« Elapsed Time » : pertinent pour la facturation ?

Emmanuel Quémener
Centre Blaise Pascal – Ecole Normale Supérieure de Lyon
46, allée d'Italie
69007 Lyon

Résumé

« Elapsed Time » reste-t-il un critère pertinent de refacturation ? Retour sur quelques années de métrologie...

Dans nos centres de ressources, l'essentiel de la métrologie (et de la facturation) repose souvent sur le « temps écoulé », parfois le « temps utilisateur ».

L'arrivée de traitements de données massives, issues notamment de communautés de biologie, dessine désormais un nouveau paradigme : le principal « stress » de nos infrastructures ne repose plus uniquement sur la sollicitation des unités de traitement mais dans le transport des données, tous les transports.

Nous verrons, au travers d'une fouille menée sur des fichiers de métrologie couvrant plusieurs années d'exploitation et issus du Centre Blaise Pascal (hôtel à projets et centre d'essais) et du Pôle Scientifique de Modélisation Numérique ((mésocentre de calcul) de l'ENS-Lyon, qu'il convient de prendre en compte désormais d'autres métriques pour évaluer au mieux ces nouveaux usages, et donc permettre une meilleure reventilation de leurs coûts..

Mots-clefs

Métrologie, facturation, ...

1 Introduction

« Une accumulation de temps écoulés n'est pas plus une consommation de ressources qu'un tas de pierres n'est une maison ! »

Comment évaluer la « consommation de ressources » de l'exécution d'une tâche en informatique ? Telle est la question que se pose le programmeur (quand il cherche à optimiser son développement) ou le responsable de ressources informatiques (quand il a la lourde responsabilité de reventiler ses coûts sur ses utilisateurs).

Jusqu'à présent, la métrologie des seconds exploite essentiellement le premier stade des informations des premiers : « time ». Le « temps écoulé » (*Elapsed Time*), voire le « temps utilisateur » (*User Time*) sont deux valeurs simplement exploitables, facilement accessibles dans les journaux des gestionnaires de ressources.

Cependant, les centres de calcul intensif ont peu à peu mué en centres de traitement de données avec l'arrivée de données massives, issues notamment de communautés de biologie. Ce « changement de contexte » dessine un nouveau paradigme : le « stress » de ces infrastructures ne repose plus uniquement sur le traitement mais aussi dans le transport et le stockage de ces données.

Il convient donc d'étendre sa métrologie au-delà de la simple « durée » pour y associer d'autres éléments liés eux à l'exploitation des ressources matérielles, mais aussi au « système » (le système d'exploitation prenant en charge tout l'ordonnancement de l'exécution des nombreux programmes ou accès aux périphériques).

Nous verrons, au travers d'une fouille menée sur des fichiers de métrologie du Centre Blaise Pascal (centre d'essais en informatique scientifique) et du PSMN, Pôle Scientifique Modélisation Numérique (mésocentre de calcul de Lyon situé à l'ENS-Lyon), qu'il convient de prendre en compte d'autres métriques pour évaluer au mieux ces nouveaux usages, et donc permettre une meilleure reventilation de leurs coûts.

2 De l'architecture de Von Neumann à l'approche « système »

L'architecture de Von Neuwann nous apporte une vision synthétique du fonctionnement interne d'un ordinateur. Après presque 3/4 de siècle, interactions entre unité de contrôle et unité arithmétique et logique, mémoire et entrées ou sorties restent toujours pertinentes pour décrire le fonctionnement dynamique d'un programme.

L'œil d'un physicien préfèrera une approche plus « système » où le matériel et le système d'exploitation se dissimulent dans une « boîte noire » : l'entrée, c'est le « cas d'usage » (les données et ses programmes) ; la sortie, c'est le résultat.

Reste maintenant à déterminer comment « instrumenter » sa « boîte noire » de la manière la moins invasive possible pour qu'elle offre, facilement, des informations sur son exploitation interne.

2.1 Du « time » comme commande au « time » comme programme

Tout utilisateur de commande en ligne a préfixé un jour son programme de « time » pour évaluer le temps écoulé. Cet appel est une commande intégrée à chaque « shell » pour offrir une métrologie minimale de son exécution : chacun, bash, zsh, tcsh, offrent une sortie différente. Par défaut, tcsh est le plus loquace (avec notamment des informations sur les entrées/sorties), bash le plus « pingre » (avec seulement des informations sur les durées).

