Gestion des ressources de calcul : 4 points de vue

Romaric David
Université de Strasbourg Direction Informatique – Pôle HPC
4 rue Blaise Pascal, CS 90032, 67081 Strasbourg Cedex

Julien Devemy, Philippe Olivero, Suzanne Poulat
Centre de Calcul de l'IN2P3
43, bld du 11 novembre 1918, 69622 Villeurbanne Cedex

Yannis Georgiou, Olivier Richard
Laobratoire d'Informatique de Grenoble
Bâtiment Inria Innovallee - INRIA Rhône-Alpes - 655 avenue de l'Europe - Montbonnot - 38334 St Ismier

Résumé

Les gestionnaires de ressources (ou gestionnaires de batch) sont vitaux dans l'accès aux clusters de calculs ou aux supercalculateurs. Ils assurent le partage (équitable ?) des ressources, contrôlent les programmes, irritent les utilisateurs, rendent les administrateurs système perplexes.

Coordonné entre plusieurs auteurs, le but de notre travail tout a fait original est de réunir 4 points de vue sur ces logiciels :

1. *un point de vue scientifique : la thèse de Yannis Georgiou soutenue 2010 à Grenoble dresse un panorama inédit et exhaustif des logiciels actuellement en compétition en proposant des métriques de comparaison*
2. *une étude technique dans un environnement de production très contraint : le Centre de Calcul de l'In2p3 a récemment changé de gestionnaire de batch. Comment a été menée leur étude ? Quels ont été les critères déterminants ? Et surtout, quel choix a été effectué ?*
3. *- l'expérience de développeurs de gestionnaires de ressources. Les équipes du CC In2p3 et les développeurs de OAR, utilisé sur le méso-centre Ciment à Grenoble nous présenteront les principaux pièges qui surgissent lorsqu'on décide d'écrire un tel logiciel.*
4. *- l'expérience du méso-centre de l'Université de Strasbourg qui répondra à l'angoissante question : "Est-ce qu'un méso-centre suffit à mettre à genoux un gestionnaire de ressources ?*"

Mots clefs

Calcul, clusters, gestion de ressources, grilles, production

1 Introduction

L'exploitation de ressources de calcul en vue de les partager entre plusieurs utilisateurs repose inévitablement sur la mise en place de logiciels spécialisés dans le partage de ressources : les gestionnaires de ressources. Très proches du système d'exploitation, ces gestionnaires ont pour fonction de démarrer des programmes, de les arréter, de fournir des informations sur la charge (CPU, mémoire, disque) des noeuds de calcul et enfin d'interagir avec l'utilisateur.

Fonctionnellement, ces gestionnaires sont couplés avec des ordonnanceurs (ou « gestionnaires de batch »), qui, à partir de données de plus haut niveau comme des demandes de ressources par des jobs, des politiques de prioritisation, des historiques de consommation de ressouces, indiquent au gestionnaire de ressources quel application (ou « jobs ») démarrer.

Il existe un grand nombre de gestionnaires de ressources/batch et choisir l'un d'entre eux n'est pas tâche aisée. De plus, au vu de la simplicité apparente de ces logiciels, il peut être tentant de se lancer dans l'écriture d'un nouveau. Dans cet article, nous présenterons une avancée méthodologique dans l'étude des gestionnaires de ressources, isssue d'un travail de thèse. Ensuite, nous étudierons les motivations qui ont poussé une équipe de recherche à développer un gestionnaire de ressources et les avantages de celui-ci. Puis, sur deux sites de taillse radicalement différentes, nous présenterons les évolutions récentes des gestionnaires de ressources, d'un point de vue gestion du changement et opérationnel.

2 Grille d'analyse et comparaison : une approche scientifique

Il existe malheureusement très peu d'études comparatives des gestionnaires de tâches et de ressources et elles sont trop anciennes pour être exploitables. Dans cette section nous présentons une synthèse d'une étude que nous avons conduite dans [1]. Erreur : source de la référence non trouvéeErreur : source de la référence non trouvéeErreur : source de la référence non trouvée

2.1 Critères d'évaluation et métrique

Fixer des critères d'évaluation n'est pas une entreprise aisée et évaluer chaque gestionnaire est un exercice long et fastidieux. Nous avons identifié et sélectionné 52 critères que nous avons regroupés en 3 catégories principales. Ces catégories correspondent à la découpe fonctionnelle souvent utilisées pour présenter les grandes fonctions des gestionnaires, soient :

1. les fonctions de gestion de ressources,
2. la gestion des tâches
3. les fonctions d'ordonnancement.

