Collecte de ressources libres dans une grille en préservant le système de fichiers : une approche autonomique

Quentin Guilloteau, Olivier Richard, Eric Rutten, Bogdan Robu

Compas'2021 : Parallélisme / Architecture/ Système/ Temps Réel
MILC - Lyon, France, 5-9 juillet 2021

Collecte de ressources libres dans une grille en préservant le système de fichiers : une approche autonomique

Quentin Guilloteau[†], Olivier Richard[†], Eric Rutten[†], Bogdan Robu[*]

[†] Univ. Grenoble Alpes, Inria, CNRS, LIG, F-38000 Grenoble France
[*] Univ. Grenoble Alpes, CNRS, Grenoble INP, GIPSA-lab, F-38000 Grenoble France
[†] : Prénom.Nom@inria.fr, [*] : Bogdan.Robu@gipsa-lab.fr

Résumé
Les systèmes de Calcul Haute Performance font face à de plus en plus de variations dans leur comportement, que ce soit en performance, consommation d'énergie, etc. Cette imprédictiblité nous amène à considérer une approche autonomique à leur gestion, utilisant des boucles de rétro-action, et des outils de l'automatique. Nous présentons une application de cette approche dans le cas de *CiGri*, un intergiciel exploitant les ressources libres d'une grille de calculs, dans le but d'en maximiser l'utilisation, en y injectant des tâches *best-effort*. Nous nous focalisons ici sur le problème d'éviter que ces dernières ne surchargent le serveur de fichiers distribué.

Mots-clés : Calculs Hautes Performances, Gestion de ressources, Systèmes auto-adaptatifs, Informatique Autonomique, Théorie du Contrôle

---

1. Introduction

Les systèmes de calculs hautes performances (HPC) ont vu leur complexité exploser avec la croissance des piles logicielles, les variabilités des processeurs ou l'évolution dynamique de la charge réseau ou mémoire. Cette complexité amène des variations de leur comportement, et notamment en performances. Ainsi, il est de plus en plus difficile de prédire précisément leur évolution. Par exemple, les variations des temps de lecture/écriture de fichiers dans une grille de calcul, et donc de la charge du serveur de fichiers, entraînent des variations des temps d'exécution des tâches soumises par les utilisateurs.
Ce type de variations apparaît également dans le contexte de l'intergiciel *CiGri* [7]. Son but est d'utiliser les ressources libres d'un ensemble de grappes de calculs (ou *clusters*), en y injectant des tâches de priorité minimale (ou *best-effort*). Ces dernières proviennent d'applications dites *Bag-of-Tasks*, constituées d'un grand nombre de tâches indépendantes, donc propices à leur parallélisation. Un des problèmes est lié à la potentielle surcharge du serveur hébergeant le système de fichiers distribué (SFD) de la grappe. En effet, il est important que l'injection de tâches de *CiGri* dans la grappe n'affecte pas les performances des tâches des utilisateurs locaux, de priorité supérieure. Il faut donc contrôler l'injection de façon à l'assurer.
La gestion des systèmes avec de telles incertitudes de comportement doit être réalisée à l'exécution et sur la base de mesures de performances. Des décisions seront alors prises en fonction de ces mesures et des connaissances des administrateurs du système afin de respecter des garanties de comportement et de performances. Cet aspect auto-adaptatif nous amène donc à considérer les notions de boucles autonomiques de rétro-action et d'automatique et contrôle.

2. Contexte

2.1. Calculs Hautes Performances et collecte de ressources libres

2.1.1. Collecte de ressources libres

La genèse de cette sous-utilisation de ressources peut provenir de plusieurs facteurs (e.g. tâches terminant plus tôt que prévu, que cela soit dû à une estimation conservative pessimiste de l'utilisateur ou à une simple erreur d'exécution, caractère hétérogène des plateformes de calculs).
Récupérer au mieux cette puissance de calcul perdue est l'objectif de plusieurs projets de recherche. On peut notamment citer *BOINC* [2] qui a pour but d'utiliser les machines personnelles des utilisateurs pendant les périodes d'inactivité pour effectuer des calculs provenant de projets communautaires. L'approche prise par *BeBiDa* [11] a pour objectif d'utiliser les ressources libres d'une grappe *HPC* avec des tâches provenant du domaine des MegaDonnées (ou *Big-Data*). Dans le cas de *BeBiDa*, deux ordonnanceurs sont utilisés : un pour les tâches *HPC*, et l'autre pour les tâches MegaDonnées. Le second ne voit alors que les ressources non utilisées par les tâches *HPC*. *CiGri* a l'avantage d'être une solution intégrable sur n'importe quelle grille de calcul, car nécessitant uniquement une interface avec les ordonnanceurs des grappes.