Pour pallier ces insuffisances de « time », GNU s'est fendu d'un développement : le programme « time » (donc à appeler avec l'appel **`/usr/bin/time`**). Dans sa version standard, il offre un sortie comparable à tcsh mais il se propose de mesurer simultanément 24 valeurs : des classiques comme les temps écoulés mais des plus abouties comme le nombre d'accès en entrée sur des fichiers ou le nombre de changement de contextes, qu'ils soient voulus ou non voulus par le système. Cependant, la sortie standard de `/usr/bin/time` n'est pas ni complète, ni facile à décomposer lors d'une extraction.

```
0.00user 0.00system 0:00.00elapsed ?%CPU (0avgtext+0avgdata
2552maxresident)k

0inputs+0outputs (0major+123minor)pagefaults 0swaps
```

Le réglage de cette sortie se réalise par la variable `$TIME` (donc veiller ne pas exploiter cette variable dans son shell). Chez nous, un fichier contenant la configuration de sortie (nommé `/etc/time_command.cfg`) sert à la définition de cette variable. Il contient :

```
TIME Command being timed: "%C"
TIME User time (seconds): %U
TIME System time (seconds): %S
TIME Elapsed (wall clock) time : %e
TIME Percent of CPU this job got: %P
TIME Average shared text size (kbytes): %X
TIME Average unshared data size (kbytes): %D
TIME Average stack size (kbytes): %p
TIME Average total size (kbytes): %K
TIME Maximum resident set size (kbytes): %M
TIME Average resident set size (kbytes): %t
TIME Major (requiring I/O) page faults: %F
TIME Minor (reclaiming a frame) page faults: %R
TIME Voluntary context switches: %w
TIME Involuntary context switches: %c
TIME Swaps: %W
TIME File system inputs: %I
TIME File system outputs: %O
TIME Socket messages sent: %s
TIME Socket messages received: %r
TIME Signals delivered: %k
TIME Page size (bytes): %Z
TIME Exit status: %x
```

La définition s'exploite avec un simple : `export TIME=$(cat /etc/time_command.cfg)`

Ainsi, il est possible, avec ce simple outil et sans modification aucune de son programme, d'avoir un aperçu de ses principales « consommations ». Cependant, cette analyse a une limitation : ce n'est qu'une analyse globale, «post mortem ».

2.2 Un peu de temps sur les « temps » : l'absence de l'attente

Ce que nous distinguons dans ces sorties de « temps » avec « time » (commande ou programme), c'est qu'il en manque une ! Nous disposons du « temps utilisateur », du « temps système » et du « temps écoulé » mais pas du temps passé à attendre que le système d'exploitation puisse accéder à ce dont il a besoin pour opérer, ou les pauses qui sont délibérément intégrées à l'exécution, le fameux « I/O wait ».

Ce manque est d'autant plus important qu'il illustre souvent la limitation voire l'incapacité du système à répondre à la sollicitation de composantes du traitement. Certaines de ces limitations sont purement « physiques », comme la limitation « brute » de débit sur un médium, qu'il soit de communication réseau ou stockage. D'autres sont « logiques », liés à la mauvaise exploitation du système.

Pour ce temps manquant, nous pourrions certes considérer que c'est la différence entre le temps écoulé et les temps système et utilisateur, mais ce n'est plus aussi simple. En effet, s'il y a des processus exploitant plusieurs ressources (parallélisées en somme), leur cumul se réalise alors et nous nous retrouvons rapidement avec des temps utilisateur ou système supérieurs au temps écoulé.

2.3 Lorsque les périphériques « de traitement » compliquent les choses

De plus, le paysage de l'informatique a changé ces dernières années, mais pas seulement par rapport à l'explosion du volume de données. Dans le classement du Top 500, sur le podium, 80 % de la puissance brute de deux machines reposent désormais sur leurs accélérateurs. 70 % en disposent dans les 10 premiers. 25 % sur le classement.

Leur part dans les traitements est ainsi vouée à devenir prépondérante, notamment dans leur nouvel El Dorado que représente le « machine learning ».