Pour les fonctions de gestions de ressources, nous avons notamment considéré la diversité des prises en compte des ressources (par type, structure et topologie, hétérogénéité, topologie), les capacités de passage à l'échelle et les aspects contrôle de l'énergie (extinction/allumage des nœuds).

Pour la catégorie liée à la gestion des tâches, nous avons, entre autres, évalué les types de tâches supportées. Par exemple nous avons considéré le support des tâches à priorité nulle[1] et différents aspects des tâches dites dynamiques. Par ailleurs, nous avons évalués les capacités de manipulation de tâche comme l'envoi de signal, la modification de priorité, la suspension et la reprise après panne (*checkpointing*). De même, nous avons pris en compte le surveillance des tâches (*monitoring*) et les moyens de soumission (CLI, service web ou autres standards).

Pour l'ordonnancement, nous avons considéré les approches classiques comme les politiques de *backfilling, fairshare, preemption* et priorité.

La métrique qui a été utilisée est relativement simple. Chaque critère est évalué suivant une échelle de notation à 4 niveaux de 0 à 3. La valeur 0 correspond à une fonctionnalité absente, 1 pour un support partiel, 2 pour un support correct et 3 pour une fonctionnalité ayant fait l'objet d'une attention particulière. Des notes intermédiaires peuvent être ainsi calculée par catégorie ainsi qu'une note globale que nous avons ramené à 10.

2.2 Evaluation comparative et synthèse des résultats

Nous avons retenu 10 systèmes pour cette évaluation comparative. Cette sélection a été faite en estimant si les logiciels étaient encore employés, pérennes, maintenus et en cours de développement. Les systèmes répondant à ces critères sont (avec leur version) : SLURM(2.2.0-pre8), CONDOR(7.5.3), TORQUE(2.6.0), OAR(2.5.0), SGE(6.2u5), MAUI(3.2.16), MOAB(5.4), LSF(7u6), PBSPro(10.2) et LoadLeveler(4.1).

Nous les avons ensuite évalués suivant la grille présentée précédemment. Nous avons obtenu une note globale ramenée à 10 où il apparaît que les systèmes LSF et SLURM (notés respectivement 6.4 et 6.2) sont légèrement en tête suivit par les système OAR, SGE, PBSPro et CONDOR (respectivement 6, 5.9, 5.8 et 5.7). Les autres systèmes sont plus en retrait.

[1] Aussi appelées *Bestefforts*, ces tâches peuvent être tuées brutalement par le système si une tâche de plus haute priorité a besoin des ressources qui leurs sont affectées pour s'exécuter. Elles sont souvent employées pour l'exploitation des ressources non-utilisées.

Il faut noter que le gestionnaire de ressources TORQUE est en pratique souvent couplé avec les logiciels MAUI ou MOAB pour fournir des politiques d'ordonnancement évoluées. Dans ce cas on se retrouve avec un niveau de fonctionnalité et une évaluation proche groupe de tête, au prix d'un complexité d'administration.

Bien entendu ces notes globales sont très réductrices, le parcours de la grille complète de résultat dans [1] permet une lecture plus fine et des comparaisons ciblées (ex : support de fonctionnalité spécifique dans les politiques d'ordonnancement).

Dans notre analyse [1] nous avons aussi considéré d'autres critères non noté comme la taille du code (ex: plus de 500 000 lignes pour SGE, moins de 60 000 pour OAR) ou le type de licence (seuls SLURM, OAR, Torque, MAUI, SGE pour la version considérée ici disposent d'une licence orientée logiciel libre).

Nous avons également mené des évaluations orientées sur les performances globales des systèmes notamment entre SLURM, OAR et Torque+MAUI avec des résultats qui varient suivant les paramètres des tests. Seul SLURM a pu être évalué à grande échelle sur 9216 cœurs avec de très bons résultats.

