Controlling the Injection of Best-Effort Tasks to Harvest Idle Computing Grid Resources

Quentin Guilloteau, Olivier Richard, Bogdan Robu, Eric Rutten

Controlling the Injection of Best-Effort Tasks to Harvest Idle Computing Grid Resources

Quentin Guilloteau, Olivier Richard
Universite Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG
38000 Grenoble, France
Quentin.Guilloteau@inria.fr,
Olivier.Richard@inria.fr

Bogdan Robu
Universite Grenoble Alpes, CNRS, Grenoble INP, GIPSA-lab
38000 Grenoble, France
Bogdan.Robu@gipsa-lab.fr

Eric Rutten
Universite Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG
38000 Grenoble, France
Eric.Rutten@inria.fr

***Abstract*—High Performance Computing systems are facing more and more variability in their performance, related to e.g., Input/Output (*I/O*) behavior and power consumption : they are less predictable, which requires more run-time management to meet the requirements. This can be addressed following feedback approach, where a management feedback loop, in response to monitored information in the systems, based on analysis of this data, decides to activate system-level or application-level adaptation mechanisms. One such regulation problem is found in the context of *CiGri*, a lightweight computing grid system which exploits the unused resources of a set of computing clusters. The computing power left over by the execution of premium cluster users' *HPC* applications, is used to execute smaller jobs, which are injected as much as the global system allows.**

The feedback loop which we design has to regulate this injection of jobs in such a way as to avoid overloading of the distributed file system (or file-server), which would be detrimental to the main performance, while self-adapting to variations in load in order to make the best use of available resources. We put in place a mechanism for feedback control in system software by controlling the number of jobs to be sent to the cluster in response to system information about the current number of processed jobs and the file-server load, which has a significant impact of the performance of the priority users jobs. We perform experimental validation by comparing several control solutions.

***Index Terms*—Dynamic resource management, High performance computing, autonomic computing, instrumentation, self-adaptive systems.**

I. Introduction

High Performance Computing (*HPC*) systems have become more and more complex to manage due to the increases in the number of machines and their usage. The inherent variations (e.g. in performance or in power consumption) make it very hard to fully predict or model their behavior [1].

Some variations concern computation itself, e.g., processor speed depending on temperature, or on DVFS (Dynamic Voltage and Frequency Scaling). Other variations concern communication and storage e.g., if the network of the cluster is heavily loaded, or number of application executing read and write operations on a file-server concurrently is high. This can also have impact on the energy consumption, as the node CPUs are idle while the network is being used [4].

Such situations require run-time management, which is usually done by human administrators. However, this run-time management is becoming too complex and too fast to be done manually therefore the need for automation, in order to avoid slow or error-prone manual manipulations [11].

The topic of harvesting the idle computing resources of a set of *HPC* machines can be seen as a typical case of run-time management. Indeed, the essential idea is that the variations of computation load can leave parts of the resources in a *HPC* computing grid unused. Unpredictability can come for example from the fact that the real duration of tasks is imprecisely known, and the worst-case considered in the scheduling makes it safe, but leads to risks of over-dimension. Run-time management enables access to real-time data w.r.t. tasks activities, and opens potentials to identify unused resources and allocate them to other tasks.

Resource harvesting has been studied in previous research. Projects such as BOINC [2] were designed to exploit unused computer cycles from personal workstations. People would install a specific saving screen on their personal machine. Once the screen saver is running due to inactivity, the CPU schedules tasks from chosen scientific community projects. The issue of such solution is the high volatility of the computing resources.

A similar approach has been taken by *Condor* [9]. *Condor* users would register their workstation on the grid and when they need more computing power than available on their machine, they can submit their job to *Condor* that will distribute it onto other workstations of the grid.

BeBiDa [10] took the approach of having a cluster with two independent schedulers: one for regular *HPC* jobs of the platform and the other for Big Data jobs. [5] is another example of resource harvesting for scientific experiments where multiple users, each of them with few machines, are connected together. Their algorithm *OurGrid* manages to scale up their application, but also to reduce the number of idle resources on registered machines.

