Comment estimer la consommation énergétique de processus avec seulement un wattmètre (et un solveur) ?

Valentin Lorentz, Laurent Lefevre, Gilles Fedak — valentin.lorentz at ens-lyon.org
{laurent.lefevre, gilles.fedak}@inria.fr
ÉNS de Lyon – Inria – Université de Lyon – LIP (UMR CNRS, INRIA, ENSL, UCBL)

Résumé
Nous proposons une solution entièrement automatique pour estimer la consommation d'énergie de processus exécutés sur une machine. Cette solution ne demande aucune configuration spécifique à la machine, et nécessite uniquement une mesure au niveau de son alimentation électrique. Cette solution combine un wattmètre physique (à la prise) et un framework logiciel (solveur) capable d'estimer automatiquement le coût électrique d'un processus ou d'une application en corrélant les variations de consommation électrique avec les intervalles de vie des processus.

Mots-clés : Wattmètre, mesure d'énergie, profil énergétique, programme linéaire

---

1. Introduction

1.1. Motivation

Dans le but d'optimiser la consommation énergétique d'un centre de calcul, il est important de connaitre la consommation des différentes machines dans ce centre, ainsi que la répartition de la consommation des calculs exécutés sur ces machines.
Ces machines étant de plus en plus denses et hétérogènes (multi-cœurs, GPUs, coprocesseurs, ...), il devient nécessaire d'effectuer des mesures de consommation à la prise pour chaque ressource à cause de l'hétérogénéité de la consommation électrique [9]. Mais le défi est alors d'extraire et d'estimer la consommation électrique d'un processus au milieu du bruit des autres processus.
Il existe de nombreux frameworks pour trouver le coût électrique d'un processus ou d'une machine virtuelle. Mais, la plupart reposent sur de la calibration complexe dépendante des architectures ; ce qui demande trop de temps dans le cadre de machines hétérogènes. De plus, une fois calibrées ce sont des solutions entièrement logicielles, qui ignorent donc la variabilité de la consommation réelle.

1.2. Consommations statique et dynamique

La consommation électrique d'un ordinateur se divise en deux parties : consommations statique et dynamique [6, 9, 11]. La première correspond à la puissance électrique consommée (watts) du simple fait d'être allumé, même si aucun processus n'est exécuté sur l'ordinateur. La seconde quant à elle est due aux activités effectuées (calculs, accès mémoire ou disque, réseau, ...).
Dans cet article nous nous intéressons à déterminer la consommation individuelle de processus « intéressants », c'est-à-dire des processus qui sont responsables d'une part importante de la consommation dynamique. Nous appelons consommation Idle la consommation du système sans ces processus, c'est-à-dire la somme de la consommation statique et de la consommation des processus négligeables.

Il est cependant possible d'attribuer la puissance Idle à des processus, par exemple de façon uniforme, ou au prorata de leur consommation dynamique.

1.3. Définitions de la consommation d'un processus

Il n'y a pas de définition unique de la consommation d'un processus, car une telle définition dépend du contexte dans lequel on veut s'en servir.
Par exemple, la consommation dynamique d'une machine sur laquelle on exécute deux processus peut être supérieure à la consommation dynamique de la même machine si on exécutait les deux processus l'un après l'autre à cause de leur compétition pour l'accès aux caches et des changements de contexte d'exécution. Ou, inversement, elle peut être inférieure à la somme des consommations des deux processus indépendemment si les deux processus peuvent mettre en commun des opérations coûteuses grâce à un cache du système d'exploitation (par exemple le cache disque).
D'autre part, on a naturellement tendance à ne considérer que la consommation dynamique pour définir la consommation d'un processus, car c'est une augmentation de puissance directement due au processus. Mais si exécuter un processus demande de démarrer une nouvelle machine dans le centre de calcul (parce qu'il nécessite plus de RAM qu'il n'y en a de disponible sur les autres, parce que les autres CPU sont déjà utilisés, etc.), alors la consommation Idle de cette machine serait bien à imputer à ce processus dans le cadre d'une facturation de l'électricité consommée.
C'est pourquoi nous parlerons dans la suite de politique d'attribution de l'énergie, plutôt que de méthode ou technique pour calculer la consommation d'énergie.