2.1.2. L'intergiciel *CiGri*

Architecture générale

Dans l'approche *CiGri*, les utilisateurs soumettent leurs applications *Bag-of-Tasks* (ou *Campagnes*), qui sont transmises par partie de manière graduelle à l'ordonnanceur *OAR* [4] avec la priorité la plus basse pour être exécutées sur les ressources libres de la grille. Cela a l'avantage de pouvoir tuer ces tâches de basse priorité dans le cas d'un besoin de plus haute priorité.
La grille de calcul est composée de grappes qui regroupent des noeuds de calcul. Chaque grappe est gérée par un ordonnanceur *OAR* et a accès au serveur de fichiers.Les ordonnanceurs peuvent recevoir des tâches provenant des utilisateurs locaux (ou *Local Users*) ou de *CiGri*.
L'architecture du système est présentée dans la Figure 1. *CiGri* prend en entrée des applications dites de type *Bag-of-Tasks*, caractérisées comme étant composées d'un large ensemble de petites tâches indépendantes (e.g. application type *Monte-Carlo*).
Toutes les 30 secondes, *CiGri* appelle la fonction de contrôle de soumission qui déterminera combien de tâches seront envoyées à *OAR*. La solution, dite originelle, de *CiGri* utilise l'algorithme 2 en Annexe B.1, qui a le défaut d'attendre que toute la soumission précédente soit terminée avant de soumettre à nouveau, pouvant mener ainsi à des situations de sous-utilisation des ressources e.g. quand il ne reste que quelques tâches de la soumission précédente en train d'être exécutées, mais avec un grand nombre de noeuds libres dans la grille. Il serait alors préférable pour *CiGri* de soumettre des tâches pour utiliser ces ressources libres.

Prise en compte du Serveur de Fichiers

Toutes les tâches du domaine du *HPC* écrivent ou lisent dans des fichiers. Dans une grappe de calcul, une machine héberge le système de fichiers distribué (SFD) (ou *Distributed File-System*). Les noeuds de calculs vont lui soumettre des requêtes de lecture/écriture lorsque la tâche exécutée le demande. Cependant, les performances de ce serveur sont limitées, et un surplus de requêtes simultanées risque d'entraîner une surcharge amenant à une baisse de performances, et ainsi, influencer négativement le temps d'exécution des tâches des utilisateurs locaux prioritaires. **Nous voulons donc contrôler l'injection pour éviter la surcharge du DFS.**
La collecte de ressources dans une grille de calcul doit se faire de manière dynamique en prenant en compte l'état actuel du système (nombre de noeuds libres, charge du serveur de fi-

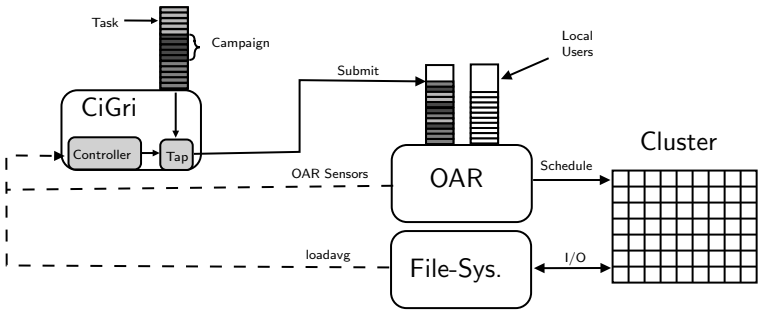

FIGURE 1 – Positionnement de l'intergiciel *CiGri* vis-à-vis de *OAR*, de la grappe et du système de fichiers distribué. Les capteurs utilisés par *CiGri* sont représentés en pointillés.

chiers, etc). Pour pallier aux variations dues à un tel système, nous nous sommes intéressés aux boucles de rétro-action afin de réguler le système en ligne, en faisant appel aux domaines de l'Informatique Autonomique et de l'Automatique.

2.2. Approche Autonomique

Le concept d'Informatique Autonomique (ou *Autonomic Computing*) [10] définit des systèmes capables de s'auto-adapter et de s'auto-réguler étant donné des objectifs de "haut niveau" donnés par les administrateurs. Il y a quatre aspects à l'Informatique Autonomique : (*i*) *l'Auto-Configuration*, (*ii*) *l'Auto-Optimisation*, (*iii*) *l'Auto-Réparation* et (*iv*) *l'Auto-Protection*. La boucle *MAPE-K* est l'outil central de l'Informatique Autonomique [9]. Elle est composée en cinq parties, présentées en Section A.2. Dans notre approche, la boucle *MAPE-K* est considérée du point de vue de la **théorie du contrôle** (i.e., l'automatique) [13, 6, 1]. Un exemple de boucle de contrôle est illustré en pointillés en Figure1 sur le cas de notre problème.
La régulation autonomique de l'injection de tâches dans *CiGri* a été considérée dans des travaux antérieurs : [14] présente un contrôleur Proportionnel-Intégrateur (PI) régulant la quantité de tâches envoyée de *CiGri* à *OAR* en se basant sur la taille de la file d'attente de l'ordonnanceur. Dans [15] un contrôleur dit *MPC* (*Model Predictive Control*) a été implémenté sur la base d'un modèle simple du système prenant en compte la taille de la file d'attente et la charge du SFD, mais ne passait pas à l'échelle . La suite de ce papier reprend et étend le contenu de [8] en proposant une nouvelle approche autonomique pour : ***(i)* minimiser les variations de la charge du SFD, *(ii)* puis collecter un maximum de ressources en gardant la charge du SFD sous contrôle.**