Sur le marché des circuits graphiques, un acteur domine, l'autre tente d'exister : Nvidia et AMD. En observant leur comportement à l'exécution, nous constatons que ces composants n'offrent pas du tout la même réponse en matière d'occupation de ressources. Pour un programme « GPUfié » avec très peu d'entrées/sorties, une carte récente Nvidia (RTX 2080 Ti) va présenter une charge « système » importante (un temps « système » représentant le quart du temps d'exécution, le reste étant le temps « utilisateur ») alors que les charges « système » et « utilisateur » sont négligeables (2%) sur un circuit AMD récent (Radeon VII).

Ainsi, même une métrologie exploitant des outils systèmes aussi simples que « time » devra s'accompagner d'une calibration. Cette dernière ne peut s'affranchir d'une récupération aussi exhaustive que possible du contexte matériel et logiciel du socle d'expérimentation, ceci pour juger de l'occupation réelle des ressources.

Reste à savoir si, sur les journaux de « gestionnaires de ressources » (batch schedulers), nous disposerons des éléments suffisants pour dessiner les contours des usages des différentes communautés.

3 Contexte de l'étude

A l'École Normale Supérieure de Lyon, le CBP et le PSMN exploitent un ordonnanceur comparable pour la distribution des tâches sur leurs clusters : GridEngine et SGE (Sun Grid Engine). Si le PSMN exploite SGE pour des raisons historiques (l'ancienne infrastructure était largement sous Sun, donc exploitait SGE), le CBP utilisait à ses débuts l'outil OAR développé à Grenoble. Cependant, lors de l'intégration du portail Galaxy pour la biologie en 2015, aucun ordonnanceur ne disposait d'une interface DRMAA (Distributed Resource Management Application API) fonctionnelle hormis Grid Engine : ce dernier a donc été choisi.

Nous avons mené notre étude sur 2 périodes : 2726 jours pour le PSMN (de février 2010 à juillet 2017) et 909 jours pour le CBP (de janvier 2015 à juillet 2017). Le format de sortie est simple, cohérent et donc comparable. Son avantage est de se présenter sous forme d'un fichier à plat présentant 45 attributs dont 18 étaient directement issus de `getrusage`. Chaque ligne se trouve ainsi associée à une exécution (complète ou avortée).

4 Quelle approche pour fouiller les journaux ?

Premier examen de ces volumineux fichiers de journaux d'événements GridEngine : des volumes de 4 Go pour le PSMN, 40 Mo pour le CBP. Leurs nombres de lignes représentent 10 millions de lignes pour le PSMN et seulement 130 mille pour le CBP. La stratégie d'analyse, comme nous disposons d'un document « à plat », peut se faire de différentes manières. Nous en avons exploité plusieurs au gré de la rapidité d'exécution et de leur consommation de ressources.

La première approche a été d'exploiter l'outil R à base de Rscript. Très rapidement, R a montré ses limites sur le document du PSMN. Les opérations de filtrages n'étaient pas si efficaces que cela et suggéraient de plutôt exploiter SQL.

Une approche Python pure à base de pandas a été également testée mais elle s'est révélée très lourde à l'usage.

Finalement, l'exploitation d'un SGBD relationnel s'est révélée la plus efficace. Cependant, il fallait récupérer les données pour les représenter : une approche hybride à base de SQLite3, puis des requêtes SQL dans pandas et en dernier lieu une représentation LibreOffice a donc été exploitée pour l'analyse.

La méthodologie de traitement exploitée comporte ainsi 6 étapes :

- la création de la base sqlite3 ;
- l'importation des journaux dans la base ;
- la création des tables de correspondance entre identifiant, laboratoire et communauté ;
- le nettoyage des journaux ;
- l'analyse selon quelques critères objectifs ;
- la synthèse pour isoler des usages par communauté.

4.1 Création de la base Sqlite3, importation des journaux

La création de la base se réalise juste par le lancement d'un `.open` préfixant le nom de sa base.