Dans ce qui précède nous avons éludé la question de l'importance et de la hiérarchie des critères. Tout d'abord l'ensemble des systèmes retenus pour cette évaluation répondent bien aux principaux critères de chaque catégorie, à savoir: le support des tâches séquentielles et parallèles, un passage à l'échelle correct, un bon niveau de contrôle des tâches, des politiques d'ordonnancement suffisamment paramétrables. La distinction entre les systèmes se concentre plus les fonctions avancées qui n'auront pas la même importance suivant les contextes d'usages (ex : importance du passage à l'échelle, possibilité accrue de configuration et personnalisation, mise à disposition ou configuration de politiques d'ordonnancement spécifiques).

Nous tenons à signaler que cette évaluation et cette grille de critères mériteraient d'être mises à jour régulièrement d'une part afin de suivre l'évolution des logiciels et d'autre part pour intégrer de nouveau critère.

Enfin, cette étude couvrait le logiciel OAR, gestionnaire de ressources et de batch développé dans le circuit universitaire à Grenoble. En partant de l'exempe de OAR, nous présentons dans la suite quelques motivations et résultats liés à l'écriture d'un gestionnaire de ressources.

3 Pourquoi développer un gestionnaire de tâches et ressources : l'exemple d'OAR

3.1 Motivations

Suite aux problèmes rencontrés avec l'utilisation d'OpenPBS (le prédécesseur de Torque) jusqu'en 2003, la robustesse et le passage à l'échelle ont constitué la motivation première du développement du système OAR [2]. Par la suite et assez rapidement, le besoin de polyvalence, c'est-à-dire la capacité à adapter le système à des contextes variés est devenu un puissant moteur d'évolution du projet.

L'enjeu du développement est alors devenu un savant équilibre entre la tentation de rajouter le plus de fonctionnalités au cœur du système et le niveau de complexité final avec le en ligne de mire le risque d'un système très complexe exploiter et difficile, voire impossible, à faire évoluer. Pour maîtriser cette problématique nous avons tout d'abord fait des choix architecturaux forts pour OAR :

1. utilisation d'une base de données comme point central ;
2. utilisation de SSH et d'un lanceur parallèle de commande générique pour le contrôle des ressources
3. utilisation majoritaire de langage de script pour le développement.

Ces choix nous assurent une complexité de code plus réduite que les approches de plus bas niveau. De plus nous mettons un soin particulier à exploiter certains abstractions pour conserver une cohérence global du système (ex : ajout de type de tâches, hiérarchie générique et flexible des ressources et règles d'admission programmable).

Finalement et ce qui est sans doute le plus important, l'adaptation aux contexte d'exploitation dès qu'il sort la norme nécessite (et s'inscrit dans) un dialogue avec les administrateurs et les utilisateurs. Il s'avère alors qu'un bon nombre de difficultés d'exploitation se résolvent en faisant varier légèrement les positions de chacun et en expliquant certain points techniques liés à ce domaine particulier (ex : précision sur le comportement des politiques d'ordonnancement).

Pour conclure, l'optimisation de l'exploitation des infrastructures de calcul est un domaine qui peut être assez complexe. Il semble illusoire qu'un système logiciel puissent résoudre l'ensemble des problèmes, il devient alors généralement nécessaire de faire appel aux experts (audit, conseil et formation).

3.2 Quel effort de développement ?

Le développement global d'un gestionnaire de tâches et de resources généralistes est devenu une entreprise de spécialistes, comparable dans une moindre mesure au développement d'un OS de type Unix. Les cycles de développement sont assez longs (1 an ou plus) et les phases de tests et de mises au point sont rendues délicates du fait de la nature distribuée de ces systèmes et de leurs problématiques toutes particulières.

Si le développement de nouvelles politiques d'ordonnancement est possible (API souvent disponibles, y compris dans des langages de script) par un développeur indépendant et pour un système particulier il doit se faire en collaboration avec les développeurs du systèmes cible. Plus généralement, les développements extérieurs se concentrent sur la périphérie des systèmes (ex: outil de visualisation, intégration dans portail web) avec le choix de se lier à un système ou d'utiliser des interfaces standard. Ce choix dépendent du compromis entre la possibilité de profiter de fonctionnalités spécifiques et une meilleure portabilité du faire de l'interopéraibilité.

Sur le site de Grenoble, le gestionnaire OAR a été mis en production. Il est intéressant de noter que dans le même temps, d'autres sites ont suivi le chemin inverse, comme le CC IN2P3.