3. Proposition d'une solution autonomique prenant en compte le serveur de fichiers

3.1. Définition du problème

Le but de notre approche est de **minimiser la sous-utilisation de la grille** en injectant des tâches venant d'applications *Bag-of-Tasks* **tout en évitant toute surcharge du serveur de fichiers distribué**, pour les raisons données dans la Section 2.1.2. En contrôlant les performances de ce serveur, il est possible de limiter l'impact des tâches venant de *CiGri* sur celles des utilisateurs locaux de la grappe. Nous utilisons le capteur *UNIX* `loadavg` pour connaître la charge du SFD. Une règle empirique des administrateurs systèmes est qu'une machine est surchargée si la valeur retournée par `loadavg` est supérieure au nombre de coeurs de la-dite machine [5].
Dans la suite de ce papier, nous considérons uniquement l'injection de tâches provenant de *CiGri* dans une unique grappe.

Originellement, les soumissions de *CiGri* à *OAR* contiennent des tâches provenant de la même campagne. Nous faisons l'hypothèse que les tâches d'une même campagne ont un comportement similaire en temps d'exécution et en entrées-sorties (*E/S*). Cependant, d'une campagne à une autre, le poids en *E/S* peut être différent. Nous définissons alors deux types de campagnes : *E/S légère*, dont les tâches effectuent des opérations *E/S courtes* et *E/S lourde* : opérations *longues*. Supposons maintenant que nous avons deux campagnes soumises à *CiGri* ; une *E/S légère* et une *E/S lourde*. Soumettre des tâches provenant d'une seule campagne pourrait amener à une sous-utilisation de la charge disponible. L'ajout de tâches *E/S légère*s dans la soumission permet alors une meilleure utilisation de la charge ainsi que des ressources disponibles.

3.2. Solution proposée

Dans l'approche que nous proposons, l'utilisateur spécifie le type de sa campagne lors de la soumission (i.e. *E/S lourde* ou *E/S légère*). Chaque soumission de *CiGri* contient maintenant des tâches provenant d'une campagne *E/S lourde* et d'une *E/S légère*. Nous avons deux actionneurs :

— La **taille de la soumission** (i.e. le nombre de tâches)
— Le **pourcentage de tâches *E/S lourde*s** dans la soumission

Nous rappelons que l'objectif est de réguler la charge du SFD puis de collecter un maximum de ressources disponibles. La régulation de la charge du serveur de fichiers s'effectuera alors en deux phases. Dans un premier temps, le contrôleur augmentera le nombre de tâches dans la soumission jusqu'à atteindre une charge proche de la charge désirée par les administrateurs de la grappe (aussi appelée *Référence* en automatique). Une fois la charge "proche" de la valeur voulue, le contrôleur modifiera uniquement le pourcentage de tâches *E/S lourde*s pour se rapprocher de l'objectif avec une granularité plus fine. Nous allons utiliser un contrôleur pour réguler la taille de la soumission, et un second pour le pourcentage de tâches *E/S lourde*s. Tous deux capteront la charge du SFD, en coordination ils définissent la soumission .

Cependant, il n'est pas suffisant de mesurer uniquement la charge du serveur de fichiers. En effet, il se peut que l'on ait une situation où toutes les ressources de la grappe sont occupées, et que le serveur de fichiers ne soit pas surchargé. Si nous utilisons seulement une solution captant cette charge, *CiGri* continuerait à soumettre de plus en plus de tâches alors qu'il n'y a pas de ressources libres. C'est pourquoi nous avons rajouté un capteur de la taille de la file d'attente de *OAR* et l'avons intégré dans un contrôleur ayant pour but de réguler cette dernière. Ce capteur nous donne une estimation de la quantité de ressources libres dans la grappe. A chaque itération de *CiGri*, le contrôleur captant la charge du SFD et celui captant la taille de la file d'attente *OAR* renvoient une taille de soumission pour respecter leurs contraintes. Nous prenons finalement le minimum de ces tailles afin de respecter toutes ces contraintes. L'Algorithme 5 décrit plus formellement notre stratégie de régulation. En tant que première approche, nous utilisons des contrôleurs *Intégral*s (voir Section A.1). Le gain pour la phase régulant la taille de la soumission est choisi de telle sorte à avoir des variations d'au minimum une tâche (comme l'erreur est nécessairement supérieure à T dans l'algorithme 5). Pour la phase régulant le pourcentage de tâches *E/S lourde*s, le gain est choisi de manière à changer au moins une tâche *E/S légère* en *E/S lourde* (ou inversement) dans la soumission.