Another idea is the one taken by *CiGri* [8], a lightweight Grid build upon the Gricad French platform and used in exploitation at the University Grenoble Alps. *CiGri* runs on top of a set of *OAR* [3] clusters and submit jobs from *Bag-of-Tasks* applications with the lowest priority in order not to disturb the priority users. For the remaining of this paper, we will focus on the context of *CiGri* and its harvesting mechanisms.

The management of systems with such characteristics and variations must be done online based on the measurement

of their current state. Moreover, decisions should be taken automatically based on those measures and by respecting the contractual requirements. These issues lead us to consider existing notions of control engineering and feedback loops for our problem.

In this paper, we build upon some preliminary work. In [12], a Proportional-Integral controller was developed in order to control the number of jobs sent to the cluster from *CiGri* but without taking into account the load of the file-server which is the most important and by considering a very small number of computing resources which is not realistic for an HPC environment. A MPC approach based on a very simplistic approximation of the behavior of our system: cluster and file-server, was used in [13] but it showed again its limitation while scaling up with the size of the cluster and the number of the high priority users. The remaining of this paper extends the previous work by introducing new approach for the regulation of the file-server load by taking into account the *I/O* weight of the *Bag-of-Tasks* applications while harvesting idle resources.

For the rest of the paper, Section II describes into details our system. We present in Section III our solution and evaluate it in Section IV. Conclusions and future work are detailed in Section V.

II. Background

A. The CiGri Middleware

This paragraph describes more into details the *CiGri* architecture. The jobs come from *Bag-of-Tasks* applications, composed of many small independent tasks that can thus be executed in parallel. The main issue with these middlewares (i.e. *CiGri* in our case) is linked to the potential overloading of the file-server. Indeed, non-attentive harvesting from the middleware may lead to an overload of the file-server due to too many *I/O* operations (i.e., reading/writing files), heavily impacting the performance of the latter. Moreover, its behavior should not disturb the high priority users of the clusters.

1) General architecture: The architecture of the system is depicted in Figure 1. The computing grid is composed of clusters, grouping compute nodes resources, connected by a network to a storage server (or file-server). *OAR* servers are managing computations on the clusters, scheduling tasks coming mainly from priority users, and also from *CiGri*.

2) Best-effort Jobs: CiGri jobs are viewed by *OAR* as *Best-effort* jobs. They are jobs 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.

3) Bag-of-Tasks Applications: CiGri itself takes work in the form of *Bag-of-Tasks* (BoT) applications, that are large sets of independent parametric tasks. Some examples of BoT applications are Monte-Carlo applications where the user has do execute thousands of small independents tasks to produce a statistical results from the results of all the tasks. Other examples can be jobs from the Big Data field, or parameter sweep applications.

4) CiGri Submission Algorithm: The actual algorithms that is used in exploitation is the following: every 30 seconds, *CiGri* will call the submission function that will decide of the quantity of jobs to submit to *OAR*. The current job submission function implemented in *CiGri*, showed in Algorithm 1, is based on a *tap mechanism*. *CiGri* will submit a number of jobs to the *OAR* scheduler (*open the tap*) and will wait until all these jobs have finished executing (*close the tap*) to submit a new subset of jobs (*open the tap again*).

Algorithm 1: *CiGri* Current Job Submission

```
Input: rate (init. 3), increase_factor (constant 1.5)
if no running jobs then
    rate = min(rate × increase_factor, 100);
    submit rate jobs;
else
    submit 0 job;
end
```

The current solution implemented in *CiGri* has the drawback that once it has submitted a set of jobs, it will wait for the completion of all those jobs before submitting another set of jobs. A situation where this strategy would lead to under-utilization is when there are only a few *CiGri* jobs left from the previous submission to execute on the cluster, but not enough to use all the idle resources of the cluster. In this case, we would have wanted *CiGri* to submit a couple of jobs from the *Bag-of-Tasks* before the end of the previous submission in order to use all the available computing resources and not waste idle computing power.

5) File-servers management: Every *HPC* job either writes a file, reads a file or both. Usually, in a cluster, there is one, or several, dedicated machines 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 not limitless, and too much requests can lead to an overload and thus to a drop in performance, affecting the priority users of the cluster (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.

