Making Control in High Performance Computing for Overload Avoidance Adaptive in Time and Job Size

Rosa Pagano, Sophie Cerf, Bogdan Robu, Quentin Guilloteau, Raphaël Bleuse, Eric Rutten

Making Control in High Performance Computing for Overload Avoidance Adaptive in Time and Job Size

Rosa Pagano[1] Sophie Cerf[2] Bogdan Robu[3] Quentin Guilloteau[1] Raphaël Bleuse[1] Éric Rutten[1]

***Abstract*— The feedback control of High-Performance Computing (HPC) has been explored as an application area of Control Theory, because of the high variability involved in their resource management. A regulation mechanism can allow to soundly automate the injection of small flexible jobs in a cluster. A trade-off is needed, to fill up the cluster's computing capacity while avoiding overload of e.g., the file server.**

In this work, we describe new results in this context, where the overload avoidance controller is made adaptive to the jobs' size, that is a time-varying unknown parameter. To do so, the original PI controller is enhanced with an online estimation algorithm that allows the controller to adapt to various working conditions, to avoid performance degradation. Parallel and robust estimation algorithms are designed, tackling the challenges of bursting and noise in the system. Validation and evaluation of the adaptive controller are performed on a large-scale experimental HPC platform, showing higher robustness than the state-of-the-art in highly varying conditions. Reproducible analysis are available at doi:10.5281/zenodo.11961696.

Keywords: Adaptive Control, Control for Computing, High Performance Computing

I. Introduction

High Performance Computing (HPC) systems are tools for scientists that stand out due to their computational power, which outstrips the capacity of a single computer. HPC platforms run large-scale computations, such as fluids mechanics simulations, molecular interactions or Artificial Intelligence training. However, HPC comes with a non-negligible energetic and monetary cost, and has a relatively short life cycle to ensure that top performing machines are always available. The optimal management of HPC infrastructures is a pivotal research domain. In this work, we focus on the optimization of the use of HPC resources, such as computing nodes.

Nowadays, such large scale distributed systems are subject to dynamical variations occurring, e.g., in the execution of the jobs, the quantity of reading and writing (called inputs/outputs or I/O) data exchanged, or the network consumption. They thus need autonomic management [1] in an online feedback control loop, in order to self-adapt to unpredictable evolutions. Leveraging methods from Control Theory for Software Engineering has been identified as a rich potential for the design of well-founded and well-understood autonomic managers [2], [3]. However, control-theory tools and methods are not straightforwardly usable and applicable, which makes control for computing a challenging research field [4], [5]: this is in part due to the lack of generally-admitted models for the controllers design, and to the great variety of problems and levels in the software stack, sometimes up to the hardware.

This work focuses on the application of control techniques for resource allocation in HPC systems. Computing jobs are scheduled to be executed on the HPC cluster, while most of the time this scheduling leaves unused resources due to jobs' high requirements. The idea behind resource harvesting is to take advantage of the idle resources left, by injecting smaller jobs. In the light to do this, two *categories* of jobs are considered: small, interruptible, and low priority are the *best-effort* (BE) jobs, and the others are the *high-priority* (HP) jobs. More particularly in this work, we consider a resource harvesting mechanism called *CiGri* [6], that regulates the injection and scheduling of BE jobs. Complementary; HP jobs are sent by the main users of the cluster, they are considered by the injection controller as an external constraint. This paper focuses on the case where the file server (the shared storage among the various nodes within a cluster) becomes the bottleneck of the system. Indeed, filling the cluster without caring about the file server status could slow down all computations, or even shut down and crash the entire system.

The approach consists in using control techniques to manage the mechanism for the harvesting of idle resources within an HPC cluster, in order to maximize its usage. Prior work [7] used a simplified linear model to build a Proportional Integral (PI) controller to avoid the overloading problem. Its performance, however, deteriorates when the working conditions deviate from the nominal ones. To overcome the dependency on modeling and PI formulation, a Model-Free Control approach [8] was proposed in [9]. Such an approach, however, cannot guarantee the performance of the closed-loop system. Building upon these results, the primary objective of this work is to make the PI controller *adaptive* to face diverse operational *scenarios* and be able to run in response to jobs of various sizes without requiring manual re-tuning. To do so, we use adaptive control [10], more specifically a self-tuning regulator [11]. The PI controller is enhanced with an online identification algorithm that allows the controller to cope with various working conditions to avoid undesired behavior observed in the previous solution.

In the remaining of the paper, the formulation of the control problem, prior model and PI is recalled (Section II), before describing the adaptive controller and the two estimation algorithms considered, and motivate its design (Section III).

[1]Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG, F-38000 Grenoble, France {firstname.lastname}@inria.fr

[2]Univ. Lille, Inria, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France sophie.cerf@inria.fr

[3]Univ. Grenoble Alpes, CNRS, Grenoble INP, GIPSA-lab, Grenoble, F-38000, France bogdan.robu@gipsa-lab.grenoble-inp.fr

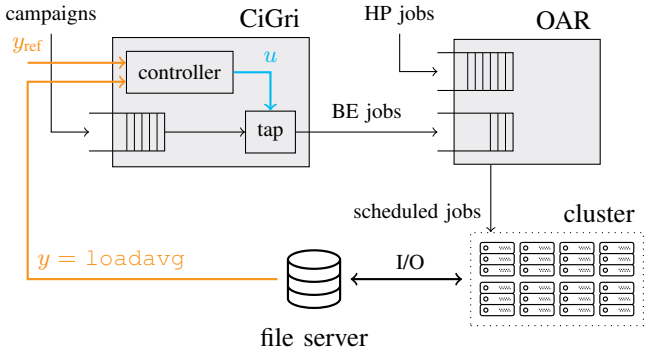


Fig. 1: Graphical representation of the *CiGri* feedback loop.