Dans cet article, nous commençons par présenter les travaux existants (section 2), puis, nous exposons différentes politiques d'attribution afin de reporter et de répartir la puissance électrique mesurée (section 3). Ensuite, nous présentons la "Politique Frontière", basée sur les variations de consommation d'énergie mesurée à la source (section 3.3). Enfin, nous présentons un exemple réel du calcul de cette politique, et évaluons le coût en temps de calcul (section 4). La section 5 conclue cet article et présente les travaux futurs.

2. Travaux connexes

PowerAPI[2], WattsKit [3] (basé sur PowerAPI), pTop[7], PowerTop, ainsi que la commande `top` de macOS sont des solutions entièrement logicielles pour déterminer la consommation d'un processus. Cependant, elles requièrent une calibration fine pour chacun des ordinateurs sur lesquelles elles sont utilisées, ce qui n'est pas souhaitable dans un environnement hétérogène.
De plus, ces outils donnent la consommation d'un processus comme étant une mesure objective ; alors que, comme nous le verrons dans la suite, il existe plusieurs façons de la définir. Notamment, ils utilisent des compteurs matériels (nombre de cache hits, cache misses, nombre d'instructions exécutées, ...), ce qui fait qu'ils ignorent la consommation statique et la consommation Idle.
Enfin, ce sont des solutions « intrusives », car elles nécessitent d'être exécutées sur les mêmes machines que les processus, ce qui demande de pouvoir les y installer, et implique une consommation supplémentaire d'énergie (par exemple la consommation de CPU de pTop est de 3%[7]).

3. Politiques d'attribution de puissance

3.1. Catalogue

Une politique simple pour définir la consommation énergétique d'un processus est d'avoir un catalogue, c'est-à-dire une correspondance entre les programmes et la puissance ou l'énergie qu'ils consomment. Elle a aussi l'avantage d'être facilement exprimable dans le cadre d'une facturation.
Cependant, cette méthode demande une calibration préalable pour déterminer la consommation

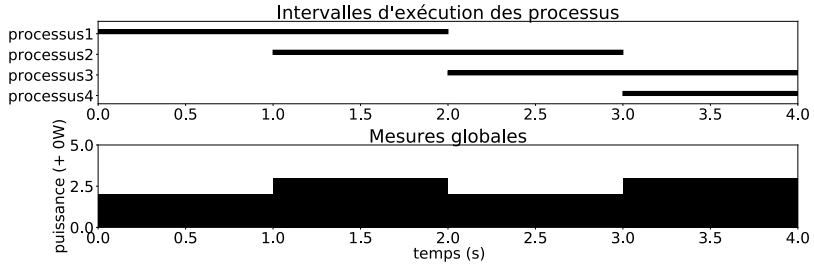

FIGURE 1 – Exemple de traces d'exécution et de mesures. La moitié haute donne les intervalles d'exécution de chaque processus, et la moitié basse donne la puissance consommée.

du programme, mais cela ne prend pas en compte toute la variabilité de l'environnement. Notamment, il faut construire ce catalogue pour chacune des machines sur lequel le programme peut être exécuté (même si elles sont a priori du même modèle [9]), le DVFS [1], la taille et le contenu de l'entrée, etc. Par exemple, en exécutant `gzip` sur un fichier d'une taille donnée, il ne consommera pas la même énergie selon que le fichier soit rempli de zéros ou d'octets aléatoires (respectivement $2 \cdot 10^2$J et $1 \cdot 10^3$J sur une machine Taurus de Grid5000 [2]).

3.2. Partage uniforme

Cette méthode est orthogonale au catalogue : au lieu d'utiliser un catalogue pour calculer la puissance consommée par un processus, elle utilise exclusivement des mesures de la puissance consommée par la machine en utilisant un wattmètre, et la divise entre chacun des processus sur la machine.
Évidemment, cette méthode ignore la variabilité des consommations des différents processus s'exécutant au même moment. Par exemple, si un processus exécute des calculs intensifs pendant qu'un autre processus concurrent est en attente passive d'une ressource (verrou, réseau, ...), le premier consomme en réalité bien plus que le second.

3.3. Politique Frontière

Nous introduisons une nouvelle politique pour définir et calculer l'énergie consommée par chaque programme au cours du temps. L'idée clé de cette politique est de corréler les dates de débuts et de fins d'exécution des processus sur une machine (les frontières) avec les variations de la consommation de puissance de cette machine, mesurée par un wattmètre.
Nous présentons d'abord cette politique par un exemple, avant de la détailler formellement.