B. 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, we could make sure that the latter is not overloaded by *CiGri* and thus is not the bottleneck of some high priority user applications.

Before implementing the controller and in order to measure the effectiveness of the proposed method we implement two `sensors`. The first sensor is the UNIX `/proc/loadavg` sensor [7] which would detect the overloading of the system.

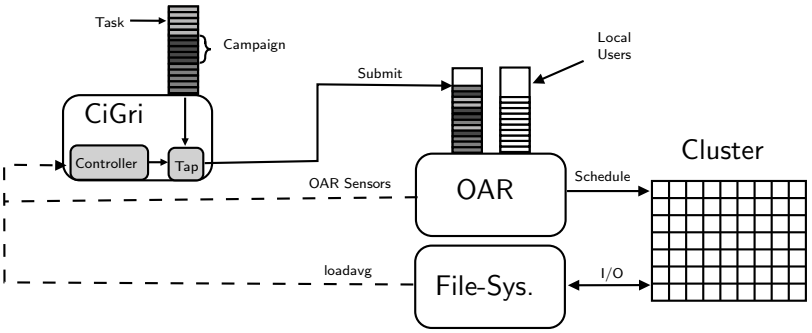

Fig. 1: Architecture of the System

A rule of thumb among system administrators 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 second set of sensors are metrics from the *OAR* scheduler: the length of its waiting queue, number of running jobs. We choose the length of the *OAR* waiting queue, as it is an indicator of the processing rate of the cluster and as it does not depend on the total number of machines in the cluster that can be variable due to errors or maintenance on the machines.

The only way of controlling the system (the `actuator`) is by adjusting the number and type of jobs that *CiGri* can submit to *OAR*: the number of jobs submitted at each time stamp but also about the type of the jobs (*I/O* heavy or light).

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

III. Proposed Solution

The work done in [13] used a simplistic approximation of the behavior of the system coupled with an MPC controller. However, experiments showed that the model is far from reality and, does not scale up and behaves incorrectly when the size of the cluster is above 15-20 nodes. Hence, we decided to implement a reaction-based feedback controller rather than a model-based to avoid faulty and imprecise models.

We present in this section, three solutions to our problem: two are "administrator friendly" solutions as they bring slight improvements to the existing solution but have the merit of being easily deployable and very comprehensible by the *CiGri* administrators (Section III-A) and an improved solution which is more complex as it requires more in depth knowledge (Section III-C).

A. Administrator friendly solutions

We present in this section two solutions which have the merit of being easily deployable in the production environment and very comprehensible by the *CiGri* administrators.

1) Scanning Algorithm: A way to regulate the size of the *CiGri* submission is to submit several potential sizes during the scanning phase: these sizes are defined by the administators of the system (e.g., 0 job, 10 jobs, 20 jobs, etc).After computing a performance index for each submission size based on a cost function (see Equation 1) taking into account the cluster usage and the load of the file-server, we pick the one with the best performance.

$$J_t = \alpha \times UtilCluster_t + (1 - \alpha) \times UtilLoad_t \quad (1)$$

where $UtilCluster$ is the number of resources utilized and $UtilLoad$ represents the load of the file-server at time t.

We would then keep submitting this number of jobs for the rest of the campaign or until the variations of the system reach a threshold value. When this happens we would start again the scanning phase.

Although this solution is independent on the cluster as well as job type and size it does not tackle directly our biggest problem which is the overloading of the file-server.

2) Controller for the load of the fileserver: A solution to manage the load of the file-server is to regulate directly the submission size of *CiGri* based on the current value of the file-server load. Here we use a simple Integral Controller which would compute the submission size based on the error between the actual load and the reference value from the requirements of the administrators.

B. Limitations of the proposed solutions

Originally, when *CiGri* sends jobs to the *OAR* scheduler, those jobs are part of the same campaign, or *Bag-of-Tasks* application. We make the natural hypothesis that every job from the same campaign have a similar behavior, and thus produce a similar impact on the *I/O* load with similar execution time. Nevertheless, jobs of different campaigns can have different *I/O* operations. Based on the campaign description we can define these campaigns as *I/O heavy* or *I/O light*.

