CIGRI : Expériences autour de l'exploitation d'une grille légère[1]

Yvan Calas
Laboratoire Informatique et Distribution -IMAG
calas@lirmm.fr

Nicolas Capit
Laboratoire Informatique et Distribution -IMAG
Nicolas.Capit@imag.fr

Estelle Gabarron
Laboratoire Informatique et Distribution -IMAG
Estelle.Gabarron@imag.fr

Résumé

Cet article présente le gestionnaire CIGRI, qui répond à une certaine problématique de la gestion de ressources en environnement de grappes de calcul. Cet outil permet de fédérer un ensemble de grappes exploitées par des gestionnaires de ressources (comme OAR) et les utilise pour contrôler l'exécution de campagnes importantes de programmes multi-paramétriques (sac de plusieurs dizaines de milliers de tâches). Dans ce contexte la grille est vue comme une fédération de grappes avec une administration simplifiée, nous parlons alors de grille légère.
Ce système utilise une base de données comme élément central d'échange et de stockage d'informations. La quantité importante de tâches soumises par le type d'applications visées nous amène à considérer la problématique des erreurs survenant dans un système complexe. Notre analyse de la base de données s'est dans un premier temps portée sur l'occupation des grappes par CIGRI. Puis nous avons analysé l'ensemble des évènements survenus au cours de l'utilisation qui a montré que sur plus d'un million de tâches soumises, moins de $0,01\%$, se sont terminées sans que le système ait su les gérer, ou que l'utilisateur n'ait tenté de les resoumettre.

Mots clefs

grille légère, exploitation des ressources inutilisées, gestion des erreurs, grande échelle

1 Introduction

Les grilles tiennent à l'heure actuelle une place de plus en plus importante dans le monde de la recherche et bien des projets tournent autour de ce sujet. Une grille consiste à partager des ressources entre plusieurs utilisateurs ou communautés d'utilisateurs. On peut partager aussi bien l'espace de stockage que les cycles processeurs, la mémoire, certaines applications, ou bien encore une architecture matérielle se trouvant sur un autre site. Il est ainsi possible pour un centre disposant de peu de moyens de pouvoir accéder à des ressources plus importantes ou dont il n'aurait pas pu disposer autrement.
Parmi les problématiques importantes liées à la mise en place d'une grille, on peut noter l'hétérogénéité des ressources, ou bien encore la volatilité de ces dernières. Se pose également un problème de sécurité. Chaque site dispose en effet généralement de sa propre politique de sécurité afin de protéger la confidentialité de ses données, ce qui empêche souvent une communication simple et efficace entre les sites.

Parmi les multiples projets dans ce domaine, on peut distinguer plusieurs types d'architectures[1] : l'architecture de type grille qui reunit un ensemble de grappes reparties sur plusieurs sites. Avec un profil plus applicatif, on peut s'appuyer également sur un modèle de type client/serveur ou MPI avec des architectures hétérogènes. Un des projets phare est le logiciel Globus [2]. Ce logiciel a débuté en 1997 avec Ian Foster du Argonne National Laboratory, et a déjà subi de nombreuses évolutions. Sa mise en place se compose de quatre piliers : la sécurité (GSI), l'allocation des ressources (GRAM), la découverte et l'indexation des ressources (MDS) et enfin la gestion des données (GridFTP et RFT). Ce système très complexe est fondé dans sa dernière version sur les services web normalisés. GSI est reconnu comme efficace dans la gestion de la sécurité, mais relativement lourd à mettre en place et n'est pas primordial dans la mise en place de la grille grenobloise comme on l'expliquera par la suite.
Les architectures de type desktop computing constituent le deuxième grand type de projets dans ce domaine. On peut citer notamment les systèmes pair à pair. Ces architectures sont décentralisées, il n'y a donc plus de structure hiérarchique comme c'est le cas pour le type client/serveur. Le plus populaire est *seti@home* [3] qui consiste à répartir l'exécution de tâches sur un ensemble de machines volontaires reliées à l'Internet.

Ce papier étudie la mise en œuvre de CIGRI, une grille dite «légère», où certains problèmes complexes à mettre en

[1]Le travail a été effectué dans le cadre du projet Mescal (CNRS, INPG, INRIA et UJF), du projet LIPS (Bull et INRIA) et la CIFRE numéro 924/2003.

œuvre (comme la sécurité ou l'hétérogénéité des machines) ne sont pas pris en compte. Cela garantit une plus grande facilité d'implantation, et une meilleure souplesse dans l'administration des applications.

2 Le gestionnaire pour grille légère CIGRI

CIGRI permet de mettre en commun l'ensemble des ressources des différents sites et utilise leurs puissances respectives quand elles sont inutilisées localement.

Le logiciel CIGRI est le résultat de l'ACI GRID CIGRI visant à développer une grille grenobloise au sein de la communauté CIMENT (Calcul Intensif, Modélisation, Expérimentation Numérique et Technologique). Cette communauté est composée de plusieurs laboratoires qui ont décidé de mettre en commun leurs différents moyens de calcul :

- *Techniques de l'Imagerie de la Modélisation et de la Cognition* (TIMC-IMAG).
- *Informatique et Distribution* (ID-IMAG).
- *Laboratoire de Physique et Modélisation des Milieux Condensés* (LPMMC).
- *Laboratoire d'AstroPhysique de Grenoble* (LAOG).
- *Laboratoire de Modélisation et Calcul* (LMC-IMAG, MIRAGE).
- *Centre d'Expérimentation du Calcul Intensif en Chimie* (CECIC).