3.3.1. Exemple

Imaginons que l'on exécute des processus pendant un intervalle de temps connu, et qu'on mesure la puissance consommée par l'ordinateur comme sur la figure 1 :

— Pendant la première seconde, seul le processus 1 existe, et le système consomme 2 watts,
— Pendant la deuxième seconde, les processus 1 et 2 existent, et le système consomme 3 watts,
— Pendant la troisième seconde, les processus 2 et 3 existent, et le système consomme 2 watts,
— Pendant la quatrième seconde, les processus 3 et 4 existent, et le système consomme 3 watts.

De cela, en supposant que toute la puissance est consommée par ces processus, et que la puissance consommée par un processus est constante (hypothèses que nous relâcherons dans la suite), nous pouvons entièrement déduire la puissance consommée par chaque processus :

— Pendant la 1ère seconde, process1 était le seul processus, donc sa consommation est exactement la consommation du système : 2W.

1. Dynamic Voltage and Frequency Scaling
2. `https://www.grid5000.fr/mediawiki/index.php/Lyon:Hardware#Taurus`

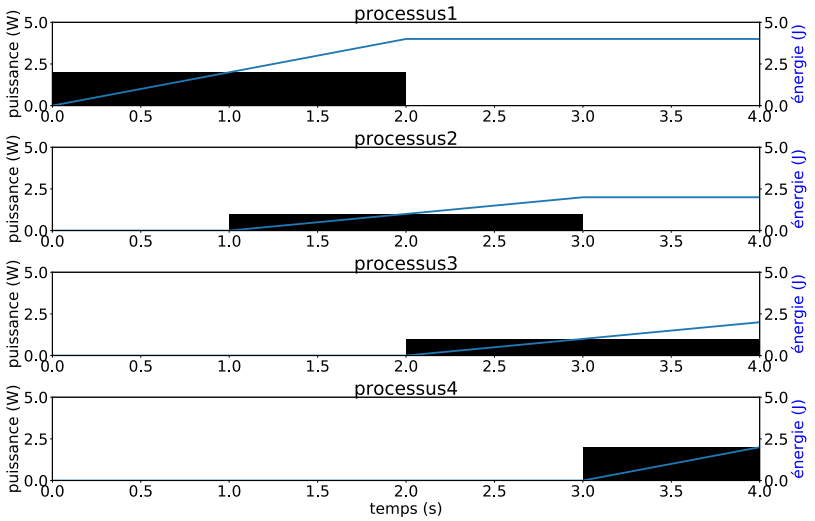

FIGURE 2 – Exemple de résolution de la consommation énergétique

— Pendant la 2ème seconde, process2 est aussi démarré. Comme process1 existe encore, il consomme encore 2W. Et comme le système consomme 3W, on en déduit que process2 consomme 1W.

— Pendant la 3ème seconde, process1 termine et process3 démarre. Comme la consommation totale est de 2W et que process2 consomme 1W, on en déduit que process3 consomme 1W.

— Enfin, de la même façon, on en déduit que pendant la 4ème seconde, process3 consomme 1W et process4 consomme 2W.

Et cela donne la résolution complète, qui est représentée graphiquement sur la figure 2. Les barres représentent la consommation instantanée (en watts), et la courbe l'énergie consommée depuis le début, cumulativement (en joules).

3.3.2. Formalisation

On suppose que les mesures sont faites toutes les secondes, et qu'un processus démarre et s'arrête sur une seconde exacte. On note V_t l'ensemble des processus vivants à l'instant t, et on imposse qu'il n'y ait pas de processus vivant au début des mesures ($V_0 = \emptyset$). On détermine la puissance statique de la machine comme étant le minimum de sa puissance sur l'intervalle mesuré.

On note $P_{p,t}$ la puissance moyenne consommée par le processus p entre les secondes t et $t+1$, et $\Delta P_{p,t} = |P_{p,t-1} - P_{p,t}|$, la variation de la consommation moyenne de p entre les intervalles $[t-1, t]$ et $[t, t+1]$ pour $t \geq 1$, et $\Delta P_{p,0} = P_{p,0}$. Alors, en notant D_t la puissance dynamique consommée par la machine, mesurée entre t et $t+1$, on peut définir les valeurs des $P_{p,t}$ comme étant une solution du système figure 3 telle que $\sum_t \sum_{p \in V_t \cap V_{t+1}} \Delta_{p,t}$ est minimale.