Then the adaptive control is first validated in simulation and then evaluated in the real setup (Section IV).

II. Background

A. System Under Control

CiGri [6] is a computing grid middleware aiming at harvesting the idle resource of a set of computing clusters. Users of *CiGri* submit campaigns of *bag-of-tasks* applications that are composed of tens or hundreds of thousands of small and independent jobs. Monte-Carlo simulations, or parameter sweep applications are examples of such applications. As those jobs are small and independent, they are the perfect candidates to use the resources left idle by larger and more rigid classical HPC jobs.

In its original form, *CiGri* submits batches of BE jobs to the scheduler (namely *OAR* in this case) [12] of the clusters and then waits for all these BE jobs to have completed before submitting again. This submission mechanism is suboptimal as it does not take into account the state of the cluster, and can lead to both an under-utilization or an overload of the system. Hence, the need for a feedback control mechanism. Figure 1 summarizes the considered feedback loop in *CiGri*.

While the introduction of a feedback loop in *CiGri* has been shown to improve cluster usage [13], it led to an increase of the load of the shared file server. This file server is the central entity hosting the files used by the computations. Any read or write (I/O) operations must go through the file server. Thus, any overload of the file server will slow down the I/O operations and impact the performance of every job being executed on the cluster. There is thus a trade-off between harvesting idle resource and degrading the I/O performance of all the jobs in the cluster.

To take this new dimension into account, Guilloteau et al. [14] extended the feedback loop proposed in [13] with a sensor on the file server. This sensor is based on the `loadavg` metric [15] which is a metric well known by system administrators of HPC centers. Moreover, this metric bears some inertia by its definition, which is a pleasant trait from the point of view of Control Theory.

Notation	Description	Unit
y	load of the file server	[1]
u	number of best-effort jobs to submit	[1]
f	file size of the best-effort jobs, (unknown)	[MiB]
$\tilde{b}$	model parameter, theoretical value	[1]
$\hat{b}$	online estimation of the model parameter b	[1]
V	covariance matrix for estimation	[1]

TABLE I: Summary of notations

B. Control Formulation

We place ourselves in the situation where the load of the file server is the limiting factor for the harvesting of idle computing resources, thus the quantity to regulate. We also consider a single cluster/scheduler.

The actuator (u) is the number of BE jobs that *CiGri* submits every $T_s = 30\,\text{s}$ to the *OAR* scheduler. The sensor (y) is the load of the file server. This load is captured via the `loadavg` metric and indirectly represents a degradation of the I/O performance. By definition, the file server load (explained in details in [7]) can take continuous values from 0 to 8. The reference value for the file server load should represent an acceptable trade-off between resources harvesting and the degradation of I/O performance. Table I summarizes the notations used.

C. Model

This section summarizes the first-order linear model of the file server from [7], that will be the base for the adaptive controller:

$$y(t+1) = a\,y(t) + b\,u(t), \tag{1}$$

where y is the file server's load, and u is the number of BE jobs sent by *CiGri* to the HPC platform.

The parameter a, describing the dynamics of the system, is analytically known: $a = e^{-T_s/60}$ [7].

The parameter b, describing the impact of the submission of jobs on the load, is not analytically known, and has to be identified from experimental data. Identification by experiments on the real system has shown that the measure of the load y depends both on the control action u (number of jobs) and on the size of the file used by the BE jobs, denoted f. The file size f is a time-varying parameter of the system. A static model has been drawn as a bi-linear regression:

$$y = c + \beta_1\,f + \beta_2\,u + \gamma\,f \times u \tag{2}$$

Numerical values of the regression parameters are reported in Table II.

In order to find an approximation of the parameter b, we consider Eq. (1) in steady state, with the control u_{ss} and the converged measure y_{ss}:

$$b = (1-a)\,\frac{y_{ss}}{u_{ss}} \tag{3}$$

Using Eq. (2), the former equation rewrites as:

$$b = (1-a)\left(\beta_2 + \gamma\,f + \frac{c + \beta_1 f}{u_{ss}}\right) \tag{4}$$

In order to find an approximation of b that is independent of the control signal, we simplify Eq. (4) with the realistic hypothesis that the last term can be neglected, provided that the control signal u is orders of magnitude larger than the numerator. We define $\tilde{b}$, the theoretical value of b given this hypothesis, as:

$$\tilde{b} = (1 - a)\,(\beta_2 + \gamma\, f) \tag{5}$$

From the expression above, one can note the connection between $\tilde{b}$ and f: the model parameter changes with the jobs file size. Let us note that the `loadavg` sensor introduces noise in the system: by definition, it integrates processes waiting on I/O but also other unrelated processes. This noise is modeled as a white noise of standard deviation 0.59.

D. Proportional Integral Control

Prior work computed a controller dealing only with the best-effort jobs. The idea was to develop a simple controller for its implementation on the real platform. A PI was chosen, implemented in discrete time [16]:

$$u(t) = u(t-1) + (K_P + K_I)\, e(t) - K_P\, e(k-1) \tag{6}$$

The controller is tuned with a pole placement design technique [16]. The new poles of the closed loop are $r\, e^{\pm j\theta}$, with $r \approx e^{-4/k_s}$ and $\theta \approx \pi \log r / \log M_p$. The two poles rely on the settling time k_s (here the number of iterations) and the maximum overshoot M_p expressed as a percentage of the reference.