Pour créer la table avec les noms des attributs, nous avons directement exploité le manuel de « accounting » associé à GridEngine. Ce dernier détaille scrupuleusement les nom des attributs et leur valeur. La commande suivante permet de créer la commande à appliquer à SQLite pour créer la bonne table à partir de la documentation.

```
echo -ne "create table accounting ( $(man -P cat accounting |
egrep "(^....\.|^.....\.)" | awk '{ print $2 }' | sed -e
's/^/ge_/g' | tr '\n' ',))\n" | sed -e "s/,)/\ );/g"
```

Pour l'importation des données, il suffit de spécifier le caractère séparateur du fichier de métrologie et d'importer carrément le document dans son ensemble. Dans notre cas :

```
.separator :
.import GridEngine_accounting_201707281508 accounting
```

4.2 Le nettoyage

Dans l'analyse massive des données, la première difficulté est le contrôle de leur cohérence.

Dans notre cas, nous avons supprimé les jobs dont les dates étaient incohérentes, ceux qui s'étaient mal terminés et ceux dont la durée totale était nulle.

```
delete from accounting where ge_exit_status!='0' or ge_failed!
='0' or cast(ge_ru_wallclock as float)<1. or ge_start_time='0' or
ge_end_time='0' or ge_submission_time='0';
```

Avec cette opération, nous avons réduit le nombre de jobs de 20% pour le CBP et de 10% pour le PSMN.

4.3 Les premières analyses et l'émergence de l'approche hybride

Nous avons évoqué l'usage de la librairie « pandas » de Python pour traiter les données.

D'abord, la première approche d'un traitement direct et d'une représentation graphique des journaux de GridEngine CSV se sont soldées par un dépassement de capacité de la machine (qui n'avait que 16 Go de RAM).

Puis, nous avons exploré un passage par des variables de cumul en 32 bits et non 64 pour réduire l'empreinte mémoire, sans plus de succès pour la rapidité d'exécution.

Ensuite, nous avons intégré les requêtes SQL directement dans les accès à la base SQLite. Le résultat était reformaté (avec des « cast » en pagaille) puis « pandas » fournissait ses fonctions statistiques pour réduire les données.

Finalement, les données «en « stdout » étaient « copiées/collées » dans LibreOffice pour représentation graphique.

C'est donc un redoutable pragmatisme qui a guidé notre analyse.

5 Premières analyses : premiers comportements

5.1 Les cumuls pour les communautés, en « tâches » et « temps »

Dans les premières analyses, nous avons réalisé un classement à partir du nombre de jobs exécutés, de la durée de réservation (la « Wall Clock ») et du temps d'exécution

processeur (le « CPU Time ») pour les différentes communautés utilisatrices (astrophysique, biologie, chimie, géologie, industriels, informatique, mathématiques, mécanique et physique).

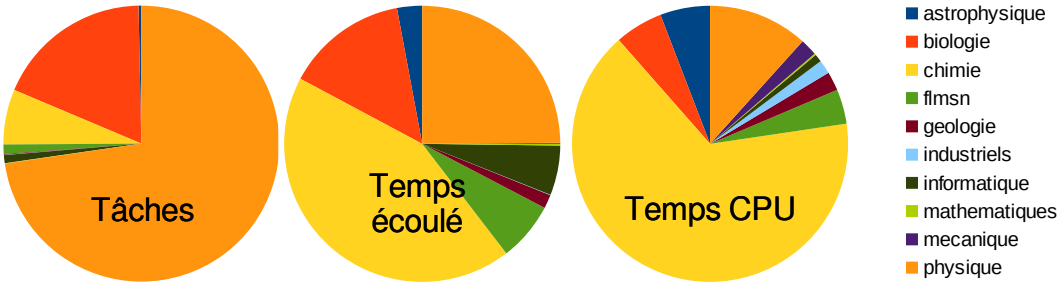

Figure 1 - Synoptique de consommations de ressources classées en fonction du nombre de tâches, du temps total écoulé et du temps CPU pour les différentes communautés exploitant le PSMN (centre de calcul)

Pour le PSMN, il est par exemple apparu que les physiciens avaient lancé les 3/4 des tâches mais pour un temps d'exécution représentant uniquement le 1/4 de l'utilisation totale et seulement 1/10 du temps CPU.