Ainsi, on cherche des valeurs pour les $P_{p,t}$ telles que la variation de la consommation de puissance de chaque processus au cours de son exécution est minimale, tout en ayant la contrainte (vu la première équation du système) que la somme des consommations de chaque processus est égale à la consommation dynamique de la machine.

Ce système d'équations et l'expression à minimiser étant linéaires, il s'agit d'un problème de programmation linéaire, qui peut donc être résolu par un algorithme de type simplexe [4].

3.3.3. Interférences dues à la politique Frontière

Cette politique a l'avantage d'avoir un impact négligeable sur la consommation de la machine exécutant les processus auxquels on s'intéresse.

En effet, les mesures sont totalement externes à la machine vu que wattmètre à la prise et que les mesures peuvent être mémorisées hors du cluster. De plus, le calcul de la politique Frontière se fait

$$\begin{cases} \sum_{p \in V_t} P_{p,t} = D_t & \forall t \text{ si } V_t \neq \emptyset \\ P_{p,t} = 0Watt & \forall t, \forall p \notin V_t \\ P_{p,t} \geq 0Watt & \forall t, \forall p \in V_t \\ \Delta P_{p,t} \geq P_{p,t-1} - P_{p,t} & \forall t, \forall p \in V_{t-1} \cap V_t \\ \Delta P_{p,t} \geq P_{p,t} - P_{p,t-1} & \forall t, \forall p \in V_{t-1} \cap V_t \end{cases}$$

FIGURE 3 – Système d'équations de la politique Frontière

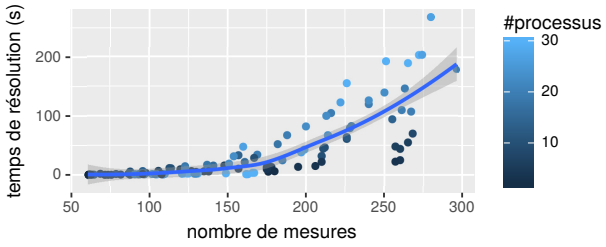

FIGURE 4 – Temps de calcul de la politique Frontière selon le nombre de mesures et de processus

après la fin de l'exécution des processus (et/ou sur une autre machine), ce qui fait que ce calcul n'a pas d'influence sur ceux-ci.
Elle demande cependant de mémoriser les dates de début et de fin d'exécution de chacun des processus intéressants, ce qui demande d'instrumenter le système exécutant les processus. Dans notre évaluation expérimentale ci-dessous, cela se fait en ayant un script qui exécute les processus par SSH en mémorisant leur intervalle d'exécution.

4. Évaluation expérimentale

4.1. Validation

Nous avons expérimenté une implémentation de notre politique Frontière sur un nœud de la plateforme Grid5000 (cluster Taurus [3] avec un processeur Intel Xeon E5-2630) [1], équipé d'un wattmètre effectuant une mesure chaque seconde, et accessible via l'API Kwapi [10].
La figure 5 montre un exemple d'exécution de six processus correspondant à des appels à la commande `gzip` sur des fichiers de différentes tailles générés aléatoirement ou remplis de zéros. 62 mesures de consommation instantanée ont été prises (une par seconde). Comme précédemment, les barres correspondent à la consommation instantanée en watts, et la courbe à la consommation d'énergie depuis le début. L'ordonnée de la courbe à droite donne donc l'énergie totale consommée par chaque processus.
On constate par exemple que compresser 1GO de zéros consomme environ 100J, et donc moins que les 400J pour compresser 300MO d'octets aléatoires. On retrouve d'ailleurs des valeurs similaires en les exécutant indépendamment.
Enfin, cela donne uniquement une répartition de la consommation dynamique. Selon la politique souhaitée, on peut également vouloir y additionner la consommation statique, répartie équitablement entre les processus, ou au prorata de leur consommation dynamique.