4. Résultats expérimentaux

4.1. Hypothèses

Dans la suite des expériences, nous faisons les hypothèses suivantes :

— Les tâches provenant de la même campagne *CiGri* ont des comportements similaires.
— La durée entre deux soumissions de *CiGri* est similaire aux temps d'exécution des tâches.

Stratégie	Algo.	Rétro-action	Proportion du temps avec le SFD surchargé (%)			Util. grappe (%)		Dist. moy. à la référence (95%)	
			Charge > Ref	Charge > Ref + 33%	Charge > Ref + 66%	absence	présence	absence	présence
Originelle	2	✗	80.49%	75.61%	60.98%	63.7	73.8	✗	✗
Scan	3	✓	17.86%	7.14%	3.57%	13.8	16.1	1.65 ± 0.16	1.63 ± 0.14
MPC	[15]	✓	89.02%	84.15%	59.76%	28.2	44.5	1.83 ± 0.16	2.64 ± 0.26
Contrôle Simple	4	✓	25.93%	11.11%	7.40%	17.4	23.0	0.56 ± 0.13	1.08 ± 0.22
Contrôle biphasé	5	✓	37%	10%	7%	19.6	25.8	0.57 ± 0.13	1.02 ± 0.18

TABLE 1 – Proportion du temps passé avec le SFD surchargé en présence d'utilisateurs locaux prioritaires (petit est mieux), utilisation de la grappe (grand est mieux) et moyenne des erreurs (petit est mieux) en fonction de la stratégie utilisée en présence (resp. absence) d'utilisateurs locaux prioritaires.

4.2. Dispositif expérimental

Les expériences ont été réalisées sur la grappe *Grisou* de la plate-forme *Grid'5000* [3]. Les machines de cette grappe ont 2 processeurs Intel Xeon E5-2630 v3 avec 8 coeurs/CPU et 128 Go de mémoire. Pour déployer notre système, nous avons utilisé 4 noeuds de la manière suivante : *(i)* un serveur *CiGri*, *(ii)* un serveur *OAR* (v3), *(iii)* un serveur de fichiers (NFS [12]) et *(iv)* une grappe de 100 ressources *OAR* ; ceci est résumé en figure 4 (en annexe C.1).

L'expérience consistait à soumettre pour les différentes stratégies une campagne *E/S lourde*, et dans le cas de la solution proposée, également une campagne *E/S légère*. Les tâches de la campagne *E/S lourde* (resp. *E/S légère*) dorment pendant 30 secondes puis écrivent un fichier de 100Mo (resp. 10Mo) dans le serveur de fichiers distribué. Le choix de faire dormir les tâches vient du fait que nous émulons une grappe *OAR* sur un simple noeud, et ne pouvons ainsi pas faire de vrais calculs sans que la machine ne devienne surchargée et impacte notre expérience en introduisant des délais.

Nous avons choisi de réguler la charge du SFD autour de la valeur 3 (en trait plein sur la figure 2), et avec une limite à $\pm 33\%$ de la référence (i.e. $T = 1$ dans l'algorithme 5). Cela signifie que lorsque la charge est entre 2 et 4 nous modifions le pourcentage de tâches *E/S lourde*s sans toucher à la taille de la soumission ; et inversement si la charge n'est pas entre 2 et 4. Ces valeurs sont choisies comme étant les objectifs de haut niveau du paradigme de l'Informatique Autonomique. En pratique, ce sont les administrateurs de la grappe qui, avec leurs connaissances du système, choisiraient les valeurs à la manière d'un système de monitoring.

4.3. Résultats

La table 1 rassemble les performances des différentes solutions (présentées en section B). Rappelons que l'objectif est de ne pas surcharger le SFD tout en collectant un maximum de ressources. Ainsi, nous souhaitons *(i)* minimiser les perturbations de la charge du SFD, c'est-à-dire l'écart moyen entre la charge et la référence à atteindre , *(ii)* puis utiliser un maximum de ressources en gardant la charge du SFD sous contrôle. On peut constater que la solution originelle de *CiGri* donne la meilleure utilisation des ressources , mais engendre de larges variations sur la charge du serveur de fichiers. Notre solution (*Contrôle biphasé*) quant à elle, renvoie la meilleure utilisation avec une erreur moyenne faible. Nous expliquons le manque de performances du *MPC* par l'imprécision du modèle utilisé. Notons que les valeurs d'utilisation sont liées à la valeur de référence choisie (dans notre cas : 3). Ainsi, différentes références nous donneront différents pourcentages d'utilisation. Il est intéressant de noter que pour des contrôles similaires de la charge du SFD, notre solution retourne une meilleure utilisation que le contrôle simple, montrant l'utilité de la seconde phase.