Let us imagine that we have two campaigns submitted to *CiGri*, one with high *I/O* load jobs and the other with light *I/O* load jobs. If we suppose that we are able to regulate perfectly the number of jobs submitted for a single *I/O heavy* campaign to keep the load of the cluster under the reference load, there will be a "gap" between the actual load of the cluster and the target load value. We could exploit this "gap" by submitting jobs that have a smaller impact on the load (i.e., jobs from a *I/O light* campaign), and thus use more cluster resources while keeping the load of the fileserver under control.

However, it is not enough to only measure the load of the file-server. Indeed, we can have situations where all the resources of the cluster are being used by the premium users, and the load of the fileserver is low. If we only measure the

load, *CiGri* would send more and more jobs to *OAR* even though there are no idle resources. This is why we also need to have a sensor for the cluster side: the length of the *OAR* waiting queue.

C. Improved Solution: Biphasic Control

In our approach, we suppose that the *CiGri* users are able to categorize their campaign into the two categories (*I/O heavy* and *I/O light*) at the moment of the submission.

As many campaigns are executed in parallel we can suppose that in each set of jobs submitted from *CiGri* to *OAR*, we have jobs at least from two distinct campaigns: an *I/O heavy* campaign and an *I/O light* campaign.

There are thus two knobs, or parameters, that we can alter at each submission to control our system:

- The **total number of jobs in a submission** (i.e. number of jobs from *I/O heavy* campaign + number of jobs from *I/O light* campaign) sent from *CiGri* to *OAR*
- The **proportion of jobs from each campaign** in the set of jobs to submit (*% I/O heavy* in the total number of submitted jobs).

As we saw in the previous Section, by considering jobs with different *I/O* loads, we should be able to control more precisely the load on the file-server. We will thus design a strategy using this observation with two phases of execution. First, the controller will increase the number of jobs in the submission until it reaches a load "close" to the desired load value. Once the load is close enough to the reference value, it will keep the submission size constant and change the percentage of *I/O heavy* jobs in the submission in order to have a finer control on the load. We say that the load is close to the reference value if the absolute value of the regulation error is less than a threshold value (named T in the following).

However, as seen in the Section III-B, we also need a controller regulating the size of the *OAR* waiting queue to avoid overloading the scheduler and be more responsive in our control and avoid pathological situations.

We thus have two controllers that can change the submission size, one sensing the file-server load, the other sensing the *OAR* waiting queue length. Those two controllers are called at every submission and each of them will return a value for the next *CiGri* submission size. To satisfy the requirements of both controllers (length of the OAR waiting queue (WQ) and load of the file-server) we pick the minimum value between the two. Along it, we have a third controller on the percentage of *I/O heavy* in the *CiGri* submission.

Figure 2 gives a high level view of the proposed strategy. The control algorithm implemented is given below (see Algorithm 2) where: k_p is the gain of the controller on percentage of I/O heavy jobs, k_w the gain of the controller on the size of the waiting queue and k_s the gain of the controller on the submission size from the file-server load.

As a first approach, we used Integral Controllers. There is a large possibility of choice for the control parameters, some guidelines and necessary stability conditions are given below:

- T: experiments have shown that this value should not be too big (as it will lead to large oscillations and overload) nor too small (as it will slow down too much the campaign execution). In our experiments, we set this value to $\pm 33\%$ of the load reference (which is chosen equal to 3 in the experiments of Section IV).
- k_p: the amount of jobs that can change from heavy to light or from light to heavy needs to be an integer ≥ 1. As the error is less than T by definition, we thus have $\frac{1}{nb\_jobs} \leq k_p|e| \leq k_pT$. As our gain can not be implemented in an adaptive way we supposed an upper bound $nb\_jobs = 10$ thus we can approximate $k_p = \frac{1}{nb\_jobs \times T} = 0.1$.
- k_w: based on numerous experiments conducted we opted for $k_w = \frac{1}{2}$ as it seems to be reactive enough.
- k_s: in this case the error is $\geq T$ and we would like to have a change in the submission size of at least one job. Hence, $k_s \leq \frac{1}{T}$, to be more conservative we set $k_s = \frac{1}{2}$.