4.2. Temps de calcul de la consommation

Nous avons également évalué le temps d'exécution de notre implémentation de cette politique, en fonction du temps d'exécution des processus et de leur nombre. La bibliothèque utilisée pour résoudre les programmes linéaires était CVXPY [5] avec le solveur ECOS [8], et les exécutions du benchmark ont été distribuées sur 7 machines du cluster Taurus.
La figure 4 montre les temps de calcul de la politique (construction du programme linéaire et résolution), en fonction du nombre de mesures et du nombre de processus exécutés. La durée de l'exécution d'un processus n'entre pas en compte, vu que le solveur ignore l'unité de temps.
On constate sur cette figure que le temps de calcul augmente fortement avec le nombre de mesures. Notre framework permet de résoudre ce problème en fusionnant automatiquement les mesures les moins intéressantes : si dans un intervalle de temps, aucun processus n'a été démarré ou ne s'est

3. `https://www.grid5000.fr/mediawiki/index.php/Lyon:Hardware#Taurus`

arrêté, on fait la moyenne $\langle P \rangle$ des mesures de puissance sur cet intervalle et on les remplace par une unique mesure sur tout l'intervalle, de valeur $\langle P \rangle$.
Ainsi, il est possible de rendre le temps de calcul de la politique plus court que la durée d'exécution des processus, permettant une analyse du profil en fenêtre glissante.

5. Conclusion et travaux futurs

Dans cet article, nous avons énoncé la notion de politique d'attribution d'énergie et son intérêt. Nous avons défini la politique Frontière, qui est – à notre connaissance – la première technique attribuant de l'énergie à des processus en se basant uniquement sur une mesure à la prise, et sans être intrusif sur la machine mesurée et sans calibration.
Cette approche demande cependant un temps de calcul important entre la fin des mesures et les résultats, ce à quoi nous souhaitons remédier dans le futur autrement qu'en diminuant la précision. Nous souhaitons également améliorer la politique Frontière par exemple en corrélant les profils de plusieurs exécutions du même programme, ainsi que comparer ses résultats avec ceux des outils existants. Enfin, nous allons développer des applications utilisant cette politique, par exemple pour de l'ordonnancement.