La figure 2 montre une comparaison de l'évolution de la charge du SFD pour la solution ori-

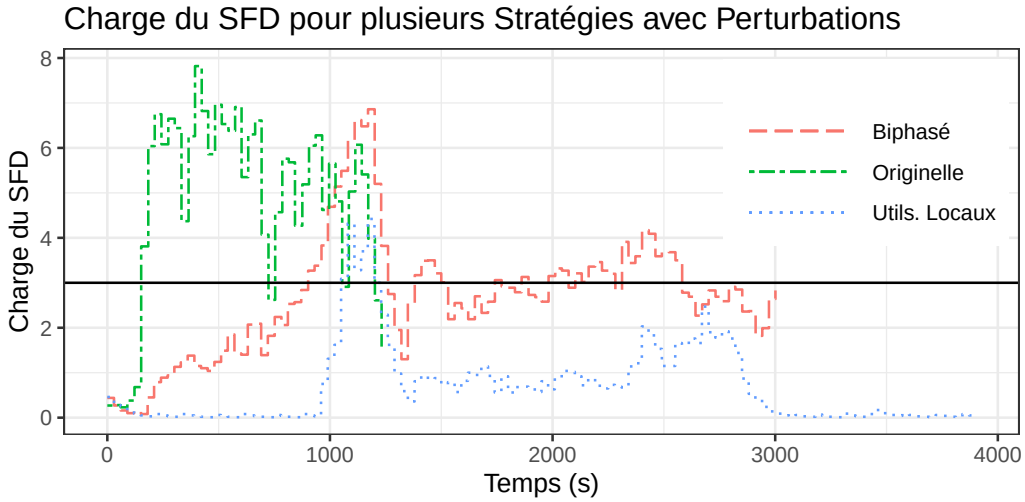

FIGURE 2 – Évolution de la charge du système de fichiers distribué (SFD) avec perturbations dues aux utilisateurs locaux prioritaires (charge seule des utilisateurs locaux en pointillés)

ginelle de *CiGri* (Algo. 2) et pour notre solution (Algo. 5). Les pointillés représentent la charge due aux utilisateurs locaux. La solution originelle n'ayant pas de rétro-action, elle mène à une charge élevée : donc elle exécute la totalité de la campagne rapidement, mais au prix d'une forte perturbation des tâches prioritaires. Notre solution quant à elle, régule la quantité de tâches par soumission, gardant la charge proche de la référence, donc perturbant moins les calculs prioritaires, en prenant plus longtemps pour compléter la campagne de tâches *best-effort*. Cette régulation peut être perturbée temporairement par les changements de charge des utilisateurs locaux, menant à une charge transitoire du SFD, mais ensuite la situation se restabilise. Cette perturbation est liée à la simplicité du contrôleur *Intégral*, avec des perspectives d'amélioration avec des contrôleurs plus élaborés, notamment *Proportionnel-Intégrateur-Dérivateur* (PID) ou avec des approches plus avancées de l'automatique.

5. Conclusion

Nous avons présenté une approche autonomique utilisant l'automatique pour minimiser la sous-utilisation d'une grappe de calculs en évitant de surcharger le serveur de fichiers. Nous avons proposé une stratégie prenant en compte la charge en *E/S* des tâches de différentes campagnes *CiGri* afin de réguler le système avec un grain plus fin. La validation expérimentale de notre solution, confrontée à d'autres algorithmes de soumission, montre que notre solution retourne la meilleure utilisation avec un minimum d'erreur sur la régulation de la charge.
En perspectives, nous travaillons *(i)* à l'amélioration du contrôleur par l'ajout de termes Intégrateurs et Dérivateurs pour former le *PID*, *(ii)* à évaluer l'impact de l'hypothèse que le temps d'exécution des tâches des campagnes est similaire à la période d'échantillonage de *CiGri*,*(iii)* réguler plusieurs grappes avec *CiGri*, *(iv)* quantifier précisément les perturbations sur le SFD, et enfin *(v)* à envisager d'autres métriques à réguler, telles que la consommation énergétique.