4 Pourquoi ne plus développer : évolution du système de batch au CC-IN2P3

4.1 Un changement s'impose !

Le Centre de Calcul de l'Institut National de Physique Nucléaire et de Physique des Particules -CC-IN2P3[3]- du CNRS héberge l'infrastructure numérique nécessaire à l'exploitation scientifique des données issues de plusieurs projets internationaux majeurs dans le domaine de la physique mais aussi des sciences de la vie et des sciences humaines. Il est par exemple l'un des centres de calcul majeurs pour les expériences du LHC[4] du CERN[5], ou encore d'AMS[6], détecteur embarqué sur la station spatiale internationale ISS. C'est au total plus d'une centaine de groupes ou expériences qui l'utilisent quotidiennement. Ces expériences requièrent l'analyse d'une grande masse de données fournies par un détecteur, un satellite, un télescope, ou de simulations numériques de processus physiques. La capacité de calcul actuelle représente une puissance de 147 K-HepSpec06 (36 M-SI2K, soit grossièrement 86 TFlops), avec 1350 machines, 17 000 cœurs, soit un potentiel de 20 000 jobs concourants. Compte tenu des temps moyens d'exécution, il s'écoule plus de 125 000 jobs par jour.

Le fort accroissement prévu de la capacité de calcul du centre a imposé une réflexion sur les capacités du système de batch, BQS[7], un développement maison, à passer le cap de cette évolution. La réflexion a commencé par un audit de cet outil, présenté à la commission internationale consultative analysant les activités du CC-IN2P3, audit dans lequel ont été mises en exergue des craintes pour l'avenir. En effet, au regard des nouveaux besoins prévus, des développements lourds devaient être opérés pour pouvoir absorber plus de charge, touchant notamment au cœur du produit, l'ordonnanceur. L'adaptation à la nouvelle tendance consistant à virtualiser les machines de calcul impliquait également une refonte de la logique de certaines fonctionnalités. De plus, il aurait été souhaitable de libérer les deux équivalents-temps-plein chargés de la maintenance évolutive du produit (hors opération) et de ses améliorations constantes. Un dernier point, non négligeable dans le contexte de grille de calcul était de rompre l'isolement du centre dans ce domaine, et par là même, diminuer les efforts d'adaptation au middleware utilisé. Ainsi, la banalisation des méthodes de calcul par des outils de grille ne justifiait plus le maintien de particularités, de solutions locales, qui furent la force du centre de calcul par le passé. En 2010, suite à cet audit, la commission a préconisé un changement du système de batch. Aussi, il a été initié une étude très poussée des candidats au remplacement.

4.2 Méthodolgie adoptée

La méthode de choix du nouveau système de batch a consisté en plusieurs phases distinctes. D'abord, il a fallu choisir les produits à étudier, puis ces produits ont été évalués à travers une grille de critères. Enfin, les deux produits retenus ont été testés plus finement afin de choisir un lauréat.

Ainsi, une enquête Web a été mise en place afin de connaître l'ensemble des systèmes de batch utilisés dans la communauté de la physique des hautes énergies (HEP) et de quelle manière ils étaient utilisés. Cette enquête a d'abord été envoyée aux responsables des systèmes de batch de sites comparables au CC-IN2P3, puis la diffusion a été élargie à des listes de discussions de la communauté. Du résultat de cette enquête ainsi que des renseignements pris au sein de différents laboratoires HEP, une liste de systèmes de batch possibles a émergé : LSF, Torque/Maui, SGE et PBS Pro. D'autres ont été écartés car ils étaient peu utilisés dans la communauté et non intégrés aux grilles de calcul Européennes : Condor et Slurm. Quant à OAR, considéré alors comme un outil de recherche, il n'a pas été retenu. Ensuite, une grille de critères pondérés a été établie pour évaluer l'ensemble de ces candidats le plus objectivement possible. Cette grille reflète les besoins du CC-IN2P3 concernant son système de batch, en fonction de ses particularités et des technologies en place à cette période. Les différents critères de cette grille portaient sur : la scalabilité, la robustesse, la mutualisation des ressources, le système d'ordonnancement, la gestion des serveurs de calcul, la facilité d'administration et d'exploitation, l'interfaçage avec les intergiciels fournis pour les grilles de calcul, le support des jobs parallèles, le support des jobs interactifs, la qualité du stockage des informations, les contraintes de mise en œuvre, le support d'AFS[8], le système de comptabilité, le support du logiciel. Dans un premier temps le coût d'acquisition et le coût d'exploitation ont été volontairement ignorés. Les points ayant la plus forte pondération étaient la scalabilité, la robustesse, la facilité d'administration et d'exploitation, le support des jobs parallèles et le support d'AFS.