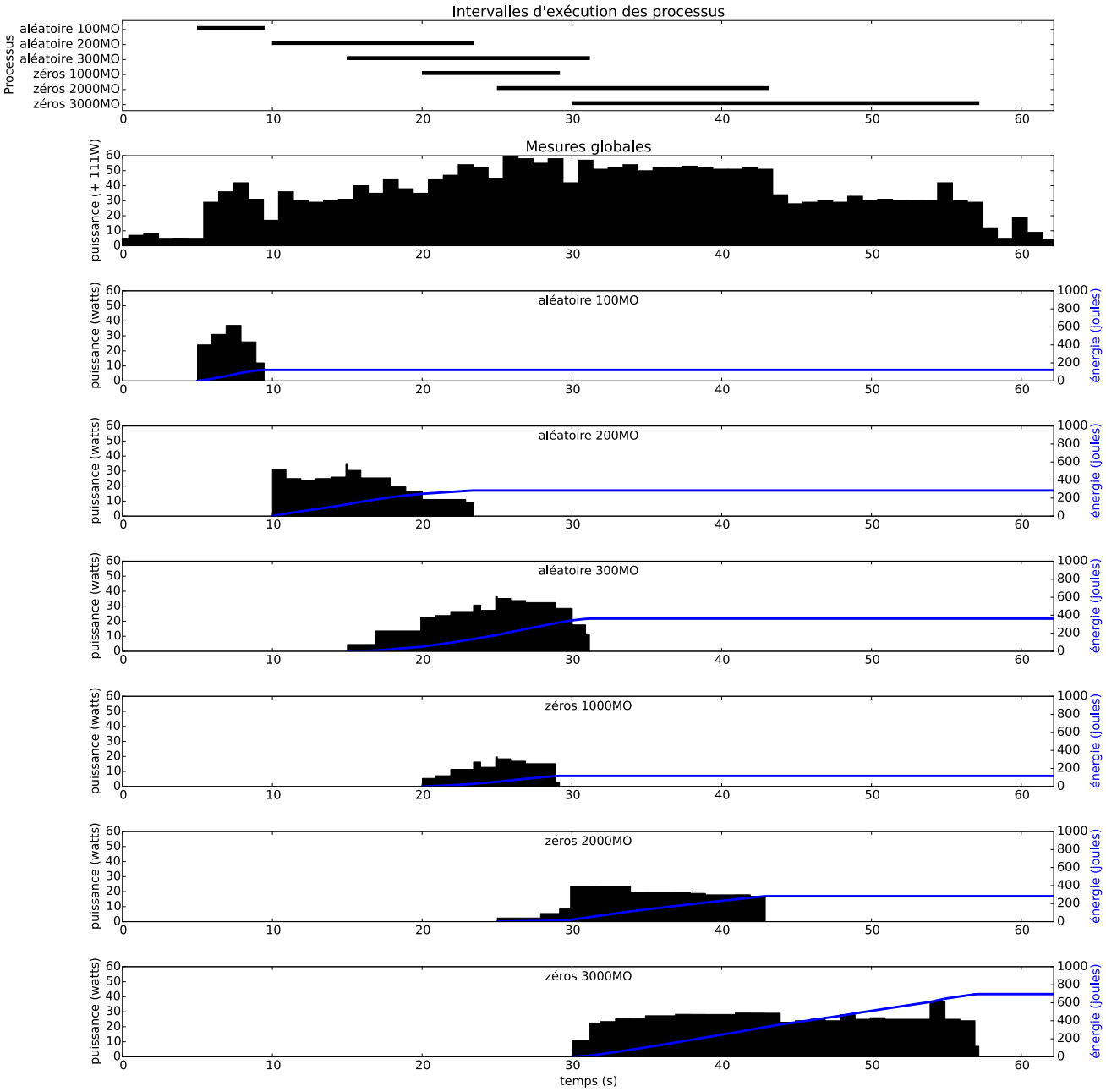

FIGURE 5 – Résolution complète du profil énergétique de `gzip` sur différents fichiers

Remerciements

Les expériences présentées dans cet article ont été exécutées sur le banc de test Grid'5000, supporté par un groupe d'intérêt scientifique hébergé par Inria et comprenant le CNRS, RENATER, et différentes Universités et organisations (voir `https://www.grid5000.fr`).

Bibliographie

1. 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.), Quesnel (F.), Rohr (C.) et Sarzyniec (L.). – Adding virtualization capabilities to the Grid'5000 testbed. *In : Cloud Computing and Services Science*, éd. par Ivanov (I.), Sinderen (M.), Leymann (F.) et Shan (T.), pp. 3–20. – Springer International Publishing, 2013.
2. Bourdon (A.), Noureddine (A.), Rouvoy (R.) et Seinturier (L.). – Powerapi : A software library to monitor the energy consumed at the processlevel. *ERCIM News*, vol. 2013, n92, 2013.
3. Colmant (M.), Felber (P.), Rouvoy (R.) et Seinturier (L.). – WattsKit : Software-Defined Power Monitoring of Distributed Systems. – In Capello (F.), Fox (G.) et Garcia-Blas (J.) (édité par), *17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), p. 10, Madrid, Spain, mai 2017. IEEE.
4. Dantzig (G. B.), Orden (A.), Wolfe (P.) et al. – The generalized simplex method for minimizing a linear form under linear inequality restraints. *Pacific Journal of Mathematics*, vol. 5, n2, 1955, pp. 183–195.
5. Diamond (S.) et Boyd (S.). – CVXPY : A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, vol. 17, n83, 2016, pp. 1–5.
6. Diouri (M. E. M.), Dolz (M. F.), Glück (O.), Lefevre (L.), Alonso (P.), Catalán (S.), Mayo (R.) et Quintana-Ortí (E. S.). – Solving some mysteries in power monitoring of servers : Take care of your wattmeters! *In : Energy Efficiency in Large Scale Distributed Systems*, pp. 3–18. – Springer, 2013.
7. Do (T.), Rawshdeh (S.) et Shi (W.). – ptop : A process-level power profiling tool. – In *Proceedings of the 2nd workshop on power aware computing and systems (HotPower'09)*, 2009.
8. Domahidi (A.), Chu (E.) et Boyd (S.). – ECOS : An SOCP solver for embedded systems. – In *European Control Conference (ECC)*, pp. 3071–3076, 2013.
9. el Mehdi Diouri (M.), Gluck (O.), Lefevre (L.) et Mignot (J.-C.). – Your cluster is not power homogeneous : Take care when designing green schedulers! – In *Green Computing Conference (IGCC), 2013 International*, pp. 1–10. IEEE, 2013.
10. Rossigneux (F.), Lefevre (L.), Gelas (J.-P.) et De Assuncao (M. D.). – A generic and extensible framework for monitoring energy consumption of openstack clouds. – In *Big Data and Cloud Computing (BdCloud), 2014 IEEE Fourth International Conference on*, pp. 696–702. IEEE, 2014.
11. Zou (Y.) et Rajopadhye (S.). – Automatic energy efficient parallelization of uniform dependence computations. – In *Proceedings of the 29th ACM on International Conference on Supercomputing*, pp. 373–382. ACM, 2015.