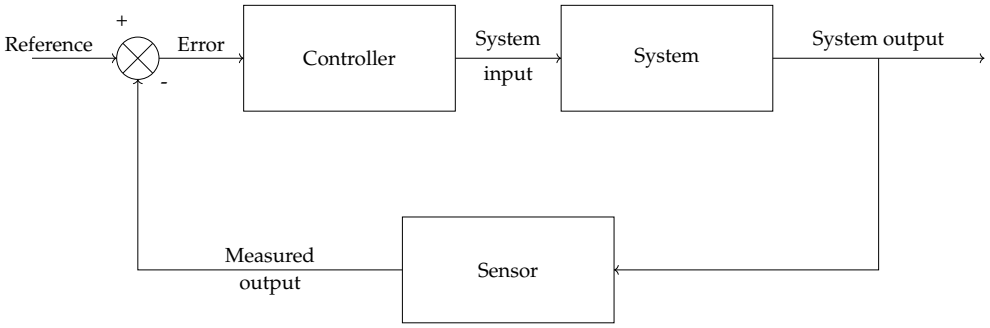

FIGURE 3 – Boucle de contrôle-commande

A. Informatique Autonomique et Contrôle

A.1. Contrôleur Intégral

De tels contrôleurs ont une réponse qui est proportionelle à l'erreur du système ; c'est-à-dire à la distance entre l'état actuel du système et l'état désiré. Si on note $u(t)$ la réponse du contrôleur au temps t, l'erreur du système $e(t)$, k_i le gain associé et Δt le temps entre deux itérations, alors le contrôleur est régi par l'équation 1 :

$$u(t+\Delta t) = u(t)+k_i \times e(t+\Delta t) = u(t)+k_i \times (reference - val\_capteur(t)) = u(0)+k_i \sum_{k=1}^{\frac{t}{\Delta t}+1} e(k\Delta t) \quad (1)$$

Algorithm 1: Algorithme pour un contrôleur intégral

Entrées : $erreur$
Variables Locales: u, k_i
$u = u + k_i \times erreur$;
return u;

A.2. La Boucle *MAPE-K* : détails

- ***M**onitor* : durant cette phase, les capteurs du système pistent les variations des variables mesurées (e.g. charge du serveur de fichiers). Selon les variations observées, cette phase notifie la suivante de la nécessité d'un changement de politique.
- ***A**nalyse* : en se basant sur les objectifs donnés et sur les mesures retournées par *Monitor*, la phase d'analyse conclut si un changement de configuration dans le système est nécessaire pour répondre aux objectifs. Si c'est le cas, cette phase va notifier la suivante.
- ***P**lan* : une fois la notification de *Analyse* reçue, le planificateur va décider des actions à mener. Cette phase peur également interroger la base de connaissance du système (ou ***K**nowledge*). Une fois le nouveau plan choisi, la phase suivante est notifiée.
- ***E**xecute* : pendant cette phase, le plan va être exécuté. Les actions de haut niveau vont être interprétées puis exécutées à travers une *API* de bas-niveau.
- ***K**nowledge* : cela contient les connaissances (objectifs, états, configurations, base de données, etc) sur le système et est partagée par toutes les phases.

B. Algorithmes

B.1. Fonction Originelle de *CiGri*

L'idée originelle pour la soumission de *CiGri* est de soumettre les tâches à la manière d'un *robinet*.

1. *CiGri* ouvre alors le robinet pendant un certain temps, i.e. soumet un certain nombre de tâches
2. puis le ferme jusqu'à ce que toute l'eau ait été évacuée, i.e. tâches exécutées par la grappe
3. *CiGri* ouvre a nouveau le robinet.

Cette solution est décrite plus formellement dans l'algorithme 2.

Algorithm 2: Algorithme de Soumission de *CiGri*

Entrées: nb_jobs, increase_factor
if *pas de tâches de CiGri en train d'exécuter* **then**
    nb_jobs = min(nb_jobs × increase_factor, 100);
    return nb_jobs;
else
    return 0;
end

B.2. Algorithme de Scan

L'idée de cet algorithme est de tester un nombre fini de tailles de soumissions, puis de choisir celle qui renvoie les meilleures performances (basées sur une fonction de coût prenant en compte l'utilisation de la grappe, la charge maximale du SFD pendant la soumission, etc). L'hypothèse principale de cette solution est que les variations du système sont plus lentes que le temps nécessaire pour tester toutes les valeurs de tailles possibles. Cette solution est décrite dans l'algorithme 3.

Algorithm 3: Algorithme de Scan

Variables Locales: iteration_scan, phase_de_scan
if *variations système* > ε *et pas* phase_de_scan **then**
    phase_de_scan = true;
    iteration_scan = 0;
end
if phase_de_scan **then**
    soumet tailles[iteration_scan] tâches;
    calcule perf sur le système;
    iteration_scan + +;
    if iteration_scan == taille(tailles) **then**
        phase_de_scan = false;
        iteration_scan = 0;
    end
    return
else
    champion = tailles[i] avec meilleure perf;
    return champion;
end

Pour calculer les performances de chaque taille de soumissions, nous utilisions une fonction de

coût J décrite en équation 2. Cette dernière prend en compte le taux d'utilisation de la grappe pendant la soumission ainsi que la charge du SFD. Chacune de ces quantités est alors normalisées entre 0 et 1. Nous introduisons un paramètre α permettant de pondérer une quantité par rapport à l'autre.