Algorithm 2: Control Algorithm of Biphasic Control

Input: $percentage$ (init. 50), nb_jobs (init. 0), T (Constant)

$errorLoad = reference - load$;
$errorWQ = referenceWQ - length(WQ)$;
if $|errorLoad| < T$ **then**
  $percentage = percentage + k_p \times errorLoad$;
else
  $nb\_jobs\_load = nb\_jobs\_load + k_s \times errorLoad$;
  $nb\_jobs\_WQ = nb\_jobs\_WQ + k_w \times errorWQ$;
  $nb\_jobs = \min(nb\_jobs\_load, nb\_jobs\_WQ)$;
end
return nb_jobs with $percentage\%$ of *I/O heavy* jobs;

As we do not have a model of the system, we are nevertheless unable to perform a rigorous stability analysis of the controller.

IV. Experimental Results

A. Experimental Goals

The goal of the following experiments is to compare the solution proposed in Section III with the solutions implemented in previous works [13] and presented in III-A. We will measure the number of jobs sent by *CiGri* at each submission, the load of the file-server, as well as the number of running jobs in the cluster. Note that we have two objectives. **We want to regulate the load of the file-server along a predefined value while having the maximum cluster utilization**.

The two hypotheses in our experiments are the following:

- Jobs from the same *CiGri* campaign have similar behavior in execution time and in *I/O*.
- The time between two *CiGri* submission is similar to the execution time of the jobs.

B. Experimental Setup

To test the controller described in Section III, we used the following setup: *(i)* One *CiGri* Server, *(ii)* One *OAR* Server (V3), *(iii)* One Fileserver (implemented with NFS), *(iv)* One Cluster of 100 *OAR* resources. The experiments were done by reserving nodes on the Grisou Cluster from Grid'5000 [6] which is a shared French testbed for experimental research in

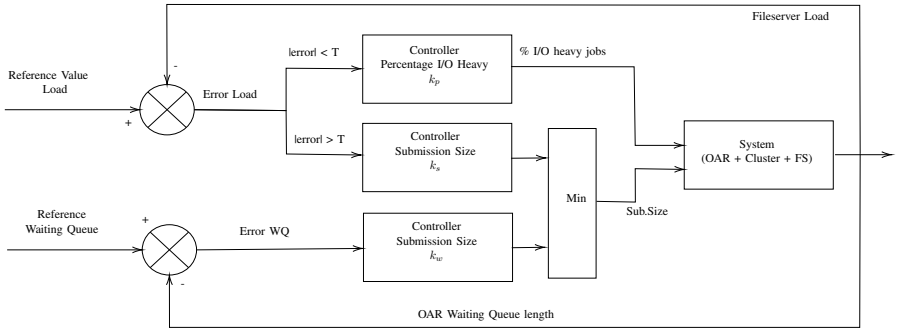

Fig. 2: Overview of the Proposed Solution

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 a *I/O heavy* campaign to *CiGri* containing 1000 jobs each, and in the case of our solution, also a *I/O light* campaign. The jobs sleep for 30 seconds then write to the file-server a file of 100Mb. The jobs from the *I/O light* campaign, sleep for 30 seconds and write a 10Mb file. The choice for the sleep jobs come from the fact that we are using a single real Grid'5000 node for emulating a cluster of 100 nodes in our experiment. Hence, we cannot have jobs running real workload otherwise the node emulating the OAR cluster might get overloaded and thus be responsible for some delay.

We regulated the load of the file-server around the **reference value 3**, with a threshold value of $\pm 33\%$ (i.e. 1). For the solution presented in Section III-C this means that with load values between 0 and 2, and above 4, we are regulating the *CiGri* submission by changing the number of jobs. When the load is between 2 and 4, we regulate using the percentage of *I/O heavy* jobs. In practice, the cluster administrators would choose the values based on their experience. In reality, 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 a acceptable overhead for their system and users.

C. Experimental Results

The control objective is first to avoid the file-server overload and, second, to use as many resources as possible.

Figure 3 shows the evolution of the different variables of the controller (percentage of *I/O heavy* jobs and submission size) as well as the load of the file-server. We can see that without any perturbation from the priority users, we are able to control relatively closely the load to the reference value.