L'évaluation des candidats s'est faite à partir de leur documentation, de réunions avec les commerciaux le cas écéhant et de retours de la part d'utilisateurs avancés. De cette évaluation, deux candidats se sont largement détachés, présentant des caractéristiques techniques très comparables : LSF (Platform) et Grid Engine (édité par SUN puis Oracle). Ils ont ensuite été évalués de manière plus précise. En plus de tests fonctionnels, d'importants tests de charge ont été effectués parmi lesquels la soumission de 500 000 jobs suivie de leur destruction, l'exécution d'un flux autoalimenté de 1 000 jobs de façon à obtenir 100 000 jobs exécutés par heure, ceci sur plusieurs jours. Ces tests ont été réalisés avec un paramétrage proche de celui de la production. Des métriques ont été relevées à cette occasion, telles que les temps de réponse, l'évolution de l'occupation de la mémoire et de l'espace disque sur le serveur. A ce stade, des critères plus subjectifs ont alors été introduits tels que les facilités d'installation, de prise en main, et de configuration. D'autres critères, comme les RoadMaps et les documentations, ont aussi été évalués. Cette seconde phase d'évaluation a été assez longue (4 mois), afin de s'assurer que le produit choisi réponde bien aux attentes, et limiter ainsi les surprises lors de sa mise en production. Ceci a déterminé le choix de Grid Engine.

Cette décision a été prise juste après le rachat de SUN par Oracle et l'annonce de ce dernier de ne plus s'investir dans les versions libres. Il se posait donc la question de pérennité du produit. Suite à une conférence téléphonique avec le directeur du programme Grid Engine chez Oracle, la pérennité du produit a été confirmée, et des développements futurs ont même été présentés. De plus, il se mettait en place une communauté autour du produit libre avec possibilité de support payant (UNIVA[9]). Ce dernier point apportait une sécurité supplémentaire dans l'hypothèse d'un changement politique d'Oracle. Tout ceci a donc conforté le choix.

4.3 Mise en production

Il a été décidé de commencer le déploiement avec la version libre puis le centre a acheté des licences OGE fin juin 2011. Des contournements techniques ont dû être mis en place, consistant principalement en l'introduction de scripts exécutés dans les prologues et épilogues des jobs, respectant ainsi une volonté d'éviter d'importants développements. Parmi les adaptations majeures, le CC-IN2P3 utilisant le système de fichiers distribué AFS, il a fallu intégrer la gestion de ce type de jetons. De même, un contrôle de l'espace disque sur les serveurs de calcul a été implémenté. Par ailleurs, l'adaptation du middleware LCG a été effectuée en collaboration avec le CESGA : le plug-in GE pour CREAM[10] a été modifié afin de prendre en compte la politique locale du site et les demandes spécifiques des utilisateurs, comme c'était déjà le cas avec BQS. Les modifications permettent d'adapter le job de l'utilisateur en fonction de critères tels que par exemple l'organisation virtuelle (VO, Virtual Organisation), la file d'attente visée, le rôle et le DN (Distinguished Name) de l'utilisateur. Ces adaptations concernent notamment le droit ou non d'écrire dans AFS, la définition de certaines variables d'environnement, l'ajout d'informations nécessaires à la comptabilité des jobs, et la mise en application des règles de sécurité du CC-IN2P3. Par contre, certaines qualités de l'ancien système BQS ne se retrouvent pas dans le nouveau service, comme par exemple la facilité d'accès aux informations sur les jobs, ou la régulation fine de la mise en exécution des jobs au regard de la charge induite sur les systèmes de stockage ou les bases de données.

La migration de ce système de batch s'est déroulée en plusieurs phases : ouverture d'une plateforme de tests aux utilisateurs début mai 2011 avec 900 cœurs, passage en production fin juin avec 3000 cœurs et, début septembre 2011, il y avait 10000 cœurs soit 58% de la puissance de calcul du centre. À ce stade il est apparu des problèmes de stabilité du produit, mais la réactivité du support Oracle permet de maintenir l'objectif de finaliser cette migration début décembre.

En conclusion, le succès de la migration vers ce nouveau système découle en grande partie de l'étude approfondie qui a permis de mettre en adéquation les caractéristiques d'un produit avec les spécificités du centre. Par ailleurs, l'un des objectifs initiaux, consistant à diminuer le cout en ressources humaines nécessaires au fonctionnement du gestionnaire de batch (exploitation, maintenance, améliorations) a été atteint et s'est traduit par une diminution de 3 équivalents temps-plein à 1.5.

Si le cas du CC IN2P3 est particulièrement complexe compte tenu de la taille de l'infrastructure et de son importance pour la communauté, des politiques d'exploitation de centres plus petits peuvent également s'avérer dimensionnantes pour les gestionnaires de ressources. Nous voyons dans la suite la cas du méso-centre de l'Univesité de Strasbourg.

5 Méso-centre et maxi-besoins

5.1 Quelques caractéristiques

Le méso-centre de l'Université de Strasbourg (UdS) opère à ce jour une centaine de noeuds de calcul sous Linux. Ces noeuds présentent les caractéristiques :

- d'être financés directement par les les budgets des laboratoires (volontairement) partenaires ;
- d'être hétérogènes en générations de processeurs, en fonction des dates d'achats dépendant des budgets ;
- de faire tourner principalement des applications parallèles sous MPI.

La politique d'exploitation garantit qu'un laboratoire ayant contribué à **x%** d'une génération de machines se verra restituer un temps de calcul correspondant exactement à **x%** du temps CPU de la génération considérée. La puissance de calcul qui resterait disponible est attribuée soit à d'autres laboratoires ayant financé des machines (*contributeurs*), soit à la communauté (*non-contributeurs*). Les application des non-contributeurs peuvent être *préemptées* (endormies) par celles des contributeurs.

En 2008 s'est posée la question du renouvellement des gestinnaires de ressources et de batch. Sur la base de la politique d'exploitation présentée ci-dessus (et précedemment dans [11]), les critères de choix étaient (par ordre d'importance) :

- capacité à intégrer finement des applications MPI : arrêtêr les processus, les redémarrer, les tuer ;
- ordonnanceur permettant de répartir les cycles CPU : fonctionnalité de type *fairshare* ;
- coût nul afin de diminuer le coût d'intégration d'un noeud de calcul dans le système (coût supporté par les laboratoires).

Les 3 produits en lice étaient Slurm, SGE, Torque. Le couplage des 3 avec l'ordonnanceur Maui a été testé, afin d'augmenter les capacités intrinsèques de ces logiciels. En 2008 ont été **éliminés** :

- SGE pour l'absence de couplage avec les jobs MPI ;
- Slurm pour la difficulté de couplage avec Maui et les faibles possibilités d'ordonnancement du produit à l'époque.

Partant sur la base de Torque couplé avec Maui, nous avons pu (dû ?) profiter des fonctionnalités avancées des produits pour traduire différentes subtilités de notre politique d'exploitation.

5.2 Ce qui a bien marché...

Dans Maui, nous utilisons la notion de *qualité de service* (QOS) qui permet d'affecter différentes caractéristiques à un job en fonction des groupes (Unix) d'appartenance, des identifiants utilisateurs, ou de la file d'attente chosie. Par exemple, un utilisateur peut bénéficier d'une QOS lui permettant de préempter des jobs quand il utilise la file d'attente *a*, et d'y renoncer s'il utilise la file *b*. De plus, certains utilisateurs peuvent choisir, *dans une file d'attente donnée*, la QOS à utiliser. En effet, certains bénéficient de plusieurs QOS, selon qu'ils accèdent aux machines achetées en leur nom propre (ANR) ou au nom de leur laboratoire d'appartenance (fonds du labo). Une fois passée la barrière de la configuration hermétique et non-documentée dans les manuels de référence, Maui calcule en général correctement la QOS attribuée à un job.

Sur notre site, les QOS sont les objets centraux de l'ordonnancement. En effet, si elles autorisent la préemtion, elles portent aussi et surtout les informations du nombre de CPUs achetés par un groupe ou un utilisateur. Le nombre de CPUs associés à un groupe se traduit par un pourcentage du total d'heures CPU à restituer sur une fenêtre de temps donnée (*fairshare target* dans la logique Maui). Nous devons donc calculer à la main (feuille de calcul) le pourcentage de cycles correspondant à un nombre donnés de machines.

Afin d'augmenter le nombre de jobs traités simultanément, nous avons décidé après 18 mois de fonctionnement de partager les noeuds entre plusieurs jobs. Pour garantir le parfait isolement des ressources, nous utilisons les fonctionnalités de Torque permettant de manipuler des *cpusets* (boîtes étanches de CPU). Au début de chaque job, Torque crée un cpuset contenant les coeurs retenus pour l'exécution. Le contenu du cpuset est transmissible à certaines implémentations de MPI (nous utilisons Openmpi) qui sait placer un processus par coeur, **et ce de manière transparente pour l'utilisateur**. Afin d'éviter la contention mémoire, nous avons également imposé la spécification de la quantité mémoire *par processus* nécessaire pour le job. Cette quantité par processus est la seule que Torque peut mesurer et correctement.

Enfin, dans le couplage Torque/Maui, il est possible d'exprimer les mêmes éléments de configuration dans l'un ou l'autre des logiciels. Dans ce cas nous en chargeons Torque, ce qui permet de renvoyer des messages d'erreur explicites aux utilisateurs.

Sur une configuration relativement homogène en générations de machines – processeurs équivalents, quantité de mémoire par coeur égale, pas de GPU – le couplage Torque/Maui est relativement stable et opérationnel. Contributions des laboratoires aidant, quand nous avons ajouté de l'hétérogénéité dans notre configuraiton, la course d'obstacles à commencé...

5.3 Les murs infranchissables... et comment les contourner

Le principe de fonctionnement du méso-centre de l'UdS est d'accueillir régulièrement des machines financées par des groupes de recherche. Ainsi, sur une base de configuration relativement homogène mise en production en 2009 se sont greffées des machines supplémentaires dès 2010. Entre temps, la technologie de processeur ayant évolué (arrivée du Nehalem), la puissance relative des nouveaux processeurs par rapport à leurs ancêtres de début 2009 était sensiblement plus élevée. En conséquence, 1 heure CPU de 2010 est bien plus précieuse qu'une heure CPU 2009, de plus *cela dépend de l'application*.

C'est pourquoi on se doute qu'un laboratoire ayant acheté des machines en 2010 pour faire tourner son application favorite ne souhaitera pas tourner sur des machines achetées en 2009. Il est donc indispensable qu'à la soumission, on puisse préciser la génération de machines à utiliser. C'est pourquoi dans Torque nous avons crée une file par génération de machines. Il reste encore à répartir les heures CPU entre laboratoires au sein d'une génération de machines. En effet, nous devons exprimer des règles comme *le groupe A peut utiliser x% des machines de génération 2010, mais 0% des machines génération 2009*. Il s'agit ainsi de autrement dit de fairshare **hiérarchique**, qui doit pouvoir être calculé sur un sous-ensemble de ressources.

Cependant, dans Maui, le Fairshare n'est par hiérarchique : toute heure CPU contribuera au total des heures CPU du cluster. Pour transormer ce fairshare à un niveau en fairshare hiérarchique, nous avons calculé pour chaque groupe le rapport entre le nombre de coeurs qu'il avait apportés et le nombre total de coeurs du système. Associé à la mise en correspondance file d'attente ⇔ génération de machines, cela permet d'assurer que les heures CPU consommées le seront sur le bon contingent de processeurs. Seule l'association de ces 2 mécanismes permet d'exprimer les règles souhaitées : Maui comptabilise les heures CPU totales et Torque restreint l'utilisation à un sous-cluster. Ouf !

16% des machines achetées en 2010 contenaient chacune 2 GPU. En termes de performances, il va du GPU comme du CPU : il n'est pas possible de partager un GPU entre plusieurs programmes. Pour cela, si l'on s'en tient à la documentation, il suffit d'associer une *ressource générique* à une machine, par exemple "Cette machine à 2 GPU". Dans le principe, chaque fois qu'un job réserve un GPU, le nombre de GPUs disponibles est décrémenté. Bien que parfaitement documenté, cette fonctionnalité est inopérante dans Torque/Maui. Pour la contourner, nous avons donc dédié 1 coeur par GPU sur chaque machine équipée, et permis à des utilisateurs nommés d'utiliser ces coeurs, par le biais des *réservations* de Maui. Ce fonctionnement présente encore des failles et nous n'avons jamais réussi à faire fonctionner les ressources génériques.

Une des fonctionnalités importantes des gestionnaires de ressources est la possibilité de lancer des tableaux de jobs. Il s'agit de jobs qui exécutent tous la même tâche et ne varient qu'aux jeu de paramètres d'éntrée. Si cette fonctionnalité est bien présente dans Torque, le fait de l'utiliser à inévitablement causé des pannes le gestionnaire de ressources sur le long terme. Dans l'impossibilité de corriger ce bug, nous en sommes donc venus à ne pas faire de publicité aux job arrays à nos utilisateurs.

Dans le couple gestionnaire de ressources/ordonnanceur, le gestionnaire de ressources (Torque) obéit relativement aveuglément au gestionnaire de batch (Maui). Ainsi, si un job n'arrive pas à démarrer sur un noeud en raison d'une erreur système, ce job repassera en file d'attente. Au prochain tour d'ordonnancement, Maui essaierai de le replacer sur le noeud en question, et ainsi de suite. Il peut arriver que 50% du cluster soit vide simplement pour cette raison (non remontée du statut « en panne » dans l'ordonnaceur).

En conclusion, le couplage Torque/Maui a été mis à mal par l'augmentation du nombre et de l'hétérogénéité des ressources de calcul. Si nous avons réussi à mettre en place un pseudo fairshare hiérarchique au prix d'une configuration truffée de valeurs obscures, la gestion des GPU, la détection de pannes et dans une moindre mesure les job arrays ne fonctionnent toujours pas. Malgré la taille modeste du méso-centre de l'UdS, nous avons atteint très vite les limites du produit et nous travaillons à son remplacement.

6 Conclusion

Dans cet article, nous avons présenté une étude méthodologique nouvelle sur les gestionnaires de ressources, ainsi que trois retours d'expérience qui montrent des tendances différentes. Il ressort de ces éléments que :

1. les gestionnaires de ressources offrent une combinatoire très large de fonctionnalités et de critères de choix ;
2. si le développement d'un gestionnaire de ressources est un travail long et délicat, certains développements universitaires héxagonaux ont atteint une maturité leur permettant d'implémenter des politiques d'ordonnancement compliquées et d'êtred déployées en production sur des sites de taille méso-centriques
3. la standardisation et l'évolution des fonctionnalités des logiciels existants permet, dans le même temps, au CC IN2P3 d'effectuer une migration réussie en libérant des ressources humaines fort utiles dans un contexte de production ;
4. les politiques d'exploitation adpatées au mieux à la demande peuvent mettre en difficulté des outils gratuits sur un méso-centre.

Le choix d'un gestionnaire de ressources est très structurant et nous espérons que cet article vous aura proposé quelques éléments de lecture.

7 Bibliographie

[1] Yiannis Georgiou, Contributions à la Gestion de Ressources et de Tâches pour le Calcul de Haute Performance . *Thése de doctorat de l'université de Grenoble* , 5 novembre 2010.

[2] Le gestionnaire de tâches et de ressources : OAR, *http://oar.imag.fr*

[3] CC-IN2P3 http://cc.in2p3.fr/

[4] LHC http://www.lhc-france.fr

[5] CERN http://public.web.cern.ch/public/

[6] AMS http://www.ams02.org

[7] BQS http://cc.in2p3.fr/docenligne/258

[8] AFS http://www.openafs.org/

[9] UNIVA http://www.univa.com/

[10] CREAM http:/LCG/www.eu-emi.eu/products/-/asset_publisher/z2MT/content/cream-1

[11] Romaric David, Mutualisation des ressources de calcul parallèle à l'Université Louis Pasteur, exemple d'article dans les actes en ligne d'une conférence. Dans *Actes du congrès JRES2007*, http://2007.jres.org/planning/pdf/17.pdf