In the end, the PI parameters are $K_P = \left(\xi - r^2\right) / b$ and $K_I = \left(1 - 2r\cos\theta + r^2\right) / b$. Note that these two parameters rely on the model parameter b, which depends on the file size f. In [14], a nominal value $f = 100\,\mathrm{MiB}$ of the file size is selected, resulting in $\tilde{b} = 0.0821$. Note that in practice, the controller does not have access to the characteristics of the jobs. Therefore, Eq. (5) cannot be used to estimate the model parameter b, as f is unknown.

E. Limitations of Previous Work

Figure 2 highlights the limits of the PI controller: we present the measured tracking performance over three experiments with different file sizes. The objective is to maintain the load of the file server at a constant level $y_{\text{ref}} = 3$. For all the experiments, the controller has been tuned with $f = 100\,\mathrm{MiB}$. Figure 2b depicts the nominal behavior, while Figs. 2a and 2c evaluate the robustness of the controller. Note that the results are the average of five similar executions for each condition, as the system is noisy. With a controller soundly tuned to the actual file size (Fig. 2b, the performance meets the requirements in terms of overshoot (none) and response time ($360\,\mathrm{s}$). When the controller is facing a file size twice larger than the value used for tuning, there is a speed-up in the system response but a significant overshoot. Conversely, when dealing with a smaller file size ($50\,\mathrm{MiB}$) the PI controller has a slower settling time than expected. Note that in the two non-nominal cases, the steady state oscillations are larger. This exhibits a lack of robustness w.r.t. variations in the jobs' file size, thus in the variations of the model parameter b.

Parameter	Value
c	$-0.507\,148\,4$
β_1	$0.008\,633\,5$
β_2	$0.045\,139\,4$
γ	$0.001\,633$
a	$0.606\,530\,7$

TABLE II: Numerical values of model parameters as experimentally identified in [7].

The objective of this paper is to devise a controller adaptive to the different submitted jobs. In practice, it is not possible to measure the file size of the jobs at runtime. We hence use an adaptation approach based on online estimation.

III. Adaptive Control Design

In order to cope with the varying size of the jobs submitted to *CiGri* (reflected in the values of the unknown parameter f), an adaptation mechanism is designed on top of the existing controller. We adopt the indirect approach, consisting in estimating the model parameters and subsequently updating the controller, over the direct approach, where the controller parameters are updated directly. Indeed, the model parameter a is fixed and known, hence the sole estimation of parameter b can be used to adapt both PI parameters. The self-tuning technique [11] is used, as it allows finding the optimal parameter for the controller, even if it does not ensure finding the true model parameters.

Using the system modeling, we can define $\hat{y}$, the estimated measure, and compute it as:

$$\hat{y}(t+1) = a\, y(t) + \hat{b}(t)\, u(t) \tag{7}$$

with $\hat{b}$ a real-time estimate of the parameter b. The considered adaptation is based on the least-squares resolution [17]. The parameter estimation $\hat{b}$ is updated as:

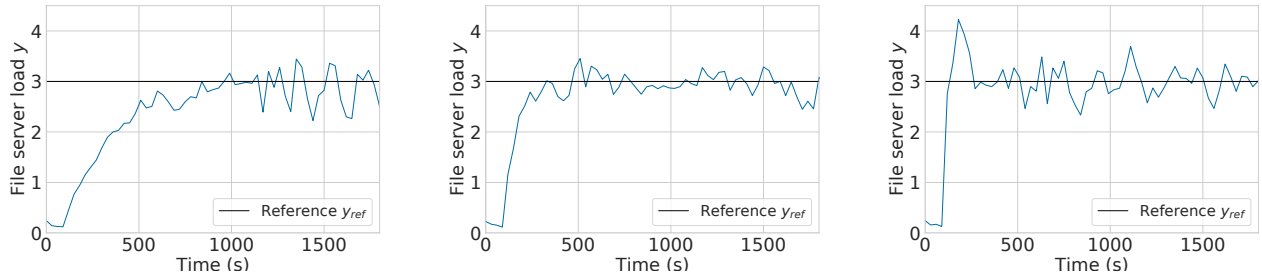
$$\hat{b}(t+1) = \hat{b}(t) + V(t+1)\, u(t)\, (y(t+1) - \hat{y}(t+1)) \tag{8}$$

where V is the covariance matrix, computed as:

$$V(t+1) = V(t) - \frac{V(t)\, u^2(t)\, V(t)}{1 + u(t)\, V(t)\, u(t)} \tag{9}$$

The estimation update of Eq. (7) relies on the difference between the measured output and the estimate one using Eq. (7). In addition, this difference is weighted by the product of the *covariance matrix* V and the information matrix in our unidimensional case is only $u(t)$. The covariance matrix allows tuning the relative impact of the new measure compared to the former estimate. V decreases with time, as the estimates improves in precision. Note that in our setup, we reset the value of the covariance matrix when a new campaign of jobs arrives. Indeed, as it results in a change of the parameter f, and thus of b, the reset allows the estimation to converge faster.

In the following, we present two different variants of the above algorithm, tackling time-variation, burst and noise challenges.