Figure 4 depicts a comparison for the load regulation between the original *CiGri* solution and our solution. The dashed line represents the load due to premium users. And the black line the reference value for the load. The original solution has no feedback mechanism and thus leads to high load values. Our solution keeps the load close to the reference value. However, it can be disturbed by perturbations. This is due to the simplicity of using Integral Controllers, and we think could be solved by adding Integrative and/or Derivative terms.

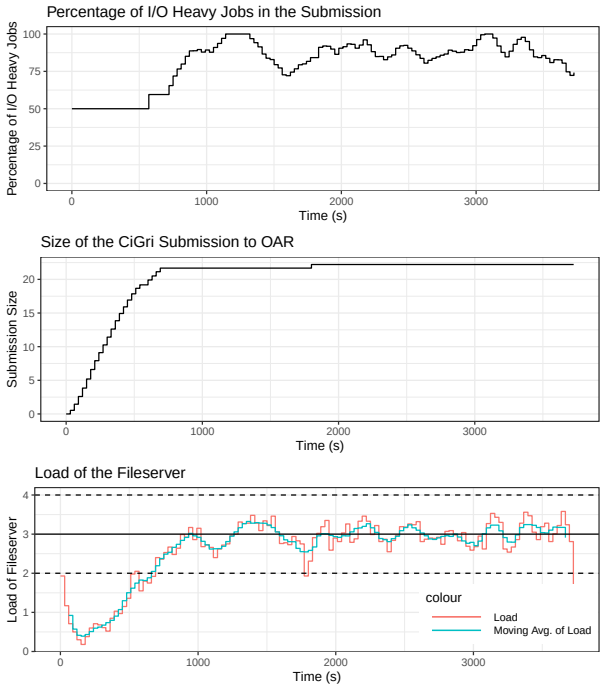

Fig. 3: Evolution of the Fileserver Load for the Biphasic control with no Perturbation from the high Priority Users

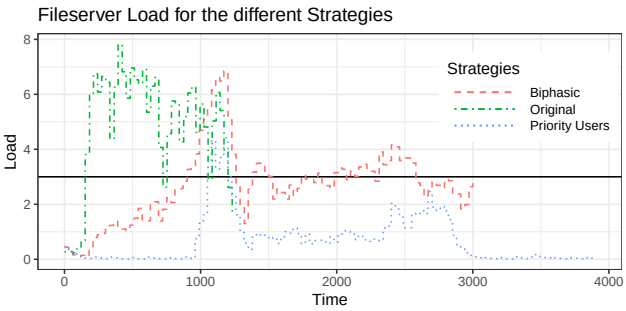

Fig. 4: Evolution of the File-server Load with Perturbation from high Priority Users

Strategy	Reference	Feedback	File-server Overloaded with Priority Users (%)			Cluster Usage (%)		Mean Distance to Ref (95%)	
			Load > Ref	Load > Ref + 33%	Load > Ref + 66%	without	with	without	with
Original	Sec. II-A4	✗	80.49%	75.61%	60.98%	63.7	73.8	✗	✗
Scanning	Sec. III-A1	✓	17.86%	7.14%	3.57%	13.8	16.1	1.65 ± 0.16	1.63 ± 0.14
MPC	[13]	✓	89.02%	84.15%	59.76%	28.2	44.5	1.83 ± 0.16	2.64 ± 0.26
Simple Control	Sec. III-A2	✓	25.93%	11.11%	7.40%	17.4	23.0	0.56 ± 0.13	1.08 ± 0.22
Biphasic Control	Sec. III-C	✓	37%	10%	7%	19.6	25.8	0.57 ± 0.13	1.02 ± 0.18

TABLE I: Proportion time spend overloaded (smaller is better), cluster usage (larger is better) and mean of the distance to the reference value (smaller is better) with and without priority users.

Table I gathers the different metrics to evaluate the strategies. We can notice the cluster usage as a percentage of the total cluster as well as the time when the file-server is overloaded as a percentage of total campaign running time.