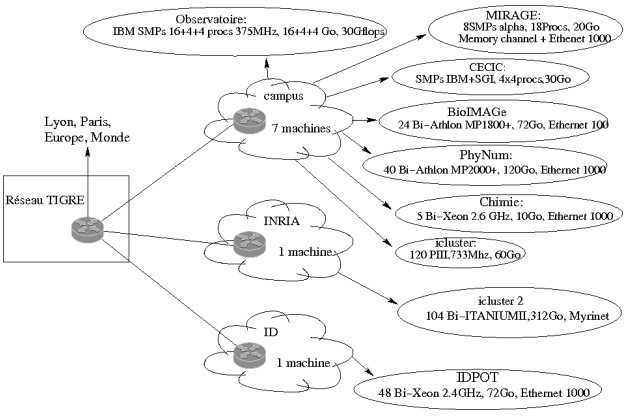

Figure 1 – *Récapitulatif des différentes machines de la communauté CIMENT.*

Le concept de grille légère

Le logiciel CIGRI est caractérisé par la notion de grille légère. En effet afin de focaliser les problèmes de recherche et développement sur certaines problématiques essentielles comme l'exécution des tâches, tous les problèmes inhérents aux grilles de calcul n'ont pas été traités. La proximité géographique et les relations suffisament étroites entre les différents laboratoires permettent dans un premier temps de ne pas avoir à centrer les efforts sur des problématiques telles que l'authentfication des utilisateurs par une unité centralisée et sécurisée, qui est un module coûteux et contraignant à mettre en place. D'autre part les différents sites (de la figure 1) sont relativement homogènes, pas forcément en terme de type de machines mais surtout en terme de choix d'administration. De plus la communauté CIMENT permet des échanges de compétences et de savoir faire entre les différents domaines (par exemple, le laboratoire ID-IMAG apporte son expertise et son savoir faire dans les grappes de PC). Les machines ne sont donc pas cloisonnées, leur administration est assez souple et elles sont ouvertes à l'expérimentation. La notion de grille légère découle de ce contexte de proximité et de souplesse.
Dans sa première version, CIGRI vise les applications de type multi-paramétriques comme par exemple les programmes reposant sur un algorithme de type Monte Carlo (comme ceux de Pierre Valiron[4] en astrophysique moléculaire). Ce type de calcul est assez répandu et génère un nombre de tâches très important. Le but est de pouvoir traiter le flot d'exécution de sacs de tâches et de garantir que tout s'est bien passé (l'ordre de grandeur d'une campagne d'évaluation est de 100 000 tâches).

L'utilisation de CIGRI nécessite la disponibilité d'un gestionnaire de ressources sur chacun des sites. OAR[5] en est un exemple. Ce gestionnaire a été développé afin d'implémenter des fonctionnalités qui n'existent pas dans les systèmes existants tels que PBS[6], Condor[7] ou bien encore SGE. Son but est de rester dans une complexité de code faible via une approche de haut niveau. Cela permet une spécialisation simplifiée du gestionnaire et par là de tester de nouveaux algorithmes d'ordonnancement ou de nouveaux modes d'exploitation.

Dans la suite de ce paragraphe, l'architecture de CIGRI est présentée ainsi que son fonctionnement.

2.1 Architecture

L'architecture du logiciel CIGRI se compose d'un serveur qui communique avec tous les gestionnaires de ressources des grappes. Le logiciel se comporte comme un utilisateur virtuel qui va s'occuper de soumettre les tâches et récuperer les résultats de manière automatique. L'objectif est d'être le moins intrusif possible sur les sites. Contrairement à des approches comme Globus[2][8] (qui utilise la notion de "GateKeeper"), aucun logiciel spécifique CIGRI n'est installé sur les grappes. Les outils système classiques sont utilisés (ssh, bash, cat,...).

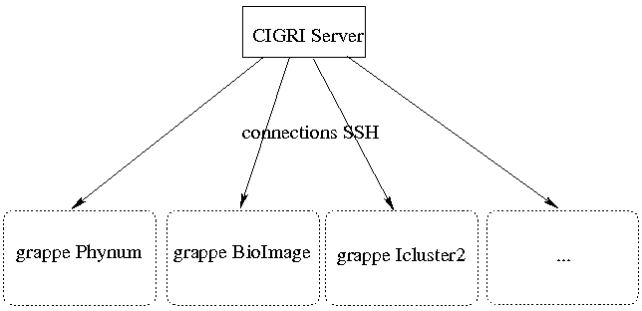

Figure 2 – *Modèle de communication avec les différents sites.*

Un problème important à résoudre est celui de la politique d'utilisation des différentes grappes. En effet grâce à ce mode grille, des utilisateurs étrangers à un site peuvent utiliser la puissance de calcul de celui-ci. La notion de tâches «best-effort» a été introduite dans OAR pour résoudre ce problème. Elles ont la particularité d'avoir une priorité nulle, et elles sont tuées si une soumission locale a besoin de leurs nœuds. Il est donc nécessaire d'avoir des programmes qui résistent à l'arrêt brutal et à la resoumission. Les applications doivent être idempotentes, c'est-à-dire que leur exécution doit toujours donner le même résultat pour un jeu de paramètres donné et ne dépend pas d'autres évènements. Cette approche résout le problème de la cohabitation entre personnes de différentes communautés.

Au niveau de la conception du logiciel, nous avons choisi une conception de haut niveau articulée autour d'une base de données, permettant ainsi d'abstraire plus facilement les structures de données présentes dans CIGRI. Celle-ci permet de gérer un grand nombre d'informations et apporte une certaine robustesse. Le logiciel CIGRI est conçu selon une décomposition en modules interagissant par l'intermédiaire de cette base de données, qui contient les informations suivantes :
– Etat de tous les nœuds de toutes les grappes.
– Etat des travaux soumis (pour le suivi).
– Evènements (erreurs, resoumissions, ...).
– "log" de tout ce qui se passe (permet de tracer et donc de faciliter le diagnostic d'une panne ; permet également de générer des statistiques d'utilisation)
– Informations sur les utilisateurs.

Les modules constituant CIGRI sont au nombre de 5, comme le montre la figure 3 :
– "Updator" : permet de mettre à jour la base de données (état des nœuds), de connaître l'état des tâches.
– "Scheduler" : détermine les tâches à soumettre (+localisation).
– "Runner" : lance les tâches sur les grappes comme un utilisateur standard.
– "Nikita" : module de nettoyage (tue les tâches).
– "Colombo" : gestion des évènements (erreurs, tâches tuées, ...), prises de décisions (retrait d'une application, d'une grappe, ...).

