CA, USA: IEEE Comput. Soc, 2001, pp. 55–63. [Online]. Available: http://ieeexplore.ieee.org/document/945176/

[20] K. J. Åström and R. M. Murray, "Feedback systems: An introduction for scientists and engineers," 2008.

[21] N. Capit, G. Da Costa, Y. Georgiou, G. Huard, C. Martin, G. Mounie, P. Neyron, and O. Richard, "A batch scheduler with high level components," in *CCGrid 2005. IEEE International Symposium on Cluster Computing and the Grid, 2005.* Cardiff, Wales, UK: IEEE, 2005, pp. 776–783 Vol. 2. [Online]. Available: http://ieeexplore.ieee.org/document/1558641/

[22] D. . Balouek *et al.*, "Adding virtualization capabilities to the Grid'5000 testbed," in *Cloud Computing and Services Science*, ser. Communications in Computer and Information Science, I. I. Ivanov, M. van Sinderen, F. Leymann, and T. Shan, Eds. Springer International Publishing, 2013, vol. 367, pp. 3–20.

[23] W. Dong, G. Liu, J. Yu, and Y. Zuo, "Characterizing i/o workloads of hpc applications through online analysis," in *2015 IEEE 34th International Performance Computing and Communications Conference (IPCCC)*. IEEE, 2015, pp. 1–2.

[24] P.-A. Gédouin, E. Delaleau, J.-M. Bourgeot, C. Join, S. A. Chirani, and S. Calloch, "Experimental comparison of classical pid and model-free control: position control of a shape memory alloy active spring," *Control Engineering Practice*, vol. 19, no. 5, pp. 433–441, 2011.