De son côté, la communauté des chimistes représentaient presque la moitié du temps total d'exécution, pour seulement 1/15 des job lancés et les 2/3 de l'occupation CPU.

La biologie, elle présentait une occupation en job de 1/6, mais un temps total de 1/10 et un temps CPU de 1/20.

Ainsi, dans ces trois communautés majoritaires, les usages sont très différents. De plus, nous voyons apparaître que le temps de calcul n'est pas nécessairement corrélé au temps d'exécution, ce qui suggère l'importance croissante des entrées/sorties dans le temps total d'exécution.

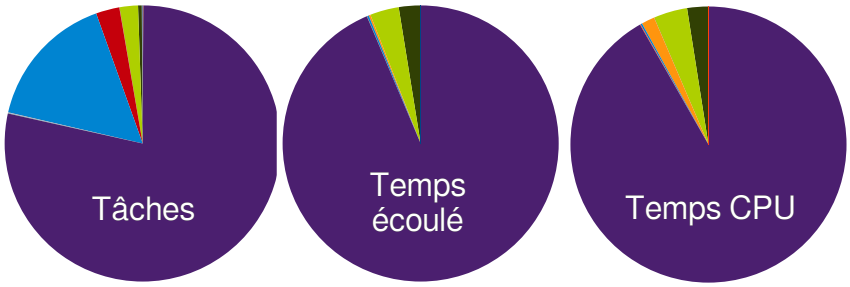

Figure 2 - Synoptique de consommations de ressources classées en fonction du nombre de tâches, du temps total écoulé et du temps CPU pour les différentes communautés exploitant le CBP. Le mauve représente un seul utilisateur, le bleu les utilisateurs du portail de biologie Galaxy.

Pour le CBP, l'utilisation se divisait essentiellement entre des calculs séquentiels représentant 4/5 des jobs pour plus de 95% des temps totaux ou d'exécution. Les jobs

liés au portail de bioinformatique Galaxy, bien que représentant 1/10 en nombre, avaient un impact imperceptible sur les temps. Nous assistions donc là aussi à deux usages très distincts.

5.2 La durée des tâches : un histogramme évocateur

L'analyse des durées d'exécutions ou de temps CPU présentait une gaussienne très marquée pour des jobs entre 100 et 1000 secondes.

Il était aussi à noter que, plus les jobs étaient courts, moins le temps CPU était faible comparé au temps d'exécution (partie entrées/sorties majoritaire) .

A contrario, pour les jobs beaucoup plus longs (entre 10000 et 100000 secondes), le *CPU time* était bien supérieur au *wall clock*, suggérant une utilisation parallélisée.

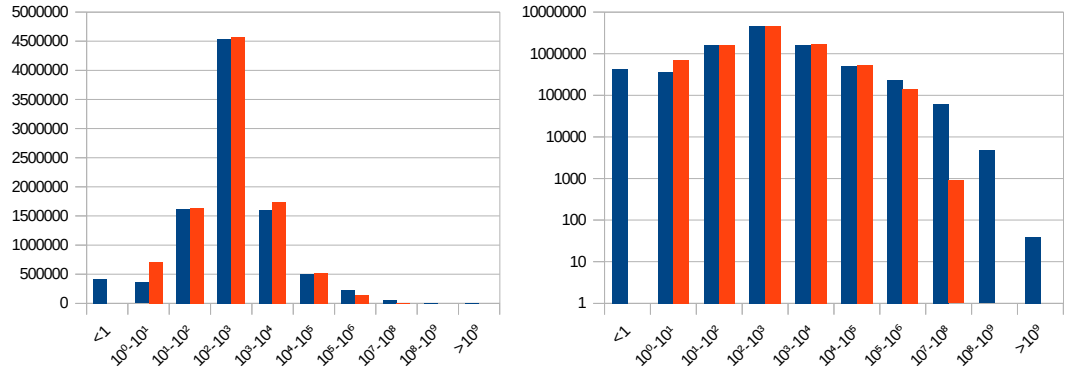

Figure 3 - Distribution des « temps écoulé » et « temps CPU »

5.3 Le ratio des durées : première mesure d'efficacité

Ainsi, un simple rapport entre les temps permet déjà de juger de l'exploitation de ressources réservées. Il révèle que, souvent, les ressources de traitement réservées sont sous-exploitées pendant l'exécution.