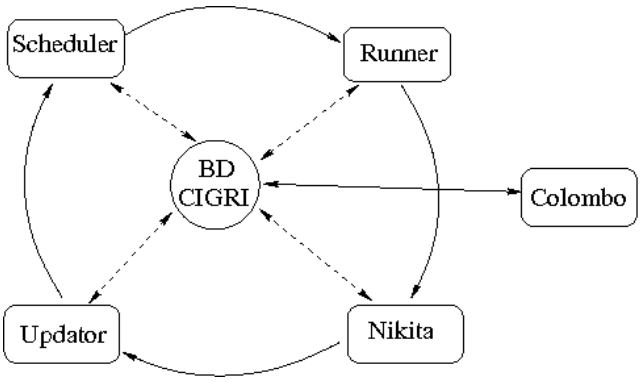

Figure 3 – *Organisation des modules de CIGRI autour de la base de données.*

2.2 Fonctionnement

Les applications exécutées sont de type multi-paramétriques. La durée de chacune doit être relativement courte pour augmenter leur probabilité de terminaison. Bien entendu, le système gère le caractère précaire des tâches (resoumission automatique si une tâche est détruite).
CIGRI permet de soumettre des campagnes complètes d'exécution (de l'ordre de 100 000 tâches) et traite un certain nombre de cas d'erreurs.
Différents dysfonctionnements sont détectés (code de retour différent de 0 des applications, pannes réseau, fin de temporisation sur exécution distante, incohérences...). Par exemple, si une communication avec une grappe est trop longue (toutes les actions distantes sont obligatoirement accompagnées d'un temps maximum d'exécution), cette dernière sera retirée et l'administrateur de la grille sera notifié que les tâches ne peuvent plus s'exécuter dessus (c'est le type de règles de décisions que l'on retrouve dans le module "Colombo").

Une autre fonctionnalité importante est la collecte des fichiers résultats et le nettoyage des fichiers temporaires. En effet, le problème provient du grand nombre de tâches à exécuter. Il n'est pas souhaitable de laisser les données sur les grappes. Il existe donc un module auxiliaire au système qui est chargé de collecter périodiquement les fichiers résultats et de les archiver sur le serveur de la grille. Ainsi il n'y a plus de problème en terme de nombre de fichiers et l'utilisateur peut récupérer ses calculs à un seul endroit centralisé. Les données sont rapatriées en trois étapes, qui sont effectuées toutes les 30 minutes :
– L'ensemble des fichiers résultats sont archivés sur le nœud dans un fichier de type tar.
– Les archives sont transférées vers le serveur par une commande scp.
– Après vérification des codes de retour attestant que le transfert s'est bien déroulé, les archives ainsi que les fichiers qui ont été archivés sont supprimés des nœuds.

Cette méthode pragmatique fonctionne, même si son efficacité n'a pas été validée. Des travaux sont en cours sur ce sujet. Il est à noter qu'en pratique actuellement, la taille des fichiers de résultats en sortie est relativement petite, ce qui ne pose pas de problème du point de vue de la saturation du serveur CIGRI.

Lors d'une soumission de type grille, l'utilisateur doit décrire sa campagne de tâches au système. Pour cela il écrit un fichier communément appelé "JDL" (*Job Description Langage*). A l'intérieur il va définir les grappes sur lesquelles il souhaite travailler et les programmes associés à chaque grappe, ainsi que le nom du fichier contenant tous les paramètres à dérouler. Chacune des lignes de ce fichier correspond à une tâche à lancer. CIGRI décidera, lors de l'exécution, sur quelle grappe celle-ci sera effectivement traitée, en fonction de la disponibilité.

L'utilisateur peut suivre l'évolution de sa campagne via un portail (Figure 4). Ses fonctionnalités sont :
– Affichage de statistique d'utilisation des grappes.
– Possibilité de suivre les tâches terminées, en cours d'exécutions et en attentes par utilisateur et par campagne.
– Possibilité de prendre des décisions concernant des tâches en erreurs (annulation ou resoumission).

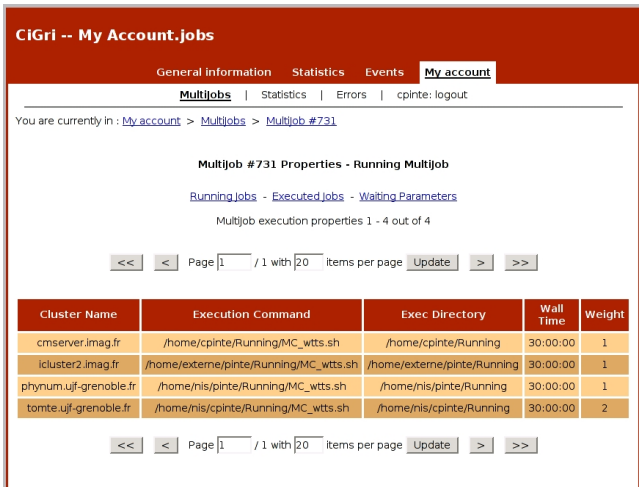

Figure 4 – *Suivi web d'une campagne de tâches.*

Voici les étapes d'un scénario type d'utilisation de la grille avec le logiciel CIGRI :

1. Un utilisateur se manifeste auprès de l'administrateur de la grille, avec une importante quantité de tâches à exécuter.
2. Un compte est créé par l'administrateur de la grille sur chacun des sites sur lesquels il veut lancer son application. Dans la mesure où on est dans un contexte de grille légère, l'aspect administration est simplifié.
3. L'administrateur de la grille s'assure que l'utilisateur a bien compris le fonctionnement de la grille, et que son application donne des résultats cohérents. En particulier, les fichiers sur les grappes locales sont gérés par un serveur nfs, et il est important d'avoir l'unicité dans les noms de fichiers résultats.
4. L'utilisateur écrit son JDL ("Job Description Langage").
5. L'utilisateur soumet son JDL et son fichier de paramètres à la grille (commande gridsub).
6. Le gestionnaire CIGRI déroule le JDL automatiquement, en soumettant les tâches sur les machines libres des grappes souhaitées.
7. Un module auxiliaire collecte les fichiers résultats et les stocke sur le serveur hébergeant CIGRI