We can see that the original *CiGri* solution yields the best usage of the cluster, but leads to large variations of the load. Our solution (*Biphasic Control*) returns the best cluster usage for a low mean distance to the reference. We explain the lack of performance of the MPC controller due to the imprecision of the model. Note that the values for the cluster usage depends on the reference value chosen (in our case 3), thus different reference values would yield different cluster usages. It is interesting to notice that for similar controls of the file-server load, our solution returns a better cluster utilization compared to the *Simple Control*, which shows the usefulness of considering *I/O heavy* and *I/O light* jobs in our approach.

V. Conclusion

We presented an autonomic approach for the harvesting of idle resources in a computing grid while not overloading the file-server. We proposed a strategy taking into account the *I/O* load from different campaigns to have a finer control. We validated our solution on synthetic behavior from high priority users and compared the results with alternative strategies. Moreover, all the solutions presented in Section III-A have the merit of being very appealing to system administrators by their simple understanding and implementation.

In perspectives, we are working at *(i)* improving the solution using different control methods (e.g., adaptive PID, Model Free Control), *(ii)* controlling several clusters from a single *CiGri*, *(iii)* quantify the perturbations on the file-server and *(iv)* investigate other metrics to control such as the energy consumption of the cluster.

References

[1] T. Abdelzaher, Y. Diao, J.L. Hellerstein, C. Lu, and X. Zhu. Introduction to control theory and its application to computing systems. In *Performance Modeling and Engineering*, pages 185–215. Springer, 2008.

[2] D.P. Anderson. BOINC: A System for Public-Resource Computing and Storage. In *Fifth IEEE/ACM International Workshop on Grid Computing*, pages 4–10, Pittsburgh, PA, USA, 2004. IEEE.

[3] 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 *IEEE International Symposium on Cluster Computing and the Grid, 2005.*, pages 776–783 Vol. 2, Cardiff, Wales, UK, 2005.

[4] S. Cerf, R. Bleuse, V. Reis, S. Perarnau, and E. Rutten. Sustaining Performance While Reducing Energy Consumption: A Control Theory Approach. In *Euro-Par*, Lecture Notes in Computer Science, August 2021.

[5] W. Cirne, F. Brasileiro, N. Andrade, L.B. Costa, A. Andrade, R. Novaes, and M. Mowbray. Labs of the World, Unite! *Journal of Grid Computing*, 4(3):225–246, September 2006.

[6] Balouek et a.l. Adding virtualization capabilities to the Grid'5000 testbed. In *Cloud Computing and Services Science*, volume 367 of *Communications in Computer and Information Science*, pages 3–20. Springer International Publishing, 2013.

[7] D. Ferrari and S. Zhou. An Empirical Investigation of Load Indices For Load Balancing Applications. In *Proceedings of Performance*, 1987.

[8] Y. Georgiou, O. Richard, and N. Capit. Evaluations of the Lightweight Grid CIGRI upon the Grid5000 Platform. In *Third IEEE International Conference on e-Science and Grid Computing*, pages 279–286, Bangalore, India, 2007.

[9] M. J. Litzkow, M. Livny, and M. W. Mutka. Condor-a hunter of idle workstations. Technical report, University of Wisconsin-Madison Dpt. of Computer Sciences, 1987.

[10] 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*, pages 347–352, Boston, MA, December 2017.

[11] E. Rutten, N. Marchand, and D. Simon. Feedback Control as MAPE-K Loop in Autonomic Computing. In Rogério de Lemos, David Garlan, Carlo Ghezzi, and Holger Giese, editors, *Software Engineering for Self-Adaptive Systems III. Assurances*, volume 9640, pages 349–373. Springer International Publishing, Cham, 2017.

[12] E. Stahl, A. Yabo, O. Richard, B. Bzeznik, B. Robu, and E. Rutten. Towards a control-theory approach for minimizing unsued grid resources. *AI-Science'18 - workshop on Autonomous Infrastructure for Science, with ACM HPDC 2018*, pages 1–8, June 2018.

[13] A. Yabo, B. Robu, O. Richard, B. Bzeznik, and E. Rutten. A control-theory approach for cluster autonomic management: maximizing usage while avoiding overload. In *IEEE Conference on Control Technology and Applications*, pages 189–195, Hong Kong, China, August 2019.