Nous pouvons ainsi, en fonction de la réservation faite de ressources, évaluer avec quelle efficacité les tâches se sont exécutées.

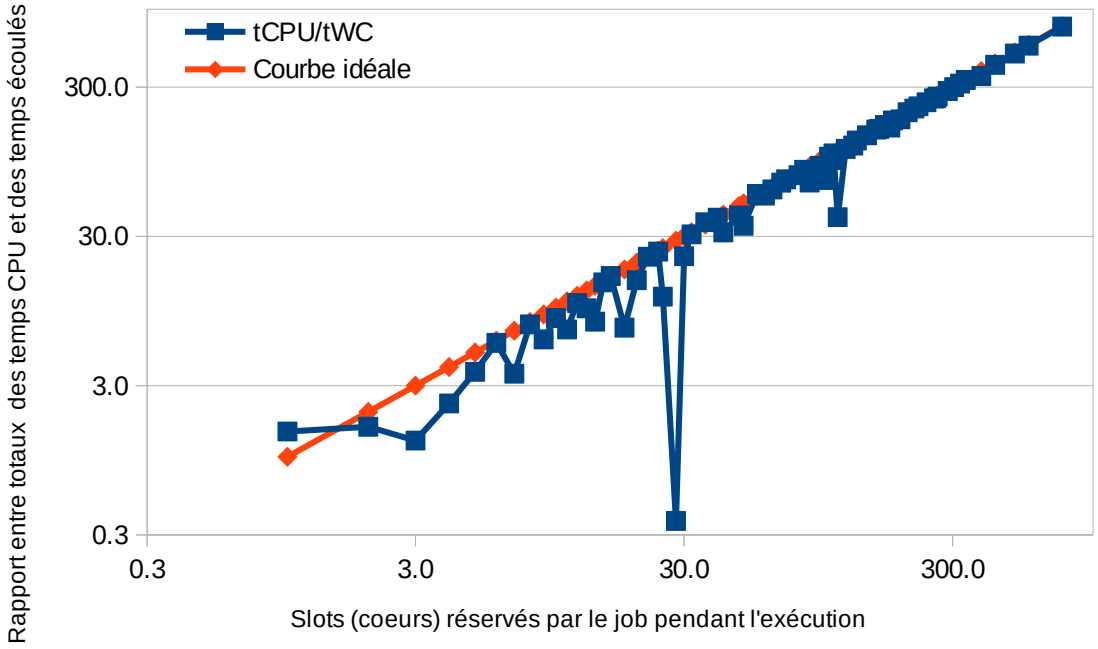

Figure 4 - Ratio des « temps » CPU et écoulés pour différentes réservations

Il apparaît, excepté pour une réservation de ressources atomique (slots=1), que la consommation de ressources est toujours inférieure à l'unité. Cela suggère finalement que les ressources sont relativement bien exploitées, sauf pour quelques valeurs.

Pour comprendre l'anomalie de « slots=1 », une poursuite d'investigations est nécessaire :

- pour combien de jobs, le temps CPU est-t-il supérieur au temps écoulé ? Environ 10 % ;
- pour combien de jobs, le temps CPU est-t-il inférieur au temps écoulé ? Environ 90 % ;
- pour combien de jobs, le temps CPU est-t-il supérieur à 10 fois le temps écoulé ? Environ 0.02 % ;
- pour combien de jobs, le temps écoulé est-t-il supérieur à 10 fois le temps CPU? Environ 1 %

Ainsi, pour plus de 100000 jobs, la machine a passé son temps à attendre de pouvoir continuer son traitement. Une fois cette constatation faite, est-il possible d'extraire, en fonction des communautés, des « comportements » typiques, malgré la grosse diversité ?

5.4 Intégration de la granularité des laboratoires

Une autre exploration s'est donc focalisée sur ces tâches pour lesquelles la machine « attendait » son travail, en distinguant les communautés. Ainsi, pour chaque communauté, nous opérons une extraction de tous les jobs pour lesquels le temps CPU est inférieur au temps écoulé. Puis nous avançons sur l'atténuation de ce ratio : plus celui-ci est important, moins c'est efficace.