Voici l'exemple d'un fichier "JDL" qu'un utilisateur peut soumettre à CIGRI :

```
DEFAULT{
    paramFile = param.tmp;
}
idpot.imag.fr{
    execFile = /users/home/capitn/test.sh ;
}
tomte.ujf-grenoble.fr{
    execFile = /users/nis/capitn/test.sh ;
}
```

Le JDL est composé de plusieurs sections, dont une première nommée «DEFAULT» pour les paramètres communs à tous les sites. Les autres sections correspondent aux grappes sur lesquelles la campagne de tâches s'exécutera. "execFile" détermine le programme à lancer et "paramFile" le nom du fichier qui contient tous les paramètres de la campagne (une ligne de ce fichier correspondra à une tâche lancée sur une grappe).

Bien entendu il existe une commande qui permet à l'utilisateur de tuer une campagne ou une tâche précise de la grille (griddel).
Dans la suite nous allons analyser dans quelle mesure les tâches soumises par CIGRI interfèrent avec les tâches soumises localement.

3 L'occupation des grappes par CIGRI

3.1 La priorité dite *best-effort*

Les tâches soumises par CIGRI sur les grappes des différents laboratoires ne doivent pas perturber l'utilisation de ces grappes par les utilisateurs locaux. En effet l'idée de CIGRI est d'utiliser seulement les cycles processeurs non

utilisés sur la grappe. Un site ne souhaite effectivement pas donner des cycles processeur dont il aurait besoin. Ainsi les tâches soumises par CIGRI sur les grappes locales sont soumises au gestionnaire de tâches dans une queue dite «best-effort». Cette tâche possède une priorité d'exécution de 0 et ne sera donc lancée que lorsqu'une ressource sera disponible et non demandée par une tâche soumise par un utilisateur local. En outre, si au cours de l'exécution de cette tâche la ressource est demandée par un utilisateur local, la tâche de CIGRI est tuée et replacée dans le sac de tâches à effectuer.
Il est donc intéressant dans ce contexte de s'intéresser à l'occupation des tâches grille sur une grappe locale, afin de voir d'une part si les tâches locales ne sont pas perturbées, et d'autre part si la grappe est bien utilisée au mieux pour utiliser le maximum de cycles processeur.

D'autres analyses ont porté sur l'occupation des ressources. Dans le cadre de [9], la disponibilité de la grille en terme de puissance de calcul est mesurée suivant deux aspects : d'une part le temps d'utilisation des cycles processeur (comme dans notre cas) et leur répartition sur les ressources, et d'autre part la disponibilité des ressources. Il faut différencier d'un coté la disponibilité d'une machine et d'un autre la disponibilité d'un processeur, c'est-à-dire le pourcentage des cycles du processeur qui sont utilisés ou bien disponibles. Une machine est disponible pour une application, mais cette application ne peut par exemple utiliser qu'un pourcentage des cycles CPU. Cette étude est réalisée avec le système Entropia[10], auquel on soumet un grand nombre de tâches simples. Ces tâches consistent en un grand nombre de boucles de temps fini d'exécution, qui effectuent des opérations sur des nombres entiers ou flottants. A intervalles réguliers, une tâche évalue la quantité de boucles traitées. Cette quantité est rapportée à une valeur de référence qui permet de déterminer le pourcentage de CPU utilisé.

3.2 La démarche et les résultats

Dans notre cas nous avons donc étudié chaque sac de tâches. Rappelons qu'un sac de tâches peut contenir de l'ordre de 100 000 tâches. En pratique, la majorité des sacs de tâches en contiennent de l'ordre de 5 000. Nous avons choisi de considérer une période pendant laquelle les tâches de CIGRI et les tâches locales sont en concurence. Nous avons concentré notre analyse sur une grappe. Une grappe contient des machines homogènes. On peut donc se limiter à regarder le temps d'exécution des tâches puisque les différents nœuds fournissent la même puissance de calcul. Nous avons considéré sur la grappe locale la période comprise entre la date de soumission de ce sac de tâches et la date de fin d'exécution de la dernière tâche contenue dans cet ensemble.
Nous avons regardé le temps occupé par les tâches locales (donc dans la queue dite *default*) qui ont donc priorité sur les tâches CIGRI, et le temps occupé par les tâches de CIGRI qui, elles, sont tuées si la ressource est demandée.

Un effort de filtrage a été nécessaire pour rendre cohérentes les deux bases de données, celle de CIGRI et celle du gestionnaire de ressources sur la grappe locale, les dates de soumissions et de fin de tâches n'étant pas toujours synchronisées entre ces deux bases.
On observe que les tâches locales ont toujours une part plus importante que les tâches CIGRI en terme de temps de calcul. Un exemple typique de mesure de comportement du système sur une journée montre dans la base CIGRI 263 tâches terminées et 51 indiquées comme ayant provoqué un évènement. Parallèlement si on regarde l'exécution des tâches dans la base de données du gestionnaire de tâches de la grappe locale, on observe : 314 tâches besteffort et 48 tâches soumises localement sur la grappe. Les 51 évènements dans la base CIGRI correspondent au type UPDATOR_JOB_KILLED, c'est-à-dire que les tâches ont été tuées par le gestionnaire de tâches, les ressources étant demandées par des tâches plus prioritaires. Il est à noter que certaines machines peuvent avoir plusieurs processeurs. Dans le cas d'une machine biprocesseurs par exemple, deux tâches CIGRI peuvent être exécutées en même temps sur une même machine. Si une tâche locale demande les deux processeurs de la machine, on dit qu'elle a un poids de deux, et elle va provoquer l'arrêt d'autant de tâches besteffort. Sur la Figure 5, on voit que les deux tâches b1 et b2 sont replacées dans le sac de tâches lorsque seulement une tâche locale, mais de poids 2 demande la ressource. On retrouve également ces caractéristiques sur des périodes plus longues d'utilisation.