$$J = \alpha \frac{r_{max} - r_t}{r_{max}} + (1 - \alpha) S(f_{ref} - f_t) \in [0, 1] \tag{2}$$

r_{max} et r_t représentent respectivement le nombre de ressources total de la grappe et le nombre de ressources occupées au temps t. f_{ref} est la référence de la charge ; donnée par les administrateurs de la grappe, et f_t est la charge du SFD au temps t.
La charge du SFD est mesurée avec le capteur `loadavg` qui, par définition, a de l'inertie. Ainsi, si la charge maximale pour une unique soumission pendant la phase se scan retourne une charge légèrement supérieure à la référence (f_{ref}), alors des soumissions consécutives de cette soumission pourrait, par cumul d'inertie, surcharger le SFD. C'est pourquoi nous introduisons la fonction S qui a pour but de favoriser les soumissions dont la charge maximale est inférieure à et proche de la référence.

$$S(x) = \begin{cases} 2 \times \left(\frac{1}{1+e^{-x}} - \frac{1}{2}\right) & \text{si } x \leq 0 \\ 2 \times \left(\frac{1}{1+e^{-4x}} - \frac{1}{2}\right) & \text{si } x > 0 \end{cases} \tag{3}$$

B.3. Algorithme de Contrôle Simple

Il est également possible de réguler la charge du serveur de fichiers en utilisant un simple contrôleur captant la charge du serveur de fichiers. Pour comparer avec la solution présentée en Section B.3, nous utilisons également un contrôleur Intégral (voir Section A.1). Cette solution est décrite dans l'algorithme 4.

Algorithm 4: Fonction de Soumission Contrôle Simple

$erreur = reference - loadavg$;
nb_taches_charge = $controleur\_nb\_taches\_charge(erreur)$;
$nb\_taches\_attente$ = $controleur\_nb\_taches\_attente$(taille file attente *OAR*);
nb_taches = $\min(nb\_taches\_charge, nb\_taches\_attente)$;
return nb_taches;

De la même façon que pour l'algorithme 5, et expliqué en section , il n'est pas suffisant de uniquement capter la charge du SFD. C'est pourquoi nous captons également la taille de la file d'attente *OAR* afin d'éviter toute accumulation de tâches dans la file d'attente, compromettant la réactivité de la régulation.

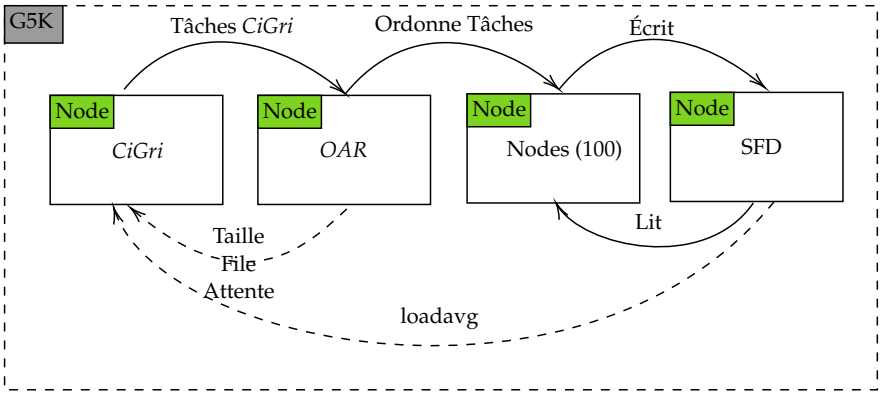

FIGURE 4 – Représentation du Dispositif Expérimental sur la plateforme Grid'5000

B.4. Algorithme de Contrôle Biphasé

Algorithm 5: Fonction de soumission pour Contrôle Biphasé

Entrées: pourcentage (initialement 50), nb_taches (initialement 0), T (Constante)
erreur = reference − *loadavg*;
if |erreur| < T **then**
  pourcentage = controleur_pourcentage(erreur);
else
  nb_taches_charge = controleur_nb_taches_charge(erreur);
  nb_taches_attente = controleur_nb_taches_attente(taille file attente *OAR*);
  nb_taches = min(nb_taches_charge, nb_taches_attente);
end
return nb_taches avec pourcentage% de tâches *E/S lourde*s ;

C. Expériences

C.1. Dispositif Expérimental

Le dispositif expérimental est représenté en Figure 4. Chacun des noeuds utilisés héberge un des composants de l'expérience. Le serveur *OAR* enregistre les ressources présentes sur le noeud représentant la grappe. Ce dernier monte, via NFS, le système de fichiers distribué (SFD) se trouvant sur un noeud dédié. Le dernier noeud héberge le serveur *CiGri* qui enregistre la grappe *OAR* afin de pouvoir lui soumettre des tâches *Best-effort*.