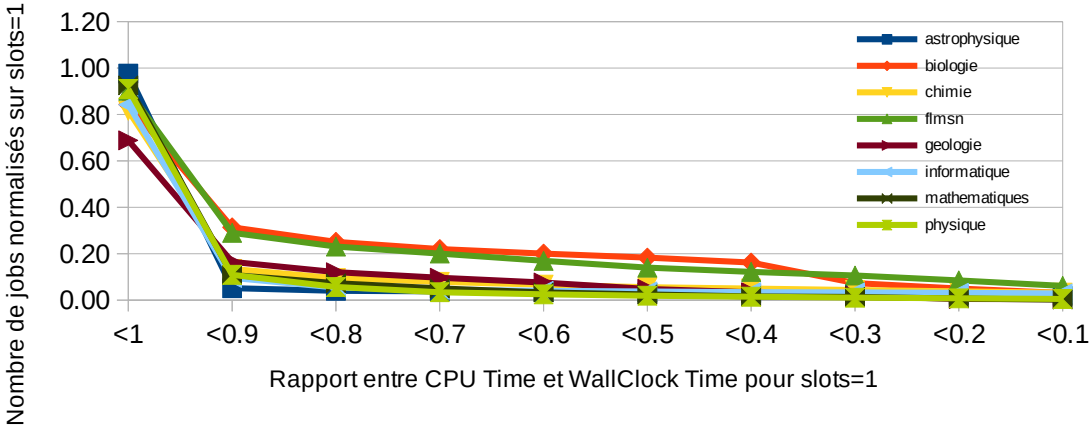

Figure 5 - Ratio des temps en fonction des communautés, une distinction s'opère

Ainsi, le ratio de nombre de jobs inefficaces en biologie est beaucoup plus important qu'en astrophysique (où, c'est vrai, les jobs sont plus rares pour slots=1). Nous avons par exemple presque 1/5 des jobs qui prennent deux fois plus de temps.

Il y a donc, clairement, pour la communauté des biologistes ou FLMSN, une carence en matière d'efficacité, certainement liée à la nature même des tâches, essentiellement des traitements nécessitant chargement et déchargement de données.

5.5 L'exploitation d'autres métriques : un exercice périlleux

Nous avons suggéré dans notre étude que les phases de lecture/écriture induisaient un décalage entre temps d'exécution total et temps CPU. A partir d'autres temps disponibles, nous avons tenté de restituer les fameux temps d'attente entrée/sortie (IO Wait) mais, sur des machines exploitant des ressources distantes et des processus parfois parallélisés, cette analyse laisse émerger des ambiguïtés sur leur origine réelle.

Il fallait donc explorer d'autres métriques, par exemple celles liées à la mémoire (*mem, maxrss*), les entrées/sorties (*io, inblock, outblock*), les temps système (*stime*) ou les changements de contextes (*nvcsw,nivcsw*) apportent des informations fondamentales sur le « stress » système.

Le souci, comme dans toute étude, c'est d'extraire la bonne valeur qui puisse permettre de discriminer les communautés. Si nous nous bornons à des statistiques à base seulement de moyennes ou écarts-types, nous tombons inéluctablement sur des écarts-types supérieurs de un à deux ordres de grandeur à la moyenne ! Cette propriété illustre la très grande diversité, au sein même de chaque communauté, de la sollicitation des ressources.

Rompus aux expérimentations pour lesquelles quelques mesures sont « hors champ » et « pourrissent » la classique moyenne, nous préférons largement exploiter la médiane, finalement plus discriminante pour comparer les communautés. Pour cela, pour chaque job, chacun des attributs est divisé par le nombre de slots de réservation et le temps écoulé. La médiane est alors calculée sur l'ensemble des jobs de la communauté. Une fois toutes les valeurs obtenues, elles sont normalisées en mettant le maximum de chaque attribut à l'unité. Les huit attributs explorés sont les temps système et temps

utilisateur *Stime* et *Utime*, les accès entrées et sorties *InBlock* et *OutBlock*, les changements de contexte non sollicités *MajFlt*, la mémoire maximale utilisée *MaxRSS*, la quantité de données échangées en entrée/sortie *IO* et enfin la quantité de mémoire moyenne utilisée *Mem*.