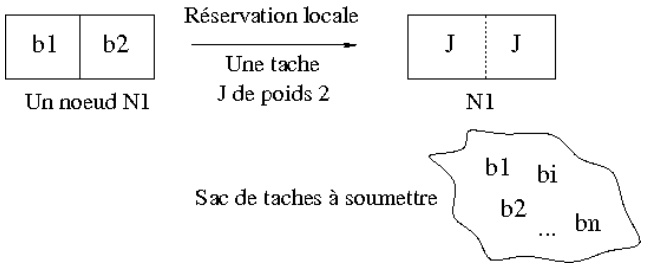

Figure 5 – *Soumission d'une tâche locale de poids 2.*

Dans la suite nous allons nous intéresser plus particulièrement à la gestion des erreurs dans CIGRI, puis nous analyserons l'ensemble des tâches ayant déjà été soumises au système.

4 La gestion des erreurs

4.1 La problématique

Comme nous l'avons déjà mentionné, les applications multiparamétriques entraînent une quantité importante de tâches soumises, et il est important de bien considérer la problématique des erreurs. Le système doit être capable de réagir effi-

cacement en cas de tâche impossible à soumettre, en cas de problème de réseau, de pouvoir faire remonter si une tâche a bien été terminée afin de savoir s'il faut la resoumettre ou non. Plus généralement, il survient des évènements sur la grille que le système peut détecter, il doit être capable de les traiter et d'informer utilement sur la cause initiale.
Chaque projet a sa propre gestion des évènements. Le gestionnaire de ressource Condor[7] utilise un module externe utilisant un langage simple, *Fault Tolerant Shell (ftsh)*, qui se base sur une approche de type Ethernet. Partant du principe qu'une tâche est soit réussie, soit erronée, chaque utilisateur peut selon ses critères et grâce à des boucles temporelles définir la manière de resoumettre ou bien le temps qu'il faut pour tenter la resoumission d'une tâche. Ce système est efficace et fiable puisqu'il a fait ses preuves dans les communications réseaux. Cependant ce principe ne permet pas de spécifier la nature des erreurs. Une tâche réussit ou échoue. Une tâche peut échouer pour de nombreuses raisons : l'exécutable est corrompu, les paramètres sont faux, un problème de réseau est survenu, etc... Cette approche permet de faire face à de nombreuses erreurs dues à des problèmes temporaires, mais ne permet donc pas d'affiner efficacement la gestion des erreurs.

Le projet pair à pair XTremWeb [11] développé à l'université de Paris-Sud se place dans le cadre du desktop computing. Il s'agit de regrouper un ensemble de ressources de calcul à travers internet afin d'exécuter comme dans le cas de CIGRI des applications multi-paramétriques. Un client soumet un ensemble de tâches à un coordinateur qui se charge ensuite de les répartir sur les nœuds de calcul. Ces nœuds peuvent être aussi bien des grappes de calcul que des ressources individuelles réparties sur internet. Ce sont donc les nœuds de calcul qui demandent au coordinateur des tâches à exécuter. Lorsqu'une tâche est exécutée sur un nœud de calcul, le résultat est ensuite retourné au coordinateur, et en cas d'erreur, la tâche est resoumise sur un autre nœud. Pendant l'exécution d'une tâche, le nœud de calcul envoie périodiquement au coordinateur un signal attestant qu'il est toujours en vie, par le système typique du «heart beat».

Le système Entropia[10] se place dans un cadre similaire à notre étude. Ce système permet d'exécuter un ensemble de tâches composées de sous-tâches dans le cadre d'un réseau d'entreprise. Dans la mesure où le système ne s'intéresse qu'aux ressources locales, la problématique de la sécurité est également moins importante, puisque les communications ne vont pas vers l'extérieur et n'ont pas à passer le firewall. Le logiciel DCGrig associé à ce projet intègre la gestion des erreurs. Le système est composé de trois modules :
– Le gestionnaire de tâches (correspondantes aux sacs de tâches dans notre cas)
– Le gestionnaire de sous-tâches (correspondantes à ce que nous avons appelé tâches)
– Le gestionnaire de ressources
Chacun de ces trois modules gère les erreurs le concernant. Dans ce système, chaque sous-tâche a une priorité différente et plus une sous-tâche a échoué, plus elle aura une priorité élévée à la prochaine exécution. Et c'est la ressource qui doit manifester au gestionnaire de tâches qu'elle est toujours active.

Dans le cas du projet Ourgrid[12], une étude particulière a été réalisée afin de déterminer les différents types d'erreurs auxquelles ont été confrontés les utilisateurs. Il en ressort 4 grandes catégories. Les erreurs de configuration ou de paramètres qui sont présentent dans trois quart des cas, les erreurs dues au logiciel de gestion de la grille, les erreurs dues directement à l'application, et enfin les erreurs dues au matériel, mettant en évidence qu'un des principaux problèmes est d'être capable de bien identifier la cause de l'erreur.

Les travaux dans le cas de [13] proposent un modèle de propagation des erreurs dans le cadre d'un système d'exploitation, et de la répercussion des erreurs d'un périphérique sur une application. Le système est vu comme un ensemble de quatre couches : le périphérique, le driver du périphérique, le système d'exploitation et l'application. Dans la mesure où ils partent du principe que chacune de ces quatre couches ne peut être modifiée, l'idée est d'étudier le comportement des erreurs au niveau de l'interface de ces quatre modules. Trois notions intéressantes sont alors définies. La notion de perméabilité aux erreurs, la notion d'exposition aux erreurs et la notion de diffusion de ces erreurs.