Bibliographie

1. Abdelzaher (T.), Diao (Y.), Hellerstein (J. L.), Lu (C.) et Zhu (X.). – Introduction to control theory and its application to computing systems. *In : Performance Modeling and Engineering*, pp. 185–215. – Springer, 2008.
2. Anderson (D.). – BOINC : A System for Public-Resource Computing and Storage. – In *Fifth IEEE/ACM International Workshop on Grid Computing*, pp. 4–10, Pittsburgh, PA, USA, 2004. IEEE.
3. Balouek (D.), Carpen Amarie (A.), Charrier (G.), Desprez (F.), Jeannot (E.), Jeanvoine (E.), Lèbre (A.), Margery (D.), Niclausse (N.), Nussbaum (L.), Richard (O.), Pérez (C.), Ques-

nel (F.), Rohr (C.) et Sarzyniec (L.). – Adding virtualization capabilities to the Grid'5000 testbed. *In : Cloud Computing and Services Science*, éd. par Ivanov (I. I.), van Sinderen (M.), Leymann (F.) et Shan (T.), pp. 3–20. – Springer International Publishing, 2013.

4. Capit (N.), Da Costa (G.), Georgiou (Y.), Huard (G.), Martin (C.), Mounie (G.), Neyron (P.) et Richard (O.). – A batch scheduler with high level components. – In *CCGrid 2005. IEEE International Symposium on Cluster Computing and the Grid, 2005.*, pp. 776–783 Vol. 2, Cardiff, Wales, UK, 2005. IEEE.
5. Ferrari (D.) et Zhou (S.). – An Empirical Investigation of Load Indices For Load Balancing Applications. *Proceedings of Performance '87*, vol. the 12th International Symposium on Computer Performance Modeling, nMeasurement, and Evaluation, 1988, pp. 515–528.
6. Filieri (A.), Maggio (M.), Angelopoulos (K.), D'Ippolito (N.), Gerostathopoulos (I.), Hempel (A. B.), Hoffmann (H.), Jamshidi (P.), Kalyvianaki (E.), Klein (C.), Krikava (F.), Misailovic (S.), Papadopoulos (A. V.), Ray (S.), Sharifloo (A. M.), Shevtsov (S.), Ujma (M.) et Vogel (T.). – Software engineering meets control theory. – In *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pp. 71–82, 2015.
7. Georgiou (Y.), Richard (O.) et Capit (N.). – Evaluations of the Lightweight Grid CIGRI upon the Grid5000 Platform. – In *Third IEEE International Conference on e-Science and Grid Computing (e-Science 2007)*, pp. 279–286, Bangalore, India, 2007. IEEE.
8. Guilloteau (Q.). – *Minimizing Cluster Under-use using a Control-Based Approach*. – Internship report, Grenoble INP Ensimag ; Université Grenoble Alpes, juin 2020.
9. Iosup (A.), Zhu (X.), Merchant (A.), Kalyvianaki (E.), Maggio (M.), Spinner (S.), Abdelzaher (T.), Mengshoel (O.) et Bouchenak (S.). – Self-Awareness of Cloud Applications. *arXiv :1611.00323 [cs]*, novembre 2016.
10. Kephart (J. O.) et Chess (D. M.). – The Vision of Autonomic Computing. *IEEE Computer*, vol. 36, janvier 2003, pp. 41–50.
11. Mercier (M.), Glesser (D.), Georgiou (Y.) et Richard (O.). – Big data and HPC collocation : Using HPC idle resources for Big Data analytics. – In *2017 IEEE International Conference on Big Data (Big Data)*, pp. 347–352, Boston, MA, décembre 2017. IEEE.
12. Pawlowski (B.), Noveck (D.), Robinson (D.) et Thurlow (R.). – The nfs version 4 protocol. – In *In Proceedings of the 2nd International System Administration and Networking Conference (SANE 2000*, 2000.
13. Rutten (E.), Marchand (N.) et Simon (D.). – Feedback Control as MAPE-K Loop in Autonomic Computing. *In : Software Engineering for Self-Adaptive Systems III. Assurances*, éd. par de Lemos (R.), Garlan (D.), Ghezzi (C.) et Giese (H.), pp. 349–373. – Cham, Springer International Publishing, 2017.
14. Stahl (E.), Yabo (A.), Richard (O.), Bzeznik (B.), Robu (B.) et Rutten (E.). – Towards a control-theory approach for minimizing unsued grid resources. *AI-Science'18 - workshop on Autonomous Infrastructure for Science, in conjunction with the ACM HPDC 2018*, juin 2018, pp. 1–8.
15. Yabo (A. G.), Robu (B.), Richard (O.), Bzeznik (B.) et Rutten (E.). – A control-theory approach for cluster autonomic management : maximizing usage while avoiding overload. – In *2019 IEEE Conference on Control Technology and Applications (CCTA)*, pp. 189–195, Hong Kong, China, août 2019. IEEE.