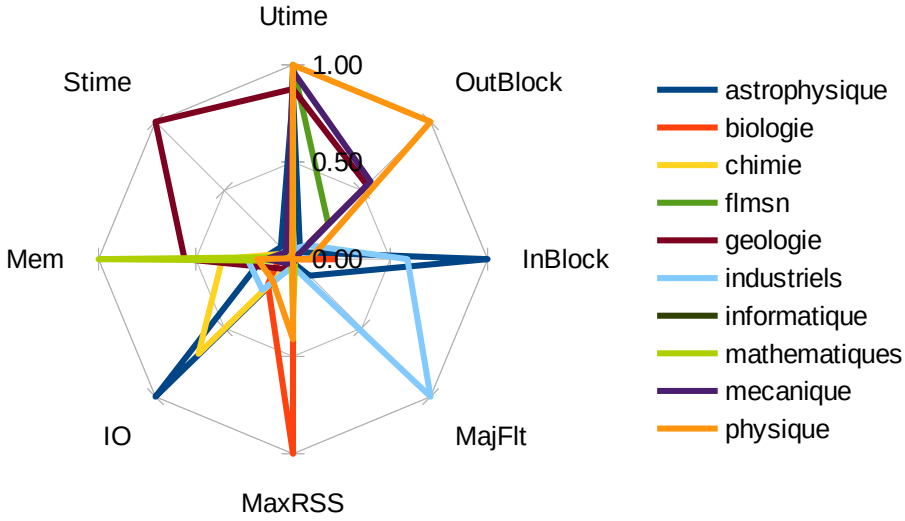

Figure 6 - Autres métriques normalisées par communauté

La représentation figure 6 illustre dans un diagramme « radio » ces 8 quantités pour les dix communautés. Il apparaît que les usages ont une signature qui assez discriminante entre les communautés.

Par exemple, la géologie dispose d'une utilisation en temps « système » et « utilisateur » et en mémoire moyenne plutôt importante. Et les industriels sont les champions du changement de contexte non sollicité.

Il semble également que les consommations maximale et moyenne en mémoire ne soient pas corrélées aux communautés tout comme les accès entrée/sortie et leur montant total.

6 La synthèse

De nos études, nous avons identifié deux comportements « à la charge », plutôt délétère à l'infrastructure, et qu'il convient de limiter :

- le « stress » : le système est exploité au delà de ses capacités
- le « burn-out » : le système « semble » inutilisé mais a des jobs en exécution

Parmi les indicateurs de « stress » système, nous avons :

- La mémoire utilisée : *mem, maxrss*
- Les entrées/sorties : *io, inblock,oublock*
- Le « temps système » : *stime*
- Les changements de contexte : *maxflt*

Identifier le « burn-out » est plus difficile, mais son origine tient essentiellement à des accès entrée/sortie. D'un côté, le ratio entre les temps CPU et écoulé (faible) et d'un autre côté, le nombre important d'entrées/sorties normalisé au temps CPU, sont un couple pertinent pour estimer ce « burn-out ».

Quoiqu'il en soit, en association au temps écoulé, une opération de bonus/malus peut accompagner l'estimation des ressources exploitées pour mieux en restituer l'utilisation réelle.

Le bonus, récompensant une « bonne » utilisation du système, s'obtient lorsque réservation et temps d'exécution sont cohérents avec le temps total CPU.

Le malus, visant à taxer une mauvaise (ou invasive) utilisation du système, comme la charge sur les entrées/sorties, l'utilisation excessive du système (comme sur les codes hybrides mal configurés) ou les opérations symétriques en entrées/sorties.

7 Conclusion

Une exploitation finalement assez simple des métriques offertes par les ordonnanceurs permet de faire émerger des usages dans l'exploitation des ressources, différents d'une communauté scientifique à l'autre. L'exploitation avancée offre une vision plus claire du stress et du burn-out des infrastructures.

Au-delà de toute destination mercantile visant à « rationaliser l'usage des ressources » (pour les participations aux financements), ces analyses ont aussi une portée pédagogique : permettre à l'utilisateur de mieux exploiter ce qui est mis à sa disposition, et éviter toute balkanisation outrancière des ressources.