Enfin [14] propose une analyse des erreurs sur le prototype BlueGene d'IBM. Ce système est constitué de milliers de machines de puissance raisonnable, mais qui grâce à ce passage à l'échelle place la machine dans les premières places du top 500 en ce qui concerne la puissance de calcul. Ce passage à l'échelle donne une importance toute particulière à la problématique de l'erreur. Cette étude décrit les trois étapes importantes pour l'analyse des erreurs pendant une utilisation très longue du système. La première étape consiste à analyser les différents types d'erreurs. La deuxième étape consiste à filtrer les erreurs correspondant à une même cause : lorsqu'une machine s'arrête ou tombe en panne, la tâche qui était en cours sur la machine va probablement être resoumise plusieurs fois sur la machine avant que le système ne s'apperçoive de la défaillance, et cela va provoquer autant de fois l'erreur avant que la machine soit définitivement bannie. Cet ensemble d'erreurs n'est cependant compté qu'une seule fois puisqu'il résulte de la même cause. Enfin la troisième étape consiste à regrouper les erreurs par classe. Deux erreurs appartiennent à la même classe si l'une résulte de l'autre. Par exemple une erreur dans une carte ou dans un cycle CPU peut engendrer une erreur sur le nœud qui peut engendrer une erreur sur l'ensemble des noeuds commandés par un même switch. L'ensemble de ces erreurs se trouvent dans la même classe.

Dans notre cas, nous avons adopté une approche proche de ce dernier exemple.

4.2 La gestion des erreurs dans CIGRI

Dans un premier temps, nous avons cherché à répertorier les différents types d'erreurs ou d'évènements qui pouvaient intervenir dans notre système et que nous pouvons détecter. Le tableau 6 présente le résultat de notre analyse. Ces erreurs sont traitées par le système en quatre classes : les erreurs relatives au gestionnaire de tâches, celles relatives à la grappe, celles relatives aux tâches soumises et enfin celles relatives aux sacs de tâches. Dans la suite nous nous intéresserons essentiellement à ces deux derniers types d'erreurs, dans la mesure où elles sont directement liées à l'utilisateur et donc à CIGRI. Les deux autres types (gestionnaire de tâches et grappe) ne sont pas liées à l'exécution des tâches mais au fonctionnement du site distant qui est géré par le site distant lui-même. On ne peut donc à notre niveau pas effectuer d'action directe pour remettre un nœud ou la grappe en état. Chaque site distant gère ses machines.

Pour CIGRI, si une erreur revient trop souvent sur un nœud, on émet l'hypothèse que l'erreur est dûe au nœud et non à l'application et le nœud est banni. De même pour une grappe, si trop de nœuds sont bannis, la grappe toute entière est bannie. L'administrateur de la grille est informé par mail des différents évènements. L'administrateur de la grappe locale concernée est alors prevenu et a en charge de remettre le nœud en état de fonctionnement. Lorsque que chaque évènement ayant entrainé le bannissement a été traité, le nœud ou la grappe selon le cas est automatiquement réinséré dans les ressources disponibles.

Afin de traiter ces erreurs, nous avons envisagé dans un premier temps d'utiliser un module externe, utilisant un langage spécifique comme par exemple Esterel[15]. Esterel est un langage de programmation de haut niveau, dédié à la spécification de systèmes réactifs, et permettant par conséquent de maintenir une interaction continue avec son environnement. Ce type de langage est largement utilisé dans les systèmes temps réel. Cependant, dans notre cas, nous souhaitons un système simple et léger à mettre en place. Ce type d'approche était trop complexe pour notre système.
Le module colombo de CIGRI permet de gérer les différents évènements. Ce module récupère les évènements du système retournés par les autres modules (Updator, Runner...) les insère dans la base de données dans un état «tofix». Ensuite, une fonction *check_event* du module permet de traiter les actions que ces évènements entraînent. Cette fonction sera rendue paramétrable par la suite.

Nous nous intéressons aux évènements de type JOB ou MJOB (cf. Figure 6). Si l'erreur vient du fait que le système n'arrive pas à lancer une tâche ou bien que le code de retour de la tâche n'est pas correct, alors le nœud est dans un premier temps indiqué comme inutilisable. Si plusieurs nœuds ont le même comportement, alors la grappe est dans ce cas indiquée comme inutilisable, soit pour toutes les applications, soit uniquement pour la campagne dont la tâche n'arrive pas à s'exécuter correctement.

Type d'évènement	Classe	Description
EXIT_VALUE	SCHEDULER	le scheduler ne se termine pas avec un code de retour = 0
ALMIGHTY_FILE	SCHEDULER	le AlmightyCigri ne trouve pas le fichier scheduler à exécuter
UPDATOR_PBSNODES_PARSE	CLUSTER	l'updatorCigri n'arrive pas à parser le résultat de la commande pbsnodes
UPDATOR_PBSNODES_CMD	CLUSTER	l'updatorCigri n'arrive pas à exécuter la commande pbsnodes
UPDATOR_QSTAT_CMD	CLUSTER	l'updatorCigri n'arrive pas à exécuter la commande qstat
UPDATOR_RET_CODE_ERROR	JOB	l'updatorCigri détecte une mauvaise terminaison du job (return code $\neq 0$)
UPDATOR_JOB_KILLED	JOB	l'updatorCigri a détecté que le job a été tué
RUNNER_JOBID_PARSE	JOB	le runnerCigri n'arrive pas à parser le retour de soumission
RUNNER_SUBMIT	JOB	le runnerCigri n'arrive pas à lancer le job
FRAG	JOB	demande d'arrêt d'un job
FRAG	MJOB	demande d'arrêt d'un multiple job
SSH	CLUSTER	détection d'une erreur du canal SSH avec un cluster
COLLECTOR	CLUSTER	erreur lors de la collecte d'un MJob
MYSQL_OAR_CONNECT	CLUSTER	erreur lors de la connexion à la base de données OAR
QDEL_CMD	CLUSTER	erreur d'une commande oardel
OAR_OARSUB	CLUSTER	erreur d'une commande oarsub
OAR_NOTIFY	CLUSTER	erreur d'un oarnotify

Figure 6 – *Les différents évènements rencontrés dans CIGRI*

En revanche, si une tâche est tuée directement par le système dans le cas d'une demande de ressources sur la grappe (puisqu'on a une tâche best-effort), les paramètres de cette tâche sont remis dans l'ensemble des paramètres à soumettre. La tâche sera donc resoumise ultérieurement. Elle ne sera pas nécessairement resoumise sur la même grappe, mais selon la disponibilité du moment et les choix de l'ordonnanceur de la grille.