(a) $f = 50\,\text{MiB}$ (b) $f = 100\,\text{MiB}$ (c) $f = 200\,\text{MiB}$

Fig. 2: Limitation of the state-of-the-art controller regarding its robustness to changing jobs' file size f. Tuning requirements are no overshoot and 360 s response time. Average of five different experiments (in blue), with a reference of 3 (in black).

A. Parallel Estimation

When dealing with time-varying parameters, it becomes essential to account for the evolving nature of the system and give more weight to recent data while gradually forgetting older data. To achieve this, a forgetting factor, denoted as μ, is introduced [11], leading to a new formulation of the covariance matrix V_μ in the parallel algorithm:

$$V_\mu(t+1) = \frac{V_\mu(t)}{\mu} - \frac{V_\mu(t)\,u^2(t)\,V_\mu(t)}{1 + u(t)\,V_\mu(t)\,u(t)} \cdot \frac{1}{\mu^2} \tag{10}$$

The forgetting factor can vary from 0 to 1. When μ is close to 1, the algorithm is slow in following the time-varying parameters, as it gives more weight to historical data and is less sensitive to rapid changes. However, this approach also makes the algorithm less susceptible to noise, as it considers a larger amount of data. On the other hand, with a small value of μ, the algorithm becomes more responsive to fast-varying parameters, enabling it to track rapid changes in the system more effectively. However, a smaller μ also means that noise has a more significant impact on the parameter estimates, potentially leading to less accurate results.

To overcome the challenge of selecting a single adequate value of μ, the parallel Estimation algorithm introduced here employs a recursive least-squares estimator with a *variable* forgetting factor. At each step, a μ is chosen among different values to minimize the discrepancy between the measured output and the estimated output. This allows the algorithm to effectively track both fast and slow dynamics, and it prevents the occurrence of blowing-up phenomena. Indeed, if the algorithm continues to forget without new information (lack of excitation), it approaches a singularity which leads to a big strike on the estimated parameter. This is known as the *bursting phenomenon* [18]. Given a vector μ of potential values within $[0, 1]$, we compute in parallel the corresponding estimated value $\hat{b}$ using Eq. (8), where the covariance matrix is computed as in Eq. (10). Each value of μ leads to a different estimation: we select the value with the smallest prediction error $\epsilon(t)$ defined as $\epsilon(t) = y(t) - \hat{y}(t)$. Note that altough many estimations are computed, it remains cheap to compute. Each estimation can be computed in constant time, hence the total work is linear in the size of μ.

B. Robust Estimation

As our system is subject to noise, the previous algorithm could have some difficulties due to the direct derivation of a least-squares cost function. A least-squares approach indeed relies on low noise levels, otherwise the estimation could significantly deviate due to the squaring in the cost function. We introduce a new formula, taking into account the noise challenge, to compute $\hat{b}$:

$$\hat{b}(t+1) = \hat{b}(t) + V(t+1)\,u(t)\,F(\epsilon(t)) \tag{11}$$

F is a filter for the prediction error $\epsilon(t)$ defined as:

$$F(\epsilon(t)) = \frac{\epsilon(t)}{1 + \alpha\ |\epsilon(t)|}. \tag{12}$$

For large errors, the function $F(\epsilon(t))$ will be more gradual, deviating from linearity, while for small errors the function will resemble a linear function. This modification allows the algorithm to handle larger errors more effectively, making it more robust in the presence of substantial noise [10]. This revised formula introduces a smoothing factor α, that is important for noise reduction: the greater the α, the greater the noise reduction. Let us observe that the computation of the robust estimation remains a constant-time algorithm, as is the PI.

C. Tuning of Adaptation Parameters

Before testing the adaptation performance of the algorithm on our system, we need to tune the algorithms. The quantities to tune are the forgetting factors μ, the smoothing parameter α, and the initial conditions $\hat{b}(0)$ and $V(0)$. We tune all parameters independently (namely varying parameter, noise smoothing, initial guess and update speed), as they impact differently the control performance.

Note that there could be a limitation in the independent tuning of μ and α. Indeed, both reduce the impact of noise in different ways: μ by using the memory of the previous data, α for the non-linear function which acts each time step. Then, we adopted two different approaches: we will avoid the selection of a single value for μ, and optimally select the best value at each iteration; while α is fixed.

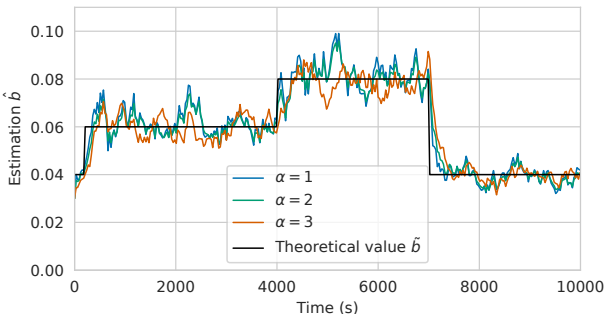


Fig. 3: Parameter estimation with the robust algorithm for various values of the smoothing factor α. Simulation results.

1) Forgetting Factor μ: The number of values in the μ vector expresses the trade-off between precision of the estimations and computing overhead. We deliberately avoided values below 0.5 due to the high level of inherent noise present in the system. We therefore choose the vector of forgetting factors in the parallel algorithm as $\mu = [0.5, 0.6, 0.7, 0.8, 0.9, 1]$.

2) Robust Smoothing Factor α: As the model parameter b is not measurable, we rely on simulation to set different values of b. We simulate the behavior of the system with a varying b value: this is achieved by changing the campaign file sizes f. We trigger two b changes: $+0.02$ at $4000\,\text{s}$ and -0.04 at $7000\,\text{s}$. The reference for the file server load y is set to the constant value 3. Figure 3 depicts the simulation results for various values of α. We observe fewer oscillations when increasing α. On the other hand, a high α value slows the algorithm: this is visible for $\alpha = 3$ around $7000\,\text{s}$. We choose $\alpha = 2$ as a good trade-off between reducing oscillations and preserving convergence speed.

3) Initial Estimation $\hat{b}(0)$: We tune the initial estimation with experiments in real platform, because the model misses the initial varying delay which exist in the real scenario. Then, the initial conditions affect the real system, but the simulation fails to capture this impact. An accurate tuning of $\hat{b}(0)$ is important as an imprecise guess could lead to an overload of the file server. Indeed, a small value of $\hat{b}(0)$ leads to an undershoot on $\hat{b}$, and thus over-increases the control action, as implied by the definition of K_P and K_I (see Section II-D). The dependency of the control action on the inverse of b advises against opting for a small initial estimation value.

Figure 4 presents the results for two initial condition values: an over-estimation ($\hat{b}(0) = 0.5$), and the theoretical value ($\hat{b}(0) = 0.15$). Both experiments exhibit the same behavior. The estimated value of $\hat{b}$, in Fig. 4a, initially decreases below the true value and then increases. We observe that the system requires some time to properly set up, behaving as an initial delay of roughly four time steps. Owing to this initial delay, the estimation is notably under-estimated during this initial phase. The consequences of this under-estimation are twofold. First, it results in an peak in the control action, see Fig. 4b. Notably, the peak is more pronounced with $\hat{b}(0) = 0.15$, primarily due to the estimated value reaching lower levels, as seen in Fig. 4a. Second, the under-estimation affects the tracking performance, leading to an overshoot in the early stages, see Fig. 4c. We therefore select an initial estimation of $\hat{b}(0) = 0.5$, significantly higher than the theoretical value. This choice facilitates a soft start of the control action, and effectively prevents overshoots that could lead to system overloads. It is worth noting that the choice of this $\hat{b}(0)$ value is not the sole solution: an initial value greater than 0.5 would probably yield satisfactory results.

4) Initial Covariance $V(0)$: The larger the initial covariance $V(0)$, the lower the impact of the inital guess $\hat{b}(0)$ on the estimation. To find a suitable value, we carried out experiments. Figure 5 presents the estimation (Fig. 5a) and tracking (Fig. 5b) performance for three different initial covariance values: 10^2, 10^4 and 10^6. As the system is noisy, we plot the average results of five experiments. The impact of $V(0)$ is relatively small. The main observed discrepancy is the overshoot within the tracking outcome. We opt for the initial value leading to the smallest overshoot: $V(0) = V_\mu(0) = 10^4$. Both the parallel and the robust algorithm are set with the same initial covariance. Note that a large initial covariance value $V(0)$ may reduce the impact of the initial condition $\hat{b}$.

IV. Experimental Evaluation

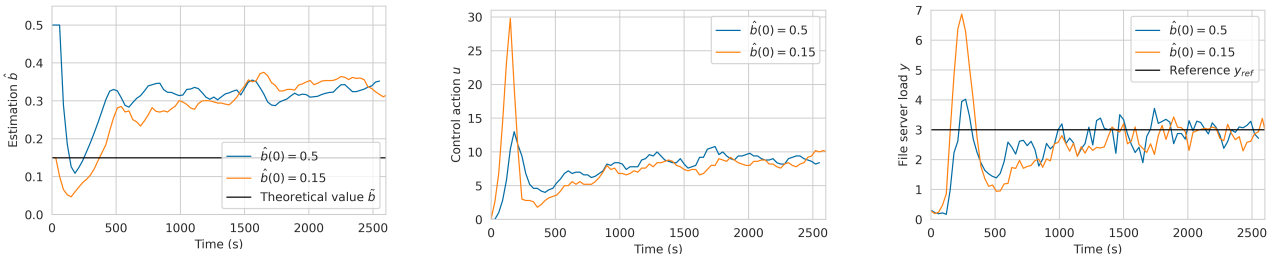
The adaptive controller and its two variants are first compared in simulation, both for their estimation and tracking performance. Second, after presenting the experimental setup, the best performing algorithm, e.g., the robust one, is evaluated experimentally on a real HPC platform. Its performance on several executions (or *runs*) is analyzed. Eventually, we compare the adaptive control to the state-of-the-art PI on varying systems.

A. Comparison of Estimation Algorithms in Simulation

The parallel and robust estimation algorithms are first compared in simulation, to allow having the ground truth value of b, unknown in experimentation. Here we are interested in evaluating both their performance in estimating the model unknown parameter $\hat{b}$, and on reference tracking.

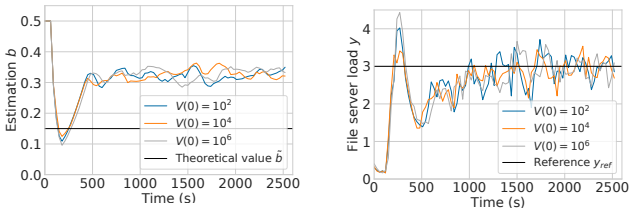
Figure 6a presents the estimation results for the parallel and robust algorithms. Both algorithms manage to estimate the parameter value, while the parallel algorithm presents larger oscillations. This is a direct consequence of the implementation of the robust algorithm, tailored to mitigate the influence of noise through the adjustment of the smoothing parameter α.

The tracking performance of both algorithms are presented in Fig. 6b. In both cases, the tracking is fairly good, however with a slight static error for the parallel algorithm. Both outputs present significant noise. The difference in tracking performance is less significant than what is observed for the estimation. It seems that the influence of the measurement noise on the output tracking is equally significant for both



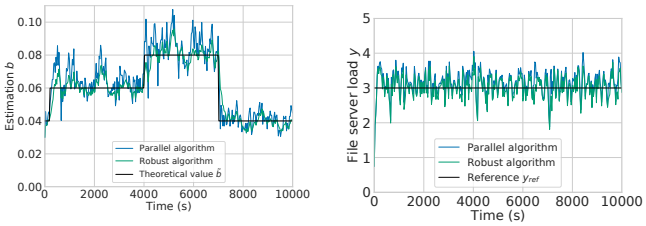
(a) Estimation $\hat{b}$ (b) Control action (c) Tracking

Fig. 4: Impact of the initial estimation $\hat{b}(0)$ on estimation, control, and tracking. Robust algorithm.



(a) Estimation $\hat{b}$ (b) Tracking

Fig. 5: Tuning the initial covariance value $V(0)$. Robust algorithm.



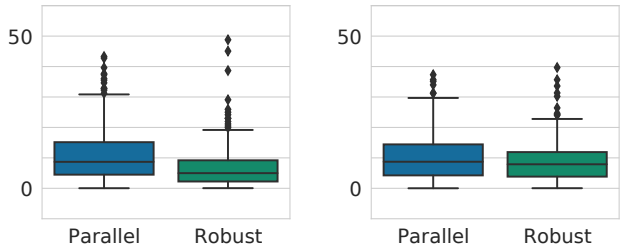
(a) Estimation $\hat{b}$ (b) Tracking

Fig. 6: Comparison of the parallel and robust algorithm in simulations.

algorithms in this simulation setup, even though less noise was observed in the estimation of $\hat{b}$ with the robust algorithm.

To objectively compare the algorithms, we quantify the estimation and tracking performances using the metric of the relative percentage error, shown in Fig. 7. These box plots aim at offering valuable insights into the distribution of the estimation and tracking errors. The results of Fig. 7a confirm that the robust algorithm outperforms the parallel one in estimating $\hat{b}$. This assertion is shown by the significant reduction in the error average and variance. With the parallel algorithm, $75\,\%$ of the data points exhibit a relative percentage estimation error below $15\,\%$ (top line of the blue box). However, with the robust algorithm, an equivalent proportion of data points exhibit errors below $10\,\%$.

Now looking at the tracking performance in Fig. 7b, one can see that the robust error values are smaller than the other algorithm. Performance disparities among algorithms are however not as significant as in the case of estimation.



(a) Estimation error (in %) (b) Tracking error (in %)

Fig. 7: Distribution of the estimation and tracking relative percentage error in absolute value for each algorithm. Data from Figs. 6a and 6b.

The similarity in tracking errors among the results can be attributed to the fact that the influence of noise can sometimes surpass the impact of the estimation of $\hat{b}$ in the controller. To further study the impact of noise, Section IV-C will present experimental results of multiple controller executions, and the average behavior.

To conclude, the robust algorithm outperforms the parallel one in estimation and tracking, hence will be selected for the experimental evaluation.

B. Experimental Setup

The experiments presented in this paper have been carried out on the `grisou` cluster of Grid'5000 testbed [19], a nation-wide infrastructure for large-scale experimental Computer Science. Each experiment required to deploy four machines: a *OAR* server, a *CiGri* server, a file server with NFS, and a cluster. We used the technique of folding described in [7] to emulate a full scale cluster of hundred nodes. The jobs submitted to *CiGri* are conceived so that to allows changing their duration and I/O load. Note that each experiment runs for several hours on the High Performance Computing platform, and are repeated five times each to cope with the inevitable variability of such shared real-life large-scale infrastructures: temperature effects, measurement noise, shared network, etc.

C. Tackling Variations

In this subsection, the robust algorithm is evaluated in experiments on the real platform, additionally allowing to

analyze the system's inherent noise and more important the variability between different executions even with the same initial conditions. Results are given in Fig. 8. The reference is constant and equal to 3. Two distinct jobs campaigns are launched, thus resulting in a varying b along the experiment. The first campaign has a file size f of 100 MiB with 2000 jobs, while the other has f equal to 200 MiB with 1000 jobs.

In an experimental context, the beginning of the second campaign is not deterministic as it will start only when all the jobs of the previous campaign are sent, which is dependent on the control action u. Thus, in different experiments, the second campaign might start at a different time. Black lines of Fig. 8a illustrate the theoretical values of b for each campaign.

In Fig. 8, we refrained from displaying solely the average of the five experiments, as such an approach would oversimplify the analysis, due to the divergence in experiment completion times for the first campaign. One can note this variability in Fig. 8a, where certain experiments initiate the new campaign around 3000 s (light blue and orange lines), while others commence closer to 4000 s.

First, we evaluate if the behavior of the closed-loop complies with the design requirements, that is no overshoot and a response time of twelve sampling times, that is 360 s. Figure 8b depicts the tracking behavior for all experiments (thin lines) and their average (bold red line), as well as the reference of 3 (black line). All experiments successfully track the reference with no steady state error, despite significant noise. There is no overshooting in average, and only one experiment out of five has an overshoot. Nevertheless, we observe that the system reached the reference value after more than double the initially projected time. It seems that the time taken by the estimation algorithm to converge to a specific value for $\hat{b}$ delays the tracking convergence.

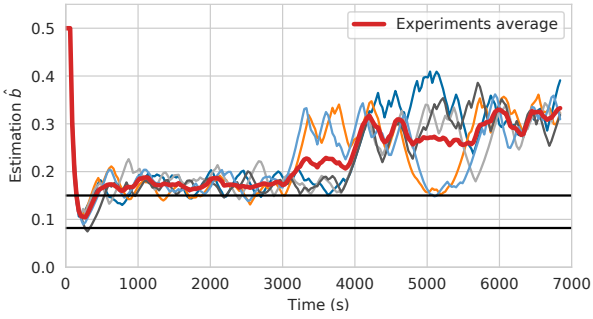
Figure 8a presents the results of the estimation. $\hat{b}$ converges in about 400 s, with some downshoot but smoothly, and similarly for all experiments. After the change of campaign (around 3000 s to 4000 s) the estimation shows an increasing trend, resulting in a reduction of the control action (see Eq. (6) and the computation of K_P and K_I) and successfully maintaining the tracking of the reference.

To conclude, the adaptive controller is stable, precise, and fast, both for individual experiments and in average. The controller successfully adapts to a changing system, i.e., a change in the jobs' campaign, despite significant noise.

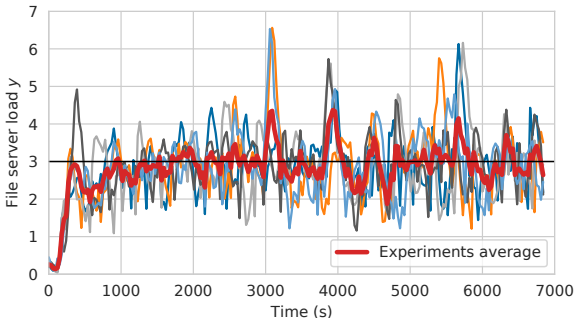
D. Comparison With State-of-the-art

To conclude on the performance of the presented adaptive controller, we compare to the state-of-the-art PI. For fair comparison, both the PI and adaptive controllers are tuned by pole placement as described in Section II-D, with $M_p = 0$ and $k_s = 12$. A range of system's variation is considered, with a file size f varying from 50 MiB to 400 MiB. Moreover, we consider a step disturbance from 2000 s to 4000 s. Tracking results are given in Fig. 9.

The adaptive controller successfully tracks the reference value in all four conditions. Note that, differently to the PI, the adaptive control does not suffer from large starting overshoots, whatever the working condition. Important overshoots are particularly detrimental in our system, as it leads to overloading the file server, which brings the system to a critical state where the jobs cannot be executed anymore. The system with the adaptive control is however slower, while this difference tends to decrease for working conditions far from the nominal one. Moreover, as expected, the further from the nominal conditions (e.g., 400 MiB) the more chaotic the PI's behavior becomes.



(a) Estimation $\hat{b}$



(b) Tracking

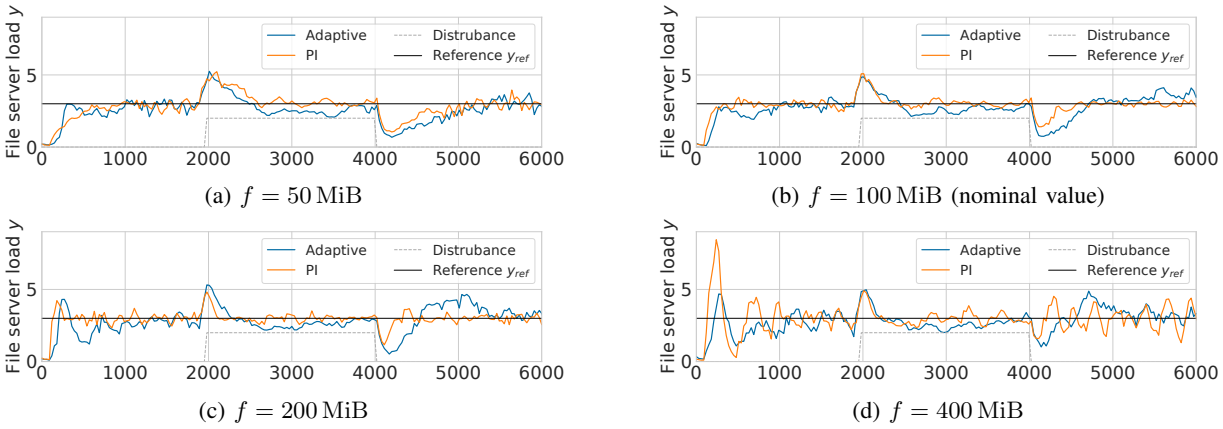
Fig. 8: Adaptive controller performance in case of system's variation. Two campaigns are successively launched, so f (and thus b) changes through the experiment. Five repetitions of individual experiments are represented in thin lines, the average is the bold red line. The setup uses the robust algorithm with $\alpha = 2$, $\hat{b}(0) = 0.5$, $V(0) = 10^4$.

The adaptation not only allows dealing with changes in the campaign (i.e., size of jobs), but is also robust to disturbances, as can be seen in all plots from 2000 s to 4000 s.