5 Analyse de l'utilisation de CIGRI

CIGRI est utilisé régulièrement depuis plus de deux ans. Depuis le début, 7 utilisateurs soumettent régulièrement des tâches pour des expérimentations essentiellement de physique portant sur plusieurs milliers de tâches. Par exemple un cas d'expérimentation est centré sur la simulation d'écho radar à la surface de Mars. La surface de la planète a été quadrillée en de nombreuses zones, correspondant chacune à une tâche à effectuer. Les tâches sont assez courtes de l'ordre de 30 secondes ou une minute. Un deuxième utilisateur a quant à lui réalisé des tâches un peu plus longues de l'ordre de dix minutes. Dans ces expérimentations, l'étude portait sur l'énergie dissipée par la collision de deux molécules en fonction de leurs trajectoires.

Le système CIGRI est comme on l'a dit centré autour d'une base de données. Ainsi toutes les informations relatives à l'exécution des tâches et au fonctionnement de CIGRI sont conservées. Une analyse de cette base permet de tirer de nombreux résultats.
On constate que la répartition des tâches sur plusieurs mois est assez localisée sur certaines périodes qui varient bien évidement selon les utilisateurs. Les campagnes de tâches ne sont donc pas soumises en continu. Cela est dû au temps de traitement et d'analyse des résultats obtenus par l'utilisateur.

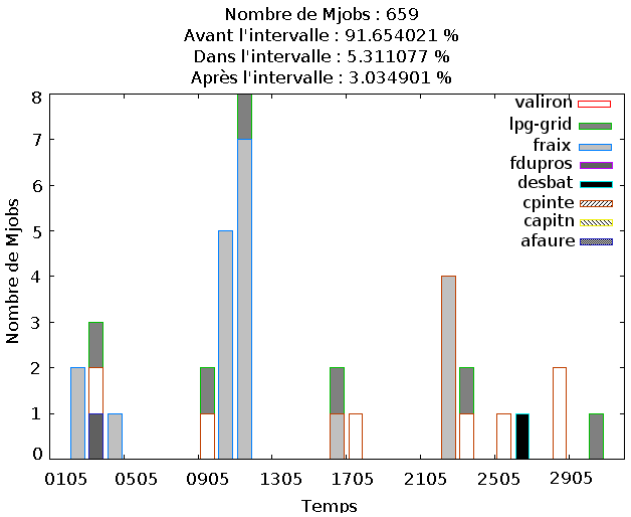

Figure 7 – *Soumission des jobs multiples pour le Mai 2005*

Depuis plus de deux ans, plus de 1 500 000 tâches ont été soumises parmi lesquelles 33 329 ne se sont pas terminées correctement et ont donc été resoumises. Ces tâches correspondent en fait à 15 788 tâches différentes, c'est-à-dire ayant le même paramètre et appartenant à la même campagne de tâches. Parmi ceux-ci 1774 tâches ont été tuées volontairement par l'utilisateur. Plusieurs raisons peuvent inciter l'utilisateur à stopper son travail, notamment s'il se rend compte d'une erreur dans son application.

L'essentiel des autres «erreurs» provient du fonctionnement normal du système. Ce sont des tâches qui ont été tuées par le système quand il y avait une tâche locale donc plus prioritaire qui était demandée. Cela correspond à l'évènement de type UPDATOR_JOB_KILLED.

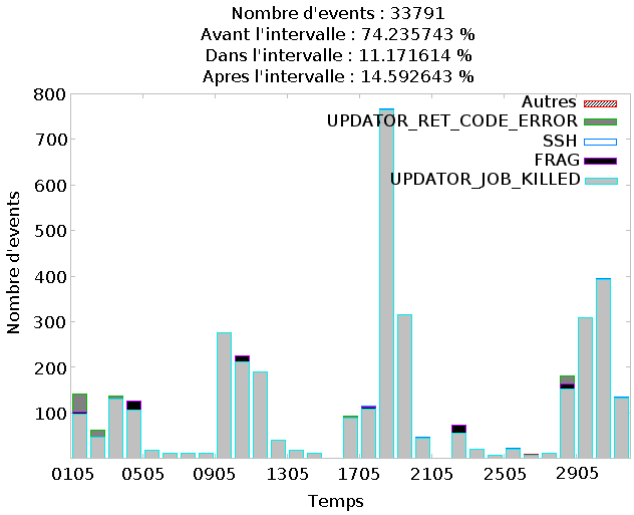

Figure 8 – *Apparition des évènements pour le mois de Mai 2005*

Ainsi les courbes des Figures 7 et 8 montrent la répartition de la soumission des sacs de tâches d'une part et l'apparition des différents évènements d'autre part au cours du mois de Mai 2005. On peut observer une corrélation entre la soumission de tâches multiples et l'apparition de plusieurs évènements de type FRAG, c'est-à-dire des tâches qui ont été tuées volontairement par l'utilisateur lui-même suite à ce qu'il ait tué lui-même le sac de tâches. Les erreurs de type UPDATOR_JOB_KILLED sont les plus fréquentes sur la période que l'on considère dans cet exemple. Ces erreurs correspondent au fait que le gestionnaire de tâches a tué la tâche CIGRI suite à la demande en local de la ressource. A la suite d'un pic de soumission de sac de tâches, on observe le jour même ou le jour suivant un pic d'évènements de type UPDATOR_JOB_KILLED.