To sum up, the adaptive control manages to successfully track the reference in all conditions, it can be applied to various jobs, and could be used also for other HPC clusters without a necessary pre-tuning.

V. Conclusion

We presented in this work a way to harvest unused resources of a HPC cluster. The harvested resources are used to execute small and low priority jobs. The injection of such jobs needs regulation to avoid overloading the platform: in particular, the file server is a critical resource. We aimed



(a) $f = 50\,\text{MiB}$ (b) $f = 100\,\text{MiB}$ (nominal value)

(c) $f = 200\,\text{MiB}$ (d) $f = 400\,\text{MiB}$

Fig. 9: Performances of the adaptive and PI controllers, on four systems with different file size.

in this work to design a control algorithm able to adapt its parameters for varying working conditions. The designed controller must prioritize the stability of the file server in order to avoid an overload, and ultimately crashing the whole HPC system. We proposed in this work two variants of an adaptive algorithm tailored for this problem. We then studied their the performance in simulation and validated the best one on a large-scale HPC platform. The experiments show the designed controller is able to adapt to varying working conditions, and can effectively limit the overload of the file server. However, the adaptive system can be slower than the original control.

Control theory usually considers adaptability w.r.t. time-varying systems. One could consider instead making the controller adaptive when deployed on a variety of systems. The variability in this case is spatial: it comes from the hardware or even the software configuration. The adaptation would then be across several repetitions on different systems, rather than a temporal adaptation. One could consider alternatives between self-tuning and adaptive control in these cases. Another perspective resides in the coordination between the controller and other already-existing decision mechanisms. For example, coordinating with the cluster scheduler would allow combining control and scheduling theories. An ongoing work considers using the scheduler knowledge of future jobs as a building block for feed-forward control. Finally, in the context of complex systems with a hierarchical structure, it is interesting to consider the coordination of controllers at different levels. Cluster-level controllers could interact with lower-level controllers such as power controllers.

Acknowledgments

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). This work has been done in the context of the Inria PULSE project (see https://www.inria.fr/en/pulse).

References

[1] J. O. Kephart and D. M. Chess, "The Vision of Autonomic Computing," *IEEE Computer*, vol. 36, pp. 41–50, Jan. 2003.
[2] A. Filieri *et al.*, "Software Engineering Meets Control Theory," in *SEAMS@ICSE*, pp. 71–82, IEEE, May 2015.
[3] É. Rutten, N. Marchand, and D. Simon, "Feedback Control as MAPE-K Loop in Autonomic Computing," in *Software Engineering for Self-Adaptive Systems*, vol. 9640 of *LNCS*, pp. 349–373, Springer, 2013.
[4] S. Shevtsov *et al.*, "Control-Theoretical Software Adaptation: A Systematic Literature Review," *IEEE Trans. Software Eng.*, vol. 44, no. 8, pp. 784–810, 2018.
[5] A. Filieri, H. Hoffmann, and M. Maggio, "Automated Design of Self-Adaptive Software with Control-Theoretical Formal Guarantees," in *ICSE*, pp. 299–310, ACM, 2014.
[6] Y. Georgiou, O. Richard, and N. Capit, "Evaluations of the Lightweight Grid CIGRI upon the Grid5000 Platform," in *eScience*, pp. 279–286, IEEE, 2007.
[7] Q. Guilloteau, *Control-based runtime management of HPC systems with support for reproducible experiments*. PhD thesis, Univ. Grenoble Alpes, France, Dec. 2023.
[8] M. Fliess and C. Join, "Model-free control," *Int. J. Control*, vol. 86, no. 12, pp. 2228–2252, 2013.
[9] Q. Guilloteau *et al.*, "Model-Free Control for Resource Harvesting in Computing Grids," in *CCTA*, pp. 384–390, IEEE, Aug. 2022.
[10] K. J. Åström and B. Wittenmark, *Adaptive Control*. Dover Publications, 2 ed., 2008.
[11] K. J. Åström and B. Wittenmark, "On Self Tuning Regulators," *Automatica*, vol. 9, pp. 185–199, Mar. 1973.
[12] N. Capit *et al.*, "A batch scheduler with high level components," in *CCGRID*, pp. 776–783, IEEE, 2005.
[13] E. Stahl *et al.*, "Towards a control-theory approach for minimizing unused grid resources," in *AI-Science@HPDC*, pp. 4:1–4:8, ACM, June 2018.
[14] Q. Guilloteau *et al.*, "Controlling the Injection of Best-Effort Tasks to Harvest Idle Computing Grid Resources," in *ICSTCC*, pp. 334–339, IEEE, Oct. 2021.
[15] D. Ferrari and S. Zhou, "An Empirical Investigation of Load Indices for Load Balancing Applications," in *Performance*, pp. 515–528, North-Holland, 1987.
[16] J. L. Hellerstein *et al.*, *Feedback Control of Computing Systems*. Wiley, 2004.
[17] L. Ljung and T. Söderström, *Theory and Practice of Recursive Identification*. Signal Processing, Optimization, and Control, MIT, Oct. 1983.
[18] B. D. Anderson, "Adaptive Systems, Lack of Persistency of Excitation and Bursting Phenomena," *Automatica*, vol. 21, pp. 247–258, May 1985.
[19] D. Balouek *et al.*, "Adding Virtualization Capabilities to the Grid'5000 Testbed," in *CLOSER*, vol. 367 of *Communications in Computer and Information Science*, pp. 3–20, Springer, 2013.