On s'est intéressé à l'exemple particulier de la grappe Tomte. Si on recoupe les informations entre la base de CIGRI et celle du gestionnaire de tâche de la grappe (OAR), on s'aperçoit que ces tâches ont été effectivement tuées suite à une demande de ressources d'un autre utilisateur de la grappe. C'est comme nous l'avons vu précédemment dû à la politique de best-effort.

Parmi les tâches non terminées correctement, la plupart ont donc été soit resoumises, soit tuées volontairement par l'utilisateur. Il reste néanmoins 1254 tâches tuées par le gestionnaire de tâches et non resoumises. Cela s'explique simplement par le fait que lorsqu'une tâche est tuée par le gestionnaire de tâches dû à la politique du besteffort, elle est

replacée dans le sac de tâches à soumettre. Si avant qu'elle ne soit resoumise, l'utilisateur tue son application, celle-ci reste donc à l'état non resoumis.

D'autre part plusieures tâches terminent avec un code de retour différent de 0. Dans ce cas, elles ne sont pas resoumises automatiquement par CIGRI, mais l'utilisateur peut au cas par cas resoumettre une tâche grâce à l'interface web de suivi. Parmi les tâches de ce type, la plupart correspondent à des sacs de tâches que l'utilisateur a annulé.
Reste finalement 32 tâches ayant un code de retour non nul, qui n'ont pas été resoumises ni tuées par l'utilisateur et donc par conséquent que le système n'a pas su traiter, ce qui représente moins de $0,01\%$ des tâches soumises.

6 Conclusion et travaux futurs

A l'origine, la communauté CIMENT souhaitait mutualiser les ressources inexploitées des grappes des différents partenaires. Le choix de Globus a été écarté à cause de sa lourdeur d'exploitation. Suite à la définition de grille légère, une approche exploitant la présence de gestionnaire de ressources sur site a été développée (CIGRI).

En deux ans d'exploitation, CIGRI a traité plus de 1 500 000 tâches. Actuellement des campagnes de tâches sont soumises régulièrement. Les performances constatées nous amènent à penser que le système est fiable. Très peu de tâches ne réussissent pas à être exécutées de manière satisfaisante.
Des travaux sont toujours en cours pour améliorer l'analyse du taux d'occupation des ressources. Il s'agit par la suite de considérer l'ensemble des grappes, et donc de prendre en compte la puissance des machines qui peut varier d'une grappe à l'autre.

D'autres améliorations sont en outre possibles afin de gérer de manière plus optimale les évènements et les erreurs de la grille. D'autre part le système actuel ne prend pas en compte l'environnement de l'utilisateur. Son application peut par exemple nécessiter des librairies particulières à l'exécution. Pour soumettre sur plusieurs grappes, l'utilisateur doit actuellement s'assurer du bon fonctionnement de son application en installant ce qui est nécessaire sur chacune des grappes. Plusieurs approches sont envisageables pour un déploiement de l'environnement, qui constitue un point important de notre recherche. Les fichiers de résultats sont également pris en compte dans ces travaux.

CIGRI permet ainsi d'utiliser de manière efficace l'ensemble des ressources de l'agglomération grenobloise, et les performances de cet outil sur une période d'expérimentation suffisament longue donnent des résultats encourageants.

Références

[1] J. Ledlie, J. Shneidman, M. Seltzer, et J. Huth. Scooped again. Dans *2nd International Workshop on Peer-to-Peer Systems (IPTPS'03)*, Berkeley, CA, USA, February 2003.

[2] Globus. `http://www.globus.org`.

[3] Seti@home. `http://setiathome.ssl.berkeley.edu`.

[4] J. Noga et P. Valiron. Explicitly correlated coupled cluster r12 calculations. Dans *Computational Chemistry : Reviews of Current Trends*, 2002.

[5] OAR. `http://oar.imag.fr`.

[6] *PBS Technical Overview*. http ://www.openpbs.org/overview.html.

[7] D. Thain et M. Livny. Condor and the grid. Dans Fran Berman, Anthony J.G. Hey, et Geoffrey Fox, éditeurs, *Grid Computing : Making The Global Infrastructure a Reality*. John Wiley, 2003.

[8] I. Foster, C. Kesselman, J. Nick, et S. Tuecke.. Grid services for distributed systems integration. *IEEE Computer*, 35(6), 2002.

[9] Derrick Kondo, Michela Taufer, III Charles L. Brooks, Henri Casanova, et Andrew Chien. Characterizing and evaluating desktop grids : An empirical study. Dans *Proceedings of International Parallel and Distributed Processing Symposium (IPDPS 2004)*. IEEE Press, 2004.

[10] A. Chien, S. Marlin, et S. Elbert. Ressource management in the Entropia system. Dans *Grid Resource Management : State of the Art and Future Trends*, pages 431–450, 2003.

[11] O. Lodygensky, G. Fedak, V. Néri, A. Cordier, et F. Capello. Auger & xtremweb : Monte carlos computation on a global computing platform. Dans *Computing in High Energy and Nuclear Physics*, 2003.

[12] R. Medeiros, W. Cirne, F. Brasileiro, et J. Sauvé. Faults in grids : Why are they so bad and what can be done about it ? Dans *4th International Workshop on Grid Computing (Grid 2003)*, November 2003.

[13] Andréas Johansson et Neeraj Suri. Error propagation profiling of operating systems. Dans *DSN '05 : Proceedings of the 2005 International Conference on Dependable Systems and Networks (DSN'05)*, pages 86–95, Washington, DC, USA, 2005. IEEE Computer Society.

[14] Yinglung Liang, Yanyong Zhang, Anand Sivasubramaniam, Ramendra K. Sahoo, José E. Moreira, et Manish Gupta. Filtering failure logs for a bluegene/l prototype. Dans *DSN '05 : Proceedings of the 2005 International Conference on Dependable Systems and Networks (DSN'05)*, pages 476–485, Washington, DC, USA, 2005. IEEE Computer Society.

[15] *Esterel*. `http://www-sop.inria.fr/